MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-14434-TDI/1133

APRENDIZAGEM POR REFORÇO NA ADAPTAÇÃO A OBSTÁCULOS EM NAVEGAÇÃO ROBÓTICA AUTÔNOMA NÃO-ESTRUTURADA BASEADA EM IMAGENS

Leandro Toss Hoffmann

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. José Demisio Simões da Silva, aprovada em 23 de fevereiro de 2006.

INPE
São José dos Campos
2006

681.03.019

Hoffmann, L. T.
Aprendizagem por reforço na adaptação a obstáculos em navegação robótica não-estruturada baseada em imagens / Leandro Toss Hoffimann. – São José dos Campos: INPE, 2006.
212p. ; (INPE-14434-TDI/1133)

1.Inteligência artificial. 2.Aprendizagem de máquina. 3.Visão computacional. 4.Navegação autônoma. 5.Robótica. 6.Redes Neurais. I.Título.

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de **Mestre** em

Computação Aplicada

Dr. Lamartine Nogueira Frutuoso Guimarães

Presidente / IEAv/CTA / SJCampos - SP

Dr. José Demisio Simões da Silva

Orientador(a) / INPE / SJCampos - SP

Dr. Rafael Duarte Coelho dos Santos

Membro da Banca / INPE / SJCampos - SP

Dr. Fernando Santos Osório

Convidado(a) / UNISINOS / São Leopoldo - RS

Aluno (a): **Leandro Toss Hoffmann**

São José dos Campos, 23 de fevereiro de 2006

"In memories we were rich. We had pierced the veneer of outside things. We had 'suffered, starved, and triumphed, grovelled down yet grasped at glory, grown bigger in the bigness of the whole.' We had seen God in His splendours, heard the text that Nature renders. We had reached the naked soul of man."

"Em memórias estávamos ricos. Tínhamos penetrado além do verniz superficial das coisas. Tínhamos 'sofrido, passado fome e triunfado, sido humilhados mas visto a glória, e crescido na grandeza do todo'. Tínhamos visto Deus em Seus esplendores, ouvido o texto que a Natureza segue. Tínhamos atingido a alma nua do homem."

Sir Ernest Shackleton
em *"South"*, 1914-1917
(descrevendo o final da travessia da Georgia do Sul)

AGRADECIMENTOS

Gostaria que esta página ultrapassasse o simples âmbito formal para um sincero registro da gratidão por aqueles que contribuíram para que este trabalho fosse concluído.

Nestes quase três anos de caminhada no programa de Pós-graduação em Computação Aplicada em Ciências e Tecnologia Espaciais, tenho a agradecer, não somente pelo crescimento profissional, mas também pelas novas amizades conquistadas.

A Deus, que me permitiu chegar ao INPE e aqui permanecer para continuar minha jornada.

Sem o suporte material e espiritual que a família proporciona, as dificuldades seriam ainda maiores. Por isso, agradeço aos eternos parceiros: minha noiva Laura, minha mãe Goret, meu irmão Cleber e minha pseudo-avó Iti. *In memoriam*, agradeço ao meu pai Onero pelo seu entusiasmo em vida, que serve de exemplo para superar os desafios.

Ao meu orientador Dr. José Demisio Simões da Silva por sua dedicação em ensinar e pela oportunidade em compartilhar de sua sabedoria.

Aos amigos e colegas da UNISINOS, Antonio Gabriel Rodrigues, Etiene Pozzobom Lazzeris Simas, Fauzi Taha da Cruz, Fernando Dapper, Hisham Hashem Muhammad, José Luis Turchiello e Márcio Oliveira, com quem pude trocar experiências, discutir dúvidas e amadurecer idéias. Também aos professores Dr. Arthur Torgo Gómez, que me mostrou os caminhos até o INPE, a ao Dr. Fernando Santos Osório, que me dispertou o gosto pelas Redes Neurais Artificiais e aceitou o convite para participar da banca.

Aos demais professores da banca, Dr. Rafael Duarte Coelho Santos e Dr. Lamartine Nogueira Frutuoso Guimarães, do qual tive o privilégio de ter sido aluno.

Ao amigo Cleber Rubert, pelo seu companherismo nos dias de luta do mestrado, desde o segundo dia de São José dos Campos. Juntos transformamos experimentos de Visão Computacional em passa-tempos para aliviar o estresse.

Aos demais colegas de curso de computação aplicada, com quem dividi horas de muito estudo e confraternização. Em especial a Isabela Drummond, que muito me

ajudou com a Teoria dos Conjuntos Nebulosos, e a Ana Paula Abrantes de Castro, pelo empréstimo do carrinho de controle remoto, utilizado nos experimentos com o robô.

Ao João Emile Louis, amigo que periodicamente me resgatava do laboratório para um café e avaliar o andamento do mestrado.

À Tâmara Chagas Machado, por sua hospitalidade na recepção dos novos alunos da computação e pelo apoio na adaptação na cidade.

Pelos novos colegas de trabalho da Divisão de Desenvolvimento de Sistemas de Solo, que compreenderam meu silêncio neste início de ano e me apoiaram no término deste trabalho.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, pela bolsa que tive a chance de receber, viabilizando meus estudos.

RESUMO

Nas últimas décadas, a robótica tem desempenhado um papel importante na sociedade, com participação de destaque na indústria de manufatura de bens. Mais recentemente, aplicações de robôs móveis, desde simples brinquedos até a exploração de outros planetas, têm demonstrado o quão promissor o uso dessas ferramentas será num futuro próximo. Contudo, atualmente o custo e a complexidade de construção de robôs móveis, que sejam suficientemente flexíveis e ao mesmo tempo úteis, têm sido uma barreira para sua ampla disseminação. Neste sentido, técnicas de Inteligência Artificial vêm sendo freqüentemente estudadas, a fim de dotar os sistemas robóticos com capacidades de aprendizado, adaptação e autonomia. Este trabalho apresenta um estudo de aprendizagem de máquina, aplicado a navegação autônoma em robótica móvel. O objetivo principal é avaliar o desempenho de técnicas de aprendizagem por reforço, no uso de robôs móveis de baixo custo e baixa precisão, equipados com sensores de visão computacional. Para tanto, um robô foi modelado à luz de uma arquitetura de agente de aprendizagem, para através de imagens obtidas por uma câmera *Charge-Coupled Device* (CCD), ser capaz de aprender a navegar de forma autônoma, em ambientes internos não-estruturados. Os operadores de visão computacional são construídos com Redes Neurais Artificiais e algoritmos de rotulação de imagem, que identificam objetos diferenciados por suas características radiométricas. As posições relativas dos objetos na imagem são utilizadas para definir o estado do agente, que através da experimentação de ações, aprende a otimizar o seu processo de tomada de decisão. A implementação da arquitetura do agente de aprendizagem é suportada pelo protótipo de sistema *Cool Autonomous Navigation Enterprise with Learning Agents* (Canela), que viabilizou principalmente a conexão de sensores CCDs e a condução dos experimentos. Duas modelagens com aprendizado por reforço foram desenvolvidas, utilizando o algoritmo *Q-learning*, sendo o intuito da primeira prover uma navegação simples, evitando-se obstáculos e da segunda a exploração homogênea do ambiente. Uma série de ensaios foram realizados em um ambiente real, para validar a primeira modelagem. Os resultados obtidos demonstraram a capacidade de aprendizagem do agente, que navegou por um ambiente inicialmente desconhecido. Com a segunda modelagem, validada em experimentos de simulação, foi possível avaliar o bom desempenho de um sistema de navegação mais complexo, orientado a multi-objetivos. Os resultados encorajam o uso da modelagem de sistemas de navegação de robôs móveis, baseados em técnicas de aprendizado por reforço, proporcionando uma alternativa interessante aos métodos de programação tradicionais.

REINFORCEMENT LEARNING FOR OBSTABLE AVOIDANCE IN IMAGE BASED ROBOTIC AUTONOMOUS NON-STRUCTURED NAVIGATION

ABSTRACT

In the last decades, robotic has become an important role for society, specially in the manufacturing industry. In recent times, applications of mobile robots, from simple toys to planets explorations, has shown how promising will be the use of these tools in a close future. Though, the cost and complexity in developing mobile robots nowadays, which should be sufficiently flexible and still helpful, has been a difficulty for its broad deployment. Hence, Artificial Intelligence techniques has been studied frequently, aiming for introducing a learnable, flexible, and autonomous behavior to robotic systems. This work presents a machine learning study, applied to autonomous navigation in mobile robotic. The main goal is to analyze reinforcement learning techniques performance, when using low cost and low accuracy mobile robots, with on-board computer vision sensors. Thus, a robot was modeled as a learning agent architecture, which is able to learn to navigate autonomously in indoor non-structured environments, using images taken by a Charge-Coupled Device (CCD) camera. Computer vision operators are made with Artificial Neural Networks and image labeling algorithms, to recognize objects by its spectral features. Relative positions from image's objects are used to define the agent state, which experiments actions, and learn to optimize its decision making process. The learning agent architecture implementation is supported by a prototype system, called Cool Autonomous Navigation Enterprise with Learning Agents (Canela), which holds the CCDs sensors connections and manages the experiments. Using the *Q-learning* algorithm, two reinforcement learning based models were developed. The first model aims to build a simple obstacle avoidance navigation system, and the second an environment's homogeneous exploration navigation system. To evaluate the first model, a series of experiments were conducted in a real environment. The results has shown agent's learning capabilities on obstacle avoidance and navigation in an unknown environment. The second model was evaluated by simulation experiments whose brought good results on applying a more complex and multi-goal oriented navigation system. The results encourage to use reinforcement learning based models in mobile robots navigation systems, bringing an interesting choice to traditional programming methods.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

3D	–	Três dimensões, tridimensional
ADF	–	Algoritmo Detector de Faixas
ALVINN	–	*Autonomous Land Vehicle in a Neural Network*
ANSI	–	*American National Standards Institute*
API	–	*Application Program Interface*
ASCII	–	*American Standard Code for Information Interchange*
CCD	–	*Charge-Coupled Device*
COA	–	Centro de Área (*Center of Area*)
FA	–	Função de Ativação
FIS	–	Sistema de Inferência Nebulosa (*Fuzzy Inference System*)
GPS	–	Sistema de Posicionamento Global (*Global Positioning System*)
IA	–	Inteligência Artificial
ID	–	Identificador
MC	–	Monte Carlo
ML	–	Aprendizagem de Máquina (*Machine Learning*)
MSE	–	Erro Quadrático Médio (*Mean Squared Error*)
MLP	–	*Perceptrons* de Múltiplas Camadas (*Multilayer Perceptrons*)
NTSC	–	*National Television System Committee*
PD	–	Programação Dinâmica
RALPH	–	*Rapidly Adapting Lateral Position Handler*
RBF	–	Função de Base Radial (*Radial Basis Function*)
RCA	–	*Radio Corporation of America*
RF	–	Rádio Freqüência
RGB	–	Sistema de cores *Red*, *Green* e *Blue*
RL	–	Aprendizado por Reforço (*Reinforcement Learning*)
RNA	–	Redes Neurais Artificiais
RMSE	–	Raiz do Erro Quadrático Médio (*Root Mean Squared Error*)
SOM	–	Mapas Auto-organizáveis (*Self-Organizing Maps*)
TD	–	Diferença Temporal (*Temporal Diference*)
TKS	–	Takagi-Sugeno
UAV	–	Veículos Aéreos Não Tripulados (*Unmanned Aerial Vehicles*)

LISTA DE SÍMBOLOS

a	–	ação pertencente ao conjunto A
A	–	conjunto de ações
c	–	coordenada de coluna de uma imagem digital
C	–	número de colunas na grade de uma imagem digital
CS	–	conjunto de neurônios que pertencem a camada de saída de uma rede MLP
d	–	saída desejada
E	–	conjunto de termos lingüísticos
$\mathcal{E}$	–	Soma dos erros quadráticos instantâneo de uma rede neural
f	–	distância focal
F	–	conjunto nebuloso
g	–	região de um terreno representada por coordenadas nebulosas
gM	–	região de um terreno, em coordenadas nebulosas, que melhor aproxima a posição do agente
$iDstVt$	–	Distância de Visitação
$iInVt$	–	Índice de Visitação
$iMdVt$	–	Média de Visitação do Terreno
$iNVt$	–	Nível de visitação
$iTxVt$	–	Taxa de Visitação
iVt	–	Número de Visitas em uma determinada região do mapa nebuloso
I	–	imagem digital 2D
K	–	níveis de intensidade de cinza de uma imagem
L	–	um tipo de função de pertinência que define um conjunto nebuloso
M	–	regra semântica que mapeia termos lingüísticos para conjuntos nebulosos
N_4	–	vizinhança-4 de um *pixel*
N_8	–	vizinhança-8 de um *pixel*
Nu	–	núcleo do conjunto nebuloso F
O	–	região conexa de *pixels*
O_A	–	área da região O
O_{bm}	–	coordenada r do *pixel* mais inferior na região O
O_{Ci}	–	índice de circularidade de uma região O
O_{lm}	–	coordenada c do *pixel* mais a esquerda da região O
O_{Me}	–	Maior eixo (vertical ou horizontal) de uma região O
O_P	–	Perímetro de uma região O
$O_{\|P\|}$	–	Comprimento do perímetro de uma região O
O_{rm}	–	coordenada c do *pixel* mais a direita da região O
$O(\bar{r}, \bar{c})$	–	centróide da região O

O_{tm} – coordenada r do *pixel* mais ao topo da região O
p – elemento de uma imagem, *pixel*
P – polígono
P_A – área do polígono P
P_{bm} – coordenada r do ponto mais inferior do polígono P
P_{Ci} – índice de circularidade do polígono P
P_{lm} – coordenada c do ponto mais a esquerda do polígono P
P_{Me} – Maior eixo (vertical ou horizontal) do polígono P
$P_{|P|}$ – Comprimento do perímetro do polígono P
P_{rm} – coordenada c do ponto mais a direita do polígono P
$P(\bar{r}, \bar{c})$ – centróide do polígono P
P_{tm} – coordenada r do ponto mais ao topo do polígono P
Pt – ponto
Q – função de Q-valores
$\mathfrak{r}$ – recompensa ou reforço
r – coordenada de linha de uma imagem digital
rt – reta
R – número de linhas na grade de uma imagem digital
s – estado pertencente ao conjunto S
S – conjunto de estados
sg – segmento
Su – suporte do conjunto nebuloso F
t – instante de tempo
T – modelo de transição de estados
u – elemento de um conjunto nebuloso F
v – ativação interna de um neurônio
V – função de valores de estados
w – peso sináptico de um neurônio
x – dado de entrada
y – valor de saída de um neurônio ou rede neural
z – distância
z^* – saída precisa de um controlador *fuzzy*
α – taxa de aprendizado
$\hat{\alpha}$ – nível de corte do conjunto nebuloso F
β – *bias* do neurônio
χ – variável lingüística (variável *fuzzy*)
ϵ – (ϵ-*greedy*) probabilidade de seleção aleatória de ações
γ – fator de desconto
Γ – um tipo de função de pertinência que define um conjunto nebuloso
κ – índice Kappa global de classificação

λ – distância de rastreio de valores visitados
Λ – função de pertinência triangular que define um conjunto nebuloso
φ – função de ativação do neurônio
μ – média
π – política de escolha de ações
Π – função de pertinência trapezoidal que define um conjunto nebuloso
ρ – raio de vizinhança topológica de um SOM
σ – desvio padrão
τ – um parâmetro de *temperatura*
ω – parâmetro de formato de uma função de pertinência
Ω – universo de discurso

CAPÍTULO 1

INTRODUÇÃO

A construção de máquinas que imitam o comportamento humano e possam servir a sociedade das mais variadas formas é um sonho antigo de nossa civilização. Já na década de 50, o russo Isaac Asimov mexia com o imaginário popular, através de suas obras de ficção científica que popularizaram o termo *robô*, como em *Eu, robô* (ASIMOV, 1950), em que foram introduzidas as *Leis da robótica*. Ainda no ramo da ficção, Arthur Clarke discute as conseqüências para a humanidade da criação de um computador dotado de inteligência, na famosa obra *2001: Uma Odisséia no Espaço* (CLARKE, 1975). Com o desenvolvimento da computação e a construção de máquinas cada vez mais eficientes e poderosas computacionalmente, na metade da década de 50, nasce uma nova linha de pesquisa na computação, batizada como Inteligência Artificial (IA). Com ela, o sonho de criar uma máquina inteligente, com a capacidade de imitar o comportamento humano, passou a ser abordado por métodos científicos. Todavia, logo se descobriu que o comportamento inteligente humano era muito mais complexo do que se imaginava, e sistemas ditos inteligentes, restringem-se atualmente a aplicações específicas (BITTENCOURT, 2001).

Assim, de uma forma mais prática e realista, várias linhas de pesquisa vêem procurando desenvolver técnicas que possam ser aplicadas a robótica, despertando interesse não somente em áreas tradicionais como engenharia (AKBARZADEH-T *et al.*, 2000) e matemática (FUJIE *et al.*, 1996), mas também em biologia (DAMPER *et al.*, 2000) e (IIDA, 2003), psicologia (MATARIC, 1998), medicina (CASALS, 1999; HOWE; MATSUOKA, 1999; RODRIGUES *et al.*, 2004), entre outras. As aplicações mais fascinantes talvez sejam as que envolvam exploração de locais inóspitos e que apresentam risco de vida a humanos, como por exemplo, crateras de vulcões ativos, regiões geladas, águas profundas, ambientes de energia nuclear, ou até mesmo outros planetas e regiões do espaço (AHLF *et al.*, 2000; SINGH *et al.*, 2000; SCHILLING *et al.*, 2002). Recentemente, por exemplo, um audacioso projeto da Agência Espacial Norte-americana (*National Aeronautics and Space Administration* - NASA) enviou duas sondas, *Spirit* e *Opportunity*, numa viagem interplanetária à Marte, onde dois robôs realizam tarefas de coleta de dados no território marciano (Figura 1.1). Este projeto exemplifica como o desenvolvimento de robôs pode ser bem sucedido, mesmo em se tratando de ambientes extremamente inóspitos (NASA, 2004).

FIGURA 1.1 - Ilustração do robô *Opportunity* sobre o solo marciano.
FONTE: NASA/JPL (NASA, 2004).

Por outro lado, aplicações no cotidiano de produção de bens também são promissoras. Neste caso, cita-se o uso de robôs autônomos para manipulação, transporte ou inspeção de materiais, em ambientes industriais (GONZÁLEZ-GALVÁN *et al.*, 2003; YASUDA, 2003).

De uma forma geral, os requisitos dos sistemas de robótica envolvem o estudo de sensores, atuadores, dispositivos de *hardware*, algoritmos de planejamento e otimização de rotas, aprendizado de máquina, representação de mapas, tolerância a falhas, controle de colisões, sistemas de navegação e eventualmente técnicas de cooperação entre robôs, entre outras disciplinas. Isto tudo para que o robô seja capaz de responder perguntas, aparentemente simples, como: Onde estou? Para onde devo ir? O que faço para chegar até lá?

Teoricamente, o problema de navegação de um robô móvel consiste em encontrar uma função de mapeamento que recebe como entrada os dados dos sensores e produz comandos que serão enviados aos atuadores do veículo, podendo inclusive ser abordado como um problema inverso (SURMANN *et al.*, 1995). Na prática, mesmo com o aumento do poder computacional dos processadores, dependendo da complexidade do problema, o aumento de dados a serem processados torna o uso de modelos matemáticos precisos inviáveis. Uma das alternativas então é o uso de técnicas emergentes, como Computação Inteligente (*Soft Computing*), que requerem menor poder computacional para processamento (YAGER; ZADEH, 1994).

Quando o sistema, dito computacionalmente inteligente, visa tomar decisões através dos seus próprios sistemas de inferência, e ao mesmo tempo é móvel, costuma-se

encontrar o termo *navegação autônoma*, como tema de pesquisa. A navegação autônoma consiste na habilidade de um robô se locomover num ambiente previamente conhecido ou desconhecido, sem que haja intervenção humana, com a finalidade de atingir objetivos e desenvolver determinadas tarefas, de forma segura (BORENSTEIN *et al.*, 1996; FIGUEIREDO, 1999; HEINEN, 2001).

Nas últimas décadas, um grande número de trabalhos que tratam o problema de navegação autônoma, tem utilizado visão artificial como o seu principal sensor. Esses trabalhos podem ser divididos em duas grandes frentes, as que englobam aplicações em ambientes internos ou em ambientes externos (DESOUZA; KAK, 2002). Parte desse sucesso deve-se ao baixo custo de câmeras *Charge-Coupled Device* (CCD), mas talvez a maior motivação seja o grande potencial de extração de informações, que esses dispositivos oferecem. Imitando muitas vezes parte do funcionamento do olho humano, operadores de visão computacional permitem também obter informações tridimensionais (3D) a cerca dos objetos no mundo (SILVA, 1999; SILVA; SIMONI, 2003).

Talvez um dos mais famosos trabalhos de navegação autônoma, utilizando visão computacional, tenha sido o projeto "No Hands Across America!" (The Robotics Institute, 1995), na qual o programa *Rapidly Adapting Lateral Position Handler* (RALPH) (POMERLEAU, 1995) guiou autonomamente a direção de um veículo em 98% do trajeto de 2849 milhas (4584 km), da cidade de Pittsburgh até San Diego, nos Estados Unidos. Liderado por Dean Pomerleau, a equipe que desenvolveu o RALPH utilizou como base para o experimento o sistema NavLab. O NavLab consiste em um laboratório móvel para experimentos de navegação autônoma em ambientes externos, no qual veículos são adaptados para incorporar sistemas computacionais e diversos tipos de sensores. Os estudos do NavLab tiveram início por volta de 1986, por pesquisadores da Carnegie Mellon University, interessados em navegação autônoma em ambientes abertos, auxiliada por visão computacional.

Anteriormente ao RALPH, Pomerleau já havia também criado o programa baseado em redes neurais *Autonomous land vehicle in a neural network* (ALVINN) (JOCHEM; POMERLEAU, 1996), que a partir de um prévio treinamento supervisionado, era capaz de guiar um veículo em vários tipos de terreno, usando o sistema de visão computacional do NavLab.

Mais recentemente, em 1999, o artefato espacial *Deep Space 1*, voando a 96.500 mil

km da Terra, foi comandado autonomamente por um agente inteligente durante 2 dias (MUSCETTOLA *et al.*, 1998; BERNARD *et al.*, 2000). Um agente é uma entidade que percebe e atua sobre seu ambiente, dotado de um mecanismo de inferência que em alguns modelos o permite avaliar e otimizar suas ações (RUSSEL; NORVIG, 2004). No *Deep Space 1* o agente era orientado a objetivos, e foi capaz de lidar com imprevistos, planejar, escalonar e executar operações, viabilizando uma maior independência de operações solo-bordo. Construído pela NASA especificamente para testar novas tecnologias promissoras, a sonda faz parte de um programa de testes *New Millennium Program* (NASA, 2005). Neste programa, a NASA aponta um possível caminho para a área de navegação autônoma, em aplicações espaciais.

1.1 Motivação e Justificativa

Seguindo a linha de aplicações no estilo NavLab, vários autores propuseram trabalhos de navegação autônoma, que visam guiar um veículos sobre uma pista estruturada, ou auxiliar o motorista humano (LI *et al.*, 1997; WEISSER *et al.*, 1999; SCHMIDT *et al.*, 2000; CASTRO *et al.*, 2001; LI *et al.*, 2003; JUNG; KELBER, 2005).

Neste trabalho, a motivação com o uso de visão computacional aplicada a navegação de robôs móveis surgiu com o trabalho de Castro *et al.* (2001), que apresentou um controlador nebuloso para guiar um robô sobre uma pista. Conforme descrito pelos autores, um veículo autônomo, equipado com uma câmera digital, captura imagens de uma pista sinalizada com faixas esquerda e direita. O robô percorre a pista, guiando-se pelas faixas, que tem suas distâncias angulares estimadas por um operador visual por gradiente, através da imagem capturada. Inicialmente, a trajetória do veículo é determinada por um sistema de controle baseado em inferência nebulosa, que recebendo como entradas as direções angulares das faixas esquerda e direita, determina a direção que o veículo deve seguir. O trabalho mostra a viabilidade do uso de controladores inteligentes para sistemas de navegação autônoma que usam sensores visuais de baixo custo e precisão.

A arquitetura do sistema de navegação utilizado nos experimentos é composta dos seguintes módulos:

Sensores - Compostos pela câmera digital, que captura imagens da pista à frente do robô. As imagens são transmitidas por Rádio Freqüência (RF), codificadas em NTSC (National Television System Committee), e capturadas pelo *software* Snappy ®, que cria arquivos no formato JPEG;

Operador de visão computacional - Abre as imagens a serem processadas, detecta as faixas da pista e fornece suas direções angulares;

Sistema de controle de trajetória - Módulo responsável pela tomada de decisão na trajetória a ser tomada pelo veículo, sendo as saídas, produzidas por controladores de IA, os valores de direção e velocidade;

Sistema de envio de comandos ao veículo - Efetua um mapeamento dos comandos enviados pelo sistema de controle para sinais do universo dos atuadores. Os sinais são então enviados aos atuadores por RF, através de uma interface em *hardware*;

Atuadores - Realizam o movimento do veículo, provendo torque das rodas e controlando o sentido e direção do movimento.

Utilizando a mesma arquitetura, em Castro *et al.* (2001), o sistema de controle foi aprimorado, incorporando-se novas regras ao sistema de inferência nebuloso, que determina não somente a direção mas também a velocidade a ser tomada pelo robô. O controle da velocidade, permitiu criar políticas de controle para diferentes situações previstas na pista, como curvas acentuadas, curvas pouco acentuadas ou retas, refinando o processo de tomada de decisão do controlador.

Mais tarde, em Hoffmann *et al.* (2004b), Hoffmann *et al.* (2004a) novas melhorias foram propostas, levando a modificações nas superfícies de controle dos sistemas de inferência nebulosa. Além disto, avaliou-se o uso de sistemas de navegação baseados em Redes Neurais Artificiais (RNA), treinadas a partir do controlador nebuloso previamente desenvolvido. Para o módulo *Operador de Visão Computacional* foi proposto um novo operador visual baseado em autômatos finitos, mais tolerante a ruídos, que possibilitou a extração de características da pista com maior acurácia. Modificações no sistema de envio de comandos ao robô também foram sugeridas, facilitando sua implementação por meio de RNAs.

A Figura 1.2 mostra o ambiente de navegação do robô utilizado nos trabalhos de Castro *et al.* (2001), Hoffmann *et al.* (2004b), Hoffmann *et al.* (2004a). Observa-se na Figura 1.2-a o carrinho de controle remoto, equipado com uma câmera digital que deve percorrer uma pista de aproximadamente 7 metros, entre duas faixas delimitadoras. Percebe-se, através da Figura 1.2-b, que a pista imita uma estrada, por onde um motorista guia seu automóvel. Na Figura 1.2-c nota-se ainda a imagem capturada pela câmera do robô, posicionado no início da pista.

(a) Robô e sua câmera montada. (b) Imagem do veículo posicionado no ponto inicial da pista. (c) A respectiva imagem capturada da pista.

FIGURA 1.2 - Experimentos com robô autônomo que navega sobre uma pista com faixas delimitadoras.

Os experimentos descritos permitiram validar um sistema de controle para navegação autônoma, que combinou o uso de diferentes técnicas de Inteligência Artificial, aprendizado de máquina e visão computacional. Todavia, o ambiente de controle no qual os experimentos foram realizados é um ambiente preparado, bem estruturado, assumindo-se que nenhum imprevisto irá ocorrer. Assim, situações inesperadas, como por exemplo, um obstáculo na pista ou o cruzamento de outro veículo, não são tratadas, ficando fora do escopo dos trabalhos desenvolvidos.

Essa limitação do robô, quanto a estruturação do ambiente, fica clara, pois o planejamento de sua trajetória está diretamente ligado a interpretação das faixas da pista, existindo pouca robustez no sistema de navegação para situações que exijam extração de novas informações da cena. Trata-se de uma navegação reativa às características das faixas da pista. Pode-se inclusive modelar o sistema de navegação do veículo utilizado nos experimentos como um agente reativo.

A literatura tem mostrado que não é possível se obter um bom sistema de navegação suficientemente robusto que seja unicamente reativo ao ambiente. Isto ocorre devido à natureza da maioria dos objetivos dos veículos autônomos envolver planejamento de trajetória, mas ao mesmo tempo evitar colisões (HEINEN, 2001; SONG; SHEEN, 2000; SURMANN *et al.*, 1995; SURMANN *et al.*, 1996). Essas duas atitudes podem gerar conflitos de objetivos, pois ao passo que o robô evita um obstáculo, pode ir contra ao objetivo de sua trajetória. Assim, arquiteturas de controle que armazenam informações do ambiente, para que possam ser utilizadas em futuras inferências, são necessárias. Uma estratégia é combinar de forma híbrida o melhor das técnicas de controle reativo e deliberativo, em uma arquitetura em camadas, como pode ser visto no trabalho de Heinen (2001).

A modelagem de um robô autônomo, pela óptica de agentes, pode auxiliar na compreensão do seu papel no ambiente e suas limitações. No caso do robô utilizado nos experimentos descritos anteriormente, fica claro que seus sensores são operadores de visão computacional e os atuadores, os motores para controle de direção e sentido. Percebe-se também que a navegação bem sucedida do veículo está limitada ao ambiente bem estruturado, no qual o seu motor de inferência foi projetado para reagir. Inspirado em outros modelos de agentes, descritos em (RUSSEL; NORVIG, 2004), é possível desenvolver sistemas de navegação mais complexos, inseridos em ambientes pouco estruturados ou não estruturados.

Contudo, é importante que a escolha do tipo de arquitetura de agente esteja adequada as necessidades impostas pelo problema abordado. Em um ambiente de navegação autônoma, sobretudo naqueles não estruturados, uma característica desejável é a capacidade de aprendizagem e adaptação do agente. Uma das técnicas que provê este tipo de comportamento, que vem sendo comumente experimentada em ambientes de navegação robótica, é a de aprendizagem por reforço (MACEK *et al.*, 2002; HAGEN; KRÖSE, 2003; ESTEVES *et al.*, 2004; WEBER *et al.*, 2004; MARTÍNEZ-MARŃ; DUCKETT, 2005; MICHELS *et al.*, 2005). Neste tipo de aprendizagem, um agente é estimulado ou inibido a realizar determinadas ações, a medida que interage com o ambiente e recebe recompensas. Após um determinado número de iterações, o agente aprende a otimizar suas ações e eventualmente se adaptar as mudanças do ambiente. Todavia, a grande maioria das aplicações estão limitadas a experimentos em ambientes simulados, ou ao uso de equipamentos (robôs ou sensores) sofisticados.

1.2 Objetivos e Métodos

Motivado pelo baixo custo de câmeras CCDs e por técnicas de aprendizagem que possibilitam um agente se adaptar a ambientes desconhecidos, o **objetivo principal** deste trabalho consiste em desenvolver um agente de aprendizagem, para navegação de um robô móvel, em ambientes não-estruturados, utilizando como sensores, operadores de visão computacional. Os experimentos de navegação serão realizados com um robô real de baixo custo e precisão nos atuadores, equipados exclusivamente por uma câmera digital e um transmissor de rádio freqüência.

Visa-se ao final do trabalho, obter um programa que incorporará uma arquitetura de agente, inteligente do ponto de vista computacional, que esteja preparado para navegar em ambientes desconhecidos, com obstáculos imóveis, tolerante a ruídos de

sensores e atuadores.

Durante a exploração do ambiente, o robô móvel assume um ciclo de percepção do ambiente, tomada de decisão e locomoção, permitindo um constante aprendizado e adaptação a novas experiências, com melhoria do processo de escolha de ações. Este mecanismo é suportado por uma arquitetura de agente de aprendizagem.

Segundo Russel e Norvig (2004), agentes de aprendizagem são aqueles dotados com um tipo especial de programa de agente dividido em quatro elementos:

Elemento de aprendizado - responsável pela adaptação do agente, através do aperfeiçoamento de suas ações;

Elemento de desempenho - que realiza a tomada de decisão, definindo as ações que serão tomadas pelo agente;

Crítico - avalia as ações do agente e determina como o *elemento de desempenho* deverá ser modificado; e

Gerador de problemas - que cria novas situações para que o agente aprenda novas experiências.

A Figura 1.3 ilustra a arquitetura geral de um agente de aprendizagem e o relacionamento dos componentes do seu programa. Ressalta-se ainda na Figura 1.3, a sua interação com o mundo externo, realizada através dos seus sensores e atuadores, mas que também pode receber informações realimentadas por meio do *crítico*.

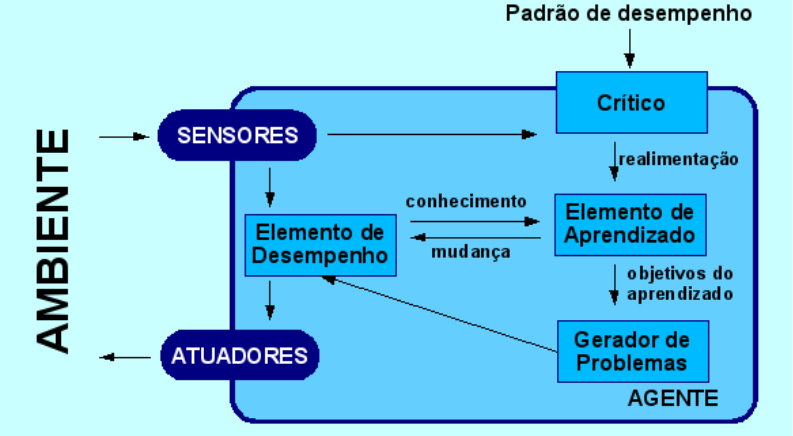


FIGURA 1.3 - Arquitetura de um agente de aprendizagem e os componentes do seu programa. FONTE: adaptada de (RUSSEL; NORVIG, 2004, p. 52).

Neste trabalho, busca-se incorporar técnicas de aprendizado por reforço aos elementos do programa do agente, avaliando-se sua capacidade não-supervisionada de mapeamento de percepções em ação.

Dotar o agente com capacidade de aprender e se adaptar a novas situações o tornará mais robusto para navegar em ambientes desconhecidos. Mesmo assim, um fator importante para o bom desempenho do agente consiste na escolha e adaptação dos sensores que constituirão sua arquitetura. Quando se emprega visão computacional, um grande número de operadores podem ser desenvolvidos para se extrair informações do ambiente. Se por um lado isto representa um farto volume de informações em potencial a serem usadas no processo de inferência do agente, por outro, o uso excessivo de operadores pode aumentar muito a complexidade do agente, sem garantir que características importantes da cena foram bem interpretadas. Além disto, sensores de visão computacional são especialmente sensíveis a ruído, uma vez que as imagens produzidas se alteram com pequenas variações na iluminação da cena. Por essas razões, a construção dos operadores será suportada por técnicas de IA, do tipo Redes Neurais Artificiais ou Lógica Nebulosa, que em geral têm boa capacidade de generalização, reduzindo assim o problema de ruídos.

Em virtude dos objetivos propostos neste documento, nota-se que o trabalho envolve várias disciplinas. Reunir técnicas de visão computacional, aprendizado de máquina e navegação autônoma numa arquitetura de agente, exige a busca por diferentes tecnologias de computação e interfaces de *software*.

1.2.1 Escopo do Trabalho

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

Além disto, os sistemas de navegação estarão restritos a ambientes internos e controlados, como por exemplo laboratórios ou escritórios, nos quais obstáculos imóveis estão arranjados. Os obstáculos serão compostos por objetos diferenciados do fundo de cena por suas características espectrais distintas.

Apesar das técnicas utilizadas serem tolerantes a ruídos, provenientes de sensores e atuadores, não se espera que o agente alcance uma otimização perfeita de suas ações.

1.3 Organização do Trabalho

O restante deste trabalho está organizado da seguinte maneira:

- *CAPÍTULO 2 - APRENDIZAGEM DE MÁQUINA:* Este capítulo introduz a disciplina de aprendizagem de máquina e discute seu uso em aplicações de robótica móvel. São apresentadas técnicas de aprendizado supervisionado e não-supervisionado com Redes Neurais Artificiais, lógica nebulosa e aprendizado por reforço. Cada secção traz uma revisão de aplicações em robótica móvel e assuntos correlatos.

- *CAPÍTULO 3 - VISÃO COMPUTACIONAL:* Neste capítulo, o processamento digital de imagens é formalizado, servindo como base para a construção de operadores de visão computacional. Dois métodos para detecção de objetos em uma cena são descritos: o primeiro com abordagem matricial, baseado em um algoritmo de rotulação; e o segundo com abordagem vetorial, que aproxima os objetos por polígonos. Um conjunto de atributos a serem extraídos dos objetos detectados são definidos. O cálculo da distância de objetos no ambiente, utilizando-se o modelo óptico da câmera, é analisado.

- *CAPÍTULO 4 - ARQUITETURA DO AGENTE:* Descreve os aspectos de modelagem do agente que será utilizado nos experimentos de navegação autônoma. Compara e determina quais métodos serão adotados nos subsistemas sensor e atuador. Sumariza o conjunto de parâmetros a serem definidos na arquitetura e apresenta o sistema computacional Canela, que dará suporte a implementação do agente de aprendizagem.

- *CAPÍTULO 5 - EXPERIMENTOS E RESULTADOS:* Este capítulo apresenta o conjunto de experimentos que foram realizados no âmbito de navegação autônoma, discutindo os resultados obtidos. Inicialmente analisa os parâmetros do algoritmo de aprendizagem de reforço, através de teste simples, passando então para ensaios com o robô móvel, em um ambiente físico real. Em seguida, experimentos simulados, objetivando a exploração de ambientes desconhecidos, são expostos.

- *CAPÍTULO 6 - CONCLUSÕES:* O último capítulo discursa sobre a relevância dos resultados obtidos e aponta as melhorias a serem aplicadas futuramente, a fim de se dar continuidade ao trabalho.

CAPÍTULO 2

APRENDIZAGEM DE MÁQUINA

Para melhor atender aos requisitos que a robótica impõe, é desejado que os robôs sejam flexíveis e dotados de um certo grau de inteligência. O desenvolvimento de técnicas que permitam que sistemas computacionais absorvam conhecimento e tomem decisões de forma automática é estudo da área de *Aprendizagem de Máquina* (ML, do inglês *Machine Learning*) (SHALOM; KULIKOWSKI, 1991). Simon (1983) define aprendizagem de máquina como *"mudanças num sistema que o permita realizar alguma tarefa ou tarefas descritas por alguma população mais eficientemente e mais eficazmente da próxima vez"*. Já em 1983, com várias publicações na área de ML, Carbonell *et al.* (CARBONELL *et al.*, 1983) propuseram que as técnicas fossem classificadas segundo seu objetivo em:

- estudos baseados em tarefas, que visavam alguma aplicação específica;
- simulação cognitiva, que procuravam entender e simular o processo de aprendizagem humano; e
- análises teóricas, com experimentos que validavam novos métodos e algoritmos, independentemente de aplicação.

Posteriormente, em 1990, Shavlik e Dietterich (SHAVLIK; DIETTERICH, 1990) reuniram uma série de trabalhos em ML, organizados em técnicas que aprendem por aquisição de conhecimento ou através de mudanças internas no sistema.

Técnicas de aprendizado por aquisição de conhecimento são conhecidas hoje como aprendizagem por indução (*inductive learning*), e normalmente se diferencia três tipos de métodos utilizados: *aprendizagem supervisionada*, *não-supervisionada* e *aprendizagem por reforço* (RUSSEL; NORVIG, 2004). Na aprendizagem supervisionada são fornecidos padrões de entrada para o sistema e as respectivas saídas que se deseja que ele aprenda e reproduza. No caso da aprendizagem não-supervisionada as saídas desejadas não são informadas e eventualmente o sistema identifica grupos de padrões (*clusters*), em função dos dados de entrada. A aprendizagem por reforço visa recompensar o sistema quando ele efetua uma ação correta, ou puní-lo quando o mesmo produz ações indesejadas.

Neste capítulo, três técnicas diferentes de aprendizagem de máquina são discutidas. A secção 2.1 trata de Redes Neurais Artificiais (RNA), arquiteturas compostas por unidades que imitam o funcionamento de neurônios, treinadas por algoritmos de aprendizagem supervisionada ou não-supervisionada. Em seguida, na secção 2.2, a Teoria dos Conjuntos Nebulosos é apresentada, servindo de base para construção de sistemas nebulosos. Por fim, na secção 2.3, algoritmos de aprendizado por reforço completam o conjunto de técnicas de ML utilizadas neste trabalho.

Cada uma das três técnicas apresentadas tem vantagens e características específicas em relação as demais. As RNA funcionam como aproximadores universais, freqüentemente utilizadas como classificadores. Os sistemas nebulosos permitem o aproveitamento do conhecimento de especialistas e modelam bem dados imprecisos. Alguns algoritmos de aprendizado por reforço são preferidos em aplicações nas quais não se conhece o modelo do ambiente, ou necessita-se de aprendizado *online*. Todos esses algoritmos de aprendizado têm sido largamente experimentados em aplicações de robótica móvel, como pode ser visto no final de cada secção. Além disto, observa-se na literatura uma tendência de combinar o que há de melhor em abordagens híbridas.

2.1 Redes Neurais Artificiais

Redes Neurais Artificiais (RNA) são sistemas computacionais compostos por vários elementos que se inter-relacionam de forma conexa, inspirados no funcionamento de neurônios biológicos (TSOUKALAS; UHRIG, 1997). Cada elemento consiste numa unidade de processamento conhecida como *neurônio*, unidade neural ou simplesmente um nó da rede. Os neurônios quando conectados entre si, por meio de ligações, formam redes. Dependendo do esquema de conexão dessas redes, diferentes *arquitetura* são definidas, das quais destacam-se as arquiteturas de redes em multi-camadas, com atalhos, recorrentes e de ordem superior (HAYKIN, 2001).

A Figura 2.1 ilustra o *Perceptron*, um neurônio artificial proposto por Frank Rosenblatt, inspirado no modelo *Psychon* de McCulloch e Pitts (HAYKIN, 2001). As percepções do ambiente chegam ao neurônio na forma de sinais de entrada x_i, que são inibidos ou estimulados por um peso sináptico w_i. A soma ponderada de todos os sinais de entrada determinará o estado interno do neurônio, conhecido por *ativação interna*, denotado por

$$v = \sum_{j=0}^{n} w_{kj} x_j, \tag{2.1}$$

onde, n é o número de entradas do neurônio.

Antes que o estado do neurônio seja propagado adiante, ele é aplicado a uma *função de ativação* (FA), também conhecida por *função de transferência* φ. A saída y de um neurônio k pode ser expressa por

$$y_k = \varphi(v_k). \tag{2.2}$$

O valor x_0 na Figura 2.1 é uma entrada especial que não faz parte das percepções externas do neurônio. Esta entrada funciona com um limiarizador (*bias*) e normalmente tem seu valor fixado em 1 ou -1. O peso da conexão *bias* algumas vezes é denotado por β.

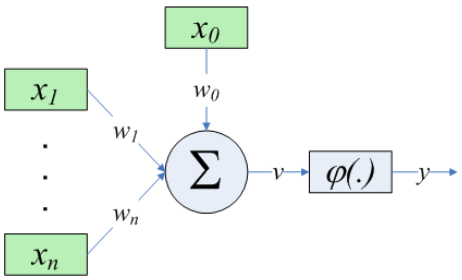


FIGURA 2.1 - Estrutura do neurônio artificial *Perceptron*.

Como função de ativação, várias opções podem ser utilizadas pelos neurônios, estando algumas ilustradas na Figura 2.2. A *função identidade* (Figura 2.2-a) é uma função linear que simplesmente propaga o valor de ativação interna do neurônio, ou seja,

$$f(x) = x. \tag{2.3}$$

A *função limiar* (Figura 2.2-b) binariza a saída do neurônio entre 0 e 1

$$f(x) = \begin{cases} 1 & \text{se } x \geq 0; \\ 0 & \text{se } x < 0; \end{cases} \tag{2.4}$$

As Figuras 2.2-c e 2.2-d representam funções sigmóides, conhecidas por seu formato em "S". Essas duas funções, conhecidas respectivamente por *função logística* e *fun-*

ção tangente hiperbólica, são comumente utilizadas em redes neurais que utilizam o algoritmo de treinamento *backpropagation* (apresentado a seguir no item 2.1.1.1), pois tem uma derivada de baixo custo computacional.

A *função logística*

$$g(x) = \frac{1}{1 + \exp(-\sigma x)} \tag{2.5}$$

limita os valores de saída dos neurônios entre 0 e 1, onde σ é o coeficiente de declividade. Sua derivada é

$$g'(x) = \sigma g(x)[1 - g(x)]. \tag{2.6}$$

De forma semelhante, a *função tangente hiperbólica*

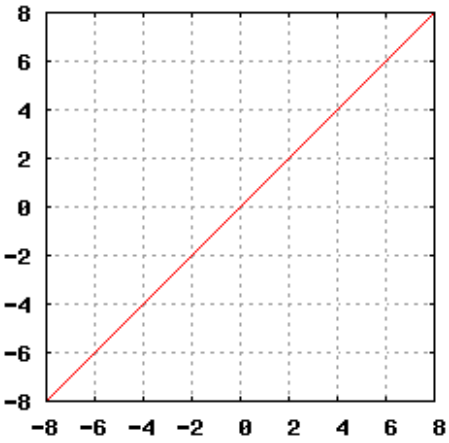
$$h(x) = \frac{1 - \exp(-\sigma x)}{1 + \exp(-\sigma x)} \tag{2.7}$$

limita as saídas dos neurônios entre -1 e 1, normalmente utilizada para dados de entrada binária (FAUSETT, 1994). Sua derivada consiste em

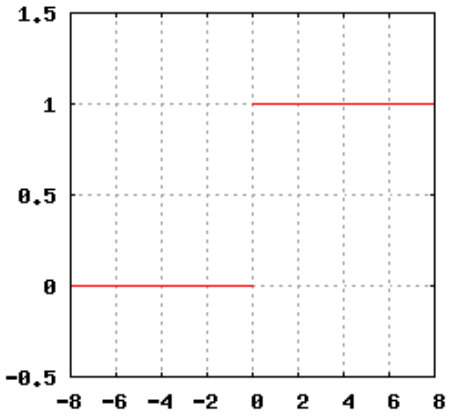
$$h'(x) = \frac{\sigma}{2}(1 - h(x)^2). \tag{2.8}$$

Quando utilizados de forma isolada, os neurônios não têm muito poder de processamento ou de armazenamento de informação. Todavia, várias arquiteturas de Redes Neurais Artificiais foram propostas, de forma que quando combinados, os neurônios possam resolver problemas complexos. Algumas dessas arquiteturas serão descritas nas próximas sub-secções.

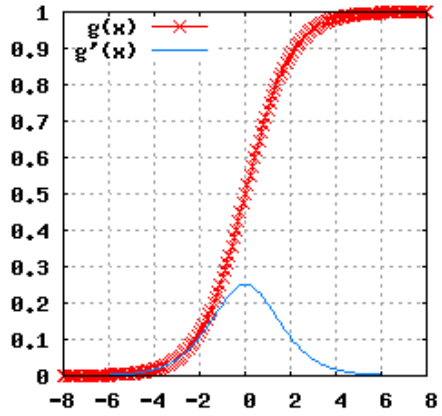
O processo de treinamento das RNA é realizado através do ajuste dos pesos de cada unidade neural. No processo indutivo de aprendizado de máquina, as RNA podem ser treinadas de forma supervisionada ou não-supervisionada. O aprendizado supervisionado é realizado fornecendo-se um par de dados à RNA: um dado de entrada e a respectiva saída desejada. A RNA utiliza o dado de entrada para produzir uma saída, que será comparada com a saída que se deseja aprender. Já os processos de aprendizado não-supervisionado, consistem na sua maioria em técnicas de agrupamento, onde os dados de entrada são organizados em classes.



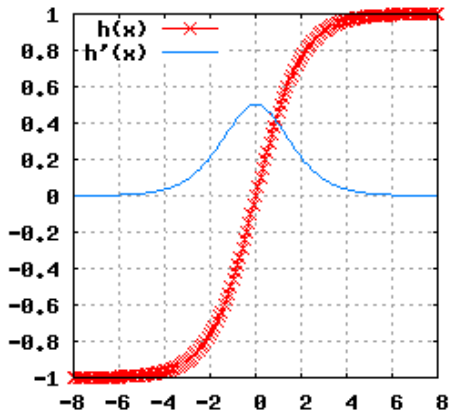
(a) Função Identidade.



(b) Função Limiar.



(c) Função Logística e sua derivada



(d) Função Tangente Hiperbólica e sua derivada

FIGURA 2.2 - Exemplos de Funções de Ativação utilizadas pelos neurônios.

2.1.1 *Perceptrons* de Múltiplas Camadas

A rede *Perceptrons* de Múltiplas Camadas (MLP, do inglês *Multilayer Perceptrons*) consiste num arranjo de várias unidades neurais *Perceptron*, normalmente dispostos em camadas de ativação (HAYKIN, 2001). Uma rede MLP, como mostra a Figura 2.3, é formada por uma camada de entrada, nenhuma ou várias camadas ocultas e uma camada de saída. As unidades da camada de entrada servem como unidades sensoriais para coleta dos dados externos, não realizando nenhum tipo de processamento, apenas propagando o sinal para a próxima camada. O uso de uma ou mais camadas ocultas é opcional, mas necessário quando aplicado a problemas linearmente não separáveis. Já a camada de saída é a interface da rede que fornece os seus resultados ao mundo externo. O número de unidades nas camadas externas (entrada e saída) é diretamente dependente da dimensionalidade dos dados de entrada e saída

da rede (FAUSETT, 1994). Quando o número de nodos e camadas estiverem adequados, as redes MLP funcionam como aproximadores universais, que podem realizar o mapeamento entre dois espaços de variáveis (HORNIK *et al.*, 1989).

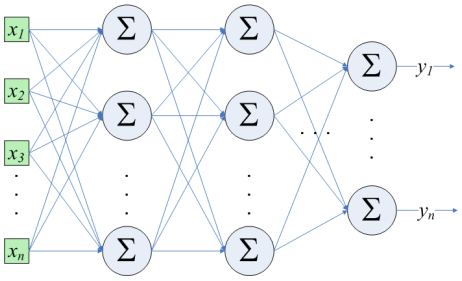


FIGURA 2.3 - Rede *Perceptrons* de Múltiplas Camadas.

A seguir, será descrito o *Algoritmo Backpropagation*, utilizado no processo de treinamento de redes MLP.

2.1.1.1 O Algoritmo *Backpropagation*

O Algoritmo de Retropropagação do Erro (*backpropagation*) é uma das abordagens mais conhecidas para se efetuar o ajuste dos pesos de uma rede MLP. Trata-se de um processo de aprendizagem supervisionada, em que compara-se a saída obtida pela rede MLP com a saída desejada fornecida, calculando-se assim o Erro Quadrático Total da rede. O algoritmo *backpropagation* é um método de descida do gradiente que visa minimizar este erro, através do ajuste dos valores dos pesos sinápticos da rede.

O ajuste do peso j do neurônio k é calculado como

$$w_{kj}(t+1) = w_{kj}(t) + \Delta w_{kj}(t), \tag{2.9}$$

onde $w_{kj}(t)$ é o peso j do neurônio k no instante t.

O termo Δw_{kj} é a correção do peso, calculado por um gradiente local do neurônio e ponderado pelo valor de entrada da ligação sináptica e por uma taxa de aprendizado. É genericamente definido como

$$\Delta w_{kj}(t) = \alpha \delta_k(t) y_j(t), \tag{2.10}$$

onde, α é a taxa de aprendizado;
$\delta_k(t)$ é o gradiente local; e
$y_j(t)$ é o valor de entrada da ligação sináptica j, ou seja, a saída do neurônio j da camada anterior a camada do neurônio k.

O gradiente local do neurônio k no instante t é definido por

$$\delta_k(t) = e_k(t)\varphi_k^{'}(v_k(t)), \tag{2.11}$$

onde, e_k é o sinal de erro do neurônio k;
φ' é a derivada da função de ativação do neurônio k; e
$v_k(t)$ é a ativação interna do neurônio k.

Percebe-se então, que o sinal de erro $e_k(t)$, na Equação (2.11), é um fator crítico para se efetuar o cálculo do ajuste dos pesos do neurônio k. Determinar o sinal de erro na saída do neurônio k é uma tarefa que varia de acordo com a posição do neurônio na rede. Quando o neurônio pertence a camada de saída da rede, o sinal de erro é simplesmente

$$e_k(t) = d_k(t) - y_k(t), \tag{2.12}$$

ou seja, a diferença entre o valor de saída do neurônio e o valor desejado $d_k(t)$.

Para os demais neurônios, pertencentes a uma camada oculta da rede MLP, o sinal de erro e_k é calculado levando em conta o gradiente local dos neurônios da próxima camada e o peso de suas ligações sinápticas

$$e_k(t) = \sum_{l=1}^{m} \delta_l(t) w_{lk}(t), \tag{2.13}$$

onde, m é o número de conexões do neurônio k, ligando-o aos neurônios da próxima camada;
δ_l é o gradiente local do neurônio l, da próxima camada; e
w_{lk} é o peso de conexão do neurônio k e l.

Em (HAYKIN, 2001) encontra-se a derivação completa das equações do algoritmo *backpropagation*.

Observa-se pela Equação (2.11) que uma restrição importante do algoritmo *backpropagation* é a função de ativação dos neurônios ser uma função contínua e diferenciável (FAUSETT, 1994).

Em resumo, pode-se identificar três etapas no processo de treinamento:

a) Ativação: os dados de entrada $\bar{x}$ são propagados adiante na rede, até que se obtenha um vetor de saída $\bar{y}$;

b) Propagação do sinal de erro: a saída obtida pela rede é comparada com a saída desejada fornecida, calculando-se o sinal de erro da rede. Em seguida o sinal é propagado para trás pelas camadas da rede, enquanto se calcula o gradiente local de cada unidade neuronal;

c) Atualização dos pesos: o gradiente local de cada neurônio é utilizado para a correção de seus pesos sinápticos.

Essas três etapas são repetidas várias vezes para todo o conjunto de dados de treinamento, visando minimizar o Erro Quadrático Médio (MSE, do inglês *Mean Squared Error*) da rede. O MSE é uma função de custo

$$MSE = \frac{1}{N} \sum_{n=1}^{N} \mathcal{E}(n), \tag{2.14}$$

obtida pelo valor instantâneo $\mathcal{E}(n)$ da soma dos erros quadráticos de todos os neurônios da camada de saída da rede

$$\mathcal{E}(n) = \frac{1}{2} \sum_{k \in CS} e_k^2(n), \tag{2.15}$$

onde, CS é o conjunto de neurônios que pertencem a camada de saída da rede neural.

O número total de padrões utilizados no conjunto de treinamento é denotado por N, sendo n uma amostra deste conjunto.

2.1.2 Redes de Função de Base Radial

As redes de Função de Base Radial (RBF, do inglês *Radial Basis Function*) são constituídas de 3 camadas de unidades neurais, como mostra a Figura 2.4. A camada de entrada tem a função de receber os sinais do ambiente, não realizando nenhum processamento. A camada de saída, é composta de neurônios do tipo *Perceptron*, com função de ativação linear. Observa-se então, que as camadas externas da rede RBF têm características semelhantes as redes MLP. O que diferencia as redes RBF das MLP são as unidades situadas na camada oculta, que têm o papel de realizar

uma transformação não-linear do espaço de entrada, compostas por funções de base radial (HAYKIN, 2001).

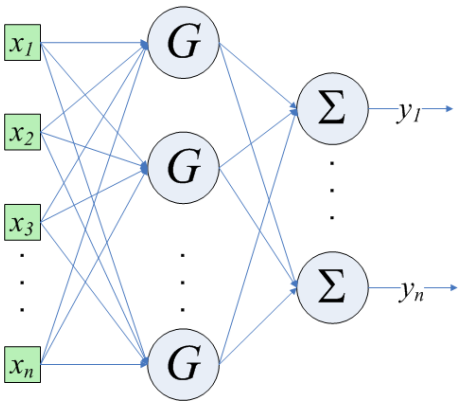


FIGURA 2.4 - Rede de Função de Base Radial.

Vários tipos de funções de base radial podem ser utilizadas nestas unidades da camada oculta, sendo a mais comum a *função Gaussiana*

$$G\left(\|\bar{x} - \bar{c}_j\|^2\right) = \exp\left(-\frac{\|\bar{x} - \bar{c}_j\|^2}{2\sigma^2}\right). \tag{2.16}$$

Nesta função, a norma euclidiana $\|\|$ é calculada a partir da diferença entre o vetor de entrada $\bar{x}$ e o vetor de centros $\bar{c}$ da gaussiana. O desvio padrão σ é o espalhamento da função que controla o campo receptivo de cada neurônio.

Existem diferentes estratégias para definição dos centros das funções de base radial, como por exemplo, simples atribuição aleatória, seleção de elementos do conjunto de dados de entrada, ou análise de agrupamento de classes a partir do conjunto de dados de entrada.

Já o coeficiente σ pode ser definido como

$$\sigma = \frac{d_{max}}{\sqrt{2m}}, \tag{2.17}$$

onde, d_{max} é a distância máxima entre os centros;
m é o número de centros.

A saída y do neurônio k da camada de saída da rede RBF é obtida por

$$y_k(\bar{x}) = \sum_{j=1}^{n} w_{kj} G_j(\|\bar{x} - \bar{c}_j\|^2) + \beta, \tag{2.18}$$

onde, n é o número de unidades na camada oculta;
w_{kj} é o peso da ligação da saída da função j ao neurônio k;
$\bar{x}$ é o vetor de entrada;
β é o *bias* do neurônio k.

Após a definição dos centros das RBF, o processo de aprendizagem da rede consiste em ajustar os pesos das unidades neurais da camada de saída, o que pode ser feito pela Regra de Aprendizado Delta (HAYKIN, 2001)

$$\bar{w}_k(t+1) = \bar{w}_k(t) + \alpha(d_k - y_k)\bar{x}^T, \tag{2.19}$$

onde, α é a taxa de aprendizado;
y_k é a saída gerada pelo neurônio k;
d_k é a saída desejada a ser aprendida pelo neurônio k;
$\bar{x}^T$ é a transposta do vetor contendo o sinal de entrada do neurônio k.

2.1.3 Mapas Auto-organizáveis de Kohonen

Os Mapas Auto-organizáveis (SOM, do inglês *Self-Organizing Maps*) de (KOHONEN, 1982) agrupa vetores de entrada $\bar{x}$ em classes de dados, sendo treinados por processo não-supervisionado. As unidades neuronais que compõem este tipo de rede têm função de ativação linear e não contêm *bias*. Os neurônios assumem uma estrutura topológica, organizada em um vetor unidimensional (conforme ilustra a Figura 2.5) ou bidimensional (FAUSETT, 1994).

O processo de formação do mapa auto-organizável inicia-se pela atribuição arbitrária de valores aos pesos dos neurônios, e em seguida, três etapas são realizadas: competição, cooperação e adaptação sináptica (HAYKIN, 2001).

A *etapa de competição* consiste em determinar um neurônio vencedor K, que melhor represente um dado vetor de entrada $\bar{x}_i$. Isto é feito por meio da função discriminante

$$K = \arg\min_k \left(\sum_j (w_{kj} - x_j)^2 \right). \tag{2.20}$$

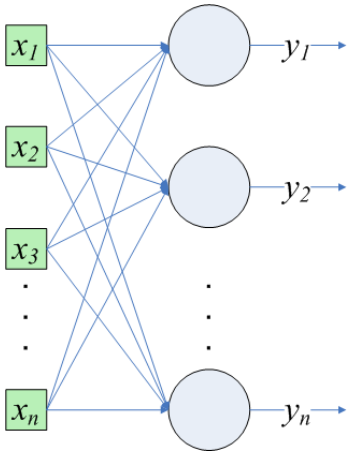


FIGURA 2.5 - Vetor unidimensional de um Mapa Auto-organizável de Kohonen. As variáveis y representam as saídas produzidas pelos neurônios. Ressalta-se porém, que apenas o neurônio com a maior saída será declarado vencedor.

Na *etapa de cooperação*, identifica-se uma vizinhança topológica de K (normalmente um raio ρ) que terá o direito de atualizar seus pesos juntamente com o neurônio vencedor.

A *adaptação sináptica*, na última etapa, atualiza os pesos dos neurônios selecionados

$$\bar{w}_k(t+1) = \bar{w}_k(t) + \alpha(\bar{x}_i - \bar{w}_k(t)), \tag{2.21}$$

diminuindo assim a distância dessas unidades com o padrão de entrada.

Durante o processo de aprendizado, se reduz a taxa de aprendizado α e o raio da vizinhança ρ.

A Figura 2.6 exemplifica o uso de um Mapa Auto-organizável unidimensional, com 50 unidades para agrupar 100 dados bidimensionais. Os dados estão representados por um círculo e a posição dos neurônios por um asterisco, após 100 épocas de treinamento. Cada época do treinamento consiste na apresentação de todo o conjunto de dados, em que o α foi inicializado em $0,5$ e decaindo a cada época. As linhas representam a topologia da rede. Durante o aprendizado considerou-se vizinhança 1 na etapa de cooperação neuronal.

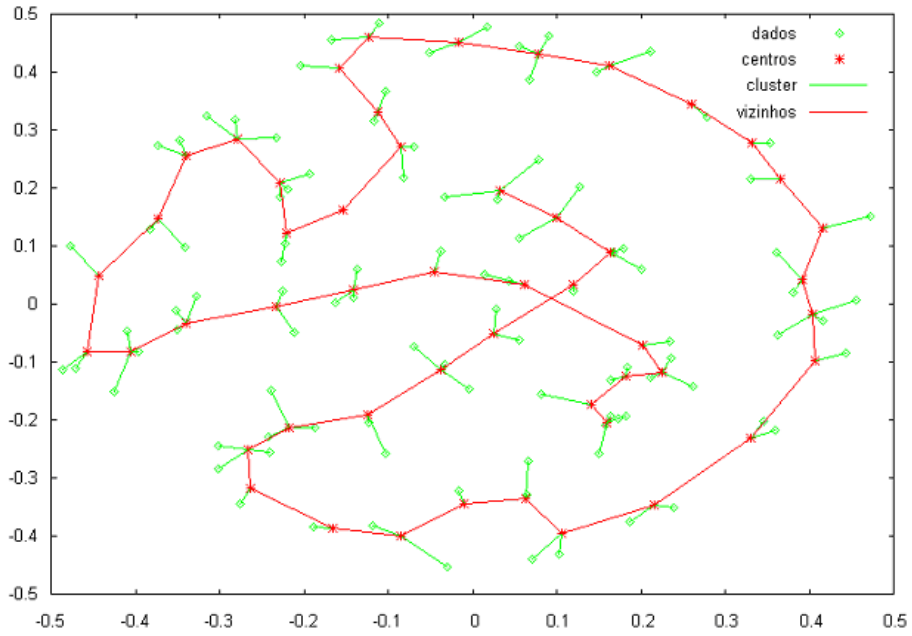


FIGURA 2.6 - Exemplo de um Mapa Auto-organizável unidimensional com 50 unidades, após 100 épocas de treinamento.

2.1.4 Aplicações

Devido a sua capacidade de aprendizado, o uso de RNA é interessante quando se tem um conjunto de dados observacionais do problema que se quer abordar, mesmo não se conhecendo exatamente o comportamento do fenômeno. Além disto, devido a característica dos neurônios estarem organizados em arquiteturas, é possível explorar ganho computacional através de processamento paralelo das unidades neuronais (RUDOLPH; MARTINEZ, 1997; KIM *et al.*, 1998; TOPPING *et al.*, 1998). Isto pode significar uma grande vantagem perante métodos exatos, principalmente em problemas de visão computacional, nos quais existe grande complexidade computacional envolvida.

Por outro lado, a escolha da arquitetura e a determinação dos seus parâmetros, mais adequados para a abordagem de um dado problema, consistem ainda em temas de pesquisa. Em navegação autônoma é comum encontrar trabalhos que utilizam RNA, como por exemplo em (CASTRO, 2003), descrito na introdução deste trabalho.

Em (HANDMANN *et al.*, 2000), uma rede MLP é utilizada em conjunto com uma arquitetura de um sistema para auxiliar o processo de guiagem de um veículo doméstico, para tempo real. O sistema permite detectar, rastrear trajetórias e classificar obje-

tos numa cena, a frente do veículo. Inicialmente, num processo mais lento do que tempo real, o sistema extrai informações do ambiente utilizando técnicas clássicas de processamento digital de imagens. Em seguida, uma rede neural MLP, previamente treinada, é utilizada para efetuar o restante do processamento de guiagem em tempo real. A utilização de redes neurais nesse sistema, além de agilizar o processamento, serve como um mecanismo de fusão de sinais obtidos por vários sensores.

O uso de outros tipos de redes também vêm sendo aplicados a robótica móvel. Em (SONG; SHEEN, 2000) é apresentada uma solução híbrida para navegação reativa de um robô, com técnicas de lógica nebulosa e mapa de Kohonen. Durante o processo de navegação, o sistema visa reconhecer padrões de comportamento aprendidos através dos dados sensoriais, para então escolher uma ação a ser tomada. O papel do mapa de Kohonen consiste em classificar os padrões de dados sensoriais entrados. Em (HOFFMANN *et al.*, 2004a), um mapa de Kohonen foi aplicado para determinar os centros das RBF utilizadas para guiar um veículo autônomo. Em (OSÓRIO *et al.*, 2002), uma rede neural do tipo *Jordan-Cascade Correlation (J-CC)* é utilizada para realizar a tarefa de estacionamento de um veículo, equipado com sensores de distância. O treinamento da rede J-CC foi realizado através de dados obtidos de um sistema baseado em autômato de estados finitos, e validado através de um simulador. Outro exemplo de uso de RNA em robótica móvel pode ser visto em (MENG; KAK, 1993).

2.2 Sistemas Nebulosos

No cotidiano, as pessoas não costumam usar informação numérica precisa nas tomadas de decisão, apesar disto, elas são capazes de realizar tarefas de controle altamente precisas. Por volta de 1965, Lotfi Zadeh desenvolveu a teoria de conjuntos nebulosos (*fuzzy sets*), que provê meios para incorporar a maneira de pensar do ser humano, em áreas aplicadas. Essa teoria, baseada na teoria clássica de conjuntos, permite lidar com dados imprecisos, modelando formalmente a incerteza, para que possa ser processada por computadores. Isto significa que se pode utilizar informações do tipo "*muito quente*", "*um pouco frio*" ou "*um pouco à esquerda*", em substituição a valores precisos, como por exemplo $25,4^oC$.

Na teoria clássica dos conjuntos, um conjunto inclui ou não um dado elemento, permitindo assim apenas dois valores de pertencimento: 0 ou 1 (falso ou verdadeiro). Já na teoria dos conjuntos nebulosos, um elemento u pode pertencer a um conjunto ne-

buloso F completamente ou com um certo grau de pertencimento. O grau é definido por uma função de pertencimento (ou pertinência) $F(u)$ num universo de discurso Ω, ou seja, $F(u) : \Omega \rightarrow [0, 1]$. Quando um elemento u pertence completamente a um conjunto F, então $F(u) = 1$. Caso o elemento u pertença parcialmente ao conjunto F, então $0 < F(u) < 1$. Assim, a Lógica Nebulosa é uma lógica de multi-valores que permite o uso de termos lingüísticos para definir valores intermediários, além de valores convencionais do tipo "sim/não", "verdadeiro/falso" ou "preto/branco" (BEZDEK, 1993; DUBOIS; PRADE, 1996; SANDRI; CORREA, 1999).

Vários sub-conjuntos podem ser extraídos de F, como por exemplo: o sub-conjunto de elementos com nível de pertinência maior ou igual a 0,6. Esses sub-conjuntos são definidos por um nível de corte, conhecidos por $\hat{\alpha}$-*corte*, sendo

$$F_{\hat{\alpha}} = \{u \in \Omega | F(u) \geq \hat{\alpha}\}. \tag{2.22}$$

Quando o maior grau de pertinência de um conjunto nebuloso for 1, diz-se que este conjunto está normalizado e sua altura é 1. Os casos particulares de $\hat{\alpha} = 0$ e $\hat{\alpha} = 1$ são denominados respectivamente *suporte*

$$Su(F) = \{u \in \Omega | F(u) > 0\} \tag{2.23}$$

e *núcleo*

$$Nu(F) = \{u \in \Omega | F(u) = 1\} \tag{2.24}$$

do conjunto F.

O suporte consiste então no conjunto formado por todos os elementos que pertencem com algum grau de pertinência a F.Já o núcleo representa o conjunto de todos os elementos que pertencem totalmente a F (TANAKA, 1997; SHAW; SIMÕES, 1999; YEN, 1999).

A Figura 2.7 ilustra um conjunto nebuloso, destacando o seu *suporte* $Su(F)$, *núcleo* $Nu(F)$ e $\hat{\alpha}$-*corte*. Observa-se ainda, o valor de pertencimento do elemento u_i, que é $0,4$.

Em aplicações de controle, os conjuntos *fuzzy* devem ser convexos, significando que os sub-conjuntos de $F_{\hat{\alpha}}$ devem ser convexos, $\forall \alpha \in [0, 1]$ (DRIANKOV *et al.*, 1996). Alguns exemplos de funções de pertencimento que formam conjuntos *fuzzy* convexos

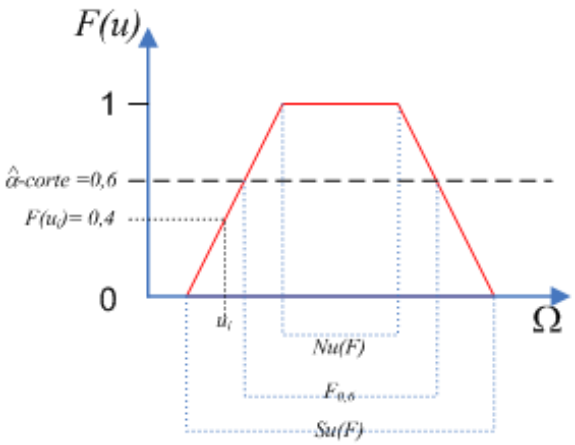


FIGURA 2.7 - Conjunto nebuloso, destacando o seu *núcleo*, *suporte*, $\hat{\alpha}$-*corte* e valor de pertencimento do elemento u_i.

estão ilustradas na Figura 2.8.

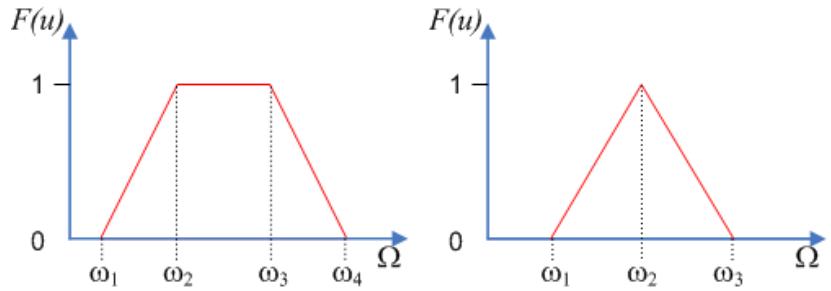
A função trapezoidal

$$\Pi(u,\omega_1,\omega_2,\omega_3,\omega_4) = \begin{cases} 0 & \text{se } u < \omega_1; \\ (u-\omega_1)/(\omega_2-\omega_1) & \text{se } \omega_1 \leq u \leq \omega_2; \\ 1 & \text{se } \omega_2 < u < \omega_3; \\ (\omega_4-u)/(\omega_4-\omega_3) & \text{se } \omega_3 \leq u \leq \omega_4; \\ 0 & \text{se } u > \omega_4; \end{cases} \tag{2.25}$$

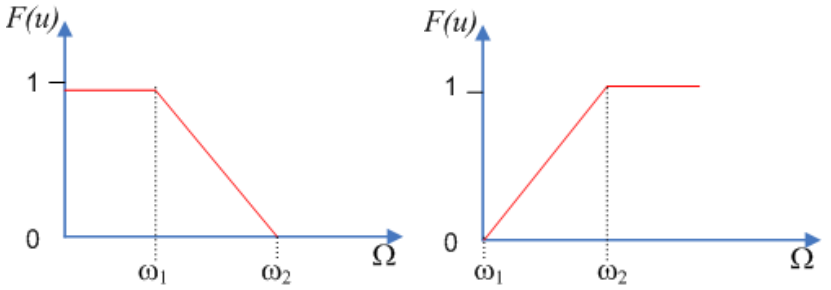
tem seu formato especificado por quatro parâmetros,definindo um conjunto *fuzzy* com vários elementos com pertencimento 1. A função triangular

$$\Lambda(u,\omega_1,\omega_2,\omega_3) = \begin{cases} 0 & \text{se } u < \omega_1; \\ (u-\omega_1)/(\omega_2-\omega_1) & \text{se } \omega_1 \leq u \leq \omega_2; \\ (\omega_3-u)/(\omega_3-\omega_2) & \text{se } \omega_2 \leq u \leq \omega_3; \\ 0 & \text{se } u > \omega_3; \end{cases} \tag{2.26}$$

define apenas um elemento preciso (*crisp*) pertencendo totalmente ao conjunto nebuloso.



(a) Função Trapezoidal Π. (b) Função Triangular Λ.



(c) Função L (d) Função Γ

FIGURA 2.8 - Exemplos de Funções de Pertencimento para Conjuntos Nebulosos Convexos.

Já as funções

$$L(u, \omega_1, \omega_2) = \begin{cases} 1 & \text{se } u < \omega_1; \\ (\omega_2 - u)/(\omega_2 - \omega_1) & \text{se } \omega_1 \leq u \leq \omega_2; \\ 0 & \text{se } u > \omega_2; \end{cases} \tag{2.27}$$

e

$$\Gamma(u, \omega_1, \omega_2) = \begin{cases} 0 & \text{se } u < \omega_1; \\ (u - \omega_1)/(\omega_2 - \omega_1) & \text{se } \omega_1 \leq u \leq \omega_2; \\ 1 & \text{se } u > \omega_2; \end{cases} \tag{2.28}$$

são semelhantes as funções trapezoidais, porém com apenas dois parâmetros que definem o seu formato.

2.2.1 Variável Lingüística

Em sistemas nebulosos, *variáveis lingüísticas* têm seus valores expressos por termos lingüísticos, mapeados por conjuntos nebulosos. Uma variável lingüística é definida pela quádrupla $(\chi, E(\chi), \Omega, M_\chi)$, onde χ é o nome da variável, $E(\chi)$ é o conjunto de

termos lingüísticos de χ. M_χ é a regra semântica

$$M_\chi : E(\chi) \rightarrow \tilde{E}(\chi) \tag{2.29}$$

que mapeia os termos lingüísticos para conjuntos nebulosos descritos em Ω, ou seja, transforma rótulos em números nebulosos,onde $\tilde{E}(\chi)$ denota um conjunto nebuloso definido em Ω (DRIANKOV *et al.*, 1996).

Por exemplo, a Figura 2.9 representa a variável lingüística *Umidade relativa do Ar*, denotada por χ. Seus termos lingüísticos são $E(\chi) = \{$ *Zero, Baixa, Média, Alta, Máxima* $\}$. Através de $\Omega \in [0, 100]$, define-se os seguintes conjuntos nebulosos:

$$\begin{aligned} \tilde{Zero} &= L(u, 15, 35) && (2.30)\\ \tilde{Baixa} &= \Lambda(u, 20, 35, 52) && (2.31)\\ \tilde{Media} &= \Lambda(u, 35, 52, 69) && (2.32)\\ &\vdots \\ \tilde{Maxima} &= \Gamma(u, 69, 90). && (2.33) \end{aligned}$$

Pode-se dizer então que uma entrada $u = 40\%$ é *Baixa* com $0,7$ de certeza e *Zero* com $0,3$.

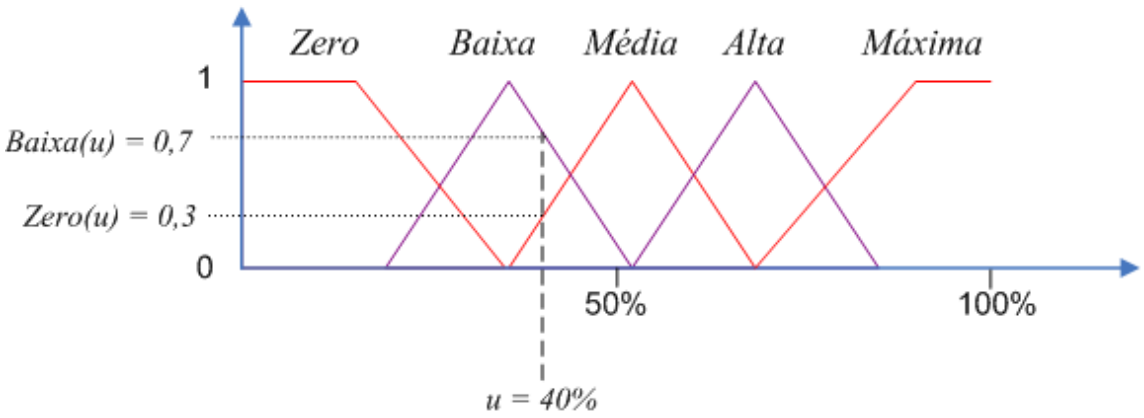


FIGURA 2.9 - Representação da variável lingüística *Umidade relativa do Ar* e seus termos lingüísticos mapeados para conjuntos nebulosos.

2.2.2 Controlador Nebuloso

A tarefa de controle visa obter informações de como um processo está transcorrendo e eventualmente interferir para ajustá-lo. A função de um processo é transformar algum bem ou informação, recebendo como entrada matéria-prima e produzindo na sua saída um novo material modificado. Independentemente de um processo ser físico ou abstrato, na maioria das vezes é possível obter dados desta ação. Sendo assim, a etapa de controle consiste em realimentar o processo, a fim de refiná-lo, ajustando-o a mudanças do ambiente. A Figura 2.10 ilustra o ambiente clássico de controle, do qual variáveis de saída são medidas por sensores, que propagam o sinal através de um instrumento de medição para normalizar os dados. Um valor de referência (*set point*) externamente fornecido é utilizado para comparar os dados coletados e produzir um erro. O erro serve de informação para a tomada de decisão de um controlador (humano ou não). A realimentação no sistema é feita por meio de atuadores.

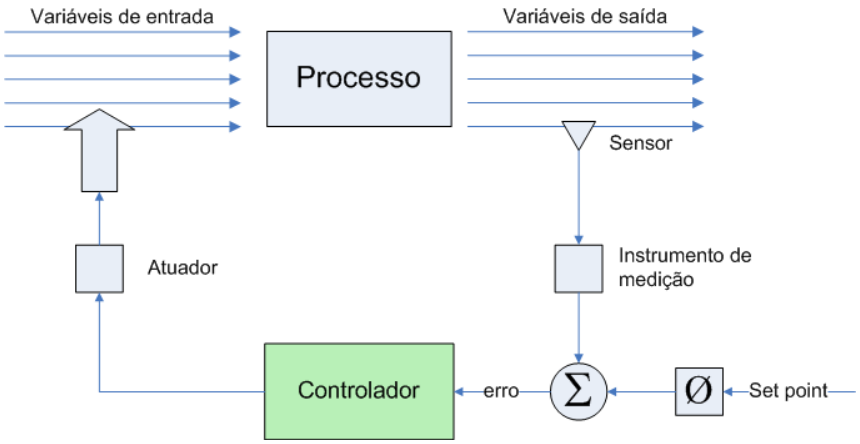


FIGURA 2.10 - Ambiente clássico de controle.

As técnicas clássicas de controle são realizadas através de métodos exatos, como por exemplo o controle Proporcional, Integral e Derivativo (PID) (TSOUKALAS; UHRIG, 1997), porém atualmente a Teoria dos Conjuntos Nebulosas vêm sendo utilizada como uma alternativa na construção de controladores.

Um *controlador nebuloso* pode ser representado pela Figura 2.11, onde variáveis de entradas, medidas do ambiente, passam por um processo de *codificação* (*fuzzificação*) a fim de serem representadas por variáveis lingüísticas. Através de um Sistema de

Inferência Nebulosa (FIS, do inglês *Fuzzy Inference System*), que utiliza uma *base de regras*, é produzido um valor de saída, representado por uma variável lingüística de saída. O valor então é *decodificado* (*defuzzificado*) a fim de se extrair um valor preciso (*crisp*) de controle (LEE, 1990; DRIANKOV, 2001; JANTZEN, 2003b; JANTZEN, 2003c).

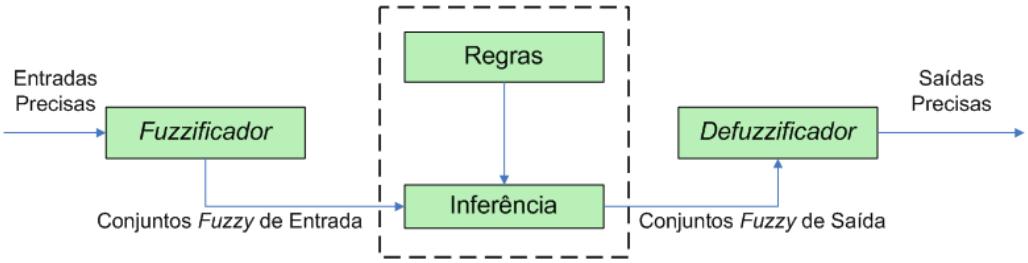


FIGURA 2.11 - Diagrama de um controlador nebuloso.

O Sistema de Inferência Nebuloso é a parte do controlador nebuloso responsável pela tomada de decisão, simulando o processo de inferência humano. Esse sistema valida uma série de regras no formato $\langle$ `if` $\dot{a}$ `is A and` $\dot{b}$ `is B then` $\dot{z}$ `is Z` $\rangle$, onde $\dot{a}$, $\dot{b}$ e $\dot{z}$ representam variáveis lingüísticas e `A`, `B` e `Z` termos. A expressão a esquerda de `then` é a causa, na forma de variáveis de entrada e a parte da direita da regra a conseqüência (variáveis de saída).

A forma de interpretar o `AND` das regras e combinar várias regras de uma base de dados varia dependendo do tipo de *operador de implicação* utilizado. Os operadores de implicação mais utilizados são:

- **Mamdani-min**: considera o menor valor de pertencimento obtido na condição das regras, e combina as regras com a operação de *união* (TSOUKALAS; UHRIG, 1997);
- **Larsen**: utiliza o produto dos valores de pertencimento obtido na condição das regras, e combina as regras com a operação de *intercessão* (TSOUKALAS; UHRIG, 1997);
- **Takagi-Sugeno (TKS)**: difere-se dos demais implicadores, principalmente porque sua saída já é um valor exato (*crisp*). Isto é possível, pois a parte conseqüente das regras são expressas diretamente por valores exatos ou funções. A combinação das várias regras ativadas, para produção de

um único valor na saída, é obtido através de interpolação de todos valores obtidos, freqüentemente num espaço $\Re^n$. As regras utilizadas pelo Sistema de Inferência Nebuloso são do tipo ⟨ `if f(a is A, b is B) then z = g(x,y)` ⟩ onde `f` é uma função lógica que conecta as sentenças que formam a condição; `z` é a saída, em função `g` das entradas. Cada regra ativada tem um valor de contribuição C, definida como $C = Min(\mu_A(a), \mu_B(b))$. A saída final z^*, inferida de n implicadores, é dada pela média de todos z's, ponderada com o valor C de cada regra (TAKAGI; SUGENO, 1985).

Exceto para o caso de uso de implicadores TKS, a *decodificação* mais comum nos controladores nebulosos consiste em encontrar o centro de massa do conjunto nebuloso de saída, conhecido como Centro de Área (COA - *Center of Area*), expresso por

$$z^* = \frac{\sum_{i=1}^{n} u_i F(u_i)}{\sum_{i=1}^{n} F(u_i)}. \tag{2.34}$$

O processo de inferência é ilustrado pela Figura 2.12, que representa um sistema nebuloso com duas variáveis lingüística de entrada (*Temperatura* (u) e *Umidade* (v) e uma de saída (*Abertura de válvula* (z)). No sistema hipotético, a base de regras foi montada com informações empíricas: constatou-se que a abertura da válvula deveria ser grande, quando a temperatura fosse baixa e a umidade próxima de zero; mas quando a temperatura fosse amena (média) e a umidade maior que zero (baixa), a abertura da válvula deveria ser pequena.

Assumindo, por exemplo, os valores de entrada $u = 9^oC$ e $v = 25\%$, duas regras foram ativadas: ⟨ `if` $\dot{u}$ `is baixa and` $\dot{v}$ `is zero then` $\dot{z}$ `is muito` ⟩ e ⟨ `if` $\dot{u}$ `is media and` $\dot{v}$ `is baixa then` $\dot{z}$ `is pouco` ⟩.

Neste exemplo, utiliza-se um implicador *Mamdani-min*, significando que a parte direita das regras (conseqüências) sofrerá um $\hat{\alpha}$-*corte* na altura do menor valor de pertencimento (região pintada nos conjuntos) obtido pelos conjuntos da esquerda das regras (condições). Todas as conseqüências das regras foram combinadas com uma operação de união, de onde foi extraído o valor preciso de saída z^*, aplicando-se a Equação (2.34).

2.2.3 Mapas Nebuloso-cartesianos

Representar o ambiente que um robô percebe durante sua exploração do mundo é uma necessidade freqüente. Ocorre que na maioria das vezes existe muita incerteza

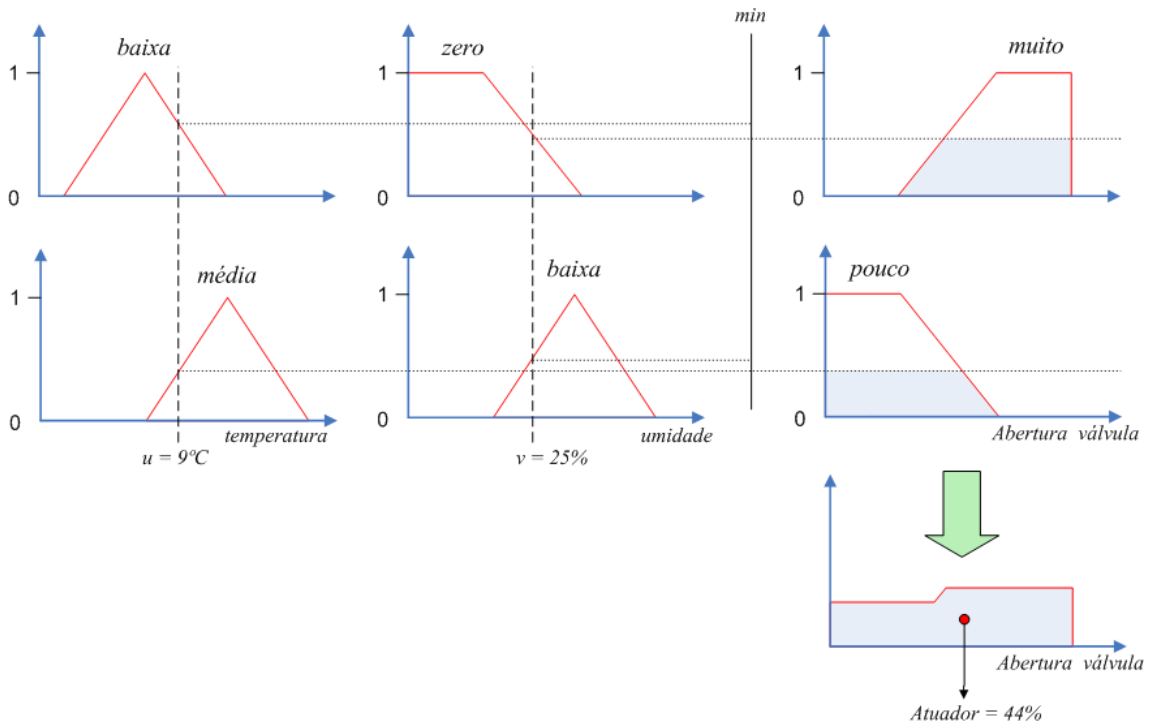


FIGURA 2.12 - Processo de inferência por *Mamdani-min*, em que foram ativadas de 2 regras, tendo como entrada 2 variáveis e 1 saída. O valor de saída preciso z^* foi obtido pelo cálculo de COA.

associada aos dados obtidos através dos sensores, devido a introdução de ruído ou pela imprecisão da localização do robô. Tunstel e Jamshidi propuseram em (TUNSTEL; JAMSHIDI, 1994) o uso de Lógica Nebulosa na construção de mapas, que auxiliariam a navegação de robôs. O mapa seria descrito em eixos de coordenadas num universo de variáveis nebulosas, ou seja, cada coordenada seria representada por termos lingüísticos. Por exemplo, na Figura 2.13 a localização final do robô pode ser expressa pela coordenada (SC, OC). Durante sua trajetória de exploração, alguns obstáculos foram percebidos, e mapeados também em coordenadas nebulosas.

Assim, com a distorção de dados de localização ao longo do tempo, inerentes a ruídos, estariam mais toleráveis, quando manipulados por variáveis lingüísticas. Além disto, numa tarefa de exploração com cobertura homogênea do terreno, é justo que registros sejam feitos em coordenadas nebulosas-cartesiana, pois no mundo real é difícil de definir fronteiras precisas entre regiões adjacentes.

Para missões nas quais exista iteração entre o agente robô e um supervisor humano,

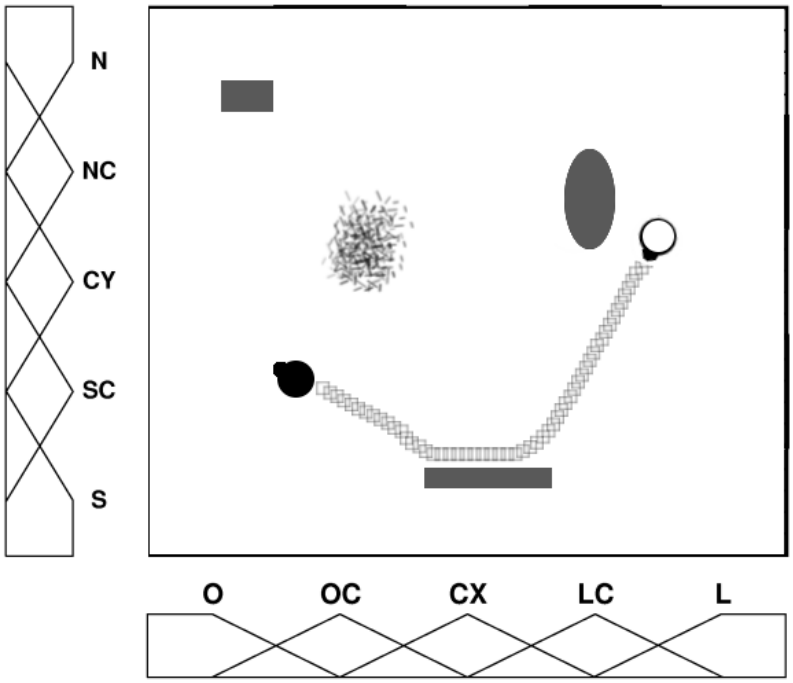


FIGURA 2.13 - Mapa nebuloso-cartesiano descrevendo a localização do robô (círculo com seta) e obstáculos em função de 2 variáveis lingüísticas, cada uma com 5 termos.

através do envio de comandos de alto nível, o uso de variáveis lingüísticas pode facilitar a construção de sistemas de controle.

Por outro lado, atualizar um mapa nebuloso-cartesiano pode se tornar mais custoso computacionalmente do que atualizar mapas tradicionais que usam apenas um valor simples de coordenada cartesiana.

Outra abordagem para construção de mapas de ambientes, baseado em lógica nebulosa é proposta em (BLOCH; SAFFIOTTI, 2003), no qual uma grade de ocupação *fuzzy* é construída a partir de dados sensoriais obtidos do ambiente. [1] Um algoritmo de segmentação é então aplicado a grade de ocupação, da qual são extraídas regiões topologicamente relacionadas. Tomando uma região como referência, *paisagens nebulosas* (*fuzzy landscapes*) são construídas através da aplicação de operadores de morfologia matemática *fuzzy* na grade de ocupação. A partir destas paisagens, a direção (norte, sul, leste, oeste) de outra região na grade de ocupação pode ser expressa em termos nebulosos.

[1] Na grade de ocupação *fuzzy*, a localização dos objetos são mapeadas em valores no intervalo [0,1], representando o grau de necessidade daquele espaço para ser considerado vazio.

2.2.4 Aplicações

Boa parte das aplicações que envolvem teoria dos conjuntos nebulosos é utilizada para controle de algum processo. Uma das vantagens do uso de controladores nebulosos está associada a sua capacidade em lidar com dados ruidosos e imprecisos. Por outro lado, isto não significa que não possa ser utilizado para sistemas que exijam um controle preciso do processo, como pode ser mostrado nos trabalhos (CZOGALA *et al.*, 1995; STONIER; STACEY A. J. MESSOM, 1998; TEIXEIRA *et al.*, 1998; MUSKINJA; TOVORNIK, 2000; JANTZEN, 2003a). Esses trabalhos utilizam controladores nebulosos para o problema do pêndulo invertido. O problema do pêndulo invertido consiste em controlar o movimento de um carrinho a fim de equilibrar verticalmente sobre ele um bastão, e é utilizado como um problema clássico (*bechmark*) para validação de controladores de sistemas dinâmicos não-lineares.

Outra vantagem do uso de sistemas de inferência nebulosa está na possibilidade da construção de um controlador baseado em experiências empíricas de um controlador humano. Como apresentado na introdução, no trabalho (CASTRO *et al.*, 2001) foi desenvolvido um FIS para controlar um robô sobre uma pista, baseado no comportamento de um motorista humano ao dirigir seu carro numa estrada. Já em (SURMANN *et al.*, 1995; SURMANN *et al.*, 1996) é descrito um sistema de navegação de um robô que utiliza módulos nebulosos no nível de controle de colisões e planejamento de rotas. Um sistema de controle hierárquico para veículos autônomos, utilizando controladores nebulosos, é utilizado em (VOUDOURIS *et al.*, 1994). O trabalho parte do uso de FIS clássicos, porém as conseqüências das regras recebem um peso, idéia conhecida como memórias associativas nebulosas (*Fuzzy Associative Memories*). As respostas dos vários controladores nebulosos são combinados numa estrutura de árvore para produzir uma saída de controle. Outros trabalhos de navegação autônoma com o uso de lógica nebulosa, podem ser encontrados também em (KELLER; KRISHNAPURAM, 1994; SAFFIOTTI, 1998; CASTELLANO *et al.*, 1999).

A medida que o processo de controle passa a ser muito complexo, a definição manual do conjunto de regras pode ser uma tarefa difícil. Neste caso, técnicas de definição automática de regras e conjuntos nebulosos, podem ser uma boa saída para o problema (SONG; SMITH, 2000a). Algumas técnicas fazem uso de Algoritmos Genéticos (KARR, 1994; RENHOU; YI, 1996; CHIN; QI, 1998), enquanto outras definem sistemas com implicadores de TKS por abordagem de problemas inversos, utilizando um conjunto de dados amostrais do processo (SONG; SMITH, 2000b).

Vale ressaltar ainda o uso de sistemas híbridos neuro-nebulosos, que combinam Sistemas de Inferência Nebuloso com Redes Neurais Artificiais, como uma alternativa de sistemas inteligentes para navegação autônoma (FULLÉR, 1995; ABRAHAM, 2001a; ABRAHAM, 2001b). Exemplos de outras aplicações podem ser vistas em Ng e Trivedi (1998), Agah e Tanie (1999), Cao (1999), Song e Sheen (2000). Em Cánovas *et al.* (2004) é apresentada ainda uma aplicação de controle nebuloso para combinar informações num sistema multi-agentes.

2.3 Aprendizado por Reforço

Um sistema de *aprendizagem por reforço* (RL - do inglês *Reinforcement Learning*) é um modelo de aprendizagem não supervisionado, em que um agente aprende a tomar suas decisões, sem que haja necessariamente a presença de um tutor externo. Inicialmente, não se sabe qual a melhor ação a ser tomada a cada instante de tempo, mas por ser um sistema de aprendizado orientado a objetivos, o agente vai se conscientizando das ações que deve tomar para atingir os objetivos (SUTTON; BARTO, 1998).

A cada ação tomada, o agente recebe uma *recompensa*, que pode ser positiva, na forma de um prêmio, ou negativa, na forma de uma punição. Muitas vezes, a recompensa não é dada imediatamente, mas somente quando se atinge um certo *estado*, após a execução de várias ações. Esse atraso na entrega da recompensa, conhecido como recompensa com atraso (*delayed reward*), dificulta a percepção de quais ações no passado levaram a uma situação ruim ou boa, pois conseqüências futuras de uma ação não são incorporadas na recompensa imediata, mas podem influenciar no desempenho global do agente. Recompensas com atraso são comuns em modelagens de problemas reais, um exemplo é um jogo da velha, no qual o agente escolhe as posições livres para posicionar sua peça, e no final do jogo recebe um prêmio se ganhou, ou uma punição se perdeu.

Assim, em um sistema de RL, o aprendizado é realizado continuamente a medida que o agente interage com o ambiente e melhora a escolha de suas ações (SINGH *et al.*, 1996).

Formalmente, a cada instante de tempo t, o agente percebe o ambiente através de seus sensores e identifica um estado s, pertencente a um conjunto de estados S. Dependendo do tipo de problema que se está abordando, podem existir *estados*

terminais e *estados não-terminais*. Estados terminais são aqueles em que o agente pára e não toma mais decisões a partir daquele estado. Diz-se que um problema que contém estados terminais é um problema de *horizonte finito* e sem estados terminais de *horizonte infinito* (RUSSEL; NORVIG, 2004).

Em cada estado s em que o agente se encontra, existe um conjunto finito de ações $a \in A(s)$ que poderão ser tomadas. A escolha das ações é feita por meio de uma política π que representará a probabilidade de escolha de uma ação, num dado estado: $\pi = (a, s)$.

Para cada ação tomada, uma recompensa $\mathfrak{r}$ poderá ser fornecida, na forma de um sinal de reforço. Utilizando o sinal de reforço, o problema de aprendizado consiste em encontrar uma política ótima π^* que maximize o recebimento de recompensas.

Dada uma política π é possível determinar uma função de valor $V^\pi(s)$, que mapeia um estado s num valor que representa a expectativa de recebimento de recompensas (KAELBLING; LITTMAN, 1996; HAGEN; KRÖSE, 1997; SUTTON; BARTO, 1998). Essa esperança de recompensas é expressa por

$$V^\pi(s) = E\left[\sum_{k=0}^{\infty} \gamma^k \mathfrak{r}_{k+1} \,|s_k = s\right], \tag{2.35}$$

onde γ, entre $[0, 1]$, é um fator de desconto para as recompensas que serão recebidas no futuro. Quando γ é 0, apenas o reforço imediato é considerado; e quando é 1, as recompensas futuras terão o mesmo peso que a recompensa imediata. O fator de desconto é especialmente importante em modelagens onde o horizonte é infinito, ou seja, não existem estados terminais. Nestes casos, γ deve ser menor que um, caso contrário a esperança do valor do estado tenderá a infinito (HARMON; HARMON, 1996).

Existe então um relacionamento direto entre o valor de um estado e o valor dos estados vizinhos s', podendo o valor de um estado ser determinado pela *Equação de Bellman* (RUSSEL; NORVIG, 2004)

$$V(s) = \mathfrak{r} + \gamma \max_{s'} (V(s')). \tag{2.36}$$

Se existir um modelo de transição de estados $T(s, a, s')$, que define a probabilidade

de mudar para o estado s', tomando a ação a a partir do estado s, então a Equação de Bellman pode ser expressa como

$$V(s) = \mathfrak{r} + \gamma \max_a \sum_{s'} T(s, a, s')V(s'). \tag{2.37}$$

Da mesma forma que conhecendo-se a política ótima π^* pode-se determinar os valores dos estados, também é possível efetuar o inverso: sabendo-se o valor ótimo dos estados V^* é possível determinar a política ótima

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s')V(s'), \tag{2.38}$$

pois será possível escolher a melhor ação a ser tomada em cada estado (SUTTON; BARTO, 1998). Trata-se então de um problema dual de otimização (HAGEN; KRÖSE, 1997).

Vários algoritmos foram desenvolvidos a fim de se aproximar os valores de π^* e V^*. A seguir, serão descritos cinco algoritmos, sendo os dois primeiros baseados em Programação Dinâmica.

2.3.1 Algoritmo de Iteração de Valor

O algoritmo de Iteração de Valor é um método baseado em Programação Dinâmica (PD), no qual a busca pela função ótima de valor é feita através da atualização dos valores dos estados em sucessivas explorações do espaço de estados, até que os valores tenham convergido (HARMON; HARMON, 1996; KAELBLING; LITTMAN, 1996; SUTTON; BARTO, 1998). Cada valor de estado é atualizado segundo o sinal de reforço imediato somado a expectativa descontada de recebimento de recompensas nos estados vizinhos:

$$V_{t+1}(s) = \mathfrak{r}(s) + \gamma \max_a \sum_{s'} T(s, a, s')V_t(s'). \tag{2.39}$$

Observa-se que para cada estado s é preciso testar todas as possíveis ações a, o que só é viável com o conhecimento do modelo de transição de estados.

O pseudo-código do Algoritmo de Iteração de Valor é representado no algoritmo da Figura 2.14.

inicializar $V(s)$ arbitrariamente, para todo $s \in S$
enquanto política não for suficientemente boa **faça**
  para cada $s \in S$ **faça**
    para cada $a \in A(s)$ **faça**
      $V_{temp}(s,a) \leftarrow \mathfrak{r}(s,a) + \gamma \sum_{s'} T(s,a,s')V(s')$
    fim para cada
    $V(s) \leftarrow \max_a V_{temp}(s,a)$
  fim para cada
fim enquanto

FIGURA 2.14 - Algoritmo *Iteração de Valor*.

2.3.2 Algoritmo de Iteração de Política

A proposta do algoritmo de Iteração de Política é partindo de uma política arbitrária inicial π, aproximar uma função de valor V^{π}, que por sua vez servirá de base para se encontrar uma nova política π' aperfeiçoada, e encontrar uma nova função de valor $V^{\pi'}$. Esse processo segue num ciclo até que algum critério de parada seja atingido (KAELBLING; LITTMAN, 1996; SUTTON; BARTO, 1998). Pode-se identificar claramente duas etapas neste algoritmo:

- **Etapa de Avaliação**: em que uma função de valor é aproximada segundo uma política atual;
- **Aperfeiçoamento**: melhoria da política atual, baseando-se na função de valor que foi encontrada na etapa anterior.

O pseudo-código do Algoritmo de Iteração de Política é representado na Figura 2.15.

Nota-se, que na etapa de Avaliação de Política, o próprio algoritmo de Iteração de Valor poderá ser utilizado.

2.3.3 Método de Monte Carlo

Os métodos baseados em Programação Dinâmica têm a desvantagem de necessitar de um modelo fiel do ambiente, no qual todas as transições de estados e ações são avaliadas. Quando esses dados não são facilmente conhecidos, pode-se utilizar um método de aproximação das funções de valor e da política ótima, baseado em Monte Carlo (MC).

inicializar $\pi(s) \in A(s)$ arbitrariamente, para todo $s \in S$
repetir
  $\pi \leftarrow \pi'$
  repetir {Avaliação da política}
    $\Delta \leftarrow 0$
    para cada $s \in S$ **faça**
      $v \leftarrow V^{\pi}(s)$
      $V^{\pi}(s) \leftarrow \mathfrak{r}(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s')$
      $\Delta \leftarrow \max(\Delta, |v - V^{\pi}(s)|)$
    fim para cada
  até $\Delta < \theta$ {θ é um número pequeno positivo}
  para cada $s \in S$ **faça** {Aperfeiçoamento da política}
    $\pi'(s) \leftarrow \arg\max_{a}(\mathfrak{r}(s, \pi(s)) + \gamma \sum_{s'} T(s, a, s') V^{\pi}(s'))$
  fim para cada
até $\pi = \pi'$

FIGURA 2.15 - Algoritmo *Iteração de Política*.

Aproximações por MC são realizadas em episódios, que são constituídos de amostras de experiências realizadas ou simuladas no ambiente. Assim, armazena-se uma série finita de estados, ações tomadas e recompensas recebidas, até que o agente atinja um estado terminal. A média das recompensas obtidas é utilizada então para se aproximar o valor dos estados (HAGEN; KRÖSE, 1997; SUTTON; BARTO, 1998).

A idéia dos algoritmos de aproximação da política ótima e função de valor é relativamente a mesma utilizada nos algoritmos de PD. Por exemplo, o algoritmo de estimação dos valores dos estados de *primeira-visita*, apresentado na Figura 2.16, aproxima o valor de $V(s)$ como sendo a média das recompensas recebidas depois da primeira visitação do estado s.

2.3.4 Diferença Temporal

A aprendizagem por Diferença Temporal (TD - *Temporal Difference*), combina as técnicas de Monte Carlo (MC) e Programação Dinâmica (PD): baseia-se em aprendizado por experiências, em episódios, sem utilizar um modelo preciso do problema, como em MC; mas atualiza os valores dos estados incrementalmente, como em PD, não necessitando atingir um estado terminal para atualizar as estimativas (SUTTON; BARTO, 1998; RUSSEL; NORVIG, 2004).

inicializar V arbitrariamente
inicializar lista $Recompensas(s)$ vazia, para todo $s \in S$
loop
  gerar um episódio usando π
  para cada s do episódio **faça**
    $R \leftarrow$ lista de recompensas depois da primeira visitação de s
    Concatenar R a $Recompensas(s)$
    $V^{\pi}(s) \leftarrow \mu(Recompensas(s))$ {onde μ denota média}
  fim para cada
fim loop

FIGURA 2.16 - Algoritmo *Método de Monte Carlo*.

De forma iterativa, o algoritmo de TD atualiza os valores dos estados sendo

$$V_{t+1}(s_t) = V_t(s_t) + \alpha \left(\mathfrak{r} + \gamma V_t(s_{t+1}) - V_t(s_t) \right), \tag{2.40}$$

onde $\alpha > 0$ é a taxa de aprendizado.

Esta é a forma mais simples do algoritmo de TD, quando apenas uma ação no passado é observada na atualização do último valor de estado. Com o intuito de acelerar o processo de aprendizagem, foi proposto o algoritmo TD(λ), no qual um determinado número de estados visitados no passado (ou no futuro na versão teórica), tem seus valores atualizados como(SUTTON; BARTO, 1998)

$$V_{t+1}(s_t) = V_t(s_t) + \alpha \left(\mathfrak{r} + \gamma V_t(s_{t+1}) - V_t(s_t) \right) e_t(s_t). \tag{2.41}$$

O termo

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{se } s \neq s_t; \\ \gamma\lambda e_{t-1}(s) + 1 & \text{se } s = s_t; \end{cases} \tag{2.42}$$

é o rastreio de elegibilidade dos valores visitados no passado.

O coeficiente $\lambda \in [0, 1]$ regula o quanto no passado, a recompensa atual irá influenciar. Para o caso TD(0) ($\lambda = 0$), apenas o valor do estado atual será atualizado, tal qual no algoritmo TD tradicional. No outro extremo, TD(1), todos os valores dos estados visitados serão atualizados, limitados por γ, aproximando-se ao método Monte Carlo. O pseudo-código do método TD(λ) é ilustrado pelo algoritmo da Figura 2.17.

inicializar V arbitrariamente
inicializar $e(s) = 0$ para todo $s \in S$
para cada episódio **faça**
  inicializar s
  para cada s do episódio **faça**
    selecionar a utilizando uma política π em s
    efetuar a e receber $\mathfrak{r}$
    perceber s'
    $\delta \leftarrow \mathfrak{r} + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    para cada $s \in S$ **faça**
      $V(s) \leftarrow V(s)\alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    fim para cada
  fim para cada
fim para cada

FIGURA 2.17 - Algoritmo *TD(λ)*.

2.3.5 Algoritmo *Q-learning*

O algoritmo *Q-learning* é um algoritmo que trouxe novos avanços em aprendizagem por reforço, tornando-se rapidamente muito popular. Ele utiliza as idéias do método TD com a vantagem de realizar apenas um mapeamento do par estados/ações para Q-valores, substituindo a função de valores e a função de transição de estados. Esse mapeamento é conhecido como função-Q, que define um Q-valor(s, a) para cada ação possível a partir de um dado estado (KAELBLING; LITTMAN, 1996; HAGEN; KRÖSE, 1997; SUTTON; BARTO, 1998; RUSSEL; NORVIG, 2004).

Assim, um Q-valor é definido como sendo a soma das recompensas recebidas ao realizar uma dada ação, seguindo uma política fornecida. Ao assumir então que o valor de um estado é dado pelo maior Q-valor desse estado, pode-se estimar um Q-valor por

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\Delta q, \tag{2.43}$$

onde,

$$\Delta q = \mathfrak{r} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t). \tag{2.44}$$

O algoritmo *Q-learning* tem a vantagem de não necessitar de um modelo de transição de estados, além de utilizar apenas uma função de valor. Isto é particularmente

útil em aprendizagem *online*, freqüente em aplicações de robótica, embora a convergência ótima da escolha das ações possa ser lenta, dependendo do algoritmo utilizado (SINGH *et al.*, 1996).

O algoritmo *Q-learning* é descrito pela Figura 2.18.

---

inicializar $Q(s, a)$ arbitrariamente
para cada episódio **faça**
  inicializar s
  para cada s do episódio **faça**
    selecionar a utilizando uma política derivada de Q
    efetuar a e receber $\mathfrak{r}$
    perceber s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha \left(\mathfrak{r} + \gamma \max_{a'} Q(s', a') - Q(s,a) \right)$
    perceber $s \leftarrow s'$
  fim para cada
fim para cada

FIGURA 2.18 - Algoritmo *Q-learning*.

Segundo (KAELBLING; LITTMAN, 1996), se cada par estado / ação for executado um número de vezes suficientemente grande e a taxa de aprendizado α decair constantemente, os Q-valores convergirão para $Q*$ com probabilidade 1. O fato de anular a taxa de aprendizado, significa estagnar o aprendizado, o que é particularmente ruim em ambientes dinâmicos, com mudanças contínuas.

Neste sentido, quando o horizonte do modelo é infinito, é preferível manter $\alpha > 0$ e utilizar $\gamma < 1$. Com o uso do fator de desconto, o valor máximo de um estado estará limitado aproximadamente em $\mathfrak{r}/(1-\gamma)$ (HARMON; HARMON, 1996).

2.3.6 Generalização

Grandes desafios no uso prático de aprendizagem por reforço surgem quando o número de estados e ações cresce muito, ou trabalha-se com variáveis contínuas. A dimensionalidade e complexidade computacional do problema torna inviável a aplicação dos algoritmos de aproximação de valor de estado e política. A solução então é recorrer a algum método aproximador de funções.

Várias abordagens são propostas, como por exemplo o uso de Redes Neurais Artificiais (RNA), codificação de *coarse*, *Cerebellar Model Articulator Controller* (CMAC), entre outras, que podem ser vistas em (SUTTON; BARTO, 1998).

Todavia, uma das aplicações mais clássicas de aprendizado por reforço, com generalização de estados, é o sistema TD-Gammon de (TESAURO, 1995), que aprende a jogar Gamão. O espaço de estados, referente a posição das peças no tabuleiro, tem milhões de combinações e o aprendizado seria inviável sem o uso de um aproximador de funções.

No TD-Gammon, os valores dos estados são aproximados por uma RNA com *Perceptrons* de Múltiplas Camadas. As unidades sensoriais da RNA representam a posição das peças no tabuleiro e a saída, a esperança de recebimento de recompensas no final do jogo (negativo ou nulo para derrotas e positivo para vitórias). A rede tem apenas uma camada oculta e as unidades neuronais utilizam a função de ativação do tipo logística sigmóide.

A cada estado do jogo, a expectativa de vitória é consultada na RNA, através das possíveis jogadas. Após uma jogada ser efetuada, a rede neural tem seus pesos atualizados pelo algoritmo *backpropagation*, em que o erro da rede é estimado na forma do algoritmo TD(λ), ou seja, baseando-se em $V_t(s_t + 1) - V_t(s_t)$.

O sistema é treinado em episódios, que representam um jogo completo de gamão, que joga contra ele mesmo durante o processo de aprendizagem. Nos episódios iniciais, as jogadas são quase aleatórias, mas com 1,5 milhões de episódios, o sistema já aprendeu a jogar no nível de um campeão mundial.

Em algumas modelagens, discretizar o espaço das ações também pode ser necessário. (SMITH, 2002) apresenta um modelo que adapta o algoritmo *Q-learning* com Mapas Alto-organizáveis de (KOHONEN, 1982) (SOM - *Self-Organizing Map)*, para aprendizado em ambientes com estados e ações contínuas. Em seu estudo, dois mapas SOM são utilizados, um para discretizar o espaço de estados, chamado de *mapa de entradas*, e outro para discretizar o espaço de ações, chamado de *mapa de ações*. Os pares de unidades neuronais do mapa de entradas e mapa de ações, formam a tabela de Q-valores. Para cada variável sensorial percebida, uma unidade vencedora no mapa de entrada é identificada, e o mapa é atualizado. Uma unidade vencedora é identificada então no mapa de ações, baseando-se nos Q-valores. Os vetores de peso

da unidade de ação vencedora são utilizados pelos atuadores após uma introdução de ruído. Se o retorno recebido com a execução da ação é maior que o retorno esperado estimado, então a unidade neuronal no mapa de ação é atualizada.

Outra proposta do uso de SOM para discretização do espaço de estados pode ser vista em (BRAGA; ARAÚJO, 2002; BRAGA, 2004).

2.3.7 *Exploration* x *Exploitation*

Os métodos que não são baseados em PD para aproximação das funções de valor e/ou política ótima dificilmente cobrem todo o espaço de estados. Isto é agravado quando o espaço de busca de estados cresce, não havendo garantia de se encontrar a solução ótima. Em cada episódio, o aprendizado do agente é guiado por uma política de exploração e dependendo desta política, hora se prioriza a exploração do espaço de estados, e em outro momento a otimização das ações aprendidas. Na realidade, ambos comportamentos são desejados, mas impossíveis de serem alcançados ao mesmo tempo. É o conhecido dilema de exploração (*Exploration x Exploitation*) do aprendizado, no qual se tenta otimizar o aprendizado, através de novas experiências, mas se corre o risco de errar (HARMON; HARMON, 1996; HAGEN; KRÖSE, 1997; RUSSEL; NORVIG, 2004).

Para amenizar esse problema, algumas políticas de exploração do tipo *soft* (*soft policies*) foram propostas, determinando de forma estocástica a escolha de ações. A mais comum é escolher as ações de forma aleatória, segundo uma distribuição uniforme. Todavia, neste caso fica-se restrito a exploração simples do espaço de busca, sem observar a otimização do aprendizado.

Uma alternativa freqüentemente utilizada é a política ϵ-*greedy*, que balanceia a escolha das ações entre puramente aleatórias e de forma gulosa (*greedy* - escolha da melhor ação aprendida até o momento). O coeficiente $\epsilon >= 0$ determina a probabilidade de seleção aleatória das ações, que é $1 - \epsilon + \frac{\epsilon}{|A(s)|}$ para a melhor ação aprendida até o momento e $\frac{\epsilon}{|A(s)|}$ para as demais (BHANU *et al.*, 2001; CROOK; HAYES, 2003). Por exemplo, quando $\epsilon = 1$ as ações serão sempre escolhidas aleatoriamente, mas se $\epsilon = 0,5$, então metade das escolhas serão aleatórias e a outra metade de forma gulosa .

Estratégias de variação do valor de ϵ podem também ser utilizadas, como por exemplo um valor alto no início do aprendizado, que vai decaindo com o tempo. Assim,

enquanto se tem pouco conhecimento do ambiente, se explora de forma abrangente, mas a medida que o conhecimento cresce, o agente fica mais cauteloso em suas tomadas de decisão.

Outra abordagem comumente utilizada é a política *softmax*, que calcula a probabilidade de escolha de uma ação, baseado nos valores de estados aprendidos até o momento(KAELBLING; LITTMAN, 1996; SUTTON; BARTO, 1998)

$$\pi(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_a e^{Q(s,a)/\tau}}, \tag{2.45}$$

onde, τ é um parâmetro de *temperatura*, ou seja, quando o limite é zero, tende a escolher as ações de forma gulosa. Este método trás a vantagem de diminuir as chances de escolha de uma ação que já se experimentou ser muito ruim.

2.3.8 Aplicações

Técnicas de aprendizagem por reforço tem atraído a atenção dos pesquisadores em aplicações de controle de robôs, o que se reflete no grande número de trabalhos que podem ser encontrados na literatura. Vários destes trabalhos dedicam-se a robótica móvel, algumas vezes combinando técnicas de RL com RNA ou Lógica Fuzzy.

Kimura *et al.* (1997) modelou um braço mecânico para aprender a tarefa de se movimentar pelo chão. O braço é constituído de duas juntas, que dependendo do movimento, arrasta o corpo do robô (sem rodas) para frente ou para trás. O experimento consistiu em avaliar métodos de discretização do espaço de estados e três algoritmos de aprendizado por reforço: *Q-learning*, Ascensão de Gradiente Estocástico e método JSJ (Algoritmo de Jaakkola e outros).

Utilizando a idéia de campos potenciais, Armstrong *et al.* (1999) simularam o aprendizado por reforço de um robô móvel, estimulado a navegar em direção de um objetivo, desviando de obstáculos. Em Gaskett *et al.* (2000), o objetivo é vaguear (*wandering*) por um ambiente desconhecido, percebido por um sistema de visão computacional, representado por uma RNA e guiado por um sistema de RL. Uma comparação do algoritmo de Busca por Profundidade em Campos Vetoriais, *Q-learning* e *Iteração de Política*, no planejamento de rotas é realizada em Aranibar e Alsina (2004).

Para validação de sistemas de RL normalmente se utiliza o ambiente de labirintos,

descritos em Sutton e Barto (1998). Bhanu *et al.* (2001) construíram então um ambiente de labirintos real, no qual vários algoritmos de RL, previamente treinados em simulação, foram validados. A implementação em *hardware*, dividida em sistema de vídeo, aprendizado, robô e interfaces, leva em média $5,4s$ para executar uma iteração com o ambiente.

Zhu e Levinson (2001) apresenta uma estratégia de representação dinâmica de estados, em aprendizado por reforço, aplicado a tarefa de aproximação de um objeto alvo imóvel, por um robô móvel. Estados contínuos são mapeados para um conjunto inicial de estados discretos, que tem seu número aumentado durante o processo de aprendizagem, pela divisão de estados.

De modo a acelerar o processo de aprendizagem e prover um comportamento inicial satisfatório a um robô móvel, Smart e Kaelbling (2001), Smart e Kaelbling (2002) propõem a arquitetura JAQL [2], na qual a tarefa de aprendizado por reforço em navegação de robôs é feita em duas etapas: primeiro o robô é guiado de forma supervisionada no ambiente, e vai aprendendo o comportamento básico de navegação; na segunda fase, o robô segue explorando o espaço de estados de maneira padrão em RL.

O problema de navegação aérea é abordado por Busquets *et al.* (2002) com uma arquitetura multi-agente. Um agente de aprendizagem é responsável em navegar um dirigível e controlar uma câmera, que busca por marcas de navegação. O agente de aprendizado foi treinado em ambiente simulado, utilizando um algoritmo de RL baseado em um modelo do ambiente.

Para melhorar a robustez do algoritmo *Q-learning* na aplicação de navegação de robôs em ambientes dinâmicos, Yen e Hickey (2002) propõem três métodos diferentes: uma arquitetura hierárquica de aprendizagem de reforço, um mecanismo de esquecimento e mapeamento de estados baseado em características. O método de arquitetura hierárquica obteve melhor resultados, pois quebrou o problema de navegação em problemas menores.

Hagen e Kröse (2003) apresentam uma arquitetura, conhecida como *lazy map*, para representar topologicamente o ambiente percebido por um robô móvel. Os nodos do mapa são utilizados como estados de um agente, que aprende por reforço caminhos

[2] Os autores não indicam em seus artigos o significado de JAQL.

a seguir para alcançar um local específico do ambiente. A aproximação da função de valor, utilizando RNA num ambiente de RL é aplicada à tarefa de navegação de robô no trabalho de Xu *et al.* (2003). O robô, equipado com sensores de radar, aprende em simulação a navegar até um objetivo de forma reativa, num ambiente desconhecido.

A tarefa específica de ancoragem (*docking*) de robôs móveis também é explorada, e pode ser vista nos trabalhos de Weber *et al.* (2004), Martínez-Marń e Duckett (2005). Ambos utilizam visão computacional para detectar o objeto alvo, e aprendizagem por reforço na etapa de controle da trajetória do robô.

Michels *et al.* (2005) usam RL com sensores de visão computacional para determinar a direção de um robô navegando em alta velocidade, num ambiente externo. O sistema de visão é baseado em RNA, treinadas a partir de medidas de distâncias, coletadas com um sensor a *laser*. O sistema de aprendizagem foi inicialmente treinado num simulador e posteriormente validado num ambiente físico.

Uma aplicação direta de RL em visão computacional é apresentada por Peng e Bhanu (1998), em um sistema de reconhecimento de objetos através de imagens, onde os parâmetros dos módulos de segmentação e extração de características são ajustados por um algoritmo *Q-learning*.

O futebol de robôs também tem sido bastante explorado pelos pesquisadores, que buscam um comportamento complexo de vários agentes coordenados, motivados pelas competições do jogo. Asada *et al.* (1995), Asada *et al.* (1996), Suzuki *et al.* (1998), Asada *et al.* (1999) trazem vários experimentos do uso de aprendizado por reforço, em especial o algoritmo *Q-learning*. Comportamentos simples, como o de aprender a chutar ao gol e a defender a bola, até comportamentos mais complexos, como a cooperação entre robôs para passe de bola, são abordados. Esteves *et al.* (2004) treina robôs para o futebol de forma simulada, discretizando os estados do ambiente com a técnica CMAC.

Técnicas híbridas de RL com Lógica *Fuzzy* e RNA são freqüentemente exploradas. Pode-se também combinar as vantagens do aproveitamento de conhecimento na construção de controladores nebulosos, com a adaptação e o auto-aprendizado dos sistemas de RL. Esta idéia está presente no trabalho de Glorennec e Jouffe (1996), em que um sistema de RL otimiza as regras de um controlador nebuloso construído

para navegador um robô. O mesmo é realizado com um controlador nebuloso do tipo Takagi-Sugeno (TS) para atitude de um satélite em Buijtenen *et al.* (1998). As saídas do controlador, que consistem na média ponderada das conseqüências individuais das regras, são inicialmente estimadas utilizando-se um controlador Proporcional Derivativo, e em seguida otimizadas por um sistema de aprendizado por reforço.

Berenji (1994) propõe o algoritmo *Fuzzy Q-learning*, que pode ser utilizado em processos de decisão nos quais os objetivos ou restrições são tratados como variáveis nebulosas. Uma idéia semelhante é apresentada por Faria e Romero (2000), modificando o algoritmo de aprendizagem por reforço *R-learning*, através da inclusão de informações nebulosas provenientes dos sensores, no cálculo de atualização dos R-valores. Zhuang *et al.* (2002) propõem um método de representação do espaço de estados em conjuntos fuzzy, que propagará a atualização dos Valores de Estados para seus membros, segundo os valores de pertencimento. A inovação proposta neste trabalho é a redução constante da abrangência dos conjuntos fuzzy, com o decorrer do aprendizado, acelerando assim o processo. Com o mesmo propósito de reduzir o espaço de variáveis, Macek *et al.* (2002) fizeram uso de redes neurais e RL para ensinar um robô móvel a desviar de obstáculos.

2.3.9 Exemplo de Uso do Algoritmo *Q-learning*

Neste item é apresentada uma aplicação simples de visão ativa, demonstrando o uso de aprendizagem por reforço. Em sistemas de visão ativa, um agente é equipado com uma ou mais câmeras, onde é possível controlar o posicionamento dessa câmera ou do seu foco. Uma das vantagens desse tipo de visão computacional é permitir maior flexibilidade ao robô, pois o mesmo pode obter várias imagens de uma cena, sem a necessidade de mover-se pelo ambiente. Em ambientes nos quais existam objetos móveis, um sistema de visão ativa permite, por exemplo, manter o foco de atenção sobre um dado objeto.

O sistema aqui ilustrado está inserido num ambiente em que existe um único objeto móvel, com características espectrais que o diferem do fundo da cena. O objetivo do robô é aprender que ações devem ser tomadas a fim de manter a projeção do objeto no centro da imagem.

O sistema sensor do agente detecta o objeto e retorna sua posição em coordenadas

discretas da imagem capturada. A variável de estado do sistema de aprendizado consiste então na coordenada (x, y) do objeto na imagem. O processo de percepção inicia com a captura da imagem colorida (em canais RGB), que é reamostrada para uma imagem de 10 linhas e 10 colunas. Essa operação visa reduzir o espaço de estados e conseqüentemente a complexidade computacional do problema. O objeto é detectado na imagem através da limiarização de cores.

A Figura 2.19 mostra um caso particular de processamento realizado pelo sensor. Nota-se na Figura (2.19-a) a imagem bruta capturada da cena, com o objeto alvo (um balão azul), posicionado na região superior esquerda da imagem. Após o processamento, a localização aproximada do objeto é destacada na Figura (2.19-b).

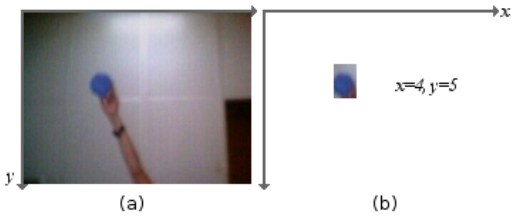


FIGURA 2.19 - Identificação da localização aproximada do objeto alvo, pelo módulo sensor: (a) imagem original capturada; (b) imagem processada.

A localização do objeto alvo em coordenadas de imagem é mapeada diretamente para o estado do agente, como ilustra a Figura 2.20. Assim, para cada coordenada discreta de localização do objeto, existe um estado associado. No centro da imagem, existem 4 estados terminais, identificando as situações em que a imagem estaria centrada no objeto alvo. Além disto, são definidos estados terminais adicionais de contorno da imagem, para quando o agente perder o objeto de foco, totalizando 144 estados. As recompensas enviadas ao agente também estão representadas na Figura 2.20, em função de cada estado: 1 para os estados terminais centrais, -1 para os estados terminais de contorno e -1/25 para os demais estados.

Em cada estado não terminal, o agente pode tomar 4 tipos de ações diferentes, acionando seus motores para mover a câmera para baixo, para cima, à esquerda e à direita. Assume-se que os movimentos realizados pelos atuadores sejam em espaços discretos e numa distância angular coincidente com o tamanho de cada ponto da imagem reamostrada.

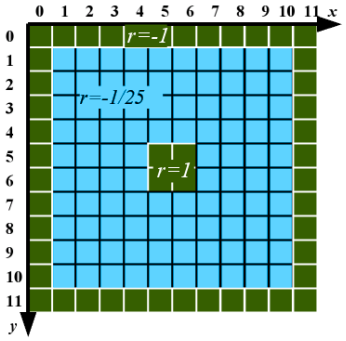


FIGURA 2.20 - Possíveis estados do agente e as respectivas recompensas. Estados no contorno e centro são terminais.

A aprendizagem foi realizada com o algoritmo *Q-learning*, que utilizou os seguintes parâmetros: $\alpha = 0.9$ e o coeficiente $\gamma = 1$. Durante a etapa de treinamento, a política adotada pelo agente foi a seleção aleatória das ações, segundo uma distribuição uniforme. O treinamento é dividido em episódios, que inicia com o posicionamento aleatório da câmera e termina quando o agente atinge um estado final. Para avaliação do aprendizado, adotou-se uma política de escolha do maior Q-valor, ou seja, ϵ-*greedy* $= 0$.

As setas nos estados da Figura (2.21) ilustram as ações que o agente deve tomar em cada estado, depois de um certo número de episódios de treinamento. A escolha da ação é um reflexo direto dos Q-valores estimados, uma vez que se segue uma política ϵ-*greedy* $= 0$. Após 40 episódios (Figura 2.21-a) percebe-se uma tendência na convergência dos valores, ficando essa tendência mais clara com 120 episódios (Figura 2.21-b). Já com 160 episódios (Figura 2.21-c) o agente já tem capacidade plena de posicionar a câmera corretamente, a partir de qualquer estado inicial, ainda que as ações não sejam ótimas. Com 1560 episódios os Q-valores convergem, resultando em ações ótimas, como ilustrado na Figura 2.21-d.

Com este exemplo de aplicação é possível observar que o agente pôde aprender a tomar decisões, utilizando-se apenas de sinais de reforço do ambiente.

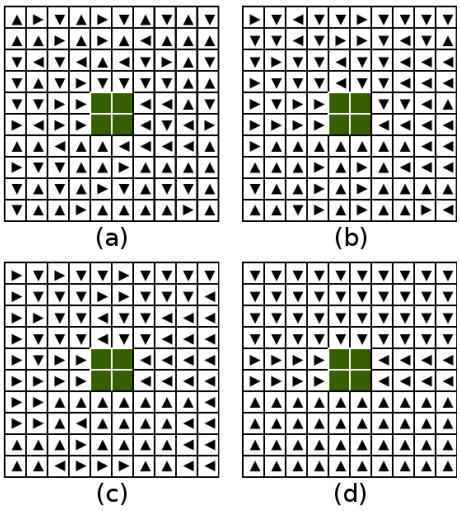


FIGURA 2.21 - Política que mapeia as ações em cada estado, após (a) 40, (b) 120, (c) 160 e (d) 1560 episódios de iteração com o ambiente. As setas indicam a ação do agente em cada estado.

CAPÍTULO 3

VISÃO COMPUTACIONAL

Segundo Shapiro e Stockman (2001) o objetivo da visão computacional "*é tomar decisões úteis a respeito de objetos físicos e cenas baseadas em imagens percebidas*". É uma área da computação que está intimamente ligada ao processamento digital de imagens, mas que tem também outras áreas aliadas, como Inteligência Artificial, reconhecimento de padrões e análise de cenas (HORN, 1986).

A visão computacional inicia-se no processo de captura da imagem, através da percepção de luz no ambiente. Vários tipos de sensores podem ser utilizados para detectar a luz, e dependendo da aplicação, uma faixa do espectro eletromagnético será selecionada. Em geral, prefere-se a faixa do espectro visível ao olho humano e a região do infravermelho próximo. Os dispositivos mais comuns para captura de imagens são as câmeras CCDs (*charge-coupled device*), que convertem a energia de luz recebida em sinais elétricos (GRAHAM, 1998), permitindo a formação de uma imagem digital da cena.

Em aplicações de navegação autônoma, a visão computacional tem se mostrado uma ferramenta promissora, que através de operadores adequados, pode extrair um grande volume de informações do ambiente. Uma das abordagens freqüentemente utilizada é a navegação do robô móvel por meio de marcas visuais (*landmarks*) espalhadas pelo ambiente. Essas marcas são artefatos artificiais ou naturais que ajudam a orientação e posicionamento do robô, além de facilitar a construção de mapas. Para tanto, pode-se utilizar placas de sinalizações já existentes no ambiente, como no trabalho de Mata *et al.* (2002) em que um sistema aprende a reconhecer placas de trânsito, com o uso de algoritmos genéticos. Já no trabalho de Gaussier *et al.* (1997), Matsumoto *et al.* (2003), algumas imagens do ambiente são fornecidas previamente ao robô, que as utiliza posteriormente para determinar sua localização e a escolha de ações. Visando amenizar a tarefa de ensinar marcas de um ambiente ao agente, Kidono *et al.* (2003) propõem um modelo em que um operador humano guia por controle remoto um robô pelo ambiente, que vai construindo um mapa a partir de imagens obtidas. Em seguida, o sistema de navegação é capaz de otimizar sua trajetória até um dado objetivo, de forma autônoma.

Outras abordagens de navegação por imagens procuram extrair uma informação es-

pecífica da cena, de modo a utilizá-la no sistema de controle. Para tanto, várias técnicas clássicas de processamento de imagens, tais como identificação de cores e texturas, extração de bordas, reconhecimento de formas, entre outras, podem contribuir para a construção de um operador visual. Por exemplo, em (CICIRELLI *et al.*, 2000) um operador visual identifica a localização de uma porta em uma imagem, da qual estima-se a posição relativa da câmera. Um robô equipado com a câmera aprende com o algoritmo *Q-learning* a calcular sua trajetória até atingir a região próxima a porta.

Contudo, é importante ressaltar, que o uso de operadores clássicos de processamento de imagens, podem levar ao aumento do custo computacional do sistema sensor de um agente. Uma alternativa é privilegiar a implementação de operadores visuais com técnicas de inteligência computacional, que aprendam o processo de computação envolvido nas técnicas clássicas (BITTENCOURT; OSÓRIO, 2002). Técnicas de aprendizado de máquina do tipo Redes Neurais Artificiais, por exemplo, são conhecidas por seu potencial na velocidade de realização de cálculos, uma vez que viabilizam a computação paralela. Em seu clássico trabalho, anteriormente introduzido, Jochem e Pomerleau (1996) descrevem o sistema *Autonomous Land Vehicle in a Neural Network* (ALVINN) que usa redes neurais para aprender a dirigir um veículo real em rodovias americanas, guiando-se por imagens da pista.

O uso de câmeras em robôs móveis, para a exploração de ambientes desconhecidos, é um sensor indicado, pois além de auxiliar na navegação, permite a captura de imagens que serão analisadas por especialistas humanos. Este é o caso de estudos com robôs que são desenvolvidos para exploração planetária, como pode ser visto nos trabalhos de Singh *et al.* (2000), Goldberg *et al.* (2002), que pode também ser combinado com outros tipos de sensores através de técnicas de fusão (HUNTSBERGER *et al.*, 2003). Para missões de reconhecimento, estuda-se ainda a aplicação de veículos aéreos não tripulados (UAVs, do inglês *Unmanned Aerial Vehicles*), que trazem uma complexidade adicional de navegação no espaço aéreo. Em virtude do objetivo da missão e a limitação de carga útil, o uso de câmeras é comum. Elfes *et al.* (1998) apresentam o projeto *AURORA* que visa desenvolver tecnologias de controle, navegação e sensoreamento de modo a prover autonomia a dirigíveis robóticos. Uma câmera é utilizada em conjunto com um *Global Positioning System* (GPS) para planejar trajetórias a partir de imagens do solo e procurar alvos específicos. O uso de helicópteros e aviões também tem atraído os grupos de pesquisa em navegação

autônoma aérea, como pode ser observado no grande número de publicações na área (SARIPALLI *et al.*, 2003; FREW *et al.*, 2004; OLLERO *et al.*, 2004; NILSSON, 2005).

Uma classe importante de operadores de visão computacional é a que visa inferir informações sobre o mundo, referenciadas em três dimensões (3D), como por exemplo, informações de profundidade em uma imagem. Isto pode ser realizado de várias maneiras, conhecendo-se um modelo do objeto alvo, como no trabalho de Brown *et al.* (2000), ou usando um conjunto de dados de textura, sombras e movimento da cena, entre outras. Porém uma das técnicas mais populares são os sistemas de visão estéreo, nos quais duas imagens de uma mesma cena são obtidas simultaneamente, geralmente por um par de câmeras posicionadas lado-a-lado, de maneira que os planos das imagens sejam coplanares (SILVA, 1999). A precisa fixação das câmeras passa a ser então um fator crítico para o correto funcionamento do sistema. Além disto, a distância entre os centros dos planos de imagem, chamada de *baseline*, deve ser conhecida. Para se calcular a distância de um alvo até as câmeras, um método de correspondência de imagens deverá ser utilizado, que encontrará a localização da projeção de pontos 3D nas duas imagens obtidas. Exemplos de aplicações de visão estéreo em navegação autônoma podem ser vistos nos trabalhos (SINGH *et al.*, 2000; GOLDBERG *et al.*, 2002; CHANG; LO, 2003; LOBO *et al.*, 2003; PARK *et al.*, 2004).

3.1 Imagens

Uma imagem pode ser definida matematicamente como uma função $g(u, v)$ que representa o nível de intensidade de brilho, na coordenada espacial u e v, de precisão infinita. Pode-se interpretar a intensidade de brilho como níveis de cinza contínuos, com limites em 0 representando o preto, e em 1 representando o branco. Na prática, obtém-se uma *imagem digital* $I[r, c]$ que é uma matriz de R linhas e C colunas, de amostras discretas de $g(u, v)$, onde $r \in R$ e $c \in C$. Para gerar $I[r, c]$ é preciso *quantizar* os níveis de cinza em K valores, e *amostrar* as coordenadas u e v em uma grade de R linhas por C colunas (GONZALEZ; WINTZ, 1987).

O processo de amostragem e quantização para a dimensão u é ilustrado na Figura 3.1. A cada intervalo discreto Δu o nível de cinza é amostrado e quantizado para níveis de intensidade de 0 a 7, ou seja, $K = 2^3$. Fica claro através da Figura 3.1 que quanto maior o número de K, R e C, melhor I aproxima g.

Cada elemento de $I[r, c]$ é chamado de *pixel*. Quando um *pixel* representa apenas

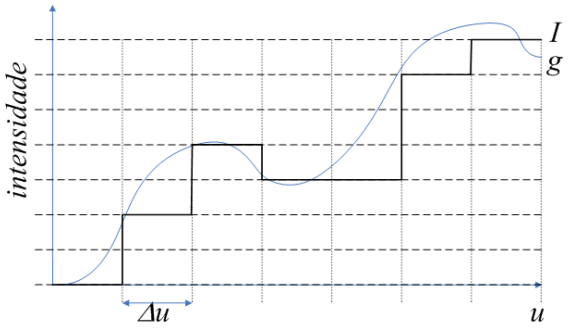


FIGURA 3.1 - Processo de amostragem e quantização para a dimensão u, com $K = 2^3$.

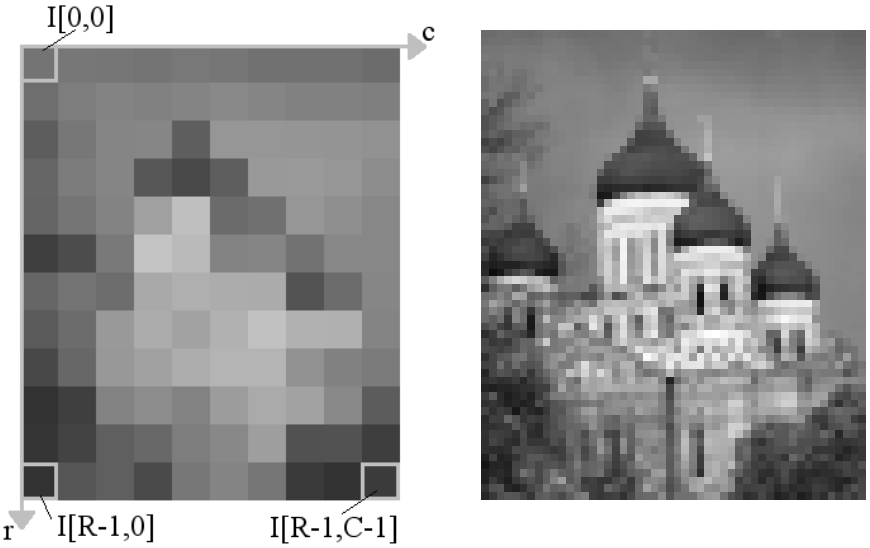
um nível de intensidade, a imagem é *monocromática*, ou em escalas de cinza. Se as coordenadas da imagem estiverem mapeadas apenas para valores 0 ou 1, diz-se que a *imagem* é *binária*. Mas é possível que para cada ponto da imagem digital exista um vetor de valores associados, compondo uma *imagem multi-espectral*. A imagem multi-espectral mais comum é a imagem colorida, que imita o sistema visual humano, contendo 3 canais de intensidades na faixa do Vermelho, Verde e Azul (RGB, do inglês, *red, green* e *blue*), no sistema de cores aditivo (BAXES, 1994).

A Figura 3.2 compara 4 imagens diferentes obtidas da mesma cena, quantificadas em 256 níveis de intensidade de brilho. Observa-se na Figura 3.2-a que muitos detalhes da imagem foram perdidos, devido a sua grade de amostragem ser muito pequena, com 12 linhas por 10 colunas. O nível de detalhes melhora a medida que se eleva o tamanho da grade para $R = 61 \times C = 50$, na Figura 3.2-b, e para $R = 720 \times C = 587$, na Figura 3.2-c. A Figura 3.2-d representa a mesma imagem da Figura 3.2-c, porém cada um de seus *pixels* contém 3 canais (RGB).

Na Figura 3.2-a pode-se observar ainda o sistema de coordenadas de imagem adotada neste trabalho. Este sistema utiliza um eixo para a linha e outro para coluna, iniciando em $[0, 0]$ no canto superior esquerdo da imagem.

Os *pixels* de uma imagem podem ainda assumir outros tipos de valores arbitrários. A *imagem rotulada* mapeia cada coordenada de *pixel* para valores pertencentes a um conjunto finito de símbolos. Pode-se utilizar esse tipo de imagem para classificar uma imagem temática, que normalmente é colorida com cores falsas (SHAPIRO; STOCKMAN, 2001).

Muitos algoritmos de visão computacional consideram o relacionamento entre os



(a) Imagem amostrada com 12 x 10 *pixels*.

(b) Imagem amostrada com 61 x 50 *pixels*.

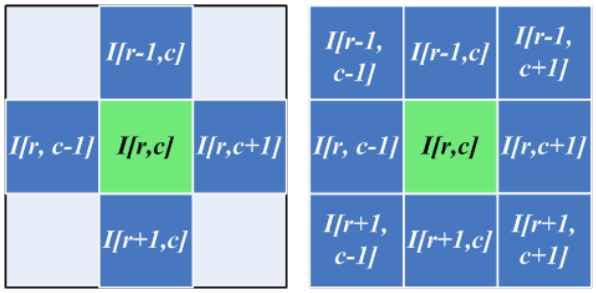
(c) Imagem amostrada com 720x 587 *pixels*.

(d) Imagem colorida amostrada com 720x 587 *pixels*.

FIGURA 3.2 - Diferentes imagens obtidas de uma mesma cena, com $K = 2^8$. (a) sistema de coordenadas da imagem com linhas e colunas, iniciando na posição $[0, 0]$.

elementos das imagens, em especial o relacionamento de *pixels* vizinhos. Seja $p = [r, c]$ o *pixel* de uma imagem, a *vizinhança-4* de p, denotada por $N_4(p)$, é composta pelos seus 4 *pixels* vizinhos acima, abaixo, a esquerda e a direita, como mostra a Figura 3.3-a. A *vizinhança-8* de p é composta por $N_4(p)$ mais os 4 pixels vizinhos nas diagonais, totalizando 8 *pixels*, ilustrados na Figura 3.3-b (GONZALEZ; WINTZ, 1987).

Dado um conjunto B de valores de intensidades de brilho, dois *pixels* p e p', com



(a) Vizinhança-4. (b) Vizinhança-8.

FIGURA 3.3 - Valores dos vizinhos de um *pixel* $[r, c]$.

valores em B, estão *conectados por um caminho* se existir uma seqüência de *pixels* $p = [r_0, c_0], [r_1, c_1], \ldots, [r_n, c_n] = p'$, na qual $I[r_i, c_i] \in B \forall i = 0, \ldots, n$ e $[r_i, c_i] \in N_8([r_{i-1}, c_{i-1}) \forall i = 1, \ldots, n$ (SHAPIRO; STOCKMAN, 2001).

Uma aplicação típica que envolve a vizinhança de *pixels* é a filtragem espacial. O filtro pode ser aplicado por uma operação de convolução, em que uma máscara é sobreposta a imagem I, gerando uma nova imagem I', conforme expressa a Equação (3.1).

$$I'[r, c] = I * h \tag{3.1}$$

$$I'[r, c] = \sum_{m=r-w/2}^{r+w/2} \sum_{n=c-w/2}^{c+w/2} I[m, n] h[r - m, c - n] \tag{3.2}$$

onde $*$ é a operação de convolução;
w é o tamanho do filtro
h é o filtro.

A Figura 3.4 ilustra o processo de convolução na imagem I a fim de se formar a imagem I'. O valor do *pixel* $[2, 2]$ é calculando levando em consideração sua vizinhança-8. Observa-se que uma situação especial ocorre nas bordas das imagem, que deverá ser desconsiderada ou calculada de forma diferente.

Entre os filtros espaciais mais comuns, destacam-se o filtro: de média, de mediana e passa-alta (GONZALEZ; WINTZ, 1987).

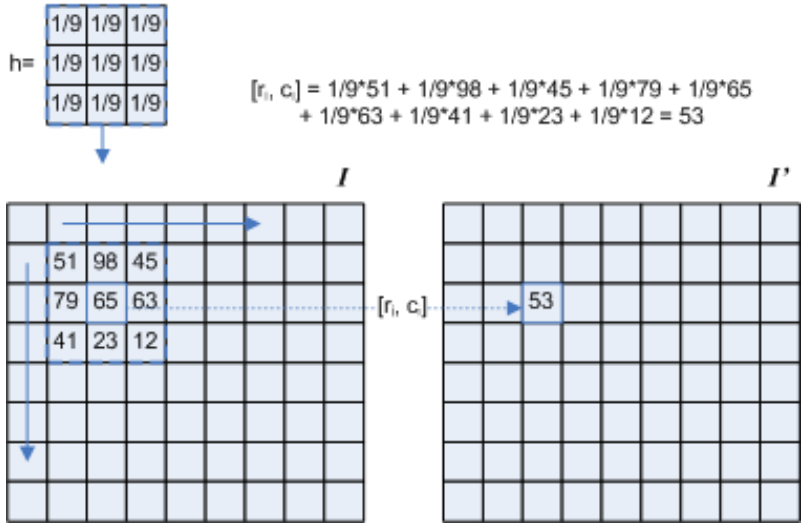


FIGURA 3.4 - Convolução da imagem I utilizando a máscara h, formando a imagem I'.

3.2 Detecção de Objetos na Cena

Um dos fatores críticos para se alcançar o objetivo deste trabalho é a percepção de objetos no ambiente. Através do conhecimento da existência de obstáculos, o agente poderá navegar com segurança e eventualmente reconhecer sua localização. Várias técnicas de processamento digital de imagens e visão computacional podem contribuir para a construção de um algoritmo de detecção de objetos, tais como: detecção de sombras, bordas, texturas, classificação e segmentação de imagens, reconhecimento de formas a partir de movimento, visão estéreo, entre outras (SHAPIRO; STOCKMAN, 2001).

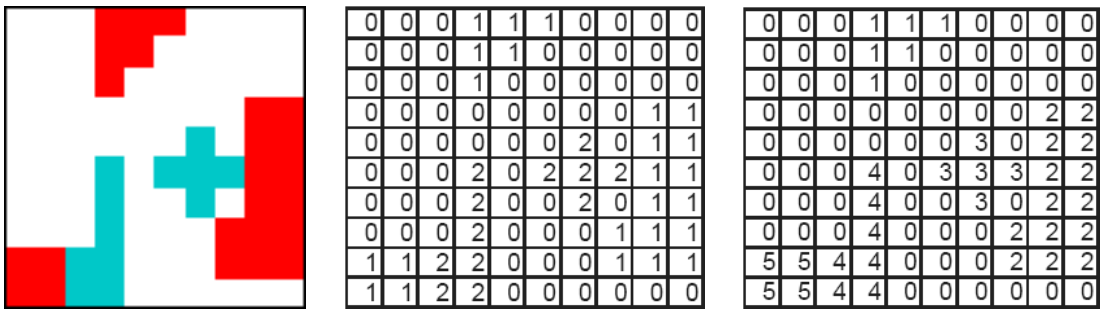
Neste trabalho, o sistema de visão computacional é equipado com apenas uma câmera e os objetos do ambiente destacam-se do fundo da cena por diferenças de cores. Uma maneira simples então de se detectar a presença dos objetos na cena é classificar os *pixels* da imagem, segundo suas características espectrais. Esta estratégia compõem o processo de detecção de objetos, representada na Figura 3.5. A primeira etapa consiste em capturar uma imagem colorida da cena que será classificada. A classificação baseada em cores, pode ser uma simples limiarização, ou algum método mais sofisticado, que produzirá uma imagem com os *pixels* rotulados com as classes. Para a identificação dos objetos, dois métodos são apresentados nas sub-secções a seguir, um matricial e outro vetorial. O processo termina com a extração de atributos dos objetos identificados, com o propósito de prover informações de alto nível

ao agente de aprendizado. Esses atributos podem ser do tipo: área aparente, classe, forma, distância, entre outros.

FIGURA 3.5 - Processo de detecção de objetos.

3.2.1 Método Matricial

A detecção de objetos pelo método matricial, visa identificar regiões formadas por conjuntos de *pixels* conectados, que representam um objeto na cena. Por exemplo, dada a imagem colorida da Figura 3.6-a, a imagem rotulada I_c resultante do processo de classificação é a Figura 3.6-b. Observa-se que os rótulos (valores 1 e 2) foram estipulados segundo as cores dos objetos. A identificação dos objetos pode então ser expressa por regiões de *pixels* rotuladas, como na imagem I_l da Figura 3.6-c, onde destacam-se 5 objetos.



0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	2	0	1	1
0	0	0	2	0	2	2	2	1	1
0	0	0	2	0	0	2	0	1	1
0	0	0	2	0	0	0	1	1	1
1	1	2	2	0	0	0	1	1	1
1	1	2	2	0	0	0	0	0	0

0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	2	2
0	0	0	0	0	0	3	0	2	2
0	0	0	4	0	3	3	3	2	2
0	0	0	4	0	0	3	0	2	2
0	0	0	4	0	0	0	2	2	2
5	5	4	4	0	0	0	2	2	2
5	5	4	4	0	0	0	0	0	0

(a) Imagem obtida da cena.

(b) Imagem rotulada com classes de objetos.

(c) Imagem rotulada com regiões de objetos.

FIGURA 3.6 - Detecção de objetos pelo método matricial.

O método descrito em Shapiro e Stockman (2001) é baseado no algoritmo clássico de Rosenfeld e Pfaltz (1966) para computar componentes conexos, melhorado com as estruturas *union-find* de Tarjan (1975).

Computar componentes conexos significa identificar regiões na imagem classificada que não estejam conectados com outras regiões. Esse processamento é efetuado em 2 etapas de varreduras pela imagem classificada I_c. A primeira consiste em varrer

a imagem classificada I_c e identificar *pixels* vizinhos para criar a imagem rotulada I_l com rótulos temporários, registrando se existe equivalência entre os rótulos. Uma segunda varredura é feita em I_l para substituir cada rótulo temporário por um rótulo que represente sua classe de equivalência.

O algoritmo da Figura 3.7 descreve esse processo, que foi adaptado para suportar uma imagem I_c com várias classes e não somente uma imagem binária. A função $encontrarRotuloVizinhosAnteriores()$ visa identificar o rótulo dos *pixels* vizinhos que pertencem a mesma classe do *pixel* atual. A busca na vizinhança é feita apenas para os *pixels* $[r, c-1]$ e $[r-1, c]$ em N_4; e $[r, c-1]$, $[r-1, c-1]$ e $[r-1, c]$ em N_8.

As estruturas de dados *union-find* armazenam e controlam uma coleção de conjuntos disjuntos. Os elementos dos conjuntos representam rótulos de uma mesma classe de equivalência que são armazenadas em uma estrutura de árvore, implementadas pela lista $PARENT$. Através da lista $PARENT$ é possível identificar um rótulo único e comum a todos os rótulos de uma região. Esse procedimento de busca na estrutura em árvore é feito pela função *find*, descrita no algoritmo da Figura 3.8. Quando um novo *pixel* a ser rotulado une dois ou mais conjuntos até então disjuntos, a função *union* é utilizada para uní-los na lista $PARENT$, como mostra o algoritmo da Figura 3.9.

Por fim, quando todos os objetos estiverem identificados em regiões O rotuladas na imagem I_l, pode-se extrair algumas informações sobre o objeto, além da classe que já foi identificada. O mais simples é calcular a *área* de cada região

$$O_A = \sum_{(r,c)\in O} 1, \tag{3.3}$$

através da contagem do seu número de *pixels*.

O *centróide* é uma localização média da região na imagem, em que as coordenadas (com precisão não inteira) $(\bar{r}, \bar{c})$ são encontradas por

$$O_{\bar{r}} = \frac{1}{O_A} \sum_{(r,c)\in O} r, \tag{3.4}$$

e

$$O_{\bar{c}} = \frac{1}{O_A} \sum_{(r,c)\in O} c. \tag{3.5}$$

Requer imagem classificada I_c com classes de 1 a n
  inicializar imagem de saída I_l com zeros
  inicializar $label \leftarrow 1$
  inicializar lista $PARENT$ vazia
  para cada $r \leftarrow 0, \ldots, R$ **faça** {Varredura 1 na imagem I_c para atribuir rótulos iniciais a I_l}
    para cada $c \leftarrow 0, \ldots, C$ **faça** {Processar linha r}
      se $I_c[r, c] > 0$ **então**
        $L \leftarrow encontrarRotuloVizinhosAnteriores(r, c, I_c[r, c])$
        se $L = \emptyset$ **então**
          $m \leftarrow label$
          $PARENT[m] \leftarrow 0$
          $label \leftarrow label + 1$
        senão
          $m \leftarrow \min(L)$
        fim se
        $I_l[r, c] \leftarrow m$
        para cada $l \in L \wedge l \neq m$ **faça**
          $union(l, m, PARENT)$
        fim para cada
      fim se
    fim para cada
  fim para cada
  para cada $r \leftarrow 0, \ldots, R$ **faça** {Varredura 2 na imagem I_l para rotulá-la com classes de equivalência}
    para cada $c \leftarrow 0, \ldots, C$ **faça**
      se $I_l[r, c] > 0$ **então**
        $I_l[r, c] \leftarrow find(I_l[r, c], PARENT)$
      fim se
    fim para cada
  fim para cada

FIGURA 3.7 - Algoritmo *Computar componentes conexos*.

As coordenadas dos *pixels* mais ao topo $O_{tm} = \min\{r | r \in O\}$, mais inferior $O_{bm} = \max\{r | r \in O\}$, mais a esquerda $O_{lm} = \min\{c | c \in O\}$, e mais a direita $O_{rm} = \max\{c | c \in O\}$, podem ser utilizadas para definir um *retângulo envolvente* da região, ou seja, o retângulo formado pelos segmentos: $\overline{tm\ rm}$, $\overline{rm\ bm}$, $\overline{bm\ lm}$ e $\overline{lm\ tm}$.

Conhecer a altura e largura aparente do objeto também é interessante, pois traz uma noção do seu espaço ocupado. O maior desses valores será armazenado, com o

nome *maior eixo*

$$O_{Me} = \max\{O_{rm} - O_{lm} + 1, O_{bm} - O_{tm} + 1\}. \tag{3.6}$$

Outro atributo comumente extraído de regiões de *pixels* é o *comprimento do perímetro*

$$O_{|P|} = |k|(rk+1, c_{k+1}) \in N_4(r_k, c_k)| + \sqrt{2}\,|k|(r_{k+1}, c_{k+1}) \in N_8(r_k, c_k) - N_4(r_k, c_k)|\,, \tag{3.7}$$

onde, os elementos (r_k, c_k) pertencem ao perímetro da região e estão ordenados de forma que cada par de *pixels* sucessivos são vizinhos (SHAPIRO; STOCKMAN, 2001).

Requer o rótulo l do qual deve-se encontrar a raiz
Requer a lista $PARENT$
  $j \leftarrow l$
  enquanto $PARENT[j] \neq 0$ **faça**
    $j \leftarrow PARENT[j]$
  fim enquanto
  return j

FIGURA 3.8 - Algoritmo *find*.

Requer os rótulos l e m dos conjuntos a serem unidos
Requer a lista $PARENT$
  $j \leftarrow l$
  $k \leftarrow m$
  enquanto $PARENT[j] \neq 0$ **faça**
    $j \leftarrow PARENT[j]$
  fim enquanto
  enquanto $PARENT[k] \neq 0$ **faça**
    $k \leftarrow PARENT[k]$
  fim enquanto
  se $j \neq k$ **então**
    $PARENT[k] \leftarrow j$
  fim se

FIGURA 3.9 - Algoritmo *union*.

O conjunto de *pixels* que pertencem ao perímetro pode ser definido através da vizinhança 8

$$O_{P4} = \{(r,c) \in O | N_8(r,c) - O \neq \emptyset\}, \tag{3.8}$$

ou vizinhança 4

$$O_{P8} = \{(r,c) \in O | N_4(r,c) - O \neq \emptyset\}. \tag{3.9}$$

Calculados a área e o comprimento do perímetro de uma região, pode-se então extrair um atributo de *circularidade* dessa região

$$O_{Ci} = \frac{{O_{|P|}}^2}{O_A} \tag{3.10}$$

A circularidade dá uma idéia do formato da região, se é mais achatada ou oval. Por exemplo, o menor valor de circularidade é aproximadamente $12,6$ para um círculo (não digital). Para um quadrado, o valor de circularidade é 16, tendendo a crescer para formas mais alongadas.

3.2.2 Método Vetorial

O método de detecção vetorial visa identificar objetos na cena, representando-os por polígonos P simples. Os polígonos resultantes podem ser convexos ou côncavos e são formados por pontos ordenados $P =< pt_0, pt_1, \ldots, Pt_{n-1}, Pt_n >$, seguindo o sentido horário. Espera-se que haja uma redução no custo computacional no processo de identificação dos objetos na cena, para os casos em que os objetos forem aproximados por polígonos convexos. Quando objetos côncavos estiverem presentes na cena, a primeira etapa de processamento identificará um ou mais polígonos convexos que se interseccionam, pertencendo ao mesmo objeto. Nestes casos, uma operação de união dos polígonos será aplicada, utilizando o algoritmo descrito posteriormente, nesta secção. A seguir o procedimento de detecção dos polígonos é descrito.

3.2.2.1 Detecção de Polígonos

Nogueira *et al.* (2004) utilizaram em seu trabalho um algoritmo para encontrar e representar de forma vetorial um triângulo com dimensões conhecidas, em uma imagem digital. Alguns pontos de bordas do triângulo são identificados e posteriormente interpolados por três retas, a fim de formarem um triângulo. O método apresentado neste trabalho é baseado na abordagem de Nogueira *et al.* (2004), mas propõe extensões para detecção de vários objetos com qualquer formato, aproximados por

polígonos.

Os polígonos são encontrados em duas etapas: inicialmente se identifica na imagem a presença de objetos; e em seguida procura-se por pontos que pertençam a borda externa desses objetos.

O processo de busca dos vários objetos pode ser efetuado por varredura na imagem classificada, em intervalos discretos ou por outro tipo de amostragem. O algoritmo para detecção de polígonos, da Figura 3.10, usa a abordagem de varredura em I_c, em intervalos discretos representados por uma grade de controle, *ctrlGrid*, de R_G linhas e C_G colunas, tal que $R_G \leq R$ e $C_G \leq C$. Quando for identificado algum *pixel* que pertence a alguma classe de objetos, toma-se esse *pixel* como inicial P_0^M e começa a etapa de detecção de pontos para formação do polígono.

Requer imagem classificada I_c com classes de 1 a n
  criar um *grid* de controle de busca de pontos iniciais $ctrlGrid[r_g, c_g] | r_g \in R_G, R_G \leq R, c_g \in C_G, C_G \leq C$
  $k \leftarrow 0$
  para cada $r_g \in R_G$ **faça**
    para cada $c_g \in C_G$ **faça**
      se $ctrlGrid[r_g, c_g] = 0 \wedge I_c[r_g, c_g] > 0$ **então**
        $SetPt = \text{detectarPontos}(r_g, c_g, I_c[r_g, c_g])$
        se $SetPt \neq \emptyset$ **então**
          $P_k \leftarrow SetPt$
          $k \leftarrow k + 1$
        fim se
      fim se
    fim para cada
  fim para cada
  return conjunto de polígonos P

FIGURA 3.10 - Algoritmo *Detectar polígonos*.

A partir da linha de P_0^M busca-se pontos extremos do objeto a esquerda P_0^L e a direita P_0^R, variando o índice das colunas em intervalos discretos Δc. Os limites de borda do objeto é identificado pela mudança da classe do *pixel*, na ausência de classe ($I_c[r, c] = 0$) ou pelo limite da imagem. Em seguida, um novo ponto P_1^M é definido pela variação da linha de P_0^M em um intervalo discreto Δr, e pela coluna

representada pela média das colunas de P_0^L e P_0^R. O processo segue então de forma sucessiva, buscando pontos extremos P_i^L e P_i^R em uma determinada linha, definindo um novo ponto P_i^M com a variação da linha e com a coluna média de P_i^L e P_i^R. A busca pára quando não se encontra mais *pixels* que pertençam a mesma classe de objeto, ou a distância entre os pontos extremos a esquerda e a direita forem inferiores a um limiar estipulado. O processo iniciado em P_0^M é repetido para busca de pontos extremos nas linhas acima de P_0^M.

A Figura 3.11 ilustra o processo de busca de pontos extremos que representam um objeto qualquer. O processo iniciou pelo ponto Pt_0^M e em seguida foram encontrados os pontos Pt_0^L e Pt_0^R. Um novo ponto Pt_1^M foi definido até que a variação abaixo das linhas terminou com os pontos Pt_D^L e Pt_D^R, em que neste exemplo $D = 7$. O último encontro de pontos (Pt_{-1}^L e Pt_U^R, com $U = 1$) ocorreu na variação acima das linhas.

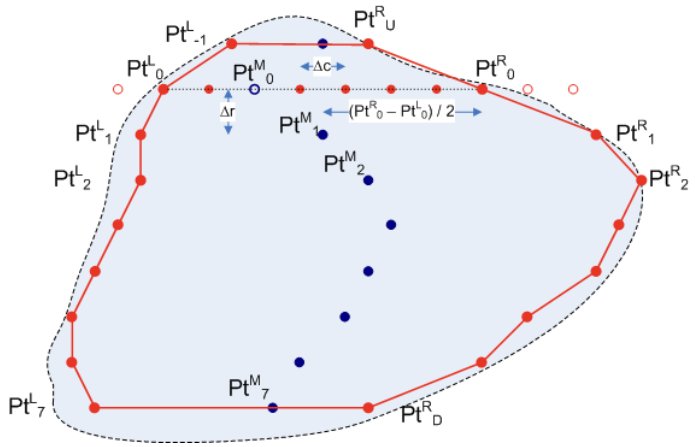


FIGURA 3.11 - Exemplo de identificação de pontos extremos para construção do polígono que representa o objeto.

Com a variação de linhas e colunas no encontro dos limites dos objetos, a grade *ctrlGrid* assume uma função de controle, registrando se uma determinada coordenada de amostra de I_c já foi alocada a algum polígono, evitando verificações desnecessárias.

Para armazenar a seqüência dos pontos que compõem o polígono, implementa-se uma lista encadeada circular, em que o último ponto da lista aponta para o primeiro. Durante a etapa de identificação dos pontos, deve-se construir o encadeamento dos pontos mantendo a coerência na ordenação em sentido horário dos pontos.

Para gerenciar a construção da lista, este trabalho propõe uma estratégia de uso de ponteiros, que guardarão referências para nodos da lista de pontos. A identificação das variáveis de ponteiros é convencionada da seguintes maneira: $\underline{NomePonteiro_{Pt}}$. O símbolo $\triangleright$ é utilizado para atribuir uma referência ao ponteiro, como por exemplo $\underline{New_{Pt}} \triangleright Pt_i$. O próximo ponteiro de um ponteiro é obtido pela operação $\underline{New_{Pt}} + 1$.

O ponteiro $\underline{First_{Pt}}$ guarda uma referência para o primeiro ponto de um polígono. Quatro ponteiros auxiliares $\underline{NW_{Pt}}, \underline{NE_{Pt}}, \underline{SE_{Pt}}$ e $\underline{SW_{Pt}}$ viabilizam a montagem da lista encadeada final, fazendo alusão aos pontos cardeais *noroeste*, *nordeste*, *sudeste* e *sudoeste*. A Figura 3.12 ilustra a lista encadeada circular com os pontos do polígono construído a partir do objeto da Figura 3.11. Percorrendo a imagem acima, abaixo, a esquerda e direita, extraiu-se os pontos extremos Pt_U^L, Pt_U^R, Pt_D^R e Pt_D^L, que têm suas referências guardadas respectivamente pelos ponteiros $\underline{NW_{Pt}}, \underline{NE_{Pt}}, \underline{SE_{Pt}}$ e $\underline{SW_{Pt}}$.

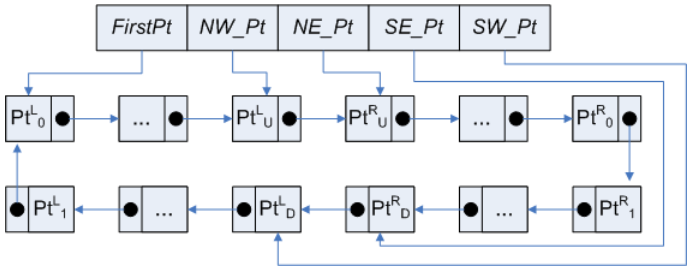


FIGURA 3.12 - Lista encadeada circular que armazena os pontos e a seqüência que compõem o polígono da Figura 3.11. Cinco ponteiros guardam referências de nodos especiais.

Percebe-se que devido a característica de varredura de linhas e colunas do algoritmo de detecção de pontos extremos, os pontos podem ser agrupados em quadrantes diferentes. A função dos ponteiros $\underline{NW_{Pt}}, \underline{NE_{Pt}}, \underline{SE_{Pt}}$ e $\underline{SW_{Pt}}$ é manter o controle dos últimos pontos identificados em cada quadrante, podendo assim se recuperar a ordem dos pontos do polígono para o correto encadeamento. A separação dos pontos extremos em quadrantes é exemplificada na Figura 3.13, na qual estão também indicados os ponteiros auxiliares. A título de exemplificação do processo de encadeamento, supõem-se o fragmento da lista no quadrante NW representado na Figura 3.14-a. O ponteiro $\underline{First_{Pt}}$ guarda a referência para o primeiro nodo da lista, que já está encadeada até o último ponto Pt_k, referenciado por $\underline{NW_{Pt}}$. Um novo ponto Pt_{k+1} está sendo criado, e temporariamente apontado por $\underline{New_{Pt}}$. A primeira medida é encadear o novo elemento na lista (Figura 3.14-b) efetuando o apontamento $\underline{NW_{Pt}} + 1 \triangleright \underline{New_{Pt}}$ e em seguida atualizar o ponteiro $\underline{NW_{Pt}}$ (Figura 3.14-c). O encadeamento é semelhante para os demais quadrantes, sempre respeitando o sentido

horário dos apontamentos. Ao final, $\underline{NW_{Pt}}$ aponta para $\underline{NE_{Pt}}$ (Figura 3.14-d) e $\underline{SE_{Pt}}$ para $\underline{SW_{Pt}}$, unindo todos quadrantes e formando uma lista circular.

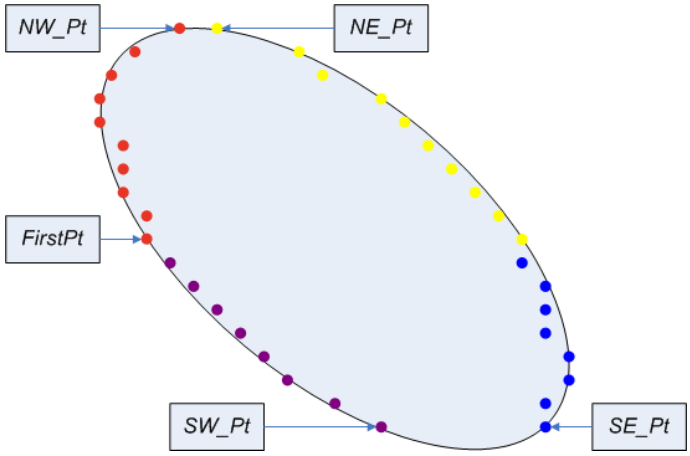
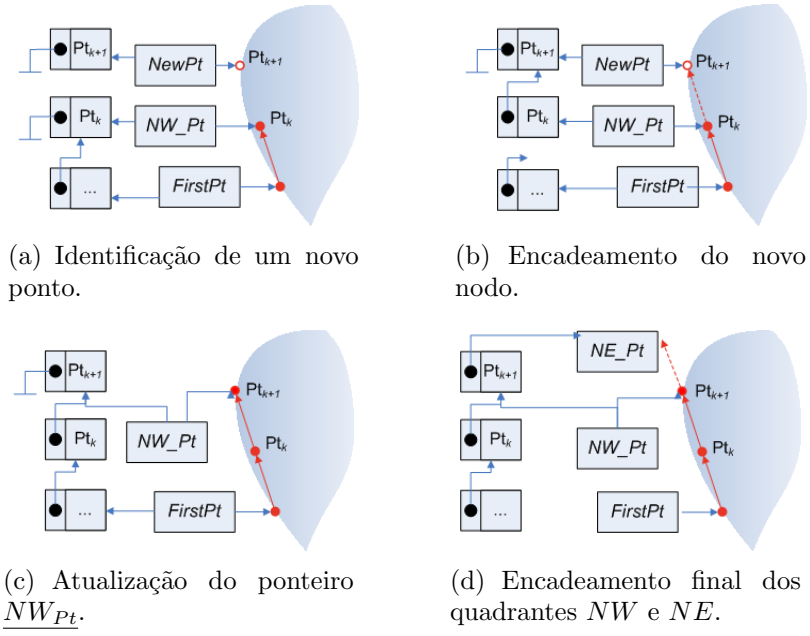


FIGURA 3.13 - Separação de pontos do polígono segundo o seu quadrante. Os últimos pontos identificados em cada quadrante são referenciados por ponteiros.



(a) Identificação de um novo ponto.

(b) Encadeamento do novo nodo.

(c) Atualização do ponteiro $\underline{NW_{Pt}}$.

(d) Encadeamento final dos quadrantes NW e NE.

FIGURA 3.14 - Atualização do ponteiro do quadrante NW, com a identificação de um novo ponto e com encadeamento final.

O procedimento completo para detecção de pontos é descrito no algoritmo da Figura 3.15, sendo as funções $variarColunas()$ e $variarLinhas()$ detalhadas nas Figuras 3.16 e 3.17, respectivamente.

Requer linha atual $i \leftarrow r_g$
Requer coluna atual $j \leftarrow c_g$
Requer classe atual $class \leftarrow I_c[r_g, c_g]$
  inicializar os passos Δr e Δc com tamanho arbitrário
  inicializar índice de pontos $p \leftarrow 0$
  inicializar ponteiro $\underline{New_{Pt}} \triangleright PtNew_p$
  inicializar os ponteiros $\underline{First_{Pt}} \triangleright \underline{NO_{Pt}} \triangleright \underline{NE_{Pt}} \triangleright \underline{SO_{Pt}} \triangleright \underline{SE_{Pt}} \triangleright \underline{New_{Pt}}$

  $lm \leftarrow percorrerLinha(i, j, -\Delta c)$ {Percorrer a linha a esquerda, variando colunas}

  $p \leftarrow p + 1$
  $\underline{New_{Pt}} \triangleright PtNew_p$
  $\underline{NE_{Pt}} \triangleright \underline{New_{Pt}}$
  $rm \leftarrow variarColunas(i, j, \Delta c)$ {Percorrer a linha a direita, variando colunas}
  se $(rm - lm) < MINLEN$ **então**
    $SetPt \leftarrow \emptyset$
    return $SetPt$
  fim se

  $j \leftarrow int(\frac{rm+lm}{2})$ {nova coluna é a média das extremidades encontradas}
  $\underline{SO_{Pt}} \triangleright \underline{NO_{Pt}}$
  $\underline{SE_{Pt}} \triangleright \underline{NE_{Pt}}$

  $tm \leftarrow variarLinhas(i, j, -\Delta r)$ {variar a linha acima}
  $bm \leftarrow variarLinhas(i, j, \Delta r)$ {variar a linha abaixo}
  se $(bm - tm) < MINLEN$ **então**
    $SetPt \leftarrow \emptyset$
    return $SetPt$
  fim se

  {Encadeamento final}
  $(\underline{NO_{Pt}} + 1) \triangleright \underline{NE_{Pt}}$
  $(\underline{SE_{Pt}} + 1) \triangleright \underline{SO_{Pt}}$

  {Criar o conjunto de pontos ordenados do polígono}
  $\underline{Temp_{Pt}} \triangleright \underline{First_{Pt}}$
  $p \leftarrow 0$
  repetir
    $Pt_p \leftarrow \underline{Temp_{Pt}}$
    $p \leftarrow p + 1$
    $\underline{Temp_{Pt}} \triangleright \underline{Temp_{Pt}} + 1$
  até $\underline{Temp_{Pt}} \neq \underline{First_{Pt}}$
  $SetPt =< Pt_0, Pt_1, \ldots, Pt_{n-1}, Pt_n >$
  return conjunto de pontos ordenados $SetPt$

FIGURA 3.15 - Algoritmo *Detectar Pontos*.

Requer variação de coluna $\Delta j \leftarrow \pm \Delta c$
Requer linha atual i
Requer nova coluna $jc \leftarrow j + \Delta j$
  enquanto $class = I_c[i, jc] \wedge jc \in C$ **faça**
    $jc \leftarrow jc + \Delta j$
  fim enquanto
  $\underline{New_{Pt}} \leftarrow [i, jc - \Delta j]$
  return $jc - \Delta j$

FIGURA 3.16 - Algoritmo *Variar Colunas*.

3.2.2.2 União de Polígonos

O método de detecção de polígonos não é suficiente para aproximar os objetos com qualidade. Ocorre que por problemas de iluminação, na geração da imagens, alguns objetos têm uma região côncava, gerando vários polígonos que se interseccionam. É possível então minimizar esse problema, efetuando uma operação de união dos polígonos.

O algoritmo para efetuar a união de dois polígonos que se interseccionam pode ser dividido nos seguintes passos:

a) Encontrar 1 par de polígonos que se interseccionam;

b) Determinar os pontos de intersecção do par;

c) Utilizando os pontos de intersecção, efetuar a união de 2 polígonos;

d) Retornar ao primeiro passo até que não existam mais polígonos que se interseccionam.

Greiner e Hormann (1998) apresentam uma coleção de algoritmos e estruturas de dados para efetuar uma série de operações com polígonos, incluindo polígonos complexos. Todavia, neste trabalho optou-se em desenvolver um algoritmo mais simplificado, que trate apenas a operação de união de polígonos simples, que resulte em um único polígono externo, ou seja, sem ilhas. Esperando-se que raramente todos os segmentos dos polígonos a serem unidos se interseccionem, o uso de um algoritmo eficiente de detecção de intersecções tende a diminuir o custo computacional.

Bentley e Ottmann (1979) propuseram um algoritmo bem conhecido para a tarefa de identificação de intersecções num conjunto de segmentos, conhecido por *sweep line*.

Requer variação de linha $\Delta i \leftarrow \pm\Delta r$
Requer nova linha $ir \leftarrow i + \Delta i$
Requer coluna atual $jc \leftarrow j$
enquanto $class = I_c[ir, jc] \wedge ir \in R$ **faça**

{Criar novo ponto a esquerda e encadeá-lo}
$p \leftarrow p + 1$
$\underline{New_{Pt}} \triangleright PtNew_p$
se $\Delta r < 0$ **então**
$(\underline{NO_{Pt}} + 1) \triangleright \underline{New_{Pt}}$
$\underline{NO_{Pt}} \triangleright \underline{New_{Pt}}$
senão
$(\underline{New_{Pt}} + 1) \triangleright \underline{SO_{Pt}}$
$\underline{SO_{Pt}} \triangleright \underline{New_{Pt}}$
fim se
$lm \leftarrow variarColunas(ir, jc, -\Delta c)$ {Percorrer a linha a esquerda}

{Criar novo ponto a direita e encadeá-lo}
$p \leftarrow p + 1$
$\underline{New_{Pt}} \triangleright PtNew_p$
se $\Delta r < 0$ **então**
$(\underline{New_{Pt}} + 1) \triangleright \underline{NE_{Pt}}$
$\underline{NE_{Pt}} \triangleright \underline{New_{Pt}}$
senão
$(\underline{SE_{Pt}} + 1) \triangleright \underline{New_{Pt}}$
$\underline{SE_{Pt}} \triangleright \underline{New_{Pt}}$
fim se
$rm \leftarrow variarColunas(ir, jc, \Delta c)$ {Percorrer a linha a direita}

se $(rm - lm) < MINLEN$ **então**
return $ir - \Delta i$
fim se
$jc \leftarrow int(\frac{rm+lm}{2})$ {nova coluna é a média das extremidades encontradas}
$ir \leftarrow ir + \Delta i$
fim enquanto
return $ir - \Delta i$

FIGURA 3.17 - Algoritmo *Variar Linhas*.

Neste algoritmo, uma linha varre o espaço bidimensional dos segmentos, controlando em uma lista quais os possíveis segmentos que podem se cruzar, minimizando o número de comparações. Uma fila de eventos XQ representa a seqüência ordenada de

todos pontos de extremidades (*endpoints* dos segmentos) que a linha de varredura encontrará. Em sua configuração clássica, os pontos de extremidades são ordenados pelos valores crescentes de suas coordenadas x e em caso de empate, pela y. (No sistema de coordenadas de imagens, essas coordenadas equivalem a c e r respectivamente.)

Os segmentos e respectivos pontos de extremidades dos polígonos são identificados através dos pontos que compõem esses polígonos. Assim, o primeiro segmento será formado pelo primeiro e segundo ponto do polígono. A identificação se um ponto é a extremidade da esquerda ou direita do segmento é feita pela ordenação das extremidades do segmento, onde o primeiro ponto é da esquerda e o segundo da direita.

Alguns cuidados devem ser tomados na ordenação dos pontos de extremidades na fila XQ. Para evitar que seja encontrado um ponto de intersecção entre dois segmentos sg_i e sg_{i+1} de um mesmo polígono, sendo sg_i o segmento antecessor de sg_{i+1}, é importante que o ponto da extremidade direita de sg_i venha antes da extremidade esquerda de sg_{i+1} na fila XQ. Adaptações também são necessárias para lidar com intersecções de segmentos colineares.

O algoritmo de Bentley-Ottman, sem as adaptações para o uso com polígonos deste trabalho, é descrito com maiores detalhes no Anexo A.

Identificados os pontos de intersecção dos segmentos dos polígonos, a idéia geral do algoritmo de união consiste em contornar os segmentos externos dos polígonos P e P' a serem unidos, formando o novo polígono P^*. Entende-se por segmentos externos aqueles segmentos que não estão na região interna de P ou P'. Cada segmento que intersecciona outro é dividido em dois, a partir do ponto de intersecção.

Inicia-se o processo escolhendo um ponto que seja externo a P e P'. A título de ilustração, assume-se que este ponto é $pt_k \in P$. Identifica-se o segmento sg formado por pt_k e pt_{k+1}. Se sg tiver alguma intersecção it_i, então adiciona-se pt_k e it_i a P^*. Caso contrário, adiciona-se pt_k e pt_{k+1} a P^* e identifica-se o próximo segmento formado por pt_{k+1} e pt_{k+2}.

Quando uma intersecção é adicionada a P^*, deve-se então identificar qual o próximo ponto ou intersecção a se seguir. Esse processo de contorno e escolha de pontos e intersecções é descrito em detalhes no algoritmo da Figura 3.18.

Requer polígonos P e P' a serem unidos
Requer Conjunto de intersecções IT
Requer Conjunto de intersecções IT_{sg} do segmento sg
  inicializar o novo polígono $P^* \leftarrow \emptyset$
  inicializar a lista de intersecções usadas $IT' \leftarrow \emptyset$
  inicializar $\underline{First_{Pt}}$ com um ponto qualquer de P ou P', mas que não seja interno a P ou P'
  inicializar $tipo \leftarrow endPoint$
  $\underline{EP_{Pt}} \triangleright \underline{First_{Pt}}$
  adicionar $\underline{First_{Pt}}$ ao polígono P^*
  repetir
    se $tipo = interseccao$ **então**
      identificar os 2 segmentos sg_0 e sg_1 participam da intersecção $\underline{EP_{Pt}}$
      se $interno(pontoEsquerda(sg_0), sg_1)$ **então**
        $sg_w = sg_1$
        $sg_l = sg_0$
      senão
        $sg_w = sg_0$
        $sg_l = sg_1$
      fim se
      se $|\{it | it \in \{IT_{sg} - IT'\} \wedge \neg interno(\underline{ponto}(it), sg_l)\}| = 0$ **então**
        $\underline{EP_{Pt}} \triangleright pontoDireita(sg_w)$
        $tipo \leftarrow endPoint$
      senão
        $it = \arg\min_{it \in \{IT sg - IT'\} | \neg interno(\underline{ponto}(it), sg_l)} (|\underline{ponto}(it) - \underline{EP_{Pt}}|)$
        $\underline{EP_{Pt}} = \underline{ponto}(it)$
        adicionar it a IT'
      fim se

    senão
      identificar o segmento sg formado pelos pontos $\underline{EP_{Pt}}$ e $(\underline{EP_{Pt}} + 1)$
      se $|\{it | it \in IT_{sg}\}| = 0$ **então**
        $\underline{EP_{Pt}} \triangleright \underline{EP_{Pt}} + 1$
      senão
        $it = \arg\min_{it \in \{IT_{sg} - IT'\}} |\underline{ponto}(it) - \underline{EP_{Pt}}|$
        $\underline{EP_{Pt}} = \underline{ponto}(it)$
        $tipo \leftarrow interseccao$
        adicionar it a IT'
      fim se
    fim se
    adicionar $\underline{EP_{Pt}}$ ao polígono P^*
  até $\underline{EP_{Pt}} \neq \underline{First_{Pt}}$
  return P^*

FIGURA 3.18 - Algoritmo *Unir polígonos*.

As seguintes representações foram convencionadas neste algoritmo:

- $\underline{EP_{Pt}}$ é uma referência para um ponto;
- $pontoEsquerda(sg)$ é uma função que retorna o primeiro ponto do segmento sg;
- $pontoDireita(sg)$ é uma função que retorna o segundo (e último) ponto do segmento sg;
- $\underline{ponto}(it)$ é uma função que retorna um ponteiro para o ponto da intersecção it;
- $interno(pt, sg)$ é uma função que retorna verdadeiro se o ponto pt é interno ao segmento sg.

Para calcular a função $interno(pt, sg)$ assume-se que a reta rt que passa pelo segmento sg divide um plano bidimensional em dois hemisférios: um hemisfério interno e outro externo. O hemisfério interno, denotado por HI_{sg}, é aquele cuja face do segmento sg está voltada para a parte interna do seu polígono. Entende-se então que um ponto pt é interno ao segmento sg se ele estiver no HI_{sg}. Esta idéia está representada na Figura 3.19. Cuidados adicionais devem ser tomados na implemen-

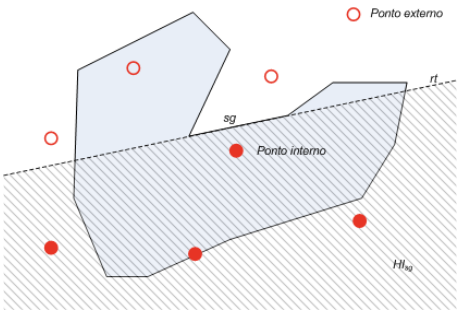


FIGURA 3.19 - Divisão de um plano bidimensional em hemisfério interno e externo, segundo um segmento sg. Diz-se que um ponto é interno ao segmento sg se ele está no hemisfério interno de sg.

tação do algoritmo de união de polígonos (Figura 3.18) quando se discretizar os pontos de intersecção, arredondando-os para números inteiros. Alguns pontos ao serem truncados serão representados por coordenadas iguais, podendo criar segmentos colineares.

3.2.2.3 Cálculo de Atributos

Alguns atributos calculados a partir da representação poligonal, diferenciam-se daqueles quando utiliza-se representação matricial. Encontrar a área de um polígono, por exemplo, pode se tornar um problema complexo quando não se tem um método de cálculo adequado. Dado que o polígono pode ser representado por um conjunto ordenado de pontos, Bourke (1988) formula o cálculo da área do polígono como sendo

$$P_A = \left| \sum_{k=0}^{nPt-1} \frac{(Pt_k^c - Pt_{k+1}^c)(Pt_k^r + Pt_{k+1}^r)}{2} \right|, \tag{3.11}$$

onde, nPt é o número de pontos do polígono.

Um método para cálculo dos centróides também é proposto por Bourke (1988),que define

$$P_{\bar{r}} = \frac{1}{6P_A} \sum_{k=0}^{nPt-1} (Pt_k^r + Pt_{k+1}^r)(Pt_k^c Pt_{k+1}^r - Pt_{k+1}^c Pt_k^r), \tag{3.12}$$

e

$$P_{\bar{c}} = \frac{1}{6P_A} \sum_{k=0}^{nPt-1} (Pt_k^c + Pt_{k+1}^c)(Pt_k^c Pt_{k+1}^r - Pt_{k+1}^c Pt_k^r). \tag{3.13}$$

Encontrar o comprimento do perímetro do polígono é mais simples, uma vez que facilmente se pode calcular o comprimento dos seus segmentos

$$P_{|P|} = \sum_{k=0}^{nPt-1} \sqrt{(Pt_k^c - Pt_{k+1}^c)^2 + (Pt_k^r - Pt_{k+1}^r)^2}. \tag{3.14}$$

Os pontos P_{tm}, P_{bm}, P_{lm} e P_{bm} para construção de um retângulo envolvente são encontrados diretamente:

$$P_{tm} = \min\{Pt^r | Pt \in P\}; \tag{3.15}$$

$$P_{bm} = \max\{Pt^r | Pt \in P\}; \tag{3.16}$$

$$P_{lm} = \min\{Pt^c | Pt \in P\}; e \tag{3.17}$$

$$P_{rm} = \max\{Pt^c | Pt \in P\}. \tag{3.18}$$

3.3 Cálculo da Distância de Objetos

O processo de captura da imagem de um objeto no mundo real por uma câmera, pode ser representado por um modelo simples de perspectiva da Figura 3.20-a. Neste modelo, a imagem de um objeto no mundo real com dimensão $X_2 - X_1$ é projetada no plano de imagem real de uma câmera, localizada a uma distância z do objeto. A imagem formada é invertida, com uma dimensão $v_2 - v_1$. A lente da câmera e o plano da imagem real estão separados a distância focal f.

Por triângulos semelhantes, sabe-se que o tamanho do objeto projetado na imagem é uma escala do objeto real, proporcionalmente a sua distância à câmera pela distância focal, logo obtém-se

$$\frac{v_2 - v_1}{f} = \frac{X_2 - X_1}{z}, \tag{3.19}$$

ou se forma genérica

$$\frac{v_i}{f} = \frac{X_i}{z}. \tag{3.20}$$

Conhecendo-se o tamanho do objeto no mundo real é possível então calcular sua distância até a câmera, através da imagem obtida

$$z = \frac{(X_2 - X_1)f}{(v_2 - v_1)}. \tag{3.21}$$

Técnicas de navegação baseada em marcos costumam utilizar informações conhecidas de objetos previamente posicionados no ambiente. Por exemplo, no trabalho de Coelho e Campos (COELHO; CAMPOS, 1998), objetos de topologia triangular guiam a navegação de um dirigível. Um sistema de visão computacional detecta os marcos visuais que tem dimensões conhecidas e calcula, através de inversão de perspectiva, a orientação e posição da câmera.

Todavia, em aplicações de navegação em ambientes não estruturados, dificilmente o tamanho dos objetos são conhecidos. Desta maneira, uma solução muito utilizada para se calcular a distância dos alvos de uma cena é utilizar visão estéreo, previamente descrita neste capítulo. Outras técnicas de processamentos digital de imagens, baseadas por exemplo em textura ou variações de tonalidade, podem fornecer informações 3D do objeto alvo, conforme citado por Pio (2002). O autor comenta também a técnica de mudança no ajuste do foco da câmera, para tal fim. Além disto, é possí-

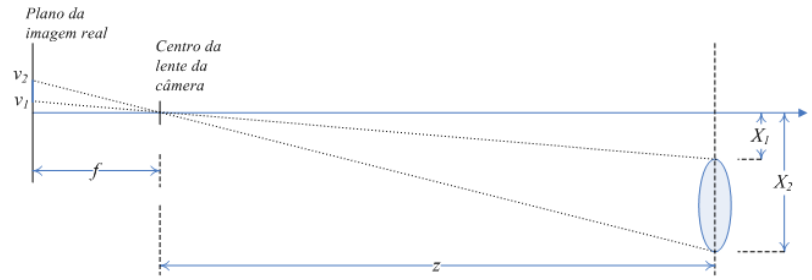
vel combinar o uso de dispositivos próprios para a medição de distância de objetos, com a captura de imagens do ambiente. Em Oliveira e Costa (2004) essa técnica de fusão de sensores é utilizada para calcular a distância de objetos identificados com uma câmera CCD, aplicado ao domínio de futebol de robôs.

Uma alternativa a essas técnicas é obter duas imagens da cena de posições diferentes e tentar inferir alguma informações sobre essas imagens. Por exemplo, na Figura 3.20-b duas imagens do mesmo objeto foram tiradas, sendo que a câmera foi transladada perpendicularmente em relação ao plano da imagem, em direção ao objeto. Tomando-se como referência a distância de deslocamento da câmera Δz e o seu modelo óptico é possível estimar a distância do objeto. Aplica-se os pontos v_1 e v_2' na Equação (3.20), obtendo-se

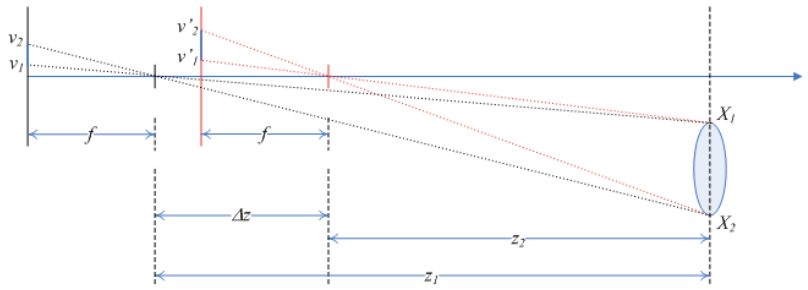
$$\frac{v_1}{X_1} = \frac{f}{z_1}, \tag{3.22}$$

e

$$\frac{v_1'}{X_1} = \frac{f}{z_2}. \tag{3.23}$$



(a) Modelo de perspectiva no qual a imagem do objeto é projetada no plano de imagem da câmera.



(b) Obtenção de duas imagens do objeto alvo através da translação da câmera.

FIGURA 3.20 - Modelo de perspectiva e cálculo da distância do objeto.

Com as Equações (3.22) e (3.23), observa-se a relação

$$v_1 z_1 = v_1' z_2. \tag{3.24}$$

Sabendo-se que $\Delta z = z_1 - z_2$ ou $z_1 = \Delta z + z_2$, o termo z_1 é substituído na Equação (3.24), obtendo-se

$$v_1(\Delta z + z_2) = v_1' z_2. \tag{3.25}$$

Finalmente, pode-se chegar a

$$z_2 = \frac{v_1 \Delta z}{v_1' - v_1} \tag{3.26}$$

para cálculo da distância do objeto.

Exemplificando o cálculo de distância pelo deslocamento da câmera, foram obtidas 5 imagens de um objeto esférico azul (Figura 3.21), deslocando-se a câmera 50 cm a frente após a captura de cada imagem. As imagens capturadas tem 320 x 240 *pixels*, das quais foram extraídas manualmente a largura do objeto, em unidades de *pixels*. A Tabela 3.1 apresenta os resultados da aplicação da Equação (3.26) tomando-se 2 imagens em seqüência. As quatro primeiras tuplas da Tabela 3.1 representam experimentos com a variação de 50 cm na distância da câmera em relação ao alvo. Comparando-se a distância obtida através de medição e cálculo, observa-se que o erro é maior quando o objeto está longe, como no caso do experimento #1. Essa anomalia era esperada uma vez que o tamanho do objeto na imagem é discretizado em um número inteiro de *pixels*, agravada pela pequena variação da distância da câmera. Considerando uma variação da distância da câmera com 100 cm, no experimento #5, o erro cai significativamente.

O diâmetro real do objeto esférico é de 25 cm, portanto para distâncias próximas o erro é considerado baixo, mas cresce linearmente com a introdução de erro na medição da distância de deslocamento da câmera Δz.

O sistema é sensível a ruído na leitura da largura do alvo, como pode ser visto nos resultados obtidos na Tabela 3.2, em que foi variada a largura do alvo em 1 *pixels* para mais e para menos. A coluna $\Delta erro$ representa a variação do erro em relação aos valores da Tabela 3.1, ou seja, valores positivos representam um aumento no erro, e negativos uma redução. Em alguns casos, a variação da largura melhorou o erro, aproximando o cálculo da distância com o valor medido, todavia, a Equação (3.26)

fica instável quando a diferença entre v_1' e v_1 é pequena.

Para amenizar o problema de discretização da largura do objeto alvo, pode-se experimentar algum método de extração de informação de sub-*pixel*, a fim de se obter uma medição contínua da largura do objeto na imagem. Por exemplo, técnicas de modelo de mistura visam identificar o percentual de classes presente em cada *pixel* (SHIMABUKURO; SMITH, 1991), o que pode viabilizar a medição contínua da largura do objeto.

Problemas encontrados em visão estéreo, tais como, correspondência e oclusão de objetos em imagens, continuarão existindo, e deverão ser tratados.

(a) Objeto a 345 cm, com 30 *pixels*.

(b) Objeto a 295 cm, com 34 *pixels*.

(c) Objeto a 245 cm, com 42 *pixels*.

(d) Objeto a 195 cm, com 52 *pixels*.

(e) Objeto a 145 cm, com 72 *pixels*.

FIGURA 3.21 - Imagens obtidas do objeto esférico azul com posicionamento da câmera em 5 distâncias diferentes. A largura em *pixels* do objeto é extraída de imagem.

TABELA 3.1 - Cálculo de distância de objeto utilizando a Equação (3.26).

ID	1ª Imagem	Nº *pixels* (v_1)	2ª Imagem	Nº *pixels* (v_1')	Δz (cm)	z_2 medido	z_2 calculado	Erro (cm)	Erro (%)
1	Figura 3.21-a	30	Figura 3.21-b	34	50	295	375,0	80,0	27
2	Figura 3.21-b	34	Figura 3.21-c	42	50	245	212,5	-32,5	13
3	Figura 3.21-c	42	Figura 3.21-d	52	50	195	210,0	15,0	8
4	Figura 3.21-d	52	Figura 3.21-e	72	50	145	130,0	-15,0	10
5	Figura 3.21-a	30	Figura 3.21-c	42	100	245	250,0	5,0	2,0
6	Figura 3.21-a	30	Figura 3.21-d	52	150	195	204,6	9,6	5,0
7	Figura 3.21-a	30	Figura 3.21-e	72	200	145	142,9	-2,1	1,5
8	Figura 3.21-b	34	Figura 3.21-d	52	100	195	188,9	-6,1	3,1
9	Figura 3.21-b	34	Figura 3.21-e	72	150	145	134,2	-10,8	7,4
10	Figura 3.21-c	42	Figura 3.21-e	72	100	145	140,0	-5,0	3,4

TABELA 3.2 - Cálculo de distância de objeto introduzindo 1 *pixel* de erro na medição de v_1'.

ID	$v_1' + 1$	z_2 calculado	Erro (cm)	Δ Erro (cm)	$v_1' - 1$	z_2 calculado	Erro (cm)	Δ Erro (cm)
1	35	300,0	5,0	-75,0	33	500,0	205,0	125,0
2	43	188,9	-56,1	23,6	41	242,9	-2,1	-30,4
3	53	190,9	-4,1	-11,0	51	233,3	38,3	23,3
4	73	123,8	-21,2	6,2	71	136,9	-8,2	-6,9
5	43	230,8	-14,2	9,2	41	272,7	27,7	22,7
6	53	195,6	0,6	-8,9	51	214,3	19,3	9,7
7	73	139,5	-5,5	3,3	71	146,3	1,3	-0,8
8	53	179,0	-16,0	10,0	51	200,0	5,0	-1,1
9	73	130,8	-14,2	3,4	71	137,8	-7,2	-3,6
10	73	135,5	-9,5	4,5	71	144,8	-0,2	-4,8

CAPÍTULO 4

ARQUITETURA DO AGENTE

Baseando-se no modelo de Agente de Aprendizagem, apresentado na introdução deste trabalho, e ilustrado pela Figura 4.1, o sistema do robô móvel é implementado. Todo o mecanismo de aprendizagem deste tipo de agente será suportado pelo Algoritmo *Q-learning*, de aprendizado por reforço, descrito na secção 2.3. Logo, o *elemento de aprendizado* é implementado na forma do algoritmo que atualiza os *Q-valores* e recebe a realimentação do ambiente através do *crítico*, por meio da *função de recompensas*. O *elemento de desempenho* escolhe suas ações seguindo uma dada *política* e estimulado por um mecanismo de exploração do *gerador de problemas*.

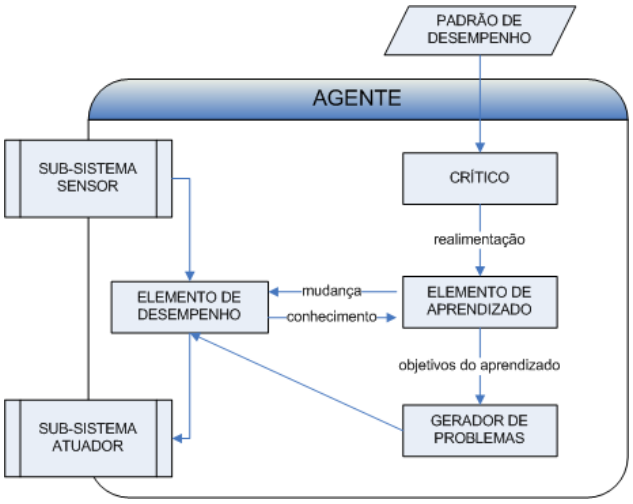


FIGURA 4.1 - Arquitetura do agente de aprendizagem.

O ambiente desconhecido no qual o agente interagirá, é composto de objetos imóveis, de tamanho e formas desconhecidos, mas que podem ser diferenciados do fundo de cena por suas características espectrais. O objetivo inicial deste agente que está sendo modelado é navegar livremente pelo terreno, desviando dos obstáculos. As ações do agente são tomadas em intervalos discretos, respeitando o ciclo de: *percepção* do ambiente, *escolha de ação*, *tomada de ação*, recebimento de *recompensa*, atualização da *base de conhecimento*. Novamente *percepção* e assim por diante...

Nesta modelagem, o agente só poderá tomar as seguintes ações:

- AC-FRENTE: andar para frente uma distância fixa e determinada;

- AC-ESQUERDA: virar à esquerda; ou

- AC-DIREITA: virar à direita.

Dado o objetivo de navegação, o sinal de reforço fornecido ao robô é simples, ao passo que para cada ação tomada a frente, o agente é premiado, e quando ocorrer alguma colisão, será punido. Esta idéia é formulada por

$$\mathfrak{r}(t) = \begin{cases} AG\text{-}PREMIO & \text{se } a_{t-1} = AC\text{-}FRENTE \\ AG\text{-}PUNICAO & \text{se o robô colidiu, ou seja, } s_t = bump \\ AG\text{-}DEFAULT & \text{caso contrário} \end{cases}, \tag{4.1}$$

onde, $AG\text{-}PREMIO$, $AG\text{-}PUNICAO$ e $AG\text{-}DEFAULT$ são constantes definidas para os experimentos, e que em geral $AG\text{-}PUNICAO < 0 > AG\text{-}PREMIO$ e $AG\text{-}DEFAULT = 0$.

O agente mapeia suas percepções em estados, através de informações obtidas pelo sub-sistema sensor, descrito na secção 4.1, que é baseado em operadores de visão computacional. Os estados representam situações de colisão e combinações da disposição dos objetos, na imagem capturada pelo sensor. A distância e a orientação de cada objeto são mapeados para 3 intervalos arbitrários de valores. O estado é armazenado então em uma estrutura de valores binários, conforme é ilustrado na Figura 4.2. Observa-se que para cada região de orientação de objetos, é possível armazenar somente uma distância de objeto. Portanto, assume-se que apenas a distância do objeto mais próximo será armazenada.

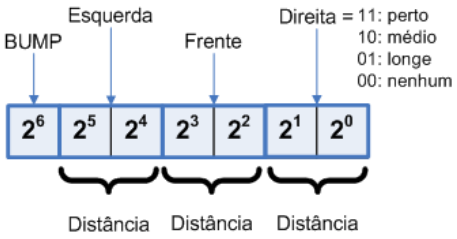


FIGURA 4.2 - Mapeamento dos estados do agente em função da orientação e distância dos objetos na cena, e detecção de colisões.

Ao total, o agente pode encontrar 65 estados, que é a soma do estado de colisão e a combinação das distâncias (ou ausência) de objetos em cada região de orientação $(1 + (3 + 1) \times (3 + 1) \times (3 + 1) = 65 = 2^6 + 1)$.

TABELA 4.1 - Relação dos parâmetros do agente.

Parâmetro	Descrição
α	Taxa de aprendizado do algoritmo *Q-learning*.
γ	Taxa de desconto do algoritmo *Q-learning*.
ϵ	ϵ-*greedy* probabilidade de seleção aleatória de ações.
AG-DefaultDst	Distância padrão a ser percorrida pela ação frente.
AG-PREMIO	Constante de premiação.
AG-PUNICAO	Constante de punição.
AG-DEFAULT	Constante de recompensa padrão.

A Tabela 4.1 sumariza os parâmetros do agente que deverão ser estipulados nos experimentos com o robô autônomo.

4.1 Sensor

O sub-sistema sensor é responsável por capturar imagens do ambiente e através da aplicação de operadores de visão computacional, fornecer ao agente informações a respeito dos objetos localizados a sua frente. A Figura 4.3 ilustra a ordenação dos módulos de processamento deste sub-sistema, que pode operar em tempo real, ou em modo simulado, ambos tomando como entrada parâmetros de configuração.

O processamento em modo simulado consiste em apenas um módulo que gera os dados sensoriais, baseando-se em variáveis de estado ou não. Inicialmente um estado é sorteado aleatoriamente, como por exemplo *um objeto longe a esquerda*. Em seguida, caso a ação do robô seja andar a frente, o estado anterior é considerado no sorteio do novo estado (no exemplo do estado anterior, o objeto a esquerda poderia se aproximar e novos objetos surgiriam ao centro ou a direita do robô). Por outro lado, se o robô tomar uma ação de virar a esquerda ou direita, a memória do estado anterior é desconsiderada no sorteio do estado atual. Em versões futuras do sensor, métodos de simulação mais sofisticados devem ser utilizados, através da construção de modelos de ambientes virtuais em 3D, gerando uma imagem de captura sintética, por técnicas de computação gráfica.

Já o modo de processamento em tempo real é composto por uma série de módulos responsáveis por extrair informações do ambiente. O primeiro passo visa acionar a câmera CCD e capturar uma imagem RGB da cena (detalhes do processo de captura serão descritos na secção 4.3). Os *pixels* da imagem então são classificados, identificando-se os elementos que pertencem a objetos e ao fundo da cena, para que

a segmentação possa ser conduzida. Para a classificação será utilizada uma RNA, descrita na secção 4.1.1.

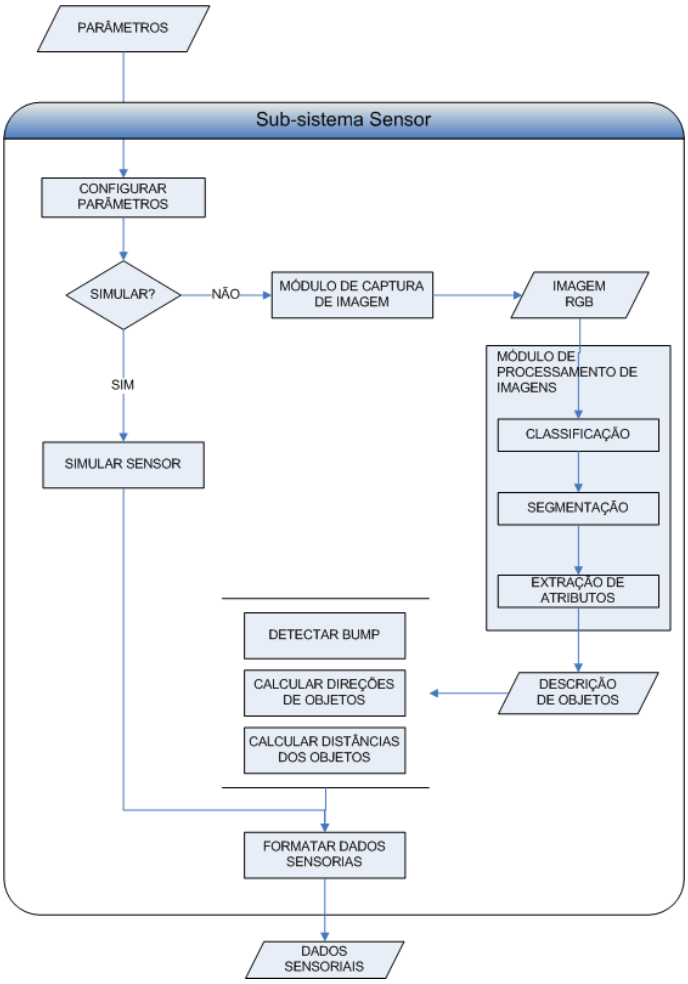
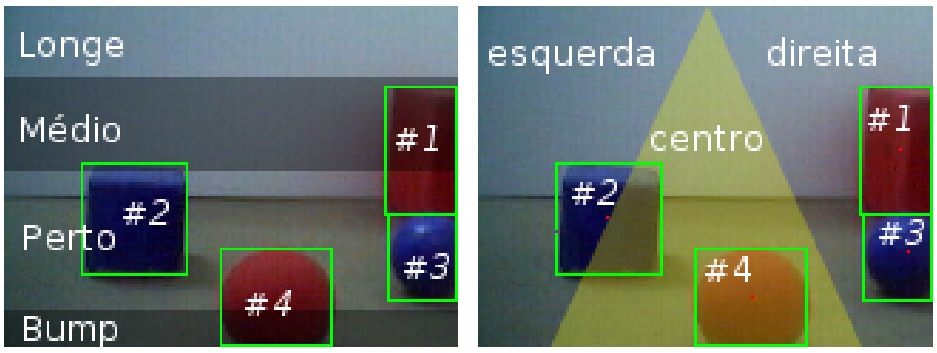


FIGURA 4.3 - Arquitetura do sub-sistema sensor do agente.

O processo de segmentação visa identificar regiões da imagem que representam os objetos alvos e viabilizar a extração de atributos destes objetos, tais como, área, largura, perímetro da região, entre outros. Um estudo comparativo entre os métodos matricial e vetorial para detecção de objetos, previamente descritos na seção 3.2, é realizado na secção 4.1.2 para que possa ser escolhido o mais adequado.

Os dados sensoriais, a serem formatados para consulta do agente, trazem informações a respeito da posição e orientação dos objetos na imagem, e são obtidos através dos atributos dos objetos identificados.

Assumindo-se que o ângulo de inclinação da câmera em relação ao solo é arbitrariamente calibrado, é possível inferir a distância aproximada dos objetos na cena até o robô. Este método é preferido nesta modelagem de sensor, em substituição a outros métodos apresentados para cálculo de distância de objetos, pois o sub-sistema não está equipado com um módulo de processamento de correspondência entre imagens. Assim, a imagem será segmentada em 4 faixas horizontais, sendo que a mais superior representa a região mais distante do robô, e a mais inferior, a região mais próxima. Essas regiões serão rotuladas pelos nomes lingüísticos *longe*, *médio*, *perto* e *colisão (bump)*, ilustradas na Figura 4.4-a. A distância dos objetos é definida pela localização dos seus *pixels* mais inferiores na imagem. Pela representação da Figura 4.4-a, o objeto #4 está em região de proximidade à colisão, e os demais objetos a uma distância *perto*.



(a) Faixas horizontais que representam regiões de distância de alvos até o robô. Os objetos #1 #2 e #3 estão *perto* e o objeto #4 está próximo a colisão.

(b) Regiões *esquerda*, *centro* e *direita* que representam a orientação dos objetos. Os centróides são representados pelos pontos vermelhos.

FIGURA 4.4 - Segmentação da imagem em regiões para estimação de distância e orientação dos alvos.

De forma semelhante, a orientação dos objetos é estimada pela localização do centróide dos objetos em relação a uma determinada região segmentada na imagem. Na Figura 4.4-b, os centróides são representados pelos pontos vermelhos nos objetos, que são rotulados por 3 orientações, *esquerda* (objeto #2), *centro* (objeto #4) e *direita* (objeto #1 e #3). As regiões de orientação são divididas por dois segmentos, definidos da seguinte maneira: $r = 0, c = C/2$ para os pontos extremos da parte superior da imagem; e $r = R - 1, c(SE\text{-}cOr, C)$ para os pontos extremos na parte inferior da imagem, sendo que $c = SE\text{-}cOr$ para o segmento da esquerda e $c = C - SE\text{-}cOr$, para o da direita. O parâmetro *SE-cOr* é definido pelo usuário e deve ser inferior ao número de colunas C da imagem.

4.1.1 Classificador

A classificação das imagens capturadas pelo sensor é efetuada por uma RNA do tipo *perceptron* em múltiplas camadas (MLP), previamente treinada por processo supervisionado. Objetiva-se diferenciar 3 tipos de objetos alvos, identificados por suas características espectrais, nas faixas do visível vermelho, azul e amarelo. Além disto, a RNA deve reconhecer a parede e o piso do ambiente, e objetos pretos, que poderão ser utilizados como um alvo adicional nos experimentos.

A partir de imagens obtidas do cenário de experimentos de navegação, foram coletadas 1138 amostras, sendo que 910 foram selecionadas para treinamento, restando 228 (20%) para teste de generalização. A Figura 4.5 demonstra a distribuição das amostras coletadas, nas 6 classes que a RNA deve identificar.

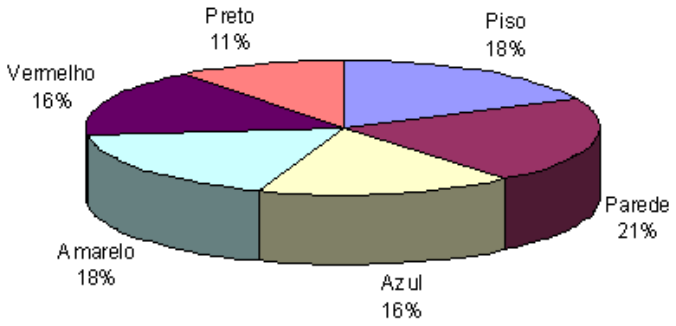


FIGURA 4.5 - Distribuição em classes das 1138 amostras coletadas para treinamento e teste de generalização da RNA.

Para definir a RNA que compõe o sensor, esta secção efetuará testes com configurações variadas de RNA. Um grupo dessas redes classificará cada *pixel* da imagem utilizando apenas as componentes RGB do pixel alvo, enquanto um outro grupo utilizará o contexto daquele *pixel*, ou seja, as componentes RGB do *pixel* mais as componentes RGB dos *pixels* de sua vizinhança 8.

A divisão das amostras em conjuntos de treinamento e teste foi realizada por escolha aleatória, resultando nas distribuições por classes apresentadas pela Tabela 4.2.

Ao total, foram realizados 8 experimentos de treinamento, variando a configuração da RNA, o número de entradas e classes, identificados e especificados pela Tabela 4.3. O uso ou não da V_8 do *pixel* a ser classificado, determina o número de entradas da rede, que pode ser 27 ou 3 unidades. Em metade dos testes, as classes *Parede* e

TABELA 4.2 - Distribuição de amostras por classes, em grupos de treinamento e teste de generalização, nos experimentos de classificação de *pixel* com e sem contexto.

Classe	Experimentos sem V_8				Experimentos com V_8			
	Aprendizado		Generalização		Aprendizado		Generalização	
	N°	%	N°	%	N°	%	N°	%
Piso	164	18	42	18	154	17	52	23
Parede	189	21	50	22	199	22	40	18
Azul	151	17	32	14	143	16	40	18
Amarelo	158	17	44	19	164	18	38	17
Vermelho	154	17	33	14	154	17	33	14
Preto	94	10	27	12	96	11	25	11
Total	910	100	228	100	910	100	228	100

Piso foram unidas por uma nova classe denominada *Fundo*, resultando num total de 5 classes, enquanto que na outra metade dos teste, o número total de classes continuou em 6. Sempre foram utilizadas 2 camadas ocultas na arquitetura das redes MLP, sendo que a representação $n_1 \rightarrow n_2$ diferencia o número de unidades neuronais, onde n_1 é o número de unidades da primeira cada oculta e n_2 o número de unidades da segunda. Optou-se por arquiteturas MLP de duas camadas internas, pois testes preliminares indicaram um melhor refinamento no processo de extração de atributos pela rede, onde a primeira camada interna agrupa informações dos dados de entrada e a segunda efetua a classificação.

O treinamento das redes foi realizado pelo Algoritmo *Backpropagation* (ver secção 2.1.1.1), através de um simulador construído em linguagem C++ para este trabalho. Para as unidades neuronais, adotou-se a função de ativação logística sigmóide (Equação (2.5)) com $\sigma = 1$, com taxa de aprendizado de $0,1$.

A saída da rede é binarizada para 0 e 1 segundo um limiar (*score threshold*) de $0,4$; ou seja, valores no intervalo $[0,6;1] \mapsto 1$, $[0;0,4] \mapsto 0$ e no intervalo $(0,4;0,6)$ a saída produzida é rejeitada.

A Tabela 4.4 sumariza os resultados de treinamento das RNAs, através do menor valor da Raiz do Erro Quadrático Médio (RMSE, do inglês *Root Mean Squared Error*) obtido e o percentual de acertos de classificação. De uma forma geral, todas configurações de RNA alcançaram um treinamento satisfatório. Porém, as taxas de acertos nos testes de generalização foram ligeiramente superiores nos experimentos que consideraram a V_8 dos *pixels*, apesar do número de épocas para se alcançar o menor RMSE ter sido maior. A união dos conjuntos de amostras das classes *Piso* e

TABELA 4.3 - Identificação dos experimentos realizados com os classificadores neurais.

Experimento	Contexto do *pixel*	Nº épocas	Nº classes	Unidades camada oculta
v1c5a	não	10.000	5	$10 \rightarrow 5$
v1c5b	não	10.000	5	$30 \rightarrow 15$
v1c6a	não	10.000	6	$10 \rightarrow 5$
v1c6b	não	10.000	6	$30 \rightarrow 15$
v8c5a	sim	20.000	5	$30 \rightarrow 15$
v8c5b	sim	20.000	5	$40 \rightarrow 20$
v8c6a	sim	20.000	6	$30 \rightarrow 15$
v8c6b	sim	20.000	6	$40 \rightarrow 20$

TABELA 4.4 - Resultados do treinamento das redes neurais utilizadas na classificação de imagens.

	Aprendizagem		Generalização		
Experimento	RMSE ($\times 10^{-2}$)	acertos %	RMSE ($\times 10^{-2}$)	acertos %	Melhor época
v1c5a	0,31	99,45	1,47	97,81	423
v1c5b	0,30	99,45	1,46	97,81	352
v1c6a	0,42	99,34	2,45	96,93	672
v1c6b	0,37	99,23	2,27	96,93	645
v8c5a	0,22	99,67	0,24	99,56	1363
v8c5b	0,17	99,67	0,23	99,56	3849
v8c6a	0,44	99,23	0,54	99,12	5393
v8c6b	0,44	99,23	0,64	98,68	6060

Parede facilitou o processo de classificação.

A Figura 4.6 traz as curvas do RMSE em função do número de épocas de treinamento das RNAs. Nas 1000 primeiras épocas, observa-se que o processo de aprendizado foi semelhante para os experimentos que não utilizaram contexto de *pixel*. Para as RNAs que consideraram a V_8 dos *pixels*, a queda do erro de aprendizagem (e conseqüente inversão do RMSE de aprendizagem $\times$ generalização), ocorreu depois da época 2000 (Figuras 4.6-c e Figuras 4.6-d).

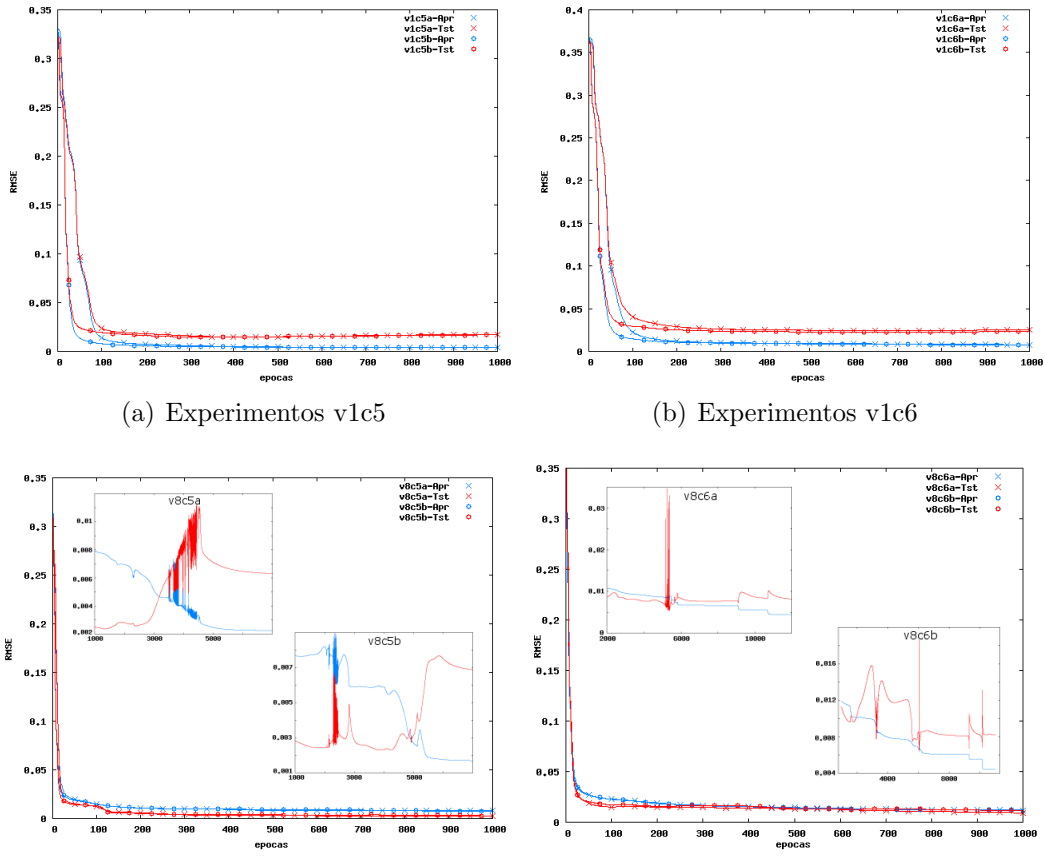
Para melhor avaliação quantitativa dos resultados de treinamento, matrizes de confusão (ou matrizes de erro) são construídas com as respostas das redes para o conjunto de dados de generalização, nas quais estão apresentadas as distribuições de elementos classificados correta e erroneamente. Esta matriz auxilia a identificação de erros específicos de classificação e pode ser tomada como base para se efetuar análises multivariadas (MOREIRA, 2001). Com base na matriz, pode-se efetuar a estatística

Kappa

$$\kappa = \frac{N \sum_{i=1}^{r} x_{ii} - \sum_{i=1}^{r} x_{i+} x_{+i}}{N^2 - \sum_{i=1}^{r} x_{i+} x_{+i}}, \tag{4.2}$$

onde, r é o número de linhas ou colunas da matriz de confusão;
x_{ii} é o número de observações dos elementos da diagonal da matriz;
N é o número total de elementos;
$x_{i+} = \sum_j x_{ij}$ é a soma dos valores da linha i; e
$x_{+i} = \sum_j x_{ji}$ é a soma dos valores da coluna i.

O cálculo do coeficiente κ traz a vantagem de incluir todos os elementos da matriz de erro e não somente a contagem simples dos acertos ou erros (MOREIRA, 2001). Em geral, obtem-se valores entre 0 (classificação péssima) e 1 (classificação excelente), mas o coeficiente pode também assumir valores negativos.



(a) Experimentos v1c5

(b) Experimentos v1c6

(c) Experimentos v8c5

(d) Experimentos v8c6

FIGURA 4.6 - Curvas de erro de aprendizado e generalização em função da época de treinamento.

Por motivos de otimização computacional, o coeficiente κ é calculado como sendo

$$\kappa = \frac{\hat{\kappa}_1 - \hat{\kappa}_2}{1 - \hat{\kappa}_2}, \tag{4.3}$$

onde,

$$\hat{\kappa}_1 = \frac{1}{N} \sum_{i=1}^{r} x_{ii}, e \tag{4.4}$$

$$\hat{\kappa}_2 = \frac{1}{N^2} \sum_{i=1}^{r} x_{i+} x_{+i}. \tag{4.5}$$

As matrizes de confusão e o índice Kappa obtidos nos experimentos são apresentados nas Tabelas 4.5, 4.6, 4.7, 4.8, 4.9 e 4.10, nas quais as linhas representam as classes alvo, e as colunas o resultado da classificação. Não houve casos de classificação rejeitada. Uma das maiores confusões de classificação ocorreu nos experimentos $v1c6$, onde a classe *Parede* foi classificada como *Piso*. Esse erro foi minimizado com a classificação com contexto de *pixel* ou pela união dessas classes na classe *Fundo*. Nos experimentos $v1c5$ as classes *Fundo* e *Amarelo*, e *Azul* e *Preto*, confundem-se entre si, o que não ocorre com a introdução de contexto de *pixel*. Os índices Kappa confirmaram o bom resultado do treinamento.

TABELA 4.5 - Tabela de confusão dos experimentos v1c5a e v1c5b, ambos com $\kappa = 0,971$.

Classe	Fundo	Azul	Amarelo	Vermelho	Preto	*Soma*
Fundo	**91**	0	1	0	0	**92**
Azul	0	**31**	0	0	1	**32**
Amarelo	1	0	**43**	0	0	**44**
Vermelho	0	0	0	**32**	1	**33**
Preto	0	0	0	1	**26**	**27**
Soma	**92**	**31**	**44**	**33**	**28**	***228***

TABELA 4.6 - Tabela de confusão dos experimentos v8c5a e v8c5b, ambos com $\kappa = 0,994$.

Classe	Fundo	Azul	Amarelo	Vermelho	Preto	*Soma*
Fundo	**92**	0	0	0	0	**92**
Azul	0	**40**	0	0	0	**40**
Amarelo	0	0	**38**	0	0	**38**
Vermelho	0	0	0	**33**	0	**33**
Preto	0	1	0	0	**24**	**25**
Soma	**92**	**41**	**38**	**33**	**24**	***228***

TABELA 4.7 - Tabela de confusão do experimento v1c6a, resultando $\kappa = 0,963$.

Classe	Piso	Parede	Azul	Amarelo	Vermelho	Preto	*Soma*
Piso	**41**	0	0	1	0	0	**42**
Parede	2	**48**	0	0	0	0	**50**
Azul	1	0	**31**	0	0	0	**32**
Amarelo	1	0	0	**43**	0	0	**44**
Vermelho	0	0	0	0	**32**	1	**33**
Preto	0	0	0	0	1	**26**	**27**
Soma	**45**	**48**	**31**	**44**	**33**	**27**	***228***

TABELA 4.8 - Tabela de confusão do experimento v1c6b, resultando $\kappa = 0,963$.

Classe	Piso	Parede	Azul	Amarelo	Vermelho	Preto	*Soma*
Piso	**41**	0	0	1	0	0	**42**
Parede	2	**48**	0	0	0	0	**50**
Azul	0	0	**32**	0	0	0	**32**
Amarelo	1	0	0	**42**	0	1	**44**
Vermelho	0	0	0	0	**32**	1	**33**
Preto	0	0	0	0	1	**26**	**27**
Soma	**44**	**48**	**32**	**43**	**33**	**28**	***228***

TABELA 4.9 - Tabela de confusão do experimento v8c6a, resultando $\kappa = 0,989$.

Classe	Piso	Parede	Azul	Amarelo	Vermelho	Preto	*Soma*
Piso	**51**	0	0	0	0	1	**52**
Parede	0	**40**	0	0	0	0	**40**
Azul	0	0	**40**	0	0	0	**40**
Amarelo	0	0	0	**38**	0	0	**38**
Vermelho	0	0	0	0	**33**	0	**33**
Preto	0	0	1	0	0	**24**	**25**
Soma	**51**	**40**	**41**	**38**	**33**	**25**	***228***

TABELA 4.10 - Tabela de confusão do experimento v8c6b, resultando $\kappa = 0,984$.

Classe	Piso	Parede	Azul	Amarelo	Vermelho	Preto	*Soma*
Piso	**51**	0	0	0	0	1	**52**
Parede	0	**40**	0	0	0	0	**40**
Azul	0	0	**40**	0	0	0	**40**
Amarelo	0	0	0	**38**	0	0	**38**
Vermelho	0	0	0	0	**33**	0	**33**
Preto	0	0	1	1	0	**23**	**25**
Soma	**51**	**40**	**41**	**39**	**33**	**24**	***228***

Uma vez que os resultados de treinamento foram satisfatórios e com pouca variação do índice Kappa, optou-se em adotar a RNA do experimento $v1c5a$ para integrar o sub-sistema sensor, minimizando assim o custo computacional de ativação do classificador. A Figura 4.7 ilustra o desempenho dessa rede, através da classificação de duas imagens capturadas do ambiente. A coluna da esquerda exibe as imagens RGB da cena, enquanto que a coluna da direita as respectivas imagens classificadas, adotando-se a cor branca para representar a classe *Fundo*. Apesar de alguns ruídos na classificação, principalmente nas regiões extremas da imagem e sombras no piso, percebe-se claramente os objetos alvos.

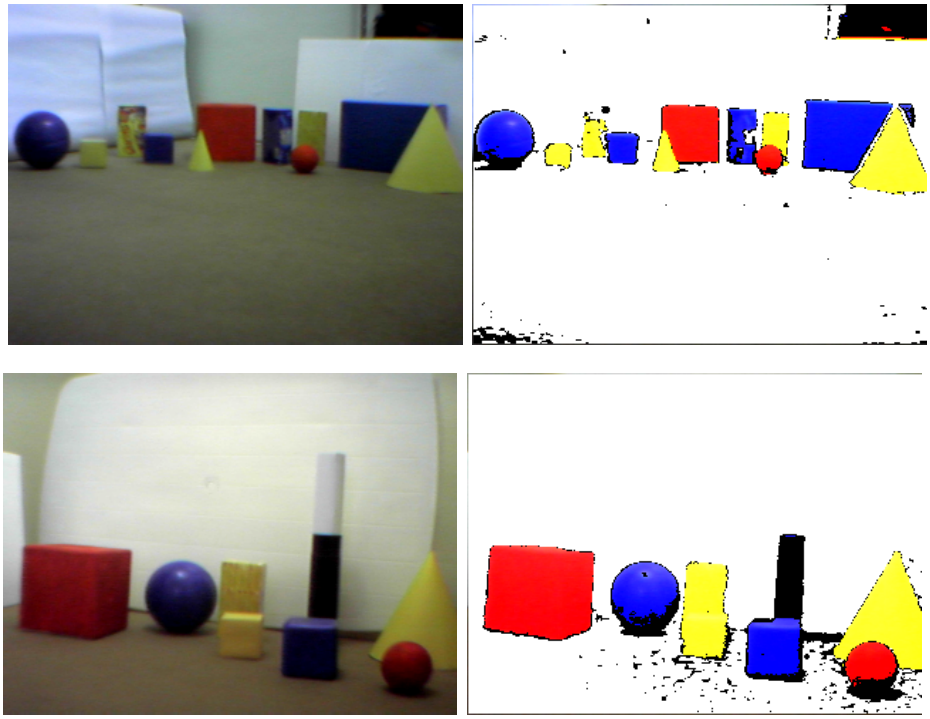


FIGURA 4.7 - Classificação de imagens utilizando a rede neural do experimento $v1c5a$. As imagens brutas estão na coluna da esquerda, e suas respectivas classes identificadas nas imagens da coluna da direita.

4.1.2 Escolha do Segmentador

Com a imagem produzida pelo classificador, a etapa de segmentação visa identificar os objetos na cena e extrair seus atributos. Para tanto, um dos algoritmos de detecção de objetos, apresentados na secção 3.2, deverá ser empregado pelo sensor.

As imagens da primeira coluna da Figura 4.8 apresentam exemplos de imagens do ambiente do agente, e os respectivos objetos identificados pelo método matricial e

vetorial. Os objetos detectados pelo método matricial estão destacados pelo seu retângulo envolvente, nas imagens da coluna do meio, e estão limitados a objetos com área maior que 20 *pixels*. A última coluna de imagens destaca os polígonos encontrados para cada objeto identificado pelo método vetorial. Nas imagens classificadas, o centróide calculado dos objetos é identificado por um ponto vermelho.

Os experimentos da primeira linha da Figura 4.8 mostram que o método matricial detectou todos os objetos presentes na cena, enquanto o método vetorial não pôde detectar dois alvos. Através dos retângulos envolventes encontrados pelo detector matricial, percebe-se que os objetos amarelos foram parcialmente identificados. Este fato deve-se a saída ruidosa do classificador neural, que encontrou dificuldades em reconhecer corretamente os alvos da classe *Amarelo*, devido a modificações na iluminação do ambiente. Para amenizar este problema, seria necessário treinar novamente a RNA, considerando um novo conjunto de amostras dos alvos, mediante diferentes condições de iluminação do ambiente. Outra proposta para aumentar a flexibilidade do classificador é submeter a imagem capturada a um processamento de realce, tais como um filtro para normalizar ou contrastar o histograma. Contudo, dentro de um limite de variação da iluminação, o classificador neural cumpriu com sua tarefa de detectar os objetos da cena.

A razão pela qual o segmentador, baseado em método vetorial, não ter sido capaz de detectar 2 alvos, consiste no tamanho da grade de controle utilizada no algoritmo de detecção de polígonos (Figura 3.10), previamente descrito. Alvos na imagem, que são pequenos em relação ao intervalo de amostragem da grade, podem ser ignorados. A Figura 4.9 exemplifica o uso de uma grade de controle que amostra a imagem a cada 10 *pixels*, representados pelos pontos vermelhos (classes de fundo da imagem) e pontos verdes (classes que pertencem a objetos alvo).

A segunda, terceira e quarta linha de testes, na Figura 4.8, ilustram a detecção correta de todos objetos na cena, incluindo objetos que se encontram nos limites da imagem capturada.

A quinta e a última linha de imagens, mostra o problema de detecção de objetos amarelos, já discutido anteriormente, mas que foi tolerado pelo método matricial. Todavia, alguns objetos espúrios foram gerados, o que sugere a necessidade da construção de um filtro mais eficiente, uma vez que nos testes da Figura 4.8, apenas um limiar de área foi utilizado. Um atributo que minimiza esse problema é a razão entre a área do objeto O_A e seu maior eixo O_{Me}, chamada neste trabalho de *espalhamento*.

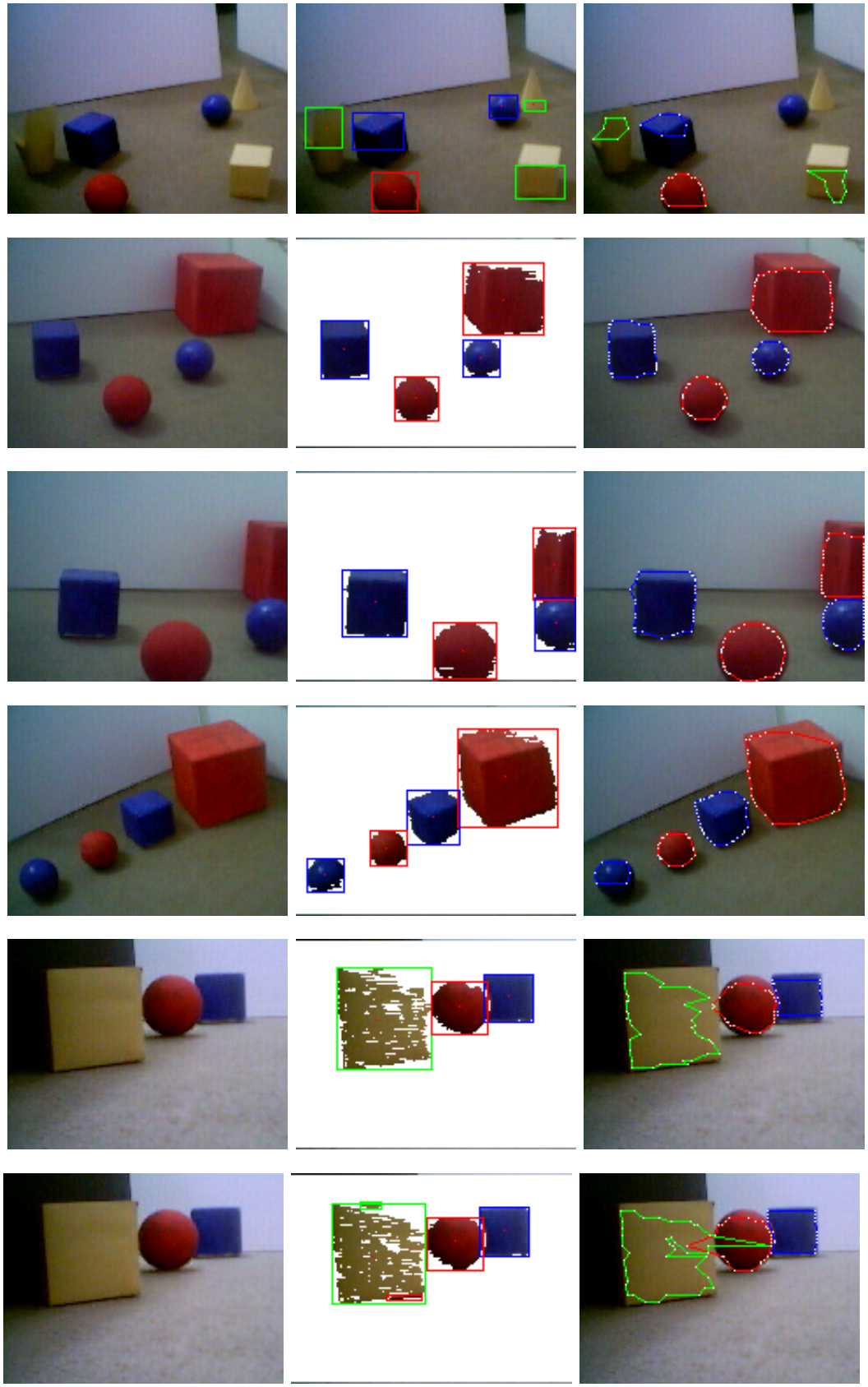


FIGURA 4.8 - Processo de detecção de objetos. A coluna da esquerda representa a imagem RGB obtida da cena, enquanto que as colunas do meio e da direita, destacam os objetos identificados pelo método matricial e vetorial, respectivamente.

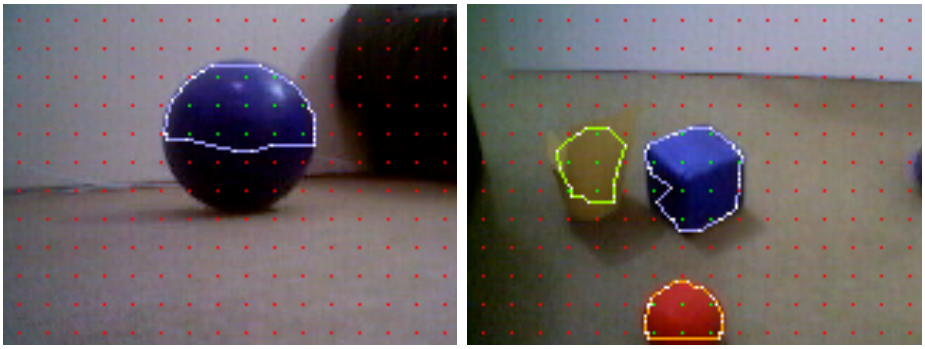


FIGURA 4.9 - Grade de controle, representada pelos pontos nas imagens, utilizada pelo algoritmo de detecção de objetos pelo método vetorial. Os pontos vermelhos representam classes de fundo da imagem, e os pontos verdes classes de objetos alvo.

Como é demonstrado na Tabela 4.11, os valores calculados para os objetos espúrios são bem inferiores quando comparados aos objetos alvos, logo, um novo limiar pode ser estipulado.

TABELA 4.11 - Cálculo de atributos dos objetos detectados na última imagem da Figura 4.8, com o propósito de eliminação de objetos espúrios.

Objeto	Classe	Espúrio	O_A	O_{Me}	O_A/O_{Me}
1	Azul	não	676	28	24,14
2	Vermelho	não	709	32	22,16
3	Amarelo	não	2043	58	35,22
4	Vermelho	sim	27	20	1,35
5	Amarelo	sim	22	12	1,83

As duas últimas imagens da Figura 4.8 mostraram ainda a dificuldade do método vetorial em detectar objetos com classificação ruidosa. O polígono gerado para o objeto da classe *Amarelo* ficou visualmente distorcido e, em alguns casos, dois polígonos de objetos diferentes se interseccionam, gerando uma representação indesejada.

A fim de se comparar o custo computacional dos métodos de detecção de objetos, foram realizados testes adicionais com as imagens da Figura 4.10-a, objetivando-se a identificação do alvo representado pela esfera azul. Nesses testes de detecção, utilizou-se um computador PC com processador de 700 Mhz, com 128 MB de memória RAM, em sistema operacional GNU/Linux distribuição Kurumin 3.0.

A classificação da imagem foi realizada por uma RNA, especialmente treinada para esta análise, com 3 unidades de entrada (referentes aos valores RBG do *pixel*) e

(a) Imagem capturadas da cena.

(b) Objeto detectado com método matricial.

(c) Objeto detectado com método vetorial.

FIGURA 4.10 - Testes com os métodos de detecção de objetos para avaliação do tempo de processamento.

uma camada oculta com 10 unidades. A camada de saída é constituída de apenas 1 neurônio, que classificará se o *pixel* pertence ao objeto alvo azul ou não. O tempo médio de ativação da RNA é na ordem de $0,4ms$, logo, para se classificar uma imagem de 160 x 120 *pixels*, leva-se aproximadamente $768ms$.

O método de detecção vetorial tem grande vantagem sobre o método matricial, pois não solicita à rede neural a classificação de toda a imagem, mas apenas uma amostra de pontos da imagem. Neste experimento, a amostragem da grade de controle foi efetuada variando-se as linhas e colunas da imagem num intervalo fixo de 10

pixels, além disto, vale ressaltar que outros *pixels* requerem classificação, conforme determinado no algoritmo da Figura 3.16, descrito na Secção 3.2.2.

A Figura 4.10-b contém os resultados de identificação de regiões que representam o objeto alvo. Nota-se que o alvo não foi detectado como uma região convexa, pois a classificação sofreu interferência de brilho especular. Nota-se que nesta análise, os *pixels* ruidosos não foram filtrados. A Figura 4.10-c apresenta o mesmo objeto da Figura 4.10-a aproximado por um polígono.

A Tabela 4.12 sumariza os tempos de processamento das imagens da Figura 4.10-a, com os métodos matricial e vetorial. Observa-se que o tempo de classificação pelo método vetorial é muito inferior em relação ao matricial, como era esperado para alvos convexos (ou côncavos de baixa complexidade). A etapa de processamento responsável por identificar o objeto no método vetorial foi maior que a etapa de rotulação no método matricial, para a segunda imagem. Isto sugere que a região côncava do alvo leva a detecção de vários polígonos independentes, penalizando o processo de união de polígonos (tempos representados entre parênteses).

Como pôde ser visto, o tempo computacional de detecção de objetos pelo método vetorial encoraja o seu uso. Contudo, optou-se em utilizar o segmentador baseado no método matricial nesta versão do sensor, pois este está mais tolerante aos ruídos do classificador. Além disto, o uso do detector vetorial requer ainda um ajuste do tamanho da grade de controle. Ainda assim, vale ressaltar que o método vetorial é promissor e deverá ser aprimorado em trabalhos futuros.

TABELA 4.12 - Tempo de processamento em milisegundos (ms) para detecção de objeto pelo método Matricial e Vetorial.

Etapas	**Matricial**		**Vetorial**	
	Fig. 4.10-a	Fig. 4.10-b	Fig. 4.10-a	Fig. 4.10-b
Classificação	733,0	934,0	6,0	7,9
Rotulação / Vetorização (União)	64,2	67,9	24,9 (8,5)	90,8 (30,7)
Total	797,0	1001,0	30,9	98,7

4.1.3 Parâmetros do Sensor

A Tabela 4.13 sumariza os parâmetros do sensor que deverão ser estipulados nos experimentos com o robô autônomo.

TABELA 4.13 - Relação dos parâmetros do sub-sistema sensor.

Parâmetro	Descrição
SE-Simular	Indicador de sensor simulado ou de tempo real.
SE-areaTH	Limiar de área para filtragem de objetos espúrios.
SE-SpTH	Limiar de espalhamento para filtragem de objetos espúrios.
SE-DstLonge	Limites da faixa horizontal na imagem que caracteriza a distância *longe*.
SE-DstMedio	Limites da faixa horizontal na imagem que caracteriza a distância *médio*.
SE-DstPerto	Limites da faixa horizontal na imagem que caracteriza a distância *perto*.
SE-DstBump	Limites da faixa horizontal na imagem que caracteriza distância de colisão.
SE-cOr	Coordenada de uma das colunas dos segmentos que definem as regiões de orientação dos objetos na imagem.

4.2 Atuador

O sub-sistema atuador tem como propósito isolar os detalhes de funcionamento do *hardware* dos motores do robô. Interagindo com ele, o agente pode tomar suas ações enviando comandos em alto nível, independente do tipo de *hardware* instalado. Com essa intenção, o atuador é composto de 3 módulos, organizados de forma hierárquica, conforme ilustrado na Figura 4.11.

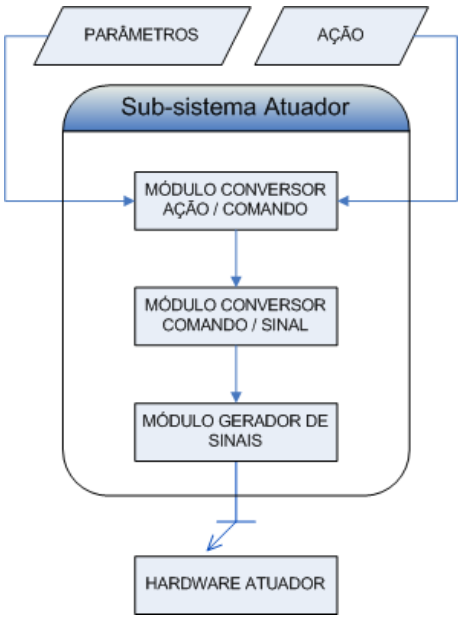


FIGURA 4.11 - Sub-sistema atuador do agente.

Através da modularização, as ações determinadas pelo agente são traduzidas gradativamente em sinais de *hardware*. Recebendo como entrada a ação a ser executada e os parâmetros do atuador, o módulo *"Conversor Ação / Comando"* produz uma seqüência de comandos básicos. Cada ação pode gerar um ou mais comandos, que serão convertidos em sinais pelo módulo *"Conversor Comando / sinal"*. Por fim, o módulo *"gerador de sinais"* formata em seqüência os sinais para envio ao *hardware*.

O tipo do robô móvel, utilizado neste trabalho, tem a configuração de um carrinho de controle remoto, cujo sistema de atuação permite controlar a orientação das rodas dianteiras (esquerda ou direita) e a direção do movimento das traseiras (para frente ou para trás). Isto determina de forma limitada o conjunto de sinais disponíveis ao sub-sistema atuador.

As Tabelas 4.14, 4.15 e 4.16 apresentam as listas de símbolos de ações, comandos e sinais utilizados nesta modelagem. As ações compreendem as mesmas modeladas para o agente, com a adição da ação *AC-PARAR* para interrupção de experimentos. Os comandos pretendem ser genéricos o suficiente para comportar mudanças nas especificações de ações e *hardware*, no futuro. Fica claro porém, que devido a configuração do robô utilizado, não será possível efetuar manobras de giro em torno do próprio eixo do carrinho. Os sinais compreendem as 8 combinações de acionamento dos motores do robô e 2 sinais de controle.

A Tabela 4.17 mostra a relação de produção de sinais a partir de ações. Nas ações em que deseja-se andar uma determinada distância a frente, esse valor deve ser convertido em tempo de acionamento do atuador, uma vez que não se tem a disposição um odômetro. Como pode ser visto, para completar a ação de virar à esquerda, o atuador executa uma seqüência de 15 sinais. Essa manobra não é exata, e o resultado visual da implementação é apresentado pela série de imagens da Figura 4.12.

TABELA 4.14 - Símbolos de ações disponíveis na implementação do atuador.

Conjunto de Ações	
Símbolo	Descrição
AC-PARAR	Desligar o atuador.
AC-FRENTE	Andar para frente a distância padrão.
AC-ESQUERDA	Virar à esquerda.
AC-ESQUERDA	Virar à direita.

TABELA 4.15 - Símbolos de comandos disponíveis na implementação do atuador.

Conjunto de Comandos		
Símbolo	Parâmetro	Descrição
CM-ABORT		Desligar o atuador.
CM-FRENTE	distância em cm	Andar para frente a distância especificada.
CM-RE	distância em cm	Andar para trás a distância especificada.
CM-FRENTE-ESQUERDA	ângulo em graus e raio em cm	Traçar um arco com determinado ângulo de abertura, descrito pela circunferência de raio especificado, andando para frente no sentido anti-horário.
CM-FRENTE-DIRENTA	ângulo em graus e raio em cm	Traçar um arco com determinado ângulo de abertura, descrito pela circunferência de raio especificado, andando para frente no sentido horário.
CM-RE-ESQUERDA	ângulo em graus e raio em cm	Traçar um arco com determinado ângulo de abertura, descrito pela circunferência de raio especificado, andando para trás no sentido horário.
CM-RE-DIREITA	ângulo em graus e raio em cm	Traçar um arco com determinado ângulo de abertura, descrito pela circunferência de raio especificado, andando para trás no sentido anti-horário.
CM-GIRAR-ESQUERDA	ângulo em graus	Rotacionar (se possível no próprio eixo) um determinado ângulo, no sentido anti-horário.
CM-GIRAR-DIREITA	ângulo em graus	Rotacionar (se possível no próprio eixo) um determinado ângulo, no sentido horário.

TABELA 4.16 - Símbolos de sinais disponíveis na implementação do atuador.

Conjunto de Sinais		
Símbolo	Parâmetro	Descrição
SG-PARAR	tempo	Desligar o *hardware* e opcionalmente aguardar um determinado tempo.
SG-FRENTE	tempo acionamento	Acionar os motores a frente, por um determinado tempo.
SG-RE	tempo acionamento	Acionar os motores em marcha ré, por um determinado tempo.
SG-FRENTE-ESQUERDA	tempo acionamento	Acionar os motores a frente, com a orientação das rodas dianteiras à esquerda, por um determinado tempo.
SG-FRENTE-DIREITA	tempo acionamento	Acionar os motores a frente, com a orientação das rodas dianteiras à direita, por um determinado tempo.
SG-RE-ESQUERDA	tempo acionamento	Acionar os motores em marcha ré, com a orientação das rodas dianteiras à esquerda, por um determinado tempo.
SG-RE-DIREITA	tempo acionamento	Acionar os motores em marcha ré, com a orientação das rodas dianteiras à direita, por um determinado tempo.
SG-ESQUERDA	tempo acionamento	Orientar as rodas dianteiras à esquerda.
SG-DIREITA	tempo acionamento	Orientar as rodas dianteiras à direita.
SG-ABORT		Desligar o *hardware* e interromper o envio de sinais.

TABELA 4.17 - Produção de sinais a partir de ações.

Símbolo entrada	Símbolo saída
AC-PARAR	CM-ABORT
CM-ABORT	SG-ABORT
AC-FRENTE	CM-FRENTE(AT-DefaultDst)
CM-FRENTE(AT-DefaultDst)	SG-FRENTE(t)
	SG-RE(t)
	SG-PARAR
AC-ESQUERDA	CM-GIRAR-ESQUERDA(90)
CM-GIRAR-ESQUERDA(90)	SG-DIREITA(t)
	SG-RE-DIREITA(t)
	SG-FRENTE(t)
	SG-PARAR(t)
	SG-ESQUERDA(t)
	SG-FRENTE-ESQUERDA(t)
	SG-RE(t)
	SG-PARAR(t)
	SG-DIREITA(t)
	SG-RE-DIREITA(t)
	SG-FRENTE(t)
	SG-PARAR(t)
	SG-ESQUERDA(t)
	SG-FRENTE(t)
	SG-PARAR
AC-DIREITA	análogo a AC-ESQUERDA

(a) Posição inicial. (b) Ré-direita. (c) Frente-Esquerda.

(d) Ré-direita. (e) Frente-Esquerda.

FIGURA 4.12 - Seqüência de movimentos realizados pelo robô para virar à esquerda.

A distância padrão a ser percorrida (parâmetro *AT-DefaultDst* do atuador, especificado por *AG-DefaultDst*) pode ser obtida em função do tempo de acionamento dos atuadores. Para tanto, dois métodos para mapeamento de distância em tempo foram desenvolvidos.

O primeiro, empírico, produz um mapeamento linear da distância para o tempo t_e = *AT-Alfa* * *AT-DefaultDst*, onde *AT-Alfa* representa um ganho. Outro coeficiente, o de frenagem *AT-Beta*, pode ser utilizado para enviar um novo sinal *SG-RE* ao atuador, com tempo de acionamento t_e = *AT-Beta* * *AT-DefaultDst*. Esse mecanismo de frenagem é utilizado para anular a inércia do robô, quando ele percorrer grandes distâncias.

O outro método realiza um mapeamento não-linear, através de uma função que aproxima a aceleração do robô. Devido a imprecisão do atuador, para cada tempo de acionamento utilizado, pode-se obter diferentes distâncias percorridas pelo robô. Logo, por definição, neste caso não existe uma função que mapeie segundos em distâncias, e nem sua inversa. Contudo, sendo possível criar uma base de dados experimental, optou-se por continuar aplicando métodos de aprendizagem supervisionada, através do uso de uma RNA MLP. Com isso, a aproximação de uma função de aceleração do robô por aprendizagem simples dispensou a estimação de parâmetros da curva. Os dados de treinamento foram obtidos em experimentos, nos quais

mediu-se a distância percorrida pelo robô na pista, após um determinado tempo de acionamento dos atuadores. A Figura 4.13 apresenta a curva de resposta da RNA (com duas camadas ocultas de 10 e 5 unidades), comparadas ao conjunto de dados de treinamento. Observa-se a capacidade de generalização da RNA trabalhando com dados ruidosos. A saída da rede pode ser ainda ponderada pelo coeficiente *AT-Alfa*, para compensar mudanças de condições da bateria do motor do robô, ou seja, o tempo de acionamento $t_n = y* \ AT\text{-}Alfa$, onde y é a saída produzida pela RNA.

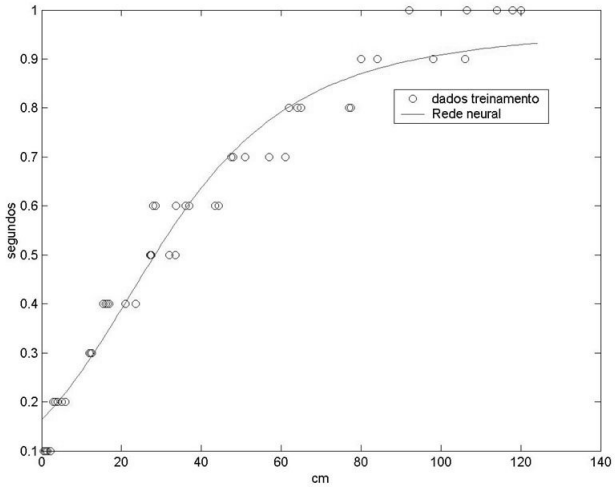


FIGURA 4.13 - Respostas da RNA para o mapeamento de distâncias a serem percorridas em tempo de acionamento dos atuadores.

4.2.1 Parâmetros do Atuador

A Tabela 4.18 sumariza os parâmetros do atuador que devem ser estipulados nos experimentos com o robô autônomo.

TABELA 4.18 - Relação dos parâmetros do sub-sistema atuador.

Parâmetro	Descrição
AT-Alfa	Coeficiente de ganho, no mapeamento de distância para tempo de acionamento do atuador.
AT-Beta	Coeficiente de frenagem.
AT-DefaultDst	Distância padrão a ser percorrida pela ação frente.
AT-Gamma	Coeficiente de rotação.
AT-MAc	Modelo de aceleração utilizado. 0: Neural; 1: Heurístico.
AT-Desligar	Desliga o atuador.

4.3 Detalhes de Implementação do Agente

A Figura 4.14 apresenta o robô utilizado neste experimento. Conforme descrito anteriormente, o robô móvel consiste em um carrinho de controle remoto adaptado, de aproximadamente 27 cm de largura, 33 cm de comprimento e 24 cm de altura. Sobre o seu pára-choque dianteiro, uma câmera CCD foi instalada (Figura 4.14-b), que transmite as imagens por rádio freqüência para um receptor.

(a) Robô móvel. (b) Posicionamento da câmera a frente.

FIGURA 4.14 - Robô móvel utilizado nos experimentos.

O robô utilizado não dispõe de *hardware* adequado para embarcar algum tipo de *software*, assim a implementação e o processamento do agente de aprendizagem serão realizados em um microcomputador, suportados por um sistema computacional, denominado *Cool Autonomous Navigation Enterprise with Learning Agents*(CANELA). Este ambiente foi desenvolvido em Visual C++, utilizando-se o pacote de classes *Microsoft Foundation Class* (MFC) para construção de janelas. O sistema CANELA tem como objetivo auxiliar a implementação do agente e seus sub-sistemas, bem como fornecer mecanismos para condução dos experimentos realizados com o robô móvel.

As principais classes que formam o núcleo de processamento do sistema CANELA estão representadas no diagrama da Figura 4.15. As classes *CCanelaDoc* e *CCanelaView*, tradicionais em implementações MFC, cuidam das janelas que exibem as imagens capturadas e processadas pelo sensor. A classe *CCanelaApp* é janela principal do aplicativo, derivada da classe *CWinApp*, que gerencia os experimentos e instancia o objeto do agente. Além disto, os detalhes técnicos de conexão de digitalizadores para captura de imagens são tratados na classe *CCanelaApp*, promovendo interoperabilidade entre o sub-sistema sensor e o sistema operacional.

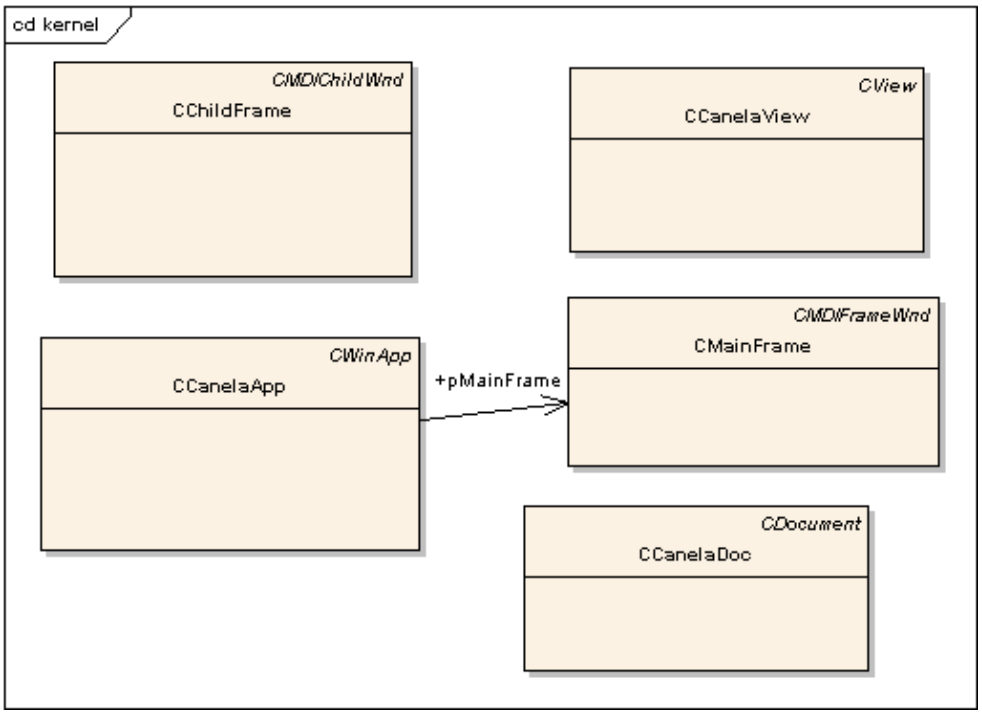


FIGURA 4.15 - Diagrama de classes do *kernel* do sistema Canela.

A Figura 4.16 traz o diagrama de classes do agente e seu relacionamento com os subsistemas. Nota-se que o retorno de recompensas é fornecido através de uma classe que representa o ambiente. O agente é instanciado como uma *thread*, que durante sua vida útil controla o seu comportamento no ambiente, por meio de uma variável de *status*. Essa variável de *status* gerencia o experimento e troca de valor a medida que recebe comandos do aplicativo. Através dela, o usuário pode controlar o andamento do experimento, interrompendo-o, reiniciando-o, confirmando ou rejeitando ações do robô. A Tabela 4.19 descreve o significado de cada *status* que o agente pode assumir. Como pode ser visto, os valores de *status* assumidos pelo agente permitem ao usuário conduzir o experimento, podendo inclusive interferir nas ações do agente. Quando o modo *debug* está ativo, ao escolher uma ação, o agente permanece com o *status Waiting* até que a ação seja confirmada ou rejeitada pelo monitor. É claro que essa intervenção só é tolerada em momentos de testes com o agente e validação da implementação, uma vez que na etapa de navegação o agente deverá agir de forma autônoma.

A transição de *status* do agente é representada pela máquina de estados da Figura 4.17, na qual percebe-se que a condição de transição é guiada pelos comandos definidos pelo usuário. O nodo *parado* representa o estado inicial e final.

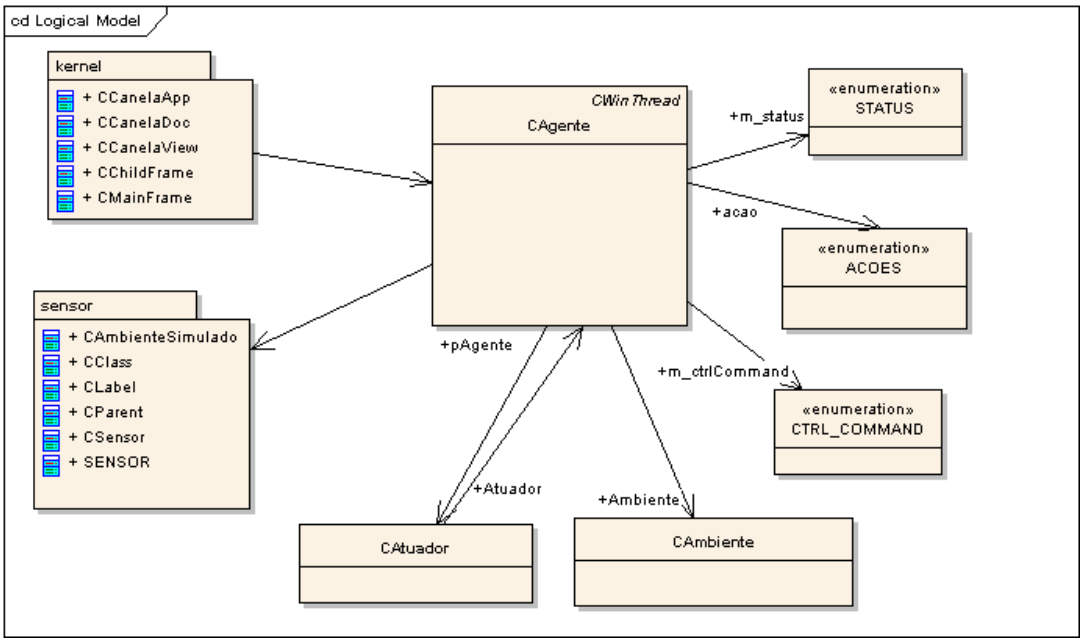


FIGURA 4.16 - Diagrama de classes do agente e seus componentes.

TABELA 4.19 - Relação de *status* assumidos pelo agente no sistema Canela.

Status	Descrição
running	O agente está em **rodando**, seguindo o ciclo: percepção - escolha de ação - tomada de ação - avaliação - atualização de conhecimento.
ready	O agente executou um ciclo e está **pronto** para efetuar a próxima percepção.
waiting	O agente está **aguardando** a confirmação do usuário para tomar a ação escolhida.
stoped	O experimento terminou e o agente está **parado** até que o seja reiniciado.
killed	O aplicativo Canela está sendo finalizado e o agente está destruindo sua instância.

As classes do sub-sistema sensor, ilustradas na Figura 4.18, completam o sistema Canela. Quando está operando em modo simulado, o sensor utiliza a classe *CAmbienteSimulado* para guardar informações da iteração do agente no ambiente e controlar as transições de estados de percepção do agente. Em modo de tempo real, a classe *CClass* classifica a imagem e as classes *CLabel* e *CParent* rotulam-na pelo método matricial para detecção de objetos.

As funcionalidades do sistema Canela podem ser resumidas pela interface da Figura 4.19. Com ela o usuário pode visualizar as imagens que estão sendo capturadas

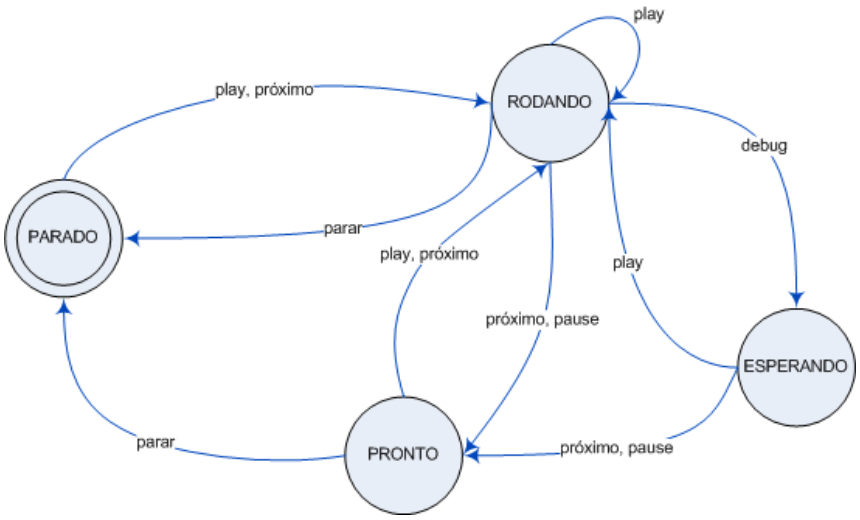


FIGURA 4.17 - Diagrama de estados.

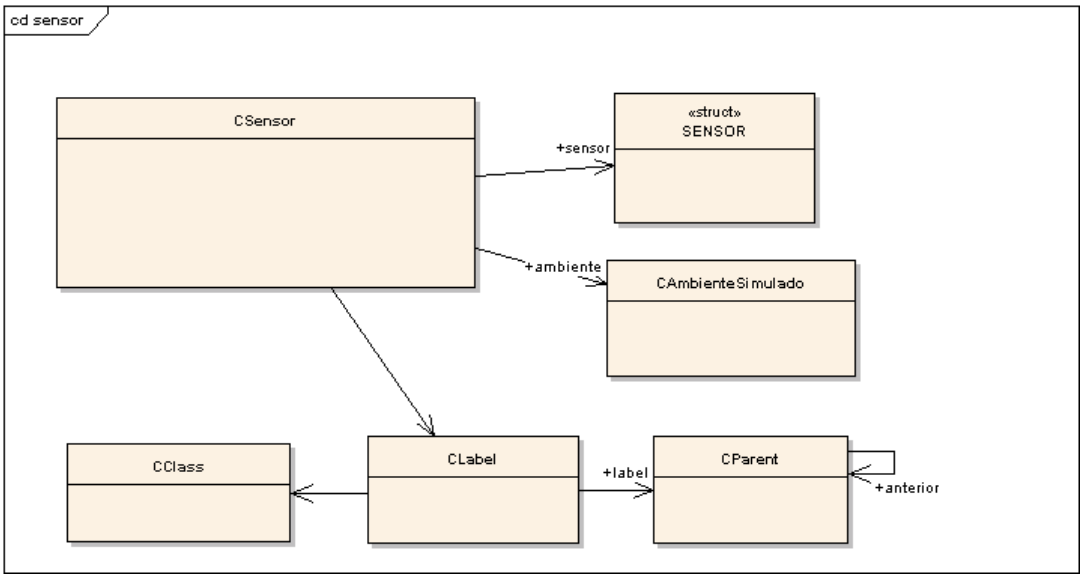


FIGURA 4.18 - Diagrama de classes do sub-sistema sensor.

pelo sub-sistema sensor e o resultado dos processamentos que vêm sendo aplicados a ela. Na janela *vision1*, da Figura 4.19, é possível verificar uma imagem que foi capturada da cena e a respectiva saída do processamento de detecção pelo método matricial, na janela *vision2*. Utilizando a barra de ferramentas, escolhe-se o tipo de processamento de imagem que a janela irá exibir, bem como a cor de fundo da imagem de classificação. Durante o processamento, o agente e os sub-sistemas produzem mensagens de *log*, exibidas na janela *debug*. Anotações podem ser registradas em documentos de textos, como o da janela *relatório*. O controle dos experimentos é realizado através da barra de ferramentas, descrita pela Tabela 4.20.

TABELA 4.20 - Descrição de funcionalidades do sistema Canela através dos ícones da barra de ferramenta.

Configuração do experimento	
	Escolha diretório de trabalho do experimento.
	Abre diálogo para configuração de parâmetros do experimento.
Controle do experimento	
	Habilita / Desabilita experimento em modo *debug*, o qual produz uma pausa antes da execução de cada ação.
	Aceita a execução de uma ação em modo *debug*.
	Rejeita a execução de uma ação em modo *debug*.
	Dispara a execução do experimento.
	Suspende a execução do experimento enquanto o agente aguarda a próxima percepção.
	Finaliza a execução do experimento.
	Dispara a execução do experimento e suspende-a após a execução de uma ação.
Visualização de processamento da imagem	
	Configura o documento para exibir a imagem mais recente capturada pelo sensor.
	Remove qualquer tipo de processamento de imagem associado ao documento.
	Configura o documento para exibir a imagem após o processamento 1.
	Configura o documento para exibir a imagem após o processamento 2.
	Configura o documento para exibir a imagem após o processamento 3.
	Configura o documento para exibir a imagem após o processamento 4.
Opções de visualização de processamento	
	Amplia a visualização da imagem reamostrando-a.
	Invoca o sensor para capturar uma nova imagem da cena.
	Configura a cor de fundo para imagens classificadas, alterando entre desligado, preto e branco.
Documentos	
	Abre uma janela de exibição de mensagens de *log* e *debug*.
	Abre um documento vazio de texto.

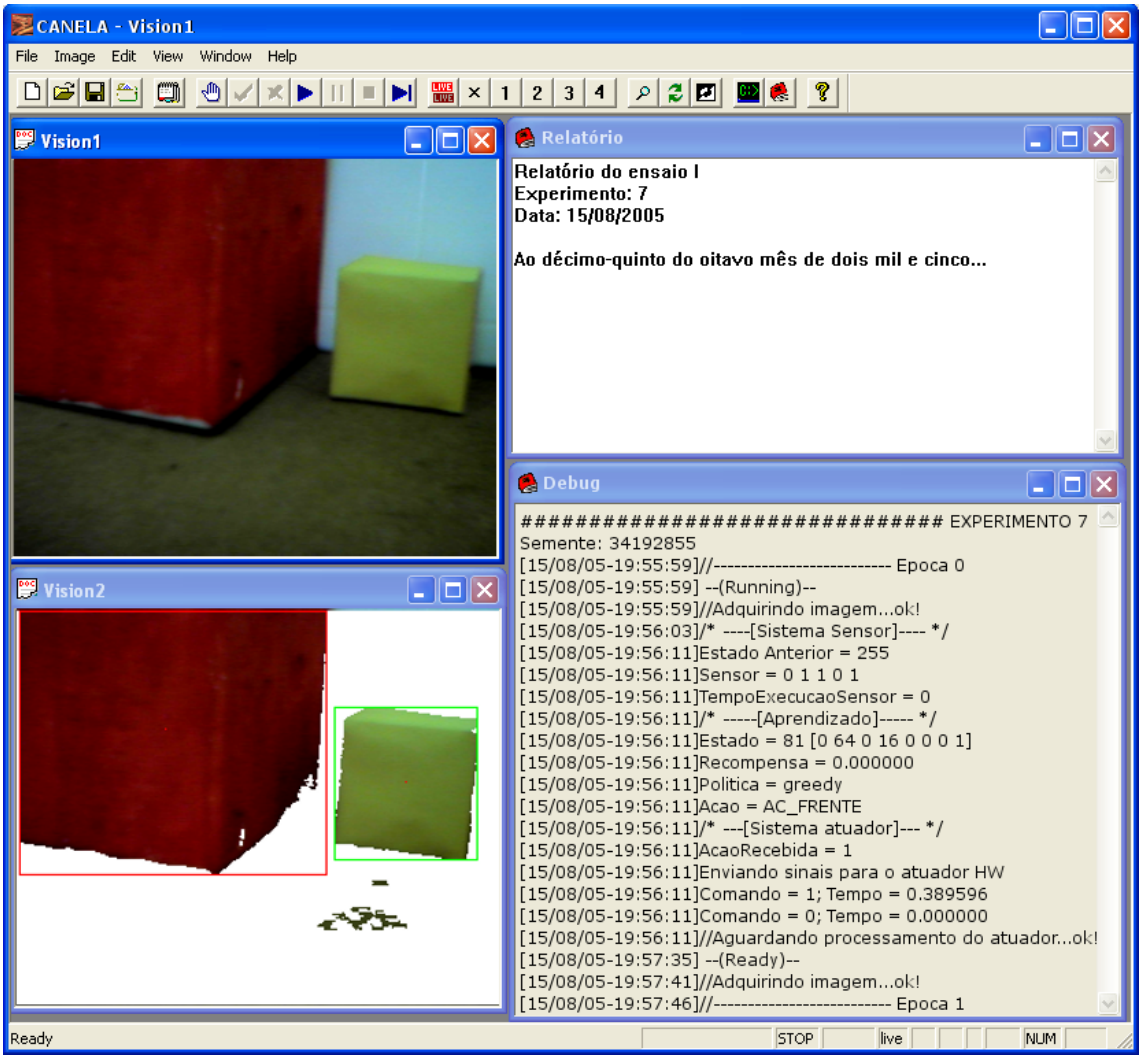


FIGURA 4.19 - Tela principal do sistema Canela.

Um diálogo de configuração permite definir os parâmetros do agente, dos sub-sistemas, além de outras facilidades, como testar o sistema de captura de imagens, gravar e carregar *Q-valores*, ligar e desligar sub-sistemas, definir a semente de geração de números pseudo-aleatórios e selecionar a gravação de arquivos e imagens de *log*.

4.3.1 Interfaces do Sub-sistema Sensor

A imagem capturada pela câmera CCD é transmitida por rádio freqüência até um receptor, que propaga o sinal analógico no formato *Radio Corporation of America* (RCA). O sinal analógico então é capturado pelo dispositivo *Snappy* ®(Play

Inc), que novamente converte a imagem em sinal digital, transmitindo para o *drive Snappy* 4.0 ©, via porta paralela (LPT). Este esquema é ilustrado pela Figura 4.20.

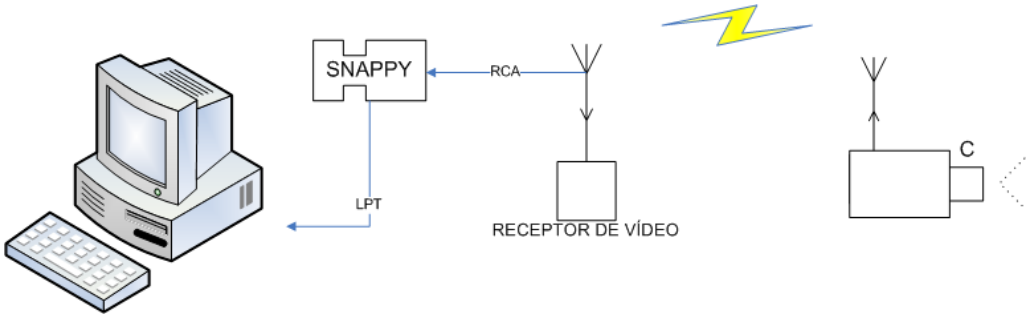


FIGURA 4.20 - Diagrama do sistema de captura de imagens.

A grande desvantagem do digitalizador *Snappy* é sua latência na captura das imagens, que no modo RGB supera os 5 segundos.

Para manipular a imagem digital diretamente via *drive Snappy*, utilizou-se o *Kit* para Desenvolvimento de *Software* de Visão (VisSDK) da Microsoft ®(MICROSOFT, 2003). Este pacote contém um conjunto de classes em C++ que se conectam com qualquer fonte de *Video for Windows* e entregam as imagens capturadas por meio de serviços, provendo transparência no uso de câmeras digitais no sistema CANELA.

O receptor do sinal da câmera CCD e o digitalizador *Snappy* são vistos na Figura 4.21-a, na qual percebe-se a ligação RCA entre os dispositivos e a saída LPT.

(a) Receptor do sinal da câmera CCD e o digitalizador *Snappy*.

(b) Controle remoto do carrinho e placa de circuito impresso.

FIGURA 4.21 - Interfaces de *hardware*.

4.3.2 Interfaces do Sub-sistema Atuador

O acionamento dos atuadores é efetuado pelo controle remoto original do carrinho, adaptado a um circuito elétrico conectado a porta paralela de um computador. A Figura 4.21-b ilustra o controle remoto do carrinho e a placa de circuito impresso. O esquemático da placa de circuito impresso está detalhada no Anexo B.

4.3.3 Montagem do Laboratório para Experimentos com o Robô

Em função das interfaces do sensor e atuador, fez-se necessário o uso de duas portas paralelas. Em razão disto, a montagem do laboratório será configurada com dois microcomputadores, interligados por uma rede TCP-IP, conforme esquematizado na Figura 4.22. O sistema CANELA é instalado no PC1, que grava um arquivo *American Standard Code for Information Interchange* (ASCII) no PC2, contendo os sinais a serem enviados ao *hardware* do atuador.

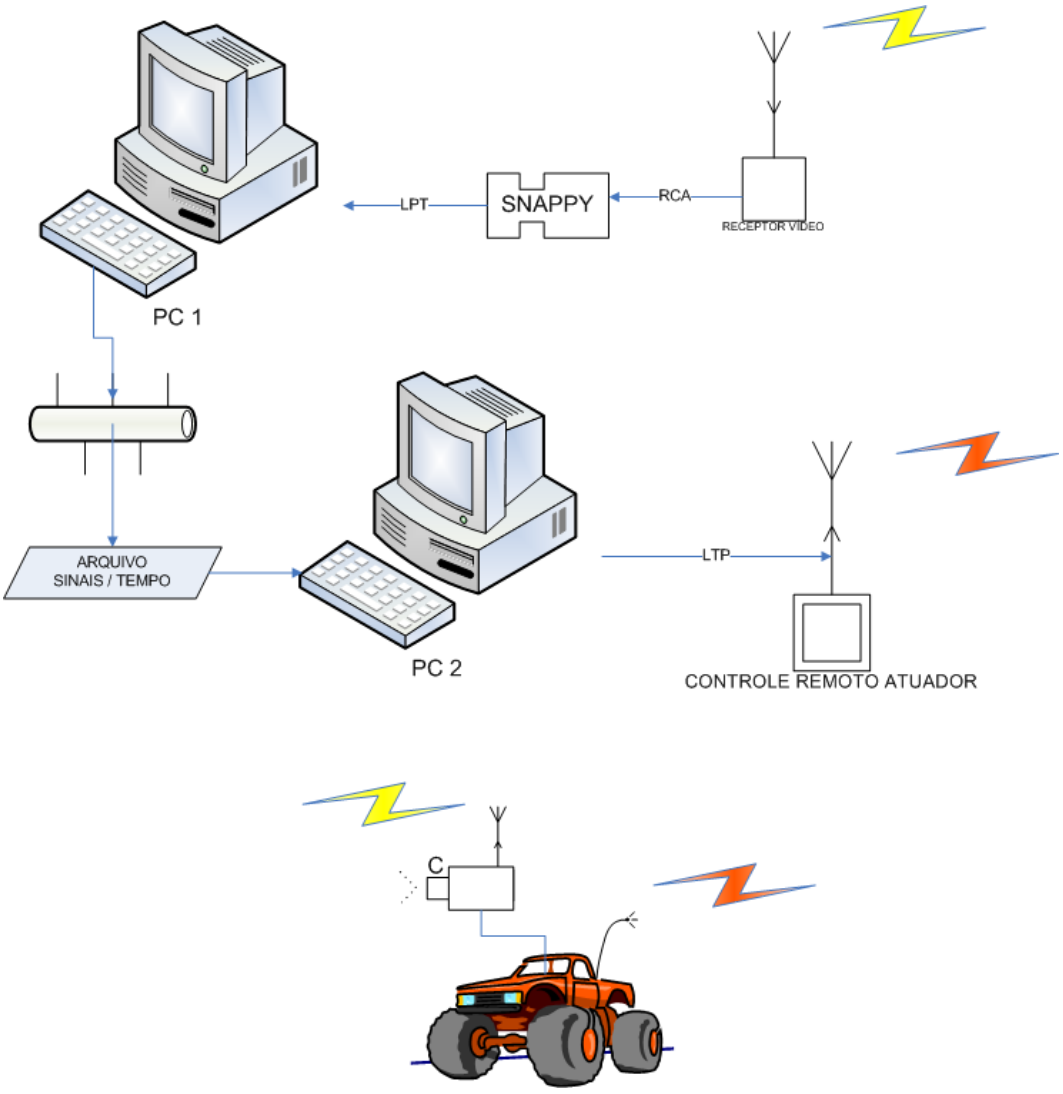


FIGURA 4.22 - Disposição dos equipamentos e configuração do ambiente para realização dos experimentos.

CAPÍTULO 5

EXPERIMENTOS E RESULTADOS

Este capítulo traz três secções descrevendo os experimentos realizados no âmbito de navegação autônoma, utilizando-se o agente de aprendizagem, com o algoritmo *Q-learning*. Inicialmente, alguns testes em um ambiente simples e simulado são realizados, com a finalidade de se avaliar os parâmetros do algoritmo *Q-learning* e a convergência dos *Q-valores*. Em seguida, são apresentados os experimentos realizados com um robô móvel real, que navega livremente em um ambiente desconhecido, evitando os obstáculos no seu caminho. Por fim, visando incrementar o sistema de navegação do robô, um novo modelo do agente de aprendizagem é utilizado, objetivando-se não somente o desvio de obstáculos, mas também a exploração homogênea de terrenos desconhecidos.

5.1 Testes Iniciais

O objetivo desses testes é avaliar como os parâmetros do algoritmo *Q-learning* interferem no aprendizado do agente e na convergência dos *Q-valores*. Para tanto, o ambiente de labirintos, ilustrado pela Figura 5.1, foi simulado no sistema Canela para treinar o agente. Neste labirinto em forma de grade, 24 objetos são posicionados arbitrariamente (representados pelas células pretas, na Figura 5.1), além da delimitação das extremidades do terreno. A grade representa coordenadas discretas que o agente pode ocupar, totalizando $25 \times 25 - 24 = 601$ posições viáveis.

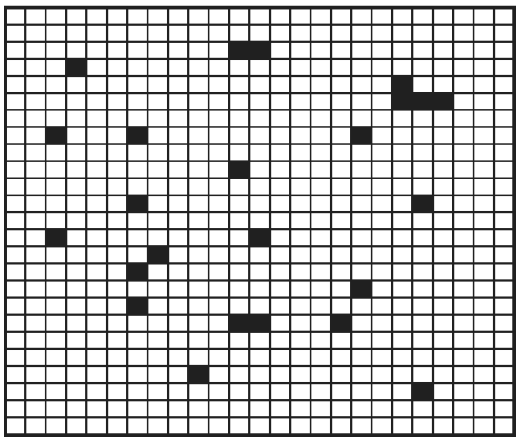


FIGURA 5.1 - Ambiente de labirinto simulado pelo sistema Canela para testes com o algoritmo *Q-learning*. Os obstáculos do ambiente são representados em preto.

A modelagem dos estados do agente é a mesma utilizada no capítulo 4, sendo que na implementação do sensor procurou-se produzir o mesmo tipo de informação do sensor real. A distância dos objetos no ambiente é medida em unidades da grade, estando os obstáculos mais longe do agente a 6 células e o mais próximo (em colisão) a uma célula. Toda codificação está ilustrada na Figura 5.2, na qual percebe-se também a codificação da orientação dos obstáculos (esquerda, frente e direita) em relação a postura do robô.

Em cada estado, o agente pode escolher entre 3 ações para tomar: *AC-FRENTE*, *AC-ESQUERDA* e *AC-DIREITA*. Assume-se que os atuadores do robô são precisos, não existindo problema de localização, e que os movimentos de giro à esquerda e à direita são exatos em 90º graus. Para cada ação tomada à frente, sem colisão, um valor positivo (+1) será fornecido de retorno. Caso o agente venha a colidir com algum obstáculo, uma punição (-1) será enviada. Para as demais ações, a função de retorno será zero.

Ao total, 8 experimentos com 50 mil passos foram realizados, tomando-se o cuidado para reposicionar o robô e reiniciar o sistema Canela com a mesma semente. Os parâmetros utilizados no algoritmo *Q-learning* estão especificados na Tabela 5.1.

A Figura 5.3 apresenta o resultado da aproximação dos *Q-valores* para o estado 0, no final dos 8 experimentos. O estado 0 é mapeado quando não existem obstáculos detectados a frente do robô, logo o aprendizado desejado é que o agente tome a ação de andar à frente. Observa-se que em todas as simulações, o valor da ação *AC-FRENTE* rapidamente ultrapassa o valor das demais ações, como era esperado. Porém, a variação dos parâmetros do algoritmo *Q-learning*, modificam consideravelmente a forma de convergência desses valores.

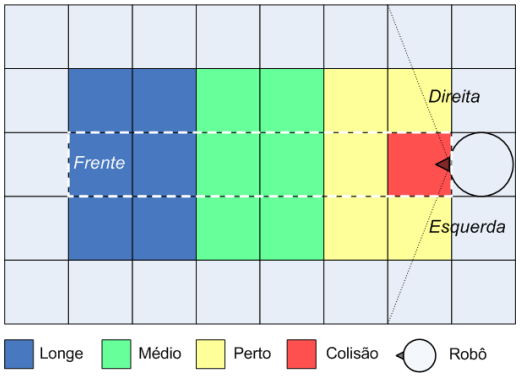
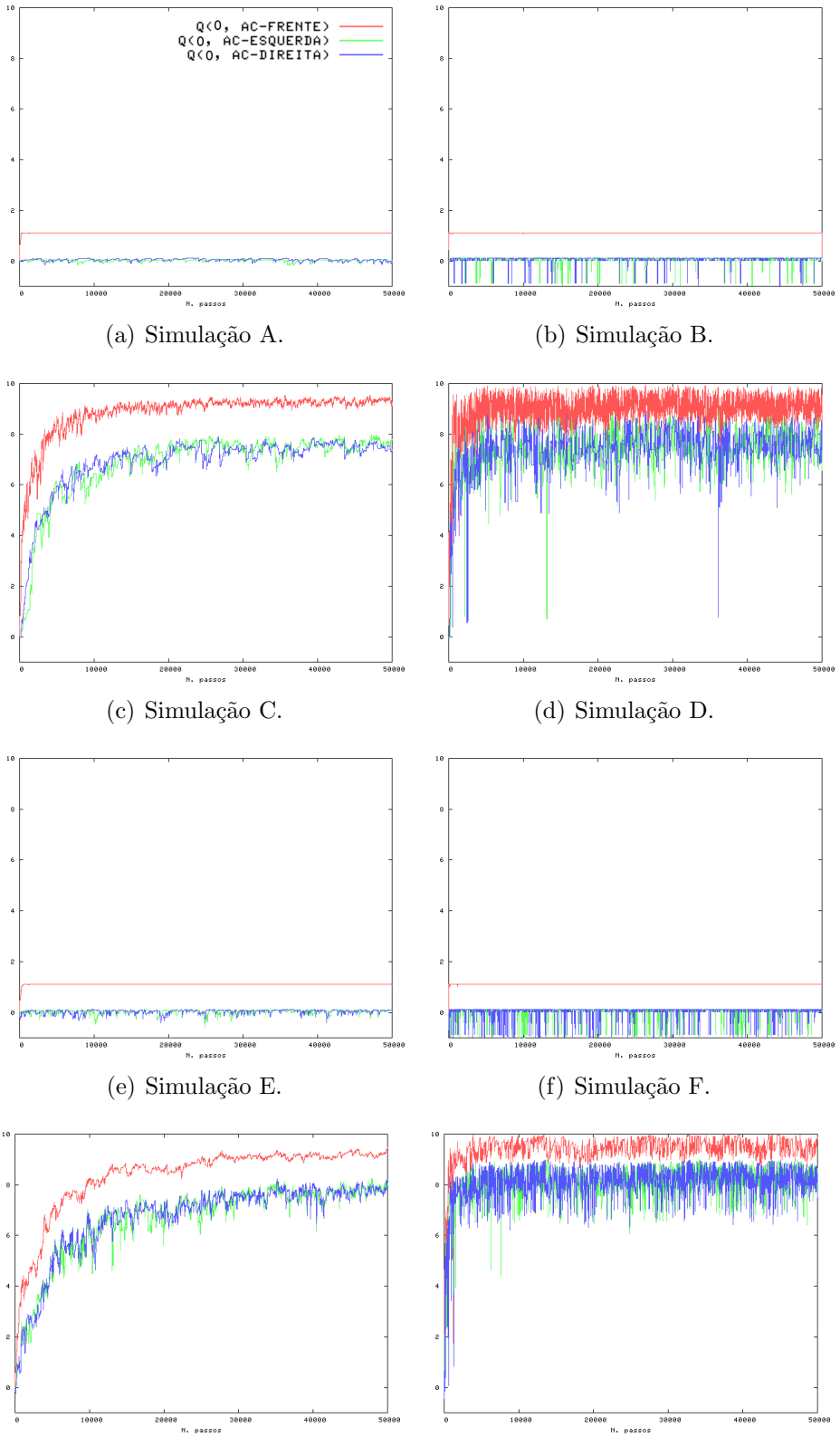


FIGURA 5.2 - Implementação do sensor no experimento simulado. A codificação da distância dos objetos está representada em diferentes cores.



(a) Simulação A. (b) Simulação B.

(c) Simulação C. (d) Simulação D.

(e) Simulação E. (f) Simulação F.

(g) Simulação G. (h) Simulação H.

FIGURA 5.3 - Curvas de aproximação dos *Q-valores* para o estado 0, nos 8 experimentos realizados.

TABELA 5.1 - Identificação dos experimentos realizados no ambiente simulado.

Exp.	Parâmetros		
	ϵ-*greedy*	γ	α
A	0,2	0,1	0,1
B	0,2	0,1	0,9
C	0,2	0,9	0,1
D	0,2	0,9	0,9
E	1,0	0,1	0,1
F	1,0	0,1	0,9
G	1,0	0,9	0,1
H	1,0	0,9	0,9

Em casos do uso de uma taxa de aprendizado α elevado, nos experimentos B, D, F e H (na segunda coluna de gráficos da Figura 5.3), os *Q-valores* tendem a ficar instáveis e não convergem suavemente. Com uma taxa pequena, as perturbações nas curvas são menores (primeira coluna de gráficos da Figura 5.3).

Quando o coeficiente γ é baixo, os *Q-valores* tendem aos valores de retorno imediato, ou seja, 1 para a ação frente e 0 para as demais, como pode ser visto nas Figuras 5.3-a 5.3-b 5.3-e 5.3-f. O uso de $\gamma = 0,9$ aumenta os *Q-valores* para próximo de 10. Ressalta-se que para os *Q-valores* não crescerem indefinidamente, esse parâmetro deve ser menor que 1.

Finalmente, comparando-se as curvas dos *Q-valores* nos experimentos C (Figura 5.3-c) e G (Figura 5.3-g), os quais usam respectivamente ϵ-*greedy* 0,2 e 1,0, percebe-se que o uso da política de escolha de ação que não é completamente aleatória, acelera a convergência dos *Q-valores*, mas pode limitar a exploração do espaço de estados.

Acredita-se que a configuração de α pequeno e γ grande, como utilizado nas simulações C e G, seja a mais adequada para os experimentos que serão realizados neste capítulo, pois aproximam os *Q-valores* de forma suave, tratando o problema de retorno com atraso. De fato, esta é a configuração indicada na literatura, como é o caso do trabalho de Crook e Hayes (2003).

A fim de se comparar o aprendizado em vários estados, o gráfico da Figura 5.4 exibe as curvas dos *Q-valores* para a ação *AC-FRENTE*, nos estados 0, 4, 8, 12 e 64, do experimento G. Esses estados representam a seqüência de aproximação de um objeto a frente do robô (0: nenhum objeto; 4: objeto longe; 8: objeto à média distância; 12: objeto perto; 64: colisão). Como era esperado, os valores decaem a medida que

o objeto se aproxima do robô, sendo os valores do estado 12 e 64 semelhantes.

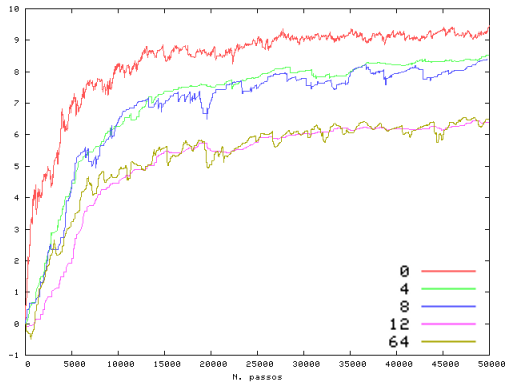


FIGURA 5.4 - Comparação da curva dos *Q-valores* para a ação *AC-FRENTE*, em estados que representam a aproximação do robô até um objeto.

Conforme discutido anteriormente, o uso de ϵ-*greedy* totalmente aleatório ou não, interferiu pouco na definição de um *Q-valor* predominante, ou seja, escolha da melhor ação aprendida até o momento. Todavia, para uma melhor aproximação global dos valores e exploração do espaço de estados, o uso do coeficiente ϵ-*greedy* $= 1,0$ é recomendado. Por outro lado, a escolha direcionada de ações (ϵ-*greedy* próximo a zero), privilegia o recebimento de recompensas durante o processo de aprendizagem, como se pode observar nas curvas de retorno médio no gráfico da Figura 5.5.

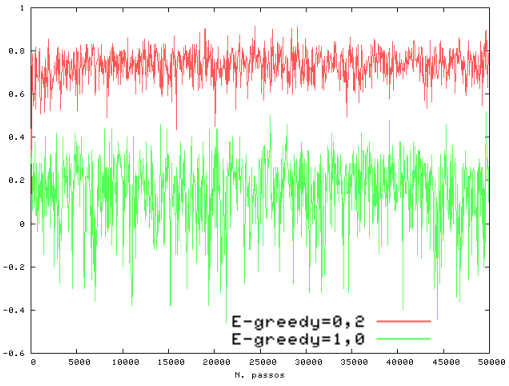


FIGURA 5.5 - Retorno médio de recompensas em simulações com escolha de ações totalmente e parcialmente aleatória.

Com o uso de ϵ-*greedy* $= 0,0$; o aprendizado simulado pode ficar facilmente preso em mínimos locais. Já em ambientes físicos reais isto é mais difícil, pois as transições de estados são bem menos determinísticas do que no ambiente simulado, uma vez que existe ruído nos atuadores e sensores.

Para minimizar o dilema de exploração do espaço de estados, é possível realizar o processo de aprendizagem em duas etapas. Na primeira, efetua-se o treinamento do agente em ambiente simulado, para aproximação básica dos *Q-valores*, com ϵ-*greedy* $= 1,0$. Posteriormente, o agente utiliza esses *Q-valores* (na forma de *bootstrap*) para aprimorar o seu conhecimento, usando um ϵ-*greedy* baixo.

Essa abordagem foi experimentada em simulações e posteriores ensaios com o robô real, mas não deram bons resultados, pois o sistema de simulação de treinamento não é muito fiel com a realidade, o que acabou forçando situações que no ambiente real não são possíveis. Por exemplo, em ambiente simulado o agente era capaz de passar por entre 2 objetos no ambiente, mas no mundo real isto nem sempre era possível. Portanto, o uso de simulação para acelerar o processo de aprendizagem é útil quando se tiver a disposição um simulador com boa fidelidade, ou quando se deseja aprender situações muito simples, do tipo: *bateu-então-não vá pra frente*.

Uma outra alternativa é dividir o aprendizado em duas fases, e utilizar o método proposto por Smart e Kaelbling (2002). Na primeira fase, o robô é guiado pelo ambiente de forma supervisionada, para que o agente seja apresentado a situações interessantes do espaço de estados. A segunda fase segue com o aprendizado padrão em *Q-learning*, em que o robô navega pelo ambiente de forma não supervisionada, utilizando o conhecimento adquirido na primeira fase. Contudo, esta abordagem não será considerada neste trabalho, pois deseja-se que o agente aprenda a tomar decisões, sem a intervenção de um tutor externo.

Deste modo, os experimentos apresentados a seguir, com o uso de um robô móvel real, partirão de uma tabela de *Q-valores* zerada.

5.2 Experimentos com Visão Computacional

A seguir, uma série de experimentos com o robô móvel são apresentados. Cada *experimento* é composto de um conjunto de *passos* efetuados pelo agente, ou seja, um conjunto de ciclos de percepção à ação. Para melhor organização e controle, os experimentos são agrupados e identificados em *ensaios*. Ao total, foram realizados 5 ensaios, 21 experimentos em 477 passos, conforme relacionado na Tabela 5.2. Para fácil referência, um determinado passo pode ser identificado por *D3:6*, denotando por exemplo, o sexto passo, do terceiro experimento, do ensaio *D*. Eventualmente, alguns experimentos poderão ser descartados das análises de navegação do robô, como é

TABELA 5.2 - Identificação dos experimentos, compostos por passos e agrupados em ensaios.

ID. Ensaio	ID. Experimento	N° de Passos
A	0	59
B	0	27
	1	4
	2	1
	3	1
	4	22
	Sub-total =	**55**
C	0	31
	1	18
	2	31
	3	52
	Sub-total =	**132**
D	0	78
	1	9
	2	2
	3	7
	Sub-total =	**96**
E	0	6
	1	19
	2	3
	3	45
	4	2
	5	32
	6	28
	Sub-total =	**135**
	Total =	**477**

o caso de B2 e B3, onde o pequeno número de passos indica que o experimento teve de ser interrompido devido a falhas técnicas. Ainda assim, esses passos foram considerados para o processo de aprendizado do algoritmo *Q-learning*.

Um conjunto de parâmetros é especificado no início de cada experimento. Por padrão (*default*), os parâmetros da Tabela 5.3 são assumidos, sendo as exceções explicitadas. Adicionalmente, a posição inicial do robô e a disposição dos objetos fixos no ambiente também podem ser modificadas nos intervalos dos experimentos. O sistema Canela possibilita ainda a definição da semente para geração de números pseudo-aleatórios e um tempo limite (*timeout*) de espera para aquisição da imagem pelo digitalizador (fixado neste trabalho em 15 segundos).

TABELA 5.3 - Parâmetros padrão assumidos nos experimentos.

Sub-sistema	Parâmetro	Valor
Q-learning	α	0,1
	γ	0,8
	ϵ-*greedy*	0,0
Agente	*AG-DefaultDst*	35 cm
	AG-PREMIO	1,0
	AG-PUNICAO	-1,0
	AG-DEFAULT	0,0
Sensor	*SE-Simular*	*não*
	SE-areaTH	20,0
	SE-SpTH	6,0
	SE-DstLonge	$[0, \frac{1}{5}R]$
	SE-DstMedio	$(\frac{1}{5}R, \frac{12}{25}R]$
	SE-DstPerto	$(\frac{12}{25}R, \frac{9}{10}R]$
	SE-DstBump	$(\frac{9}{10}R, R)$
	SE-cOr	$\frac{1}{4}C$
Atuador	*AT-Desligar*	*não*
	AT-Alfa	1,0
	AT-Beta	0,0
	AT-Gamma	1,0
	AT-DefaultDst	35 cm
	AT-MAc	0

O ambiente físico, arranjado para os experimentos, consiste em uma sala de piso plano, com aspecto retangular de $300cm \times 260cm$. Nos dois primeiros ensaios (A e B), o ambiente não era delimitado por objetos, ficando inviável ao agente determinar a presença da parede, acarretando na interrupção do experimento quando o robô colidia. Assim, nos ensaios seguintes, os limites externos do ambiente foram todos delimitados por objetos planos, formando uma barreira para o robô, conforme ilustrado pela Figura 5.6-a. Durante os ensaios, outros objetos, de variadas formas, tamanhos e cores, são espalhados pelo interior do ambiente. Os tamanhos dos objetos variam de $10cm$ a $35cm$. Algumas imagens de configurações do ambiente podem ser vistas na Figura 5.6.

(a) Objetos que delimitam o ambiente são posicionados nas extremidades.

(b) Exemplo de configuração de obstáculos no ambiente.

(c) Exemplo de configuração de obstáculos no ambiente, sem delimitação.

FIGURA 5.6 - Ambiente físico onde são realizados os experimentos.

5.2.1 Ensaio A

O experimento do ensaio A partiu de uma tabela de *Q-valores* zerada, escolhendo suas ações aleatoriamente em aproximadamente 20% dos 59 passos efetuados. O objetivo desse ensaio é permitir que o agente adquira um comportamento básico (e vital), como por exemplo, *não ir para frente quando bateu*. Isto permitirá uma navegação mais cautelosa nos próximos experimentos. Sendo assim, não se está preocupado em analisar trajetórias ou escolhas de ações, mas apenas em realizar uma exploração inicial do espaço de estados. Optou-se em estipular ϵ-*greedy*$= 0,2$, pois uma navegação com um índice elevado de aleatoriedade dificulta o andamento do experimento, uma vez que o agente facilmente fica preso em algumas regiões do ambiente.

O resultado do aprendizado do ensaio A está sumarizado na Tabela 5.4, na qual as

ações a serem estimuladas e evitadas estão explicitadas. Percebe-se que de forma geral, quando o caminho está livre, o robô privilegia a ação andar para frente, como é o caso nos estados 0, 1, 2, 8, 16, 32 e 34 (este comportamento só não ocorreu no estado 17). Se for detectada uma colisão (estado 64), o agente deve evitar a ação frente, escolhendo como alternativa a ação virar à direita. Para os demais estados visitados da Tabela 5.4, objetos foram detectados próximo ao robô, e na maioria das vezes a ação frente passou a ser evitada. No caso particular dos estados 03, 51 e 52, as ações tomadas sempre foram virar a esquerda ou a direita, não havendo retorno de recompensas.

Neste ensaio, o parâmetro *AT-ALFA* foi valorado em $1,35$.

TABELA 5.4 - Espaço de estados visitados pelo agente durante o experimento do Ensaio *A* e o respectivo aprendizado de ações a serem estimuladas e evitadas.

Estado					Ação	
ID	B	E	C	D	estimulada	evitada
0	0	00	00	00	*AC-FRENTE*	-
01	0	00	00	01	*AC-FRENTE*	-
02	0	00	00	10	*AC-FRENTE*	-
03	0	00	00	11	-	-
07	0	00	01	11	-	*AC-FRENTE*
08	0	00	10	00	*AC-FRENTE*	-
12	0	00	11	00	*AC-ESQUERDA*	*AC-FRENTE*
16	0	01	00	00	*AC-FRENTE*	-
17	0	01	00	01	*AC-ESQUERDA*	-
32	0	10	00	00	*AC-FRENTE*	-
34	0	10	00	10	*AC-FRENTE*	-
35	0	10	00	11	-	*AC-DIREITA*
48	0	11	00	00	-	*AC-FRENTE*
50	0	11	00	10	-	*AC-FRENTE*
51	0	11	00	11	-	-
52	0	11	01	00	-	-
64	1	00	00	00	*AC-DIREITA*	*AC-FRENTE*

B: colisão; E: esquerda; C: centro; D: direita.
Objetos: 00 = nenhum ; 01 = longe; 10 = médio; 11 = perto.

5.2.2 Ensaio B

No ensaio B foram realizados 5 experimentos, sendo que os experimentos B1, B2 e B3 foram interrompidos por problemas na câmera do robô, e conseqüentemente

descartados das análises de aprendizado. Os seguintes parâmetros, fora do padrão estabelecido na Tabela 5.3, foram alterados: $AT\text{-}ALFA = 1,00$ e $\epsilon\text{-}greedy = 0,2$.

Nos experimentos B0 e B4 são efetuadas 2 trajetórias tímidas, baseadas no conhecimento adquirido no ensaio anterior, incluindo a exploração de novos estados. No passo B0:16 o agente volta a visitar o estado 3 $\rightarrow$ [0|00|00|11] e desta vez escolhe a ação frente (Figura 5.7-a). Apesar de existir um objeto próximo ao robô, não houve colisão, nem transição de estados. Assim, com o recebimento de premiação, o robô novamente executa a ação de andar para a frente (Figura 5.7-b), acabando por colidir no obstáculo (Figura 5.7-c). Posteriormente, no passo B4:11, o agente encontra-se no estado 3 (Figura 5.7-d), mas desta vez aprendeu que deve girar para a esquerda ou direita. A evolução do aprendizado, através da estimação dos *Q-valores* do estado 3, é apresentada na Tabela 5.5.

Apesar da promissora capacidade de aprendizado do agente, observada neste ensaio, alguns problemas no sub-sistema sensor e atuador foram detectados. Em razão do limitado campo de visão da câmera acoplada ao robô, alguns objetos muito próximos, em rota de colisão, não são visíveis. A imagem do passo B0:5, da Figura 5.8-a, ilustra esta situação, em que o objeto representado pela bola azul ficou oculto. Ao se deslocar para frente, o robô colide com o objeto, sem que haja tal conhecimento (Figuras 5.8-b e 5.8-c).

(a) Estado 3, ação frente. (b) Estado 3, ação frente (c) Colisão. (d) Estado 3, ação direita.

FIGURA 5.7 - Desenvolvimento do aprendizado a partir do estado 3.

TABELA 5.5 - *Q-valores* do estado 3 nos passos B0:16, B0:17 e B4:11.

Passo	*AC-FRENTE*		*AC-ESQUERDA*		*AC-DIREITA*	
	Q	NV	Q	NV	Q	NV
B0:16	0,00	0	0,00	2	0,00	0
B0:17	0,10	1	0,00	2	0,00	0
B4:11	$-8,9 \times 10^{-3}$	2	0,00	2	0,00	0

Q: *Q-valor*; NV: Número de visitas ao estado.

O segundo problema está vinculado a característica de guiagem do robô, que não permite executar manobras de giro em torno do seu próprio eixo. Com esta restrição, o atuador é forçado a efetuar várias manobras para executar a ação de girar à esquerda ou à direita, ocupando uma área significativa. Conseqüentemente, durante as manobras, pode ocorrer a colisão com algum objeto, como é o caso ilustrado na Figura 5.9, em que o robô colide com a bola vermelha.

Ainda que estes fatores venham a afetar o desempenho do agente, no que tange a sua capacidade em navegar sem sofrer colisões, acredita-se que o seu desempenho global seja satisfatório.

(a) Imagem capturada pela câmera.

(b) Posição do robô no instante da figura (a).

(c) Colisão do robô com a bola azul.

FIGURA 5.8 - Impossibilidade de detecção de obstáculos, em virtude do restrito campo de visão da câmera.

(a) Robô com objeto a frente, antes de executar a ação.

(b) Robô executa a ação de virar a direita e atinge objeto.

FIGURA 5.9 - Colisão em objeto durante manobra de virar à direita.

5.2.3 Ensaio C

Com a necessidade de se delimitar o ambiente do experimento, a partir deste ensaio, objetos foram colocados nas extremidades do terreno, conforme descrito anteriormente. Com a área central do ambiente livre de objetos, o agente pôde explorar a região, utilizando-se de vários valores do coeficiente ϵ-*greedy*: iniciando em $0,2$ nos dois primeiros experimentos, aumentando para $0,9$ no terceiro e finalmente sendo zerado no último. Foi estipulado o valor $1,20$ para o parâmetro $AT\text{-}ALFA$, no primeiro experimento, e $1,00$ para os demais.

O objetivo neste ensaio foi prover o primeiro contato do agente com a nova configuração do ambiente, permitindo uma transição de ϵ-*greedy* para zero. Espera-se que com a política determinística de escolha de ações, o agente navegue de forma cautelosa, evitando novas experiências, o que lhe levaria a colisões e eventuais interrupções do experimento.

A partir do quarto experimento, percebeu-se que o agente já havia adquirido o comportamento de evitar as paredes, como é observado na trajetória ilustrada pela Figura 5.10, referente aos passos C4:21 a C4:34.

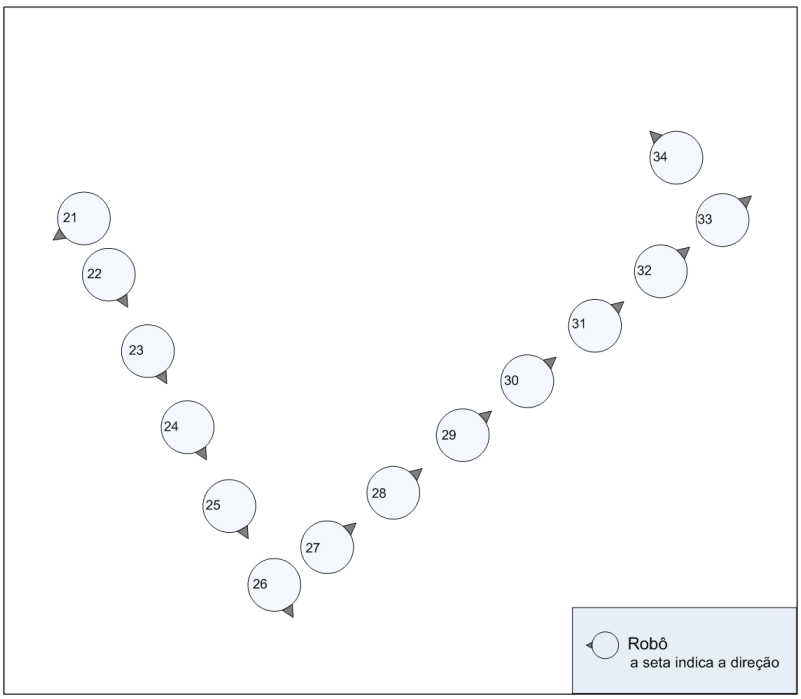


FIGURA 5.10 - Trajetória do robô nos passos C4:21 a C4:34.

Chama-se a atenção para o fato de que mesmo com o coeficiente ϵ-*greedy* $= 0$, o agente ainda pode tomar algumas ações de forma estocástica e continuar o seu processo de aprendizagem. Em estados que ainda não foram avaliados pelo agente (sem retorno de recompensas), ou quando houver empate dos *Q-valores*, a escolha de ações se dará de forma aleatória.

Mudanças de comportamento também podem ocorrer. Por exemplo, no passo C3:10, o sub-sistema sensor detectou dois objetos na cena (Figura 5.11-a), identificando o estado $35 = [0|10|00|11]$. Porém, o comportamento aprendido até então, com $Q\text{-}valor(35,\ AC\text{-}FRENTE\ ) = 0,1001$, $Q\text{-}valor(35,\ AC\text{-}ESQUERDA\ ) = 0,0084$ e $Q\text{-}valor(35,\ AC\text{-}DIREITA\ ) = -0,1$, indicou a escolha da ação *AC-FRENTE*. Esta ação ocasionou a colisão do robô (Figura 5.11-b) e uma posterior punição, forçando o agente a modificar o seu conhecimento (agora $Q\text{-}valor(35,\ AC\text{-}FRENTE\ ) = 0,0015$). Ao retornar ao mesmo estado 35 (e por coincidência ao mesmo local do ambiente), no passo C3:26 (Figura 5.11-c), o agente, entendendo que a ação *AC-FRENTE* era ruim, tomou a ação *AC-ESQUERDA*, encontrando um caminho livre no ambiente (Figura 5.11-d).

(a) Estado 35 no passo C3:10.

(b) Colisão em C3:11 após andar a frente.

(c) Estado 35 no passo C3:26.

(d) Caminho livre em C3:27 após virar a esquerda.

FIGURA 5.11 - Modificação de conhecimento adquirido com uso de ϵ-*greedy* $= 0$.

5.2.4 Ensaio D

Antes que obstáculos sejam inseridos no interior do ambiente, este ensaio é realizado ainda com o terreno livre. Ao total, o agente efetua 96 passos, distribuídos em 4 experimentos. Não há reposicionamento do robô entre os experimentos, apenas o ajuste do parâmetro *AT-ALFA*, a medida que a bateria do atuador vai enfraquecendo. Os valores foram ajustados para 1,00; 1,35; 1,50 e 1,75, respectivamente nos experimentos.

Analisando a seqüência de ações ilustrada pela Figura 5.12, é possível perceber que

neste ambiente com obstáculos delimitando a parede, o robô assume um comportamento que visa evitar o contato com a parede. Os cantos oferecem um desafio adicional, uma vez que o robô não consegue encontrar um caminho livre, com apenas uma ação. Isto é observado nas duas primeiras imagens da Figura 5.12, nos passos D0:58 e D0:59.

No final da primeira linha de imagens da Figura 5.12, nota-se que o robô toca levemente a parede, a sua esquerda. Esta colisão ocorre em virtude de falha no sistema sensor, que como já havia sido previsto, tem dificuldades em identificar os objetos de cor amarela. No passo D0:63, o objeto amarelo já estava em posição de colisão, que não foi detectada, como pode ser visto nas Figuras 5.13-a, 5.13-b e 5.13-c. Ainda assim, no próximo passo D0:64, o sistema foi tolerante o suficiente para detectar a colisão (Figuras 5.13-d, 5.13-e e 5.13-f) e mudar de direção.

Uma análise mais cuidadosa das ações do robô no estado 14 = [0|00|11|10], no passo D0:63, revela que o agente já deveria estar preparado para evitar um objeto tão próximo, a sua frente. Olhando-se então os *Q-valores* do estado 14, apresentados na Tabela 5.6, percebe-se que de fato o agente não havia ainda experimentado as ações de *AC-FRENTE* e *AC-DIREITA* neste estado, mas já sabia que virar a esquerda era uma opção ruim. Com a colisão no passo D0:64, a ação *AC-DIREITA* passa então a ser priorizada no estado 14. Essa adaptação aumenta a tolerância do sistema aos ruídos do sensor, pois caso o agente venha enfrentar essa mesma situação, evitará as ações já experimentadas.

TABELA 5.6 - *Q-valores* do estado 14 nos passos D0:63 e D0:64.

Passo	*AC-FRENTE*		*AC-ESQUERDA*		*AC-DIREITA*	
	Q	NV	Q	NV	Q	NV
D0:63	0,00	0	-0,0934	1	0,00	0
D0:64	-0.0766	1	-0,0934	1	0,00	0

Q: *Q-valor*; NV: Número de visitas ao estado.

5.2.5 Ensaio E

Neste último ensaio com o robô móvel, 7 experimentos foram realizados, sendo que os experimentos E0 e E4 foram descartados. Os experimentos E1, E2 e E3 formam um único experimento de 67 passos, no qual apenas o valor de *AT-ALFA* foi modificado (1,0; 1,5 e 1,2 respectivamente). Após o rearranjo dos obstáculos no ambiente e o

FIGURA 5.12 - Seqüência de imagens do robô, demonstrando sua trajetória no ambiente, iniciando no passo D0:58 ao D3:4.

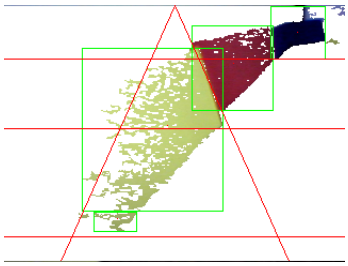
reposicionamento do robô, os experimentos E5 e E6 também consistem em um único experimento, com 60 passos, utilizando *AT-ALFA* 1,2 e 1,5, respectivamente.

A trajetória realizada pelo robô nos experimentos E1, E2 e E3 está documentada nas imagens da Figura 5.14. Com a maioria dos objetos posicionados na periferia do ambiente, o robô manteve um comportamento semelhante ao descrito no ensaio anterior. Ao total, o robô colidiu 5 vezes (tocou em obstáculos), ainda encontrando dificuldades nas regiões de canto.

A trajetória realizada pelo robô na última seqüência de experimentos (E5 e E6) está ilustrada na Figura 5.15, detalhando as 60 ações tomadas. Percebe-se que o agente pôde, na maioria das vezes, evitar as paredes e obstáculos do ambiente, tomando o

(a) Posição do robô no passo D0:63

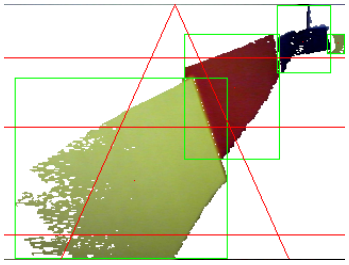
(b) Imagem capturada no passo D0:63



(c) Estado 14 mapeado no passo D0:63

(d) Posição do robô no passo D0:64

(e) Imagem capturada no passo D0:64



(f) Estado 64 mapeado no passo D0:64

FIGURA 5.13 - Colisão do robô no passo D0:63 e D0:64 e recuperação da trajetória.

rumo a frente sempre que possível. Por volta do passo E6:10 (ação número 40), o robô encontrou dificuldades em sair da região, pois ficou isolado entre uma parede e um objeto.

Das 60 ações tomadas, 9 levaram o robô a sofrer colisões com obstáculos, sendo que 7 foram no momento de manobras. Conforme demonstrado anteriormente, essas colisões ocorrem porque o atuador do robô não permite manobras de giro em torno do seu próprio eixo.

Uma colisão ocorreu devido a tomada de decisão incorreta, em uma situação que o agente não havia ainda experimentado (todos os Q-valores eram zero) (Figura 5.16). No passo E5:22 (Figura 5.16-a) o agente percebe o estado 57 = [0|11|10|01] e sorteia a ação *frente*. Esta ação leva o robô a uma colisão (Figura 5.16-b). Mais tarde, no passo E6:44 (ação número 12) (Figura 5.16-a) o robô percebe novamente o estado 57, mas desta vez sabe que a ação frente não é a mais indicada (Q-valor$(57, frente) = -0.0468$), então escolhe a ação virar a *direita* (Figura 5.16-b).

A seqüência de ações ilustradas pela Figura 5.16 demonstra novamente a capacidade de aprendizado e adaptação do agente, frente a situações nunca experimentadas.

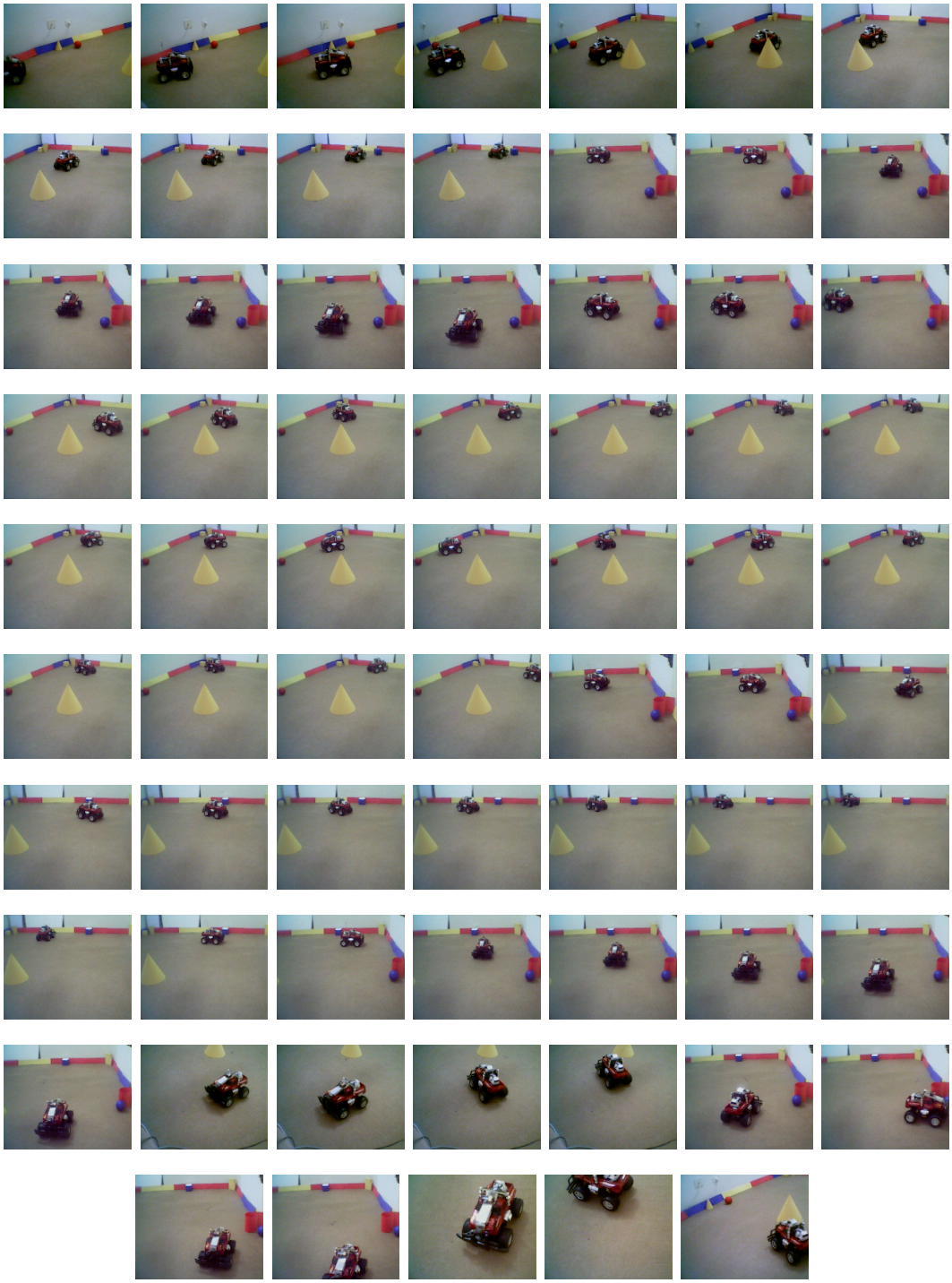


FIGURA 5.14 - Seqüência de imagens do robô, demonstrando sua trajetória no ambiente, durante os experimentos E1, E2 e E3.

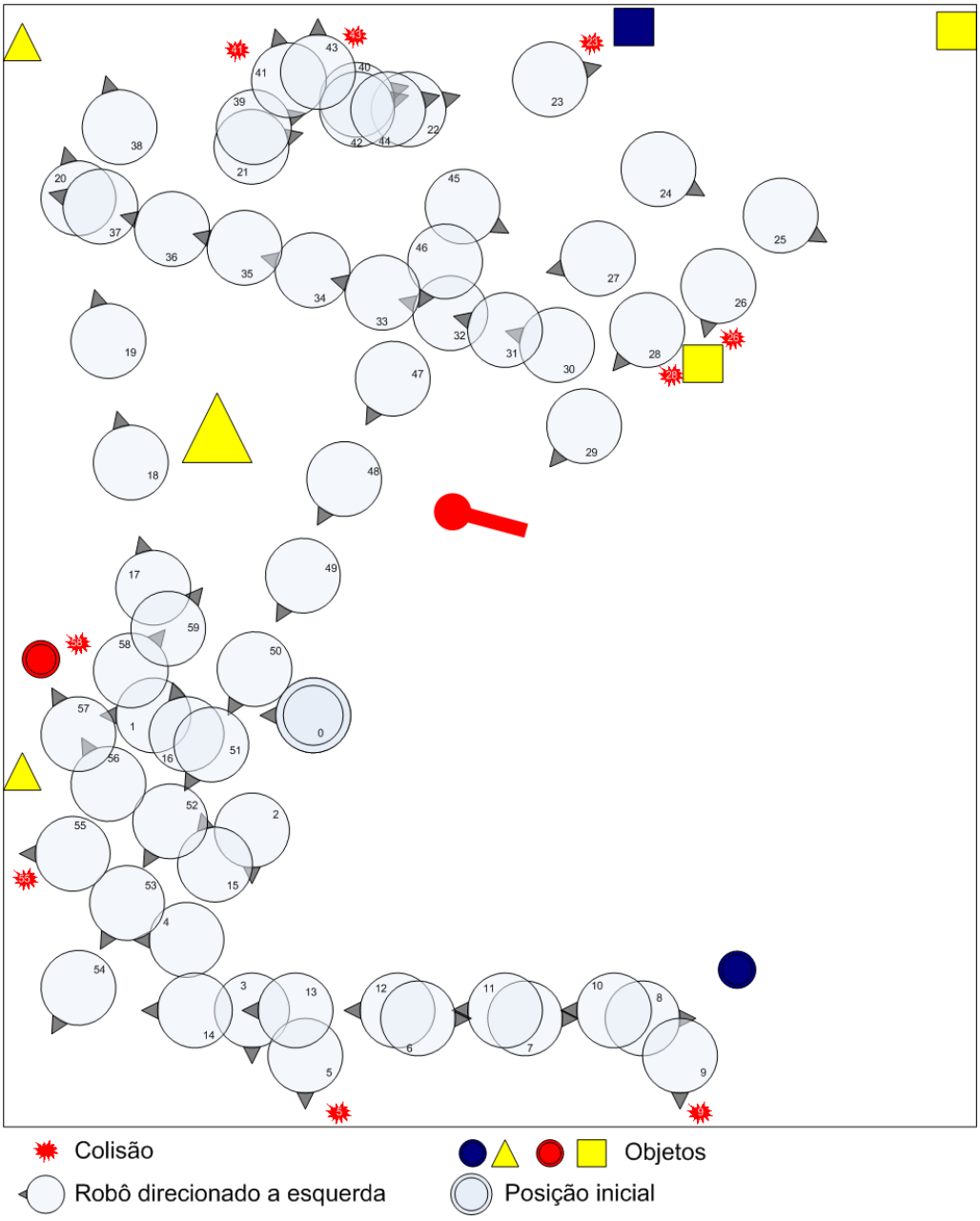


FIGURA 5.15 - Descrição das trajetórias executadas pelo robô, num ambiente de 310 x 260 cm. A direção do robô é indicada por uma seta preta.

5.2.6 Análise de Taxa de Retorno de Recompensas

A Tabela 5.7 sumariza o retorno de recompensas recebidas pelo agente nos 5 ensaios. Somente os 16 experimentos, considerados válidos para análise, estão relacionados. Nota-se que a partir do experimento C3, a quantidade de punições começa a cair quando comparada a quantidade de premiações. Apenas no último experimento E6 essa relação volta a ficar um pouco desequilibrada. Ainda assim, através do gráfico da

(a) O robô no passo E5:22 percebe o estado 57, até então desconhecido, e sorteia a ação *frente*, que o leva a colisão

(b) Colisão no passo E5:23.

(c) O robô no passo E6:12 percebe o estado 57, que desta vez é conhecido, e então toma a ação *direita*.

(d) robô evita a colisão no passo E6:13, desviando do objeto a sua frente.

FIGURA 5.16 - Demonstração de aprendizado.

Figura 5.17, que relaciona o valor acumulado de recompensas em função do número de passos, observa-se que a taxa de premiações tende a crescer a medida que o aprendizado evolui. A região de cada ensaio está indicada no gráfico da Figura 5.17 através de linhas verticais e letras, na qual percebe-se que a melhor taxa de retorno foi obtida no ensaio E, com pequena queda no último experimento.

5.2.7 Análise de Tempo Gasto pelo Agente

Esta secção visa discutir o tempo de processamento e atuação do agente, gastos em cada passo. A Tabela 5.8 apresenta o tempo médio dos passos, agrupados em experimentos. Os valores, expressos em segundos, estão separados em 3 conjuntos. A coluna *Câmera* representa o tempo necessário para o dispositivo *Snappy* ® capturar e digitalizar a imagem. A coluna *Sensor* descreve o tempo total gasto pelo

TABELA 5.7 - Recompensas.

Ensaio	Exp.	N° Passos	Prêmios	Punição
A	0	59	25	7
B	0	27	13	2
	4	22	12	2
C	0	31	9	4
	1	18	9	1
	2	31	11	3
	3	52	32	3
D	0	78	53	8
	1	9	5	0
	2	2	2	0
	3	7	7	0
E	1	19	15	1
	2	3	3	0
	3	45	24	5
	5	32	18	2
	6	28	13	7

sub-sistema sensor, ou seja, o tempo de aquisição da imagem somado a etapa de processamento de imagens. Por último, a coluna *T. Passo* exibe a totalização do tempo gasto em cada passo, composto pelos tempos dos sub-sistemas sensor, atuador e agente. Além das médias de tempo, o desvio padrão de cada conjunto de amostras também é apresentado. A última linha da Tabela 5.8 traz a média e o desvio padrão dos tempos coletados nos 477 passos.

À luz desses valores, observa-se que cada passo leva em média $16,63s$, sendo que aproximadamente 80% do tempo é consumido pelo sub-sistema sensor. Assim, um experimento com 50 passos tomaria 13 minutos e 51 segundos para ser executado. Logo, se a cada ação (independentemente da direção) o carrinho se deslocar 30 cm, ao final do experimento o agente terá percorrido 15 metros, a uma velocidade média de $0,018m/s$ (ou $1,8cm/s$).

Ainda assim, vale ressaltar que o tempo de cada passo varia em função da ação escolhida pelo agente. Em média, o tempo de atuação é de $3,4$ segundos, mas pode variar entre 1 e 2 segundos para ação frente (*AC-FRENTE*) e entre 5 a 7 segundos para as ações de mudança de direção (*AC-ESQUERDA* ou *AC-DIREITA*).

Os maiores tempos médios de processamento foram registrados nos experimentos do ensaio B, provavelmente em decorrência de mudanças nas condições do ambiente. É

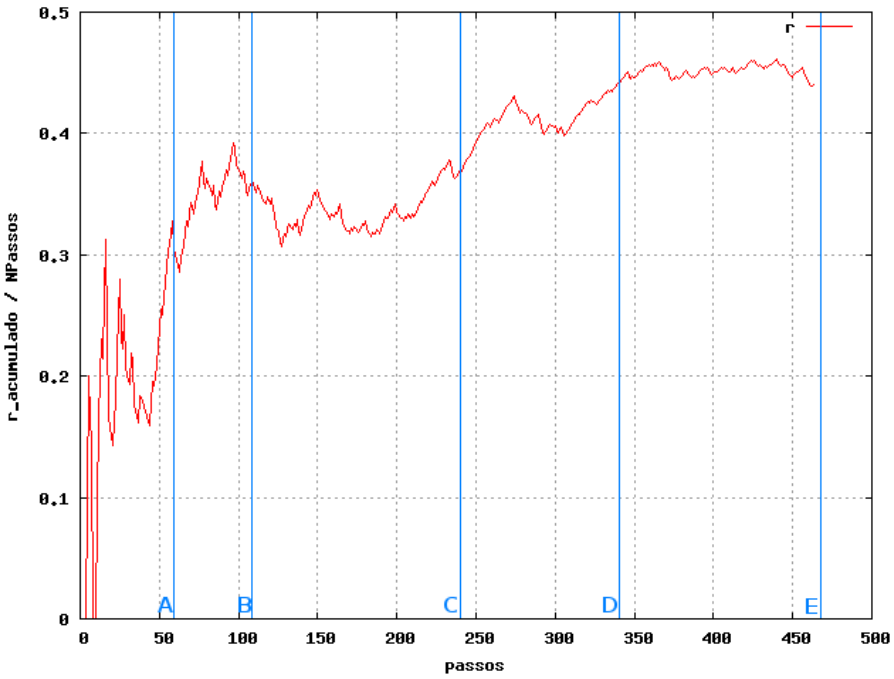


FIGURA 5.17 - Valor acumulado de recompensas recebidas em razão do número de passos.

provável que um dos fatores tenha sido o enfraquecimento da bateria do dispositivo *Snappy* ®, que acarretou em aumento no tempo de digitalização das imagens, como pode ser avaliado nos valores da Tabela 5.8, para o ensaio B.

O tempo mínimo registrado de um passo foi de 8 segundos e o máximo 118. Em casos particulares, o tempo de processamento do agente é elevado, sobretudo quando observa-se o processamento do sub-sistema sensor. Essa etapa de processamento é crítica, pois além de efetuar o processamento da imagem para extração de atributos da imagem, o sistema CANELA efetua processamentos adicionais em cópias da imagem original, para que possam ser exibidas ao operador do experimento. Além disto, os demais processos do sistema operacional estão concorrendo pelo uso do processador. Como no sistema CANELA as medições de tempo não foram coletadas por funções de tempo real, representam médias sujeitas a variações na demanda de processamento computacional. Operações de *swap* de memória, por exemplo, levariam a variações nos tempos de processamento do agente.

Contudo, utilizando-se um sistema computacional dedicado, e reduzindo-se gargalos de captura e digitalização da imagem, acredita-se ser possível reduzir consideravelmente o tempo de processamento do sub-sistema sensor, chegando-se a valores próximos aos apresentados na secção 4.1.2.

TABELA 5.8 - Tempos Médios de aquisição da imagem pela câmera, processamento do sensor e duração de cada passo, agrupados por experimentos.

Exp. ID		**Nº Passos**	**Câmera** (s)		**Sensor** (s)		**T. Passo** (s)	
			μ	σ	μ	σ	μ	σ
A	0	59	3,95	1,15	11,24	5,79	14,59	6,18
B	0	27	9,07	3,04	28,03	13,54	32,33	13,18
	1	4	13,75	1,26	34,00	2,16	34,75	2,63
	2	1	13,00	-	39,00	-	41,00	-
	3	1	12,00	-	38,00	-	38,00	-
	4	22	8,82	1,87	28,73	17,23	34,00	19,98
C	0	31	3,42	0,56	9,94	2,43	14,16	3,61
	1	18	3,39	0,50	8,94	0,64	12,83	2,60
	2	31	3,42	0,50	10,23	3,92	14,29	4,59
	3	52	3,40	0,50	9,13	0,99	12,85	4,33
D	0	78	5,08	3,07	14,23	8,27	16,59	8,35
	1	9	3,44	0,53	8,56	0,72	12,67	3,12
	2	2	3,00	0,00	8,50	0,71	10,00	1,41
	3	7	3,57	1,13	10,43	1,99	13,57	2,99
E	0	6	5,50	3,14	14,33	1,06	17,00	5,06
	1	19	5,42	0,51	11,39	1,39	13,32	2,50
	2	3	5,00	0,00	9,33	0,58	10,67	0,56
	3	45	4,91	0,29	9,56	0,78	12,82	2,66
	4	2	5,00	0,00	9,50	0,71	10,50	0,71
	5	32	5,00	0,51	11,66	4,60	15,22	7,37
	6	28	5,43	1,35	12,79	4,50	16,36	5,93
Geral		477	4,94	2,46	13,23	8,91	16,63	9,46

5.3 Navegação Visando a Exploração de Ambientes Desconhecidos

Os experimentos da secção anterior alcançaram os seus objetivos de guiar o robô, evitando os obstáculos pelo caminho. Contudo, observou-se que as trajetórias do robô ficaram um tanto quando restritas, limitando-se a algumas área do ambiente. Visando encontrar um mecanismo de exploração homogênea do terreno, inicialmente desconhecido, esta secção propõem uma modificação no modelo de aprendizado por reforço do agente.

O objetivo é induzir o robô a uma navegação que explore o ambiente de forma homogênea, primando por regiões pouco visitadas do terreno, ao mesmo tempo que evita colisões. Para isto, um mapa nebuloso-cartesiano será utilizado pelo agente para registrar suas ações no ambiente. A seguir, a modelagem do mapa nebuloso-cartesiano e do agente serão descritas, bem como os experimentos realizados em ambientes simulados no sistema Canela.

5.3.1 Modelagem do Mapa Nebuloso-cartesiano

O mapa nebuloso-cartesiano é utilizado pelo agente para demarcar as regiões visitadas no terreno explorado, que posteriormente o ajudará a escolher suas ações. O modelo de mapa nebuloso adotado nestes experimentos é composto de 2 variáveis lingüísticas: eixo de *coordenadas* x e y. Assumindo que o agente move-se sempre numa distância constante dst, optou-se em delimitar a área do ambiente a ser explorada em $25dst \times 25dst$ unidades, sendo os universos de discurso discretizados da seguinte forma: $\Omega_x \in [0, 24dst]$ e $\Omega_y \in [0, 24dst]$. Cada uma das variáveis lingüísticas contém 5 termos lingüísticos, $E(x) = \{O, OC, CX, LC, L\}$ e $E(y) = \{N, NC, CY, SC, S\}$, fazendo alusão aos pontos cardeais, *Norte*, *Sul*, *Leste* e *Oeste*, e C denotando *Centro*. De forma empírica, os conjuntos nebulosos que representam os termos lingüísticos foram definidos pelas funções

$$\tilde{O} = L(u_x, 2, 7); \tag{5.1}$$

$$\tilde{OC} = \Delta(u_x, 2, 7, 12); \tag{5.2}$$

$$\tilde{CX} = \Delta(u_x, 7, 12, 17); \tag{5.3}$$

$$\tilde{LC} = \Delta(u_x, 12, 17, 22); \tag{5.4}$$

$$\tilde{L} = \Gamma(u_x, 17, 22); \tag{5.5}$$

$$\tilde{S} = L(u_y, 2, 7); \tag{5.6}$$

$$\tilde{SC} = \Delta(u_y, 2, 7, 12); \tag{5.7}$$

$$\tilde{CY} = \Delta(u_y, 7, 12, 17); \tag{5.8}$$

$$\tilde{NC} = \Delta(u_y, 12, 17, 22); e \tag{5.9}$$

$$\tilde{N} = \Gamma(u_y, 17, 22). \tag{5.10}$$

Define-se uma *região* $g(u_x, u_y)$ do terreno, como sendo uma região representada por um par de termos lingüísticos (gX, gY), tal que $u_x \in \Omega_x, u_y \in \Omega_y, gX \in E(x)$ e $gY \in E(y)$. Devido a configuração dos conjuntos nebulosos e a característica discreta do universo de discurso, sempre existirá uma região $gM(u_x, u_y)$ que melhor represente a posição do robô, ou seja, que contenha os conjuntos nebulosos com maior valor de pertinência $gX = \arg\max_{gx \in E(x)} \tilde{gx}(u_x)$ e $gY = \arg\max_{gy \in E(y)} \tilde{gy}(u_y)$. A Figura 5.18 ilustra a modelagem do mapa nebuloso-cartesiano, indicando as 25 regiões predominantes do terreno.

O *nível de visitação* $(iNVt(g))$ de uma dada região do terreno, em um instante de tempo t (passo do agente), é definido pelo produto dos valores de pertinência dos conjuntos nebulosos que representam aquela região, ou seja, $g\tilde{X}(u_x) \times g\tilde{Y}(u_y)$.

Por exemplo, o objeto representado pelo círculo duplo na Figura 5.18, está posicionado na coordenada cartesiana $(11, 10)$. Os seguintes valores de pertinência são obtidos nos conjuntos nebulosos: $\tilde{OC}(11) = 0,2$, $\tilde{CX}(11) = 0,8$, $\tilde{CY}(10) = 0,6$ e $\tilde{SC}(10) = 0,4$. Com isso, quatro regiões com $iNVt > 0$ são identificadas: $iNVt(OC, CY) = 0,12$, $iNVt(OC, SC) = 0,08$, $iNVt(CX, CY) = 0,48$ e $iNVt(CX, SC) = 0,32$. A melhor região que representa a posição do objeto no mapa nebuloso-cartesiano é (CX, CY).

Fica claro então que a cada passo dado pelo agente, todas as regiões do mapa estão sendo visitadas, proporcionalmente ao valor de pertencimento dos seus conjuntos nebulosos. Com esta característica, sempre existirá regiões menos visitadas e mais visitadas do terreno, o que auxiliará o agente na escolha de suas ações. O método de contagem do número de visitas de uma determinada região, bem como outros indicadores utilizados na avaliação de resultados de navegação do robô, são descritos no próximo item.

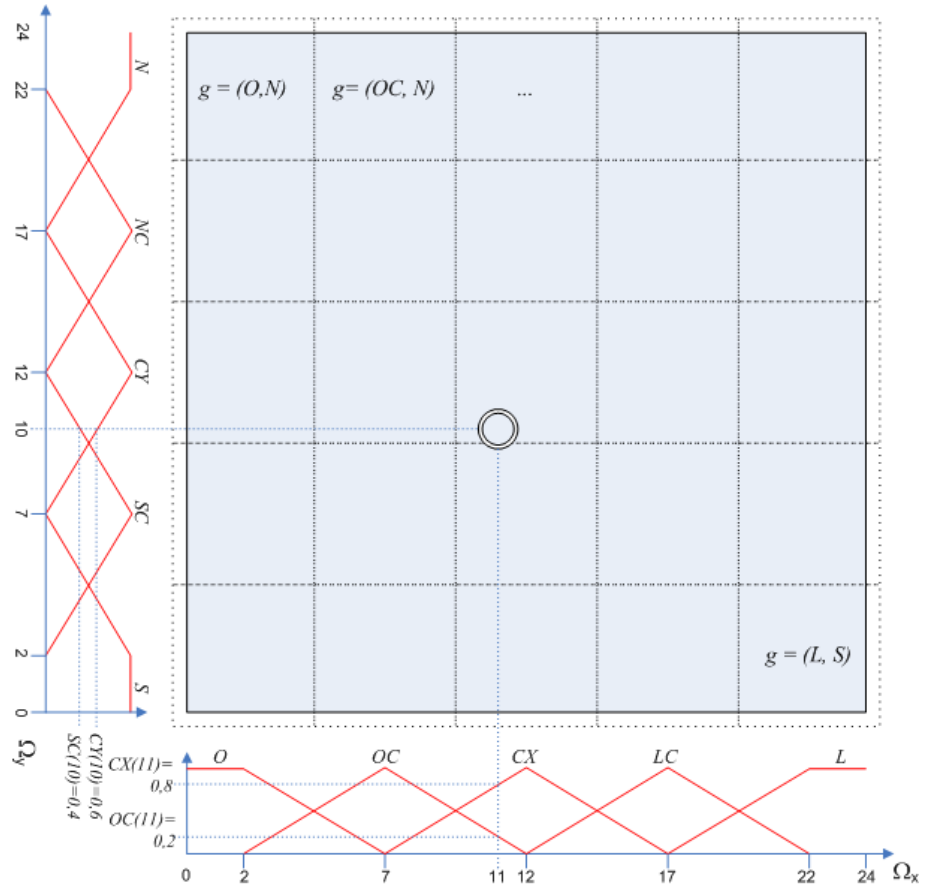


FIGURA 5.18 - Representação da modelagem do mapa nebuloso-cartesiano e exemplificação da leitura da posição de um obstáculo.

5.3.2 Indicadores para Avaliação dos Resultados

Para auxiliar o processo de avaliação dos resultados obtidos nos experimentos desta secção, a seguir serão definidos alguns indicadores específicos. Esses indicadores representam as métricas e a linguagem comum para a quantificação do desempenho do sistema de navegação do agente:

- **Número de Visitas** (iVt): é a quantidade de passeios efetuados pelo agente em uma dada região $g(u_x, u_y)$ do terreno explorado, num instante de tempo t.

$$iVt(g) = \sum_{i=1}^{t} iNVt(g)_i. \tag{5.11}$$

  Representa a soma de todas marcações efetuadas pelo agente em seu mapa nebuloso-cartesiano em g.

- **Média de Visitação do Terreno** ($iMdVt$): é o número médio de visi-

tação de um terreno, no instante t

$$iMdVt_t = \frac{\sum_{i=1}^{|E(x)|.|E(y)|} iVt(g_i)}{|E(x)|.|E(y)|}. \tag{5.12}$$

Todavia, em razão da modelagem do mapa nebuloso-cartesiano, pode ser calculada simplesmente por $iMdVt_t = \frac{t}{25}$.

- **Distância de Visitação ($iDstVt$)**: representa a homogeneidade de visitação do terreno. É calculada pelo desvio padrão [1] do número de visitas de cada região, normalizado pela Média de Visitação do Terreno

$$iDstVt_t = \sqrt{\frac{1}{|E(x)|.|E(y)|} \sum_{i=1}^{|E(x)|.|E(y)|} \left(\frac{iVt_t(g_i)}{iMdVt_t} - 1\right)^2}. \tag{5.13}$$

Quanto maior a distância, maior a heterogeneidade no número de visitas das regiões.

- **Taxa de Visitação ($iTxVt$)**: compara o quão distante o número de visitas de uma dada região, em t, está da região mais visitada

$$iTxVt_t\,(g) = \frac{iVt_t\,(g)}{\max_{h \in g} iVt_t(h)}. \tag{5.14}$$

- **Mapa de Visitação**: representação gráfica de todas Taxas de Visitação pertencentes a um mapa nebuloso-cartesiano. Útil para visualização da evolução da navegação do agente e do seu aprendizado em tempo real.

- **Índice de Visitação ($iInVt$)**: indica a média de visitação de um terreno, tomando como referência o valor de visitação da região mais visitada

$$iInVt_t = \frac{1}{|E(x)|.|E(y)|} \sum_{i=1}^{|E(x)|.|E(y)|} \left(\frac{iVt_t(g_i)}{\max_{h \in g} iVt_t(h)}\right). \tag{5.15}$$

Assim, quanto mais próximo de 1 esse indicador estiver, maior o número de regiões com visitação próxima ao máximo.

[1] A rigor, no cálculo do desvio padrão da amostra, a variância tem como divisor o número de amostras menos 1.

5.3.3 Modelagem do Agente

O agente móvel deve explorar o ambiente, inicialmente desconhecido, tomando a cada instante de tempo t, apenas uma das seguintes **ações**: deslocar-se à frente a uma distância *dst* (*AC-FRENTE*), rotacionar 90° à esquerda (*AC-ESQUERDA*) ou 90° à direita (*AC-DIREITA*).

Através dos sensores e suas variáveis internas, o agente obtém um conjunto de percepções que serão utilizadas por uma função que mapeia os seus **estados**. Cada estado S é definido por uma quádrupla $S = (s^1, s^2, s^3, s^4)$, onde s^1, s^2, s^3 e s^4 são símbolos que representam a situação da região adjacente à esquerda, à frente, à direita e atrás, respectivamente. Isto significa que o estado é composto por informações da vizinhança 4 da região atual em que o agente está no mapa. Os símbolos s^i podem assumir os diferentes símbolos apresentados na Tabela 5.9.

TABELA 5.9 - Símbolos que podem compor a codificação de um estado do agente.

ID	Símbolo	Ocasião de escolha
0	x	se existe um obstáculo imediato na direção da região adjacente
1	:	se existe um obstáculo não imediato na direção da região adjacente, mas sua distância é menor do que a distância até a região adjacente
2	*	se a região adjacente é a menos explorada da vizinhança
3	~	se a região adjacente é a mais explorada da vizinhança
4	|	se a região adjacente pertence ao limite do mapa, ou seja, existe uma parede no limite externo da região vizinha
5	.	caso nenhuma das opções anteriores se adeqüar

A Figura 5.19 exemplifica o mapeamento dos dados sensoriais do agente para o seu estado atual S_t. O agente, posicionado na região (S, OC) e representado pelo círculo cinza, está apontado para o Leste (seta preta no agente). Tem como regiões adjacentes: à frente (S, CX), à esquerda (SC, OC), atrás (S, O). À direita não existe vizinhança, logo o símbolo $s^3 = 4$. Existe um obstáculo (região escura no mapa), imediatamente à esquerda do agente, assim o símbolo $s^1 = 0$. Assumindo que o agente vem da região (S, O) e esta é a mais visitada da vizinhança, então $s^4 = 3$. Se, apontando para a região (S, CX) percebe que a próxima é a menos visitada, o seu estado atual finalmente é definido como $S_t = (0, 2, 4, 3)$ ou $S_t = (x * |\tilde{})$.

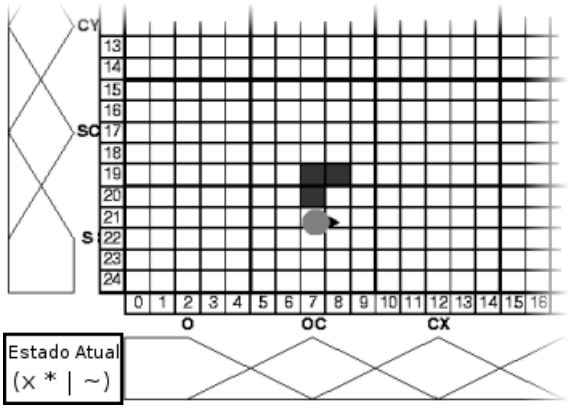


FIGURA 5.19 - Ilustração da posição de um agente no seu mapa nebuloso-cartesiano e a respectiva codificação do seu estado. A região escura representa obstáculos e o círculo cinza a posição do agente, com a seta preta indicando sua direção.

Para completar a modelagem do agente de aprendizagem, define-se sua função de **reforço**

$$\mathfrak{r} = \begin{cases} -1,0 & \text{se o agente colidiu} \\ -0,2 + PA + PR & \text{caso contrário} \end{cases}, \tag{5.16}$$

onde, PA é um Prêmio por Afastamento e PR um Prêmio por Região.

O objetivo imediato da Equação 5.16 é punir o agente caso ele venha colidir com algum obstáculo. Se os termos PA e PR forem nulos, o agente terá uma pequena punição, ou seja, toda ação será punida com uma perda de energia. O termo

$$PA = \begin{cases} +0,25 & \text{se } s^4_{t-1} = 3 \text{ e a ação foi } AC\text{-}FRENTE \\ 0 & \text{caso contrário} \end{cases} \tag{5.17}$$

é uma pequena premiação caso o agente se afaste da região adjacente mais visitada.

Já o termo

$$PR = \begin{cases} extra + g\tilde{X}(u_x)g\tilde{Y}(u_y) - iTxVt(gM) & \text{se } gM_t \neq gM_{t-1} | (gX, gY) \in gM \\ 0,0 & \text{caso contrário} \end{cases} \tag{5.18}$$

é a parte da recompensa responsável por induzir o agente a navegar por regiões g pouco exploradas. Ela só é ativada quando o agente muda de região predominante no mapa. A premiação consiste no valor de pertencimento da nova região explorada, em função da localização atual do agente. Soma-se a isto um prêmio extra e subtrai-se um desconto, que indica como a região atual vem sendo explorada em relação as demais regiões do mapa.

A premiação extra

$$extra = \begin{cases} 2,0 & \text{se } s_{t-1}^2 = 2 \\ 0,0 & \text{se } s_{t-1}^2 = 3 \\ 0,5 & \text{caso contrário} \end{cases} \tag{5.19}$$

induz o agente a navegar para uma região pouco explorada e o inibe a navegar para uma região muito explorada.

Para treinamento do algoritmo *Q-learning*, foram utilizados os seguintes valores: taxa de aprendizado $\alpha = 0{,}1$; fator de desconto $\gamma = 0{,}8$; número de ações tomadas pelo agente = 1 milhão; e probabilidade de escolha aleatória de ações $\epsilon - greedy$ inicialmente fixado em 0,9, decaindo linearmente a partir do passo de número 200 mil, até ser zerada.

5.3.4 Um Ambiente Simples sem Obstáculos

A fim de validar o modelo proposto, a seguir são apresentados alguns experimentos realizados com o agente de aprendizado, num ambiente simulado. Inicialmente, a primeira simulação é realizada num ambiente onde não existem obstáculos, além daqueles que delimitam o mapa. Para comparar a estratégia de navegação baseada em aprendizado por reforço, experimentos com uma política de escolha de ações aleatória (ϵ-*greedy*= 1) são efetuados (algoritmo *Randômico*). Foram realizadas 10 simulações, de 10 mil passos, com o algoritmo *Q-learning* já treinado em 150 mil passos, reposicionando o agente aleatoriamente a cada nova simulação.

As curvas médias de Distância de Visitação do algoritmo *Q-learning* e da política aleatória, estão ilustradas no gráfico da Figura 5.20. Observa-se que o algoritmo *Q-learning* converge rapidamente para um *iDstVt* inferior, permanecendo estável ao longo das iterações. Ainda na Figura 5.20, nota-se a curva de convergência do *iDstVt* para a política aleatória, ao longo de 1 milhão de passos.

5.3.5 Ambientes com Obstáculos

Com a inserção de obstáculos no ambiente, a navegação se torna mais complexa, exigindo do agente a capacidade de evitar colisões, enquanto vagueia pelo terreno. Nos experimentos simulados, foram utilizados três ambientes com obstáculos, na forma de labirintos, ilustrados na Figura 5.21. Os obstáculos estão representados em preto, na grade do mapa do ambiente.

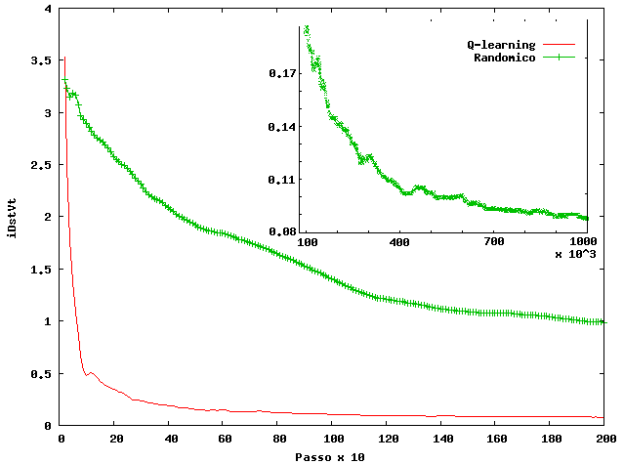
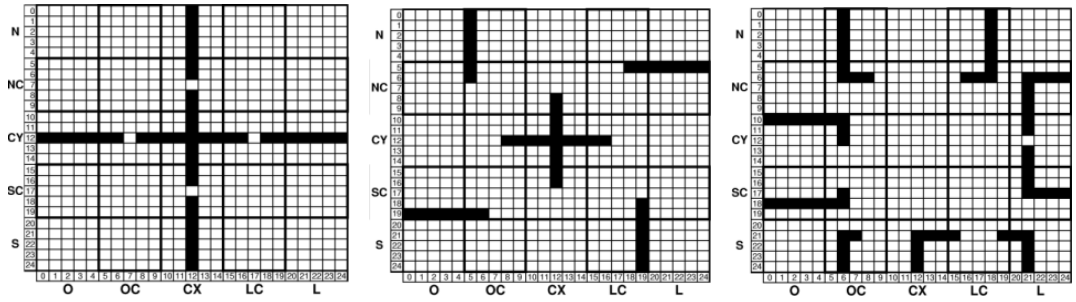


FIGURA 5.20 - Curvas das médias de *iDstVt* obtidas pelo algoritmo *Q-learning* e navegação Aleatória, em função do número de passos, em ambiente sem obstáculos.

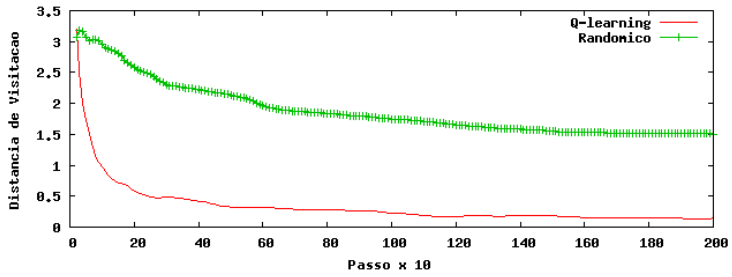


(a) Labirinto Cruz. (b) Labirinto Cata-vento. (c) Labirinto Escritório.

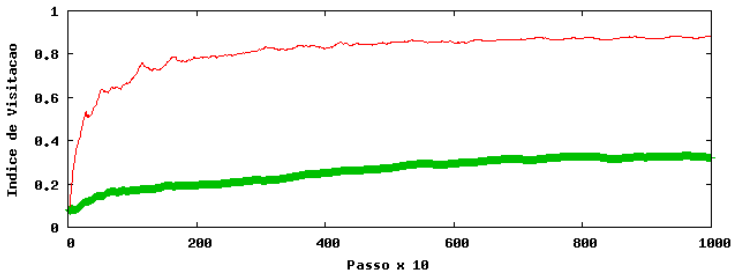
FIGURA 5.21 - Representação dos ambientes de labirintos utilizados nos experimentos de simulação.

Utilizando o labirinto *Cruz*, o algoritmo *Q-learning* foi novamente treinado em 150 mil passos (com ϵ-*greedy*$= 1,0$), e em seguida o desempenho da navegação do agente foi avaliado em 10 mil passos (com ϵ-*greedy*$= 0$). A Figura 5.22 traz os gráficos das curvas das médias dos indicadores *iDstVt* e *iInVt*, nos 2 mil e 10 mil passos inicias do experimento (as curvas sofrem pouca variações no restante dos passos). Nota-se, através do indicador *iDstVt*, que ambos algoritmos têm uma maior dificuldade em manter uma cobertura homogênea do terreno, mas o algoritmo baseado em aprendizado por reforço obtém uma melhor convergência. Essa homogeneidade é confirmada pelo indicador *iInVt*.

Simulações com o labirinto *Cata-vento* foram efetuadas a fim de se observar a generalização no aprendizado do algoritmo *Q-learning*. Para tanto, o algoritmo de aprendizado por reforço foi treinado no labirinto *Escritório* (com ϵ-*greedy*$= 1,0$), mas avaliado no labirinto *Cata-vento* (com ϵ-*greedy*$= 0$).



(a) Curvas médias do indicador *Distância de Visitação*.



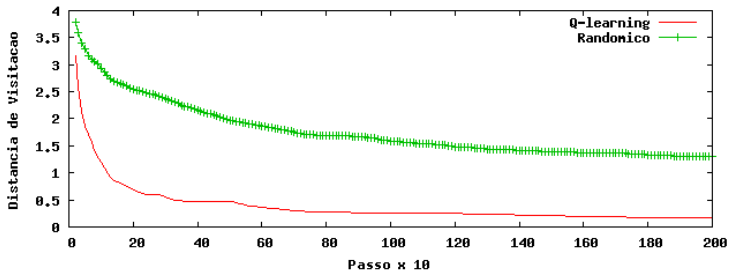
(b) Curvas médias do indicador *Índice de Visitação*.

FIGURA 5.22 - Curvas das médias de iDstVt e iInVt obtidas pelo algoritmo *Q-learning* e navegação Aleatória, em função do número de passos, no labirinto *Cruz*.

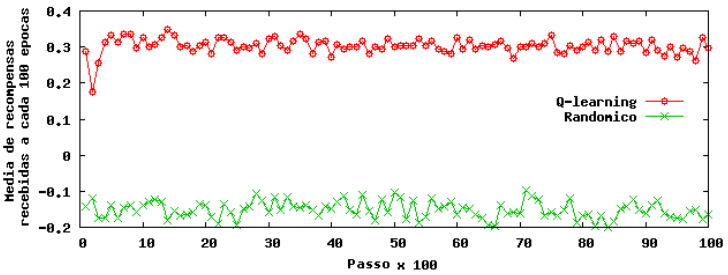
O primeiro gráfico, na Figura 5.23-a, demonstra a capacidade de navegação do agente de aprendizagem, através da convergência do indicador *iDstVt* em relação a estratégia de navegação aleatória. O segundo gráfico, na Figura 5.23-b, indica a média de recompensas coletadas a cada 100 ações tomadas na simulação. Neste último gráfico é possível observar a otimização da navegação aprendida pelo algoritmo *Q-learning*, segundo a função de recompensas estabelecida.

Não é incomum que a estratégia adotada na navegação entre em conflito com as ações tomadas para evitar colisões. Este fato foi observado durante experimentos de navegação, com ϵ-*greedy*$= 0$, no labirinto *Escritório*. O resultado é que o agente fica preso a uma determinada região do terreno e não consegue, por conta do seu mapeamento de ações, navegar adequadamente. Isto ocorre, porque mesmo em ambientes para o qual o agente foi treinado, existem situações em que o mapeamento de dados sensoriais para estados-ações é ambíguo (existe *perceptual aliasing*), como já foi apontado por outros autores (CROOK; HAYES, 2003). Todavia, devido a convergência do algoritmo *Q- learning*, a ação tomada será aquela que maximiza o recebimento de recompensas, o que pode levar o agente a uma situação que limitará sua navegação.

Para minimizar esse problema, propôs-se uma modificação na arquitetura do agente, que será descrita a seguir.



(a) Curvas médias do indicador *Distância de Visitação*.



(b) Curvas médias de retorno de recompensas.

FIGURA 5.23 - Curvas das médias de *iDstVt* e Recompensas obtidas pelo algoritmo *Q- learning* e navegação Aleatória, em função do número de passos, no labirinto *Cata-vento*.

5.3.6 Nova Modelagem da Arquitetura do Agente

Uma abordagem para solucionar o problema de ambigüidade na percepção do agente, seria a implementação de um mecanismo de memória de estados e ações tomadas. Todavia essa estratégia poderia levar a um crescimento direto no número de estados do agente e conseqüentemente aumento na complexidade do aprendizado.

Optou-se então em definir uma nova arquitetura hierárquica de controle para o agente de aprendizagem, composta por três módulos:

- Módulo de **anti-colisão**, um sub-sistema de aprendizado por reforço, específico para evitar colisões de objetos próximos;
- Módulo de **navegação**, idêntico a modelagem que foi utilizada até o momento;
- Módulo de **detecção de armadilhas**, que contém variáveis de estado para monitorar o mapa nebuloso-cartesiano e identificar possíveis armadilhas.

O módulo de navegação continua encarregado em selecionar as ações a serem tomadas, mas pode ser censurado pelo módulo de anti-colisão, caso venha a escolher uma ação que leve a uma colisão.

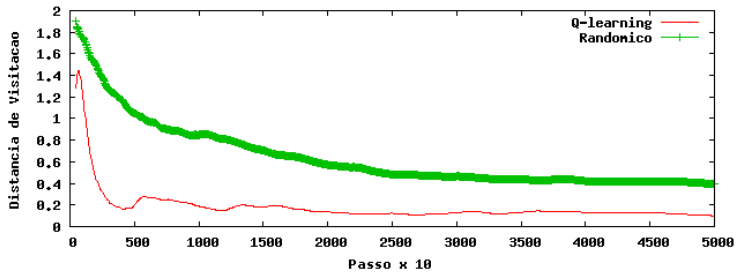
O módulo de detecção de armadilhas, por sua vez, aumenta o coeficiente ϵ-*greedy*, sempre que uma situação de armadilha for detectada. Essa detecção é feita através do monitoramento do indicador $iDstVt$ em função do tempo. Para tanto, o agente armazena duas variáveis internas:

$$iDstVtD_t = iDstVt_t - iDstVt_{t-1}, e \tag{5.20}$$

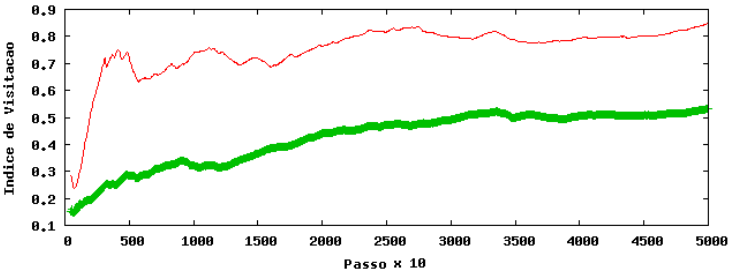
$$iDstVtI_t = iDstVtI_{t-1} + iDstVt_t - SetPoint, \tag{5.21}$$

onde, $SetPoint$ é uma constante indicadora de $iDstVt$ ideal. Sempre que essas variáveis ultrapassarem um determinado limiar, o coeficiente ϵ-*greedy* é incrementado, caso contrário decrementado.

Com essas medidas, o agente foi capaz de escapar de armadilhas, e continuar a navegar pelo labirinto *Escritório* de forma homogênea. Esse comportamento é observado nos gráficos da Figura 5.24, no qual os indicadores de Distância de Visitação e Índice de Visitação convergiram de maneira semelhante aos experimentos anteriores.



(a) Curvas médias do indicador *Distância de Visitação*.



(b) Curvas médias do indicador *Índice de Visitação*.

FIGURA 5.24 - Curvas das médias de iDstVt e iInVt obtidas pelo algoritmo *Q-learning* e navegação Aleatória, em função do número de passos, no labirinto *Escritório*.

Para comparar a homogeneidade na cobertura de navegação do agente, a Figura 5.25 ilustra os respectivos Mapa de Visitação médio, em que as regiões escuras correspondem àquelas pouco visitas em relação a região mais visitada. Conclui-se que o agente de aprendizado obteve uma navegação média mais homogênea, em 10 simulações realizadas.

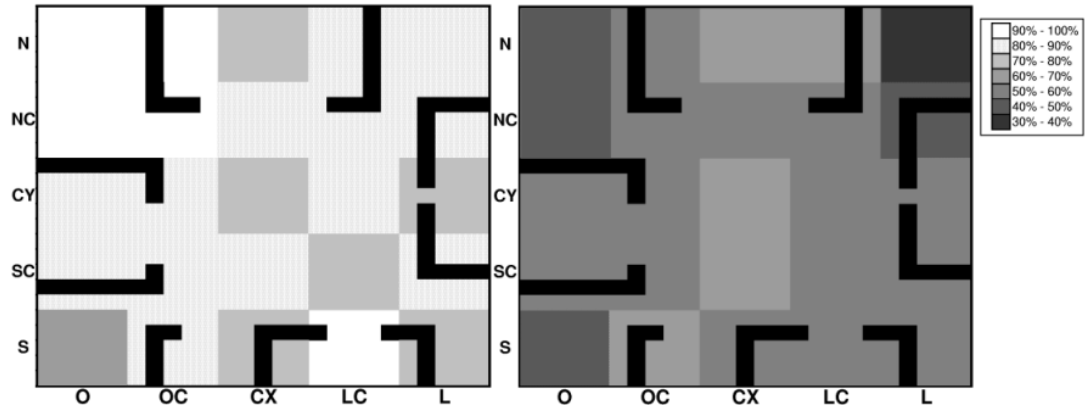


FIGURA 5.25 - Mapas de Visitação médio obtidos no labirinto *Escritório*, após 10 mil épocas simuladas com o algoritmo *Q-learning* e navegação Aleatória, respectivamente.

Os experimentos de navegação, apresentados nesta secção, mostram a viabilidade de uso da técnica de aprendizado por reforço na navegação de robôs móveis, visando a exploração homogênea de um terreno desconhecido. Os resultados foram quantificados em termos de indicadores, descritos neste trabalho, que indicaram o melhor desempenho da navegação baseada no algoritmo *Q-learning*, quando comparado a uma estratégia puramente aleatória de escolha de ações. Essa capacidade de orientar a navegação do agente se manteve mesmo em ambientes sem prévio treinamento do agente, o que é particularmente interessante para exploração de terrenos desconhecidos e inacessíveis. A Tabela 5.10 confirma o bom desempenho do algoritmo *Q-learning*, especialmente em ambientes com obstáculos, através dos valores mínimos das médias de Distância de Visitação e máximos das médias de Índice de Visitação, obtidos nos experimentos.

Por outro lado, dependendo da configuração do ambiente, o agente pode ter dificuldade para escapar de armadilhas, devido a disposição dos obstáculos. Sendo assim, no futuro, experimentos adicionais devem ser realizados, com outras configurações de labirintos, comparando-se o modelo de navegação com outras técnicas.

TABELA 5.10 - Valores mínimos das médias de Distância de Visitação e máximos das médias de Índice de Visitação obtidos pelos algoritmos de navegação em vários ambientes.

Labirinto	**Indicador**	***Q-learning***	**Aleatório**
Vazio	$\min(iDstVt)$	0,067	0,087
	$\max(iInVt)$	0,926	0,825
Cruz	$\min(iDstVt)$	0,099	0,935
	$\max(iInVt)$	0,883	0,330
Cata-vento	$\min(iDstVt)$	0,073	0,604
	$\max(iInVt)$	0,868	0,393
Escritório	$\min(iDstVt)$	0,100	0,395
	$\max(iInVt)$	0,847	0,532

CAPÍTULO 6

CONCLUSÕES

Este trabalho apresentou um estudo e modelagem de um agente de aprendizagem, aplicado a navegação em robótica móvel. Visando dotar o robô com capacidade de aprender autonomamente a navegar em ambientes não estruturados, técnicas de aprendizagem por reforço foram empregadas na implementação do programa do agente.

Para viabilizar a experimentação em um ambiente real e avaliar a tolerância a ruídos do agente, um robô móvel de baixo custo e precisão foi utilizado, equipado com uma câmera CCD que funcionou como sensor.

Os experimentos realizados cobriram a integração de um sistema completo de navegação robótica autônoma, através da montagem de sensores, robô móvel e atuadores, sistemas de comunicação, algoritmos operacionais e de aprendizagem, inseridos em um ambiente interno não estruturado.

Os resultados obtidos demonstraram a capacidade de aprendizado do agente, que partindo sem nenhum conhecimento prévio do ambiente no qual estava inserido, e sem saber quais ações eram boas ou ruins, pôde gradativamente otimizar seu processo de tomada de decisão e navegar livremente pelo terreno. Devido as limitações dos atuadores, não foi possível ao robô evitar, em todo instante, a colisão com obstáculos no ambiente. Todavia é importante ressaltar que o agente se adaptou a este fato, e encontrou soluções para se recuperar de situações em que o robô ficava preso, e prosseguir com a navegação.

De forma semelhante, os ruídos do sensor, na maioria das vezes provenientes de mudanças nas condições de iluminação, foram até um certo nível tolerados pelos operadores de visão computacional, que utilizaram Redes Neurais Artificiais. Nas ocasiões em que as distorções foram grandes, como por exemplo a apresentada no Ensaio D, apesar de não se sido detectado o estado de colisão, o agente foi capaz de retomar a trajetória. Além disto, caso essa falha persistisse, o agente estaria adaptado para evitá-la. Este comportamento é particularmente interessante em missões, nas quais ocorram falhas de componentes, e o sistema de controle adapta-se as novas condições.

O algoritmo *Q-learning*, que traz vantagens teóricas sobre alguns algoritmos de aprendizagem por reforço, se mostrou adequado para a tarefa. Contudo, ficou claro que a convergência dos *Q-valores* é lenta e nem sempre as ações do robô foram ótimas. Assim, um método para aceleração do aprendizado poderia ser avaliado em experimentos futuros. Uma das opções é a construção de um simulador para treinamento em *off-line*, que seja suficientemente fiel na representação do ambiente real. Outra abordagem a ser testada é o uso de generalização no aprendizado por reforço, conforme discutido na secção 2.3.6, que apoiado pelo treinamento simulado, poderá suportar um modelo sensorial com um maior número de estados. Talvez com isso, fosse possível acrescentar à modelagem dos estados um sistema de memória, no qual o agente pudesse manter um controle de suas ações, para evitar ciclos de ações. Neste caso, por exemplo, se o robô tomou a ação *AC-FRENTE* e bateu, em seguida escolheu virar à direita, não deveria tomar a ação *AC-ESQUERDA* (o que provavelmente lhe levaria a um estado ruim, recentemente visitado).

A modelagem permitiu ao agente guiar em um ambiente desconhecido, em contraste aos experimentos realizados sobre uma pista, apresentados na introdução deste trabalho. Com a elaboração de técnicas de chaveamento de controladores, seria possível então combinar os dois tipos de navegadores, com os quais o robô vagaria pelo ambiente não estruturado até encontrar uma pista para guiar.

Embora o objetivo tenha sido atingido, no que tange navegar equipado com sensores e atuadores ruidosos, evitando obstáculos, alguns pontos devem ser trabalhados para se alcançar melhores resultados. O primeiro refere-se ao fato operacional de haver um laboratório adequado para a realização dos experimentos de navegação, com o robô móvel. Isto significa ter um ambiente no qual a iluminação é controlada (até mesmo para se introduzir ruído controlado), com dimensões suficientes para o uso de obstáculos com tamanhos diversos e se possível com irregularidades no terreno. Alguns equipamentos auxiliares, tais como estações de trabalho para a condução dos experimentos, suportes para recarregamento de baterias e dispositivos para registro de som e imagem (câmeras fotográficas e filmadoras), facilitam o registro e a repetibilidade dos ensaios.

Outro ponto importante para a melhoria dos resultados é o uso de digitalizadores mais rápidos. O emprego de câmeras CCDs simples não introduziu ruídos ao ponto de comprometer o funcionamento dos operadores de visão computacional. Contudo, o processo de digitalização das imagens, através do dispositivo *Snappy* ®, acarretou

em aumento excessivo no tempo total de processamento do sub-sistema sensor.

Além de operadores de visão computacional, a instalação de sensores de toque (*bumpers*), também traria uma melhora significativa na capacidade sensorial do agente. Fazendo uma analogia, o sistema sensorial humano, que mesmo com visão estéreo, tem no tato uma forte fonte de informação para navegação em ambientes com muitos obstáculos.

Os últimos experimentos conduzidos em ambientes simulados, apresentaram a capacidade de aprendizado do agente para realização de tarefas mais complexas. Nestes ensaios, visando a exploração homogênea de um terreno, a navegação foi auxiliada por um mapa nebuloso-cartesiano, que contabilizava as regiões visitadas pelo agente. Foram experimentados quatro ambientes, na sua maioria com obstáculos, onde a qualidade da navegação foi quantificada em termos de indicadores, descritos neste trabalho. Os indicadores obtidos nas simulações do sistema de navegação com aprendizagem por reforço foram comparados com resultados obtidos por uma estratégia de navegação aleatória.

Notou-se que em todos ambientes simulados, o algoritmo de navegação baseado em aprendizado por reforço obteve os melhores índices de desempenho, demonstrando sua capacidade de orientar a navegação do agente por ambientes inexplorados, de forma homogênea. Além disto, através da simulação da navegação do agente em um ambiente sem prévio treinamento, observou-se a capacidade de adaptação do modelo desenvolvido e do algoritmo *Q-learning*, bem como a viabilidade do seu uso na exploração de ambientes desconhecidos e pouco acessíveis para treinamento.

A modelagem, primando a visitação homogênea de terrenos, pode ser aproveitada em missões de robôs, em que exista a restrição de tempo para exploração, mas deseja-se uma cobertura geral do terreno. Por outro lado, dependendo da configuração do ambiente, o agente terá dificuldades para escapar de armadilhas, devido a disposição dos obstáculos.

Os resultados apresentados, encorajam o uso da modelagem do sistema de navegação de robôs móveis, baseada em aprendizado por reforço, mesmo em se tratando de tarefas complexas, orientadas a multi-objetivos. Ainda que essas tarefas tragam um aumento na complexidade de modelagem e implementação da função de retorno do agente, acredita-se que esta abordagem traga vantagens sobre os métodos de

programação tradicional. Em sistemas de aprendizagem por reforço, ao invés de se tentar prever todas as situações em que o robô virá enfrentar, um mecanismo de aprendizagem e adaptação contínua as condições do meio é implementado.

A construção de sistemas robóticos com essas características de adaptação, pelo menos em alguma parte de sua arquitetura, são interessantes, sobretudo num futuro próximo, quando os robôs móveis deixarão de ser artefatos do meio industrial / científico para se tornar parte do cotidiano da sociedade.

6.1 Trabalhos Futuros

Como pontos de referência no horizonte, a seguir são descritos alguns aspectos a serem abordados para a continuidade dos trabalhos aqui iniciados.

No âmbito operacional, serão buscadas alternativas para implementação dos conceitos descritos, primando pelo uso de sistemas livres. Atualmente, a implementação do sistema CANELA está restrita a ambientes *Windows* ®, principalmente em razão do uso da biblioteca VisSDK e MFC. Utilizando uma câmera CCD compatível com a *Application Program Interface* (API) *Video for Linux*, bibliotecas alternativas para manipulação de imagens, como por exemplo de Faria e Paulo (2005), podem ser testadas. Além disto, todos os códigos fundamentais da arquitetura, já estritos em C++ *American National Standards Institute* (ANSI), serão integrados por classes de interface de usuário em QT (TROLLTECH, 2005). Adicionalmente, parte dos algoritmos poderão ser implementados em *hardware*.

Os seguintes itens serão revistos e consolidados:

- em se tratando de um método promissor para detecção de objetos, o algoritmo baseado em método vetorial deverá ser validado e testado, a fim de se obter um código confiável para aplicações;
- a RNA que classifica os *pixels* da imagem capturada da cena será treinada novamente, com um conjunto de amostras mais abrangente, objetivando uma maior tolerância a ruídos; e
- a construção de um simulador para treinamento do agente, que imite com maior fidelidade as condições do ambiente no qual se deseja realizar os experimentos.

Em relação aos experimentos de exploração homogênea de ambientes, objetiva-se

realizar um número maior de experimentos a fim de se observar o comportamento do agente em situações de armadilhas e aprimorar a estratégia de detecção e escape dessas regiões. Para tanto, a comparação do método de navegação baseado em *Q-learning* com outros algoritmos será imprescindível. Pode-se experimentar abordagens híbridas, combinando as técnicas de aprendizagem com técnicas de controle deliberativo ou algoritmos clássicos de navegação (como por exemplo o A*).

Algumas modificações no uso do mapa nebuloso-cartesiano também serão estudadas. Uma delas será a atualização do indicativo de número de visitas de uma região, considerando a visibilidade daquela região. Neste caso, o agente pode levar em conta a localização de obstáculos, para modificar a função de ativação das variáveis nebulosas do mapa. Chama-se a atenção que com essa abordagem, o método de cálculo do indicador $iMdVt$ deverá ser modificado.

Espera-se avaliar também o uso dos mapas nebuloso-cartesianos em diferentes dimensões de ambientes, buscando-se uma adaptação dinâmica do tamanho do mapa. A medida que algumas área do terreno foram suficientemente exploradas, pode-se alterar a granularidade do mapa e aumentar a área de exploração. Ou de forma similar, para uma pequena região de interesse, modificar o mapa para explorá-la com maior atenção. Com isso, pode-se incorporar ao modelo a idéia de exploração gradual de um ambiente. Inicialmente o agente explora a região que está próxima de si, e tendo um conhecimento suficiente da mesma, amplia-se a área a ser explorada. Nessa abordagem, pode-se manter os índices de visitação para a área já explorada. Esta estratégia é utilizada intuitivamente na exploração de ambientes abertos. Caso o ambiente seja o interior de edificações, pode-se aumentar a abrangência do mapa a medida que novos caminhos são descobertos, ou reduzindo-a a fim de se explorar com mais detalhes uma região nova pouco explorada. Por fim, novas configurações de conjuntos nebulosos, otimizados a um dado tipo de terreno, poderão se avaliadas.

Em casos de aplicação dos mapas apresentados em ambientes reais, é muito provável a ocorrência gradativa de erro na marcação da localização do robô no mapa. Este fato se deve a imprecisão nos atuadores e sensores do agente. Sendo assim, pode-se tirar vantagem da modelagem nebulosa para que no decorrer do mapeamento do ambiente e revisitas de regiões, sejam aplicados algoritmos para determinação da estimativa da localização do robô.

Finalmente, adaptações dos modelos desenvolvidos, visando a navegação do robô móvel em ambientes externos, serão estudadas.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHAM, A. **It is time to fuzzify neural networks: intelligent multimedia, computing and communications - technologies and applications of the future**. Dakota: John Wiley & Sons, 2001. 68

_____. Neuro fuzzy systems: state-of-the-art modeling techniques. In: INTERNATIONAL WORK-CONFERENCE ON ARTIFICIAL AND NATURAL NEURAL NETWORKS, 6., 2001, Granada, Spain. **Proceedings...** Granada, Spain: Springer Berlin / Heidelberg, 2001. p. 269–276. 68

AGAH, A.; TANIE, K. Fuzzy logic controller design utilizing multiple contending software agents. **Fuzzy Sets and Systems**, v. 106, n. 2, p. 121–130, 1999. 68

AHLF, P.; CANTWELL, E.; OSTRACH, L.; PLINE, A. Mars scientific investigations as a precursor for human exploration. **Acta Astronautica**, v. 47, n. 2–9, p. 535–545, 2000. 35

AKBARZADEH-T, M.-R.; KUMBLA, K.; TUNSTEL, E.; JAMSHIDI, M. Soft computing for autonomous robotic systems. **Computers and Electrical Engineering**, v. 26, n. 1, p. 5–32, 2000. 35

ARANIBAR, D. B.; ALSINA, P. J. Reinforcement learning-based path planning for autonomous robots. In: ENCONTRO DE ROBóTICA INTELIGENTE - CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAçãO, 24., 2004, Salvador. **Anais...** Salvador: SBC, 2004. p. 10. ISBN 85-88442-93-0. (1 CD-ROM). 78

ARMSTRONG, W.; COGHLAN, B.; GORODNICHY, D. Reinforcement learning for autonomous robot navigation. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1999, Washington DC. **Proceedings...** Washington DC: IEEE, 1999. v. 4, p. 2282 – 2287. 78

ASADA, M.; NODA, S.; TAWARATSUMIDA, S.; HOSODA, K. Vision-based reinforcement learning for purposive behavior acquisition. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1995, Nagoya, Japan. **Proceedings...** Nagoya, Japan: IEEE, 1995. p. 146–153. 80

_____. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. **Machine Learning**, v. 23, n. 2–3, p. 279–303, 1996. 80

ASADA, M.; UCHIBE, E.; HOSODA, K. Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. **Artificial Intelligence**, v. 110, n. 2, p. 275–292, 1999. 80

ASIMOV, I. **Eu, robô**. Rio de Janeiro: Círculo do Livro, 1950. 294 p. 35

BAXES, G. A. **Digital image processing: principles and applications**. New York, USA: John Wiley & Sons, Inc., 1994. 88

BENTLEY, J. L.; OTTMANN, T. Algorithms for reporting and counting geometric intersections. **IEEE Transactions on Computers**, C-28, n. 9, p. 643–649, 1979. 102, 207

BERENJI, H. Fuzzy Q-Learning: a new approach for fuzzy dynamic programming. In: IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE, 3., 1994, Orlando, USA. **Proceedings...** Orlando, USA: IEEE, 1994. p. 486–491. 81

BERNARD, D. E.; DORAIS, G. A.; GAMBLE, E.; KANEFSKY, B.; KURIEN, J.; MILLAR, W.; MUSCETTOLA, N.; NAYAK, P.; ROUQUETTE, N.; RAJAN, K.; SMITH, B.; TAYLOR, W.; TUNG, Y. W. **Final Report on the Remote Agent Experiement**. Pasadena, 2000. 48 p. 38

BEZDEK, J. C. Editorial: Fuzzy models - what are they, and why? **IEEE Transactions on Fuzzy Systems**, v. 1, n. 1, p. 1–9, 1993. 58

BHANU, B.; LEANG, P.; COWDEN, C.; LIN, Y.; PATTERSON, M. Real-time robot learning. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2001, Seoul, Korea. **Proceedings...** Seoul, Korea: IEEE, 2001. p. 491–498. Disponível em: <citeseer.ist.psu.edu/546875.html>. Acesso em: 11 fev. 2004. 77, 79

BITTENCOURT, G. **Inteligência artificial: ferramentas e teorias**. Florianópolis: Editora da UFSC, 2001. 362 p. 35

BITTENCOURT, J. R.; OSÓRIO, F. S. Processamento de imagens inteligente usando redes neurais artificiais. In: ROCHA, M. A.; CRUZ, T. R. (Ed.). **Aprendizado, criação e integração na Iniciação Científica**. Porto Alegre: Editora da UFRGS, 2002. p. 11–35. 86

BLOCH, I.; SAFFIOTTI, A. On the representation of fuzzy spatial relations in robot maps. In: BOUCHON-MEUNIER, B.; FOULLOY, L.; YAGER, R. (Ed.). **Intelligent Systems for information processing**. Netherland: Elsevier, 2003. p. 47–57. 66

BORENSTEIN, J.; EVERETT, H. R.; FENG, L. **Where am I?** sensors and methods for mobile robot positioning. Ann Arbor, USA: J. Borenstein, 1996. 37

BOURKE, P. **Calculating the area and centroid of a polygon**. 1988. Disponível em: <http://astronomy.swin.edu.au/~pbourke/geometry/polyarea/>. Acesso em: 21 ago. 2004. 107

BRAGA, A. P. de S. **Agente topológico de aprendizado por reforço**. 133 p. Tese (Doutorado) — Universidade de São Paulo, São Carlos, 2004. 77

BRAGA, A. P. S.; ARAÚJO, A. F. R. Applying topological maps to accelerate reinforcement learning in mobile robot navigation. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 2002, Hammamet, Tunísia. **Proceedings...** Hammamet, Tunísia: IEEE, 2002. v. 2, p. 7–12. 77

BROWN, M.; DRUMMOND, T.; CIPOLLA, R. 3D model acquisition by tracking 2D wireframes. In: BRITISH MACHINE VISION CONFERENCE, 11., 2000, Bristol. **Proceedings...** Bristol: University of Bristol, 2000. p. 10. 87

BUIJTENEN, W. M. van; SCHRAM, G.; BABUSKA, R.; VERBRUGGEN, H. B. Adaptive fuzzy control of satellite attitude by reinforcement learning. **IEEE Transactions on Fuzzy Systems**, v. 2, n. 6, p. 185–194, 1998. 81

BUSQUETS, D.; MÀNTARAS, R. L. de; SIERRA, C. Reinforcement learning for landmark-based robot navigation. In: INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 1., 2002, Bologna, Italy. **Proceedings...** Bologna, Italy: ACM Press, 2002. p. 841–842. Disponível em: <citeseer.ist.psu.edu/477978.html>. 79

CÁNOVAS, J.-P.; LEBLANC, K.; SAFFIOTTI, A. Robust multi-robot object localization using fuzzy logic. In: NARDI, D.; RIEDMILLER, M.; SAMMUT, C. (Ed.). **RoboCup 2004: Robot Soccer World Cup VIII**. Berlin: Springer-Verlag, 2004, (LNAI). 68

CAO, J. **Vision techniques and autonomous navigation for an unmanned mobile robot**. Dissertação (Mestrado) — University of Cincinnati, 1999. 68

CARBONELL, J. G.; MICHALSKI, R. S.; MITCHELL, T. M. An overview of machine learning. In: MICHALSKI, R. S.; CARBORELL, J. G.; MITCHELL, T. M. (Ed.). **Machine Learning: an artificial intelligence approach**. San Mateo: Morgan Kaufmann, 1983. 45

CASALS, A. Medical robotics at UPC. **Microprocessors and Microsystems**, v. 23, n. 2, p. 69–74, 1999. 35

CASTELLANO, G.; ATTOLICO, G.; STELLA, E.; DISTANTE, A. **Reactive navigation by fuzzy control**. 1999. Disponível em: <http://citeseer.ist.psu.edu/364443.html>. Acesso em: 21 ago. 2003. 67

CASTRO, A. P. A. **Detecção de bordas e navegação autônoma utilizando redes neurais artificiais.** 154 p. Dissertação (Mestrado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2003. 56

CASTRO, A. P. A.; SILVA, J. D. S.; SIMONI, P. O. Image based autonomous navigation with fuzzy logic control. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2001, Washington, USA. **Proceedings...** Washington: IEEE, 2001. v. 3, p. 2200–2205. 38, 39, 67

CHANG, Y.-C.; LO, R.-C. A study on outdoor guidance of autonomous land vehicle by binocular computer vision based on artificial intelligent policy. In: CONFERENCE ON COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING, 16., 2003, Kinmen, Taiwan. **Proceedings...** Kinman: R.O.C., 2003. p. 733–739. 87

CHIN, T. C.; QI, X. M. Genetic algorithms for learning the rule base of fuzzy logic controller. **Fuzzy Sets and Systems**, v. 97, n. 1, p. 1–7, 1998. 67

CICIRELLI, G.; D'ORAZIO, T.; DISTANTE, A. Visual state recognition for a target-reaching task. In: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 15., 2000, Barcelona, Spain. **Proceedings...** Barcelona, Spain: IEEE, 2000. v. 4, p. 854–857. 86

CLARKE, A. C. **2001: odisséia espacial**. 4. ed. São Paulo: Edibolso, 1975. 227 p. 35

COELHO, L. de S.; CAMPOS, M. F. M. Navegação de dirigíveis autônomos baseada em visão computacional. In: II SEMANA DE PóS-GRADUAçãO EM CIêNCIA DA COMPUTAçãO DA UFMG, 1998, Belo Horizonte. **Proceedings...** Belo Horizonte: UFMG, 1998. p. 7. 108

CROOK, P. A.; HAYES, G. Learning in a state of confusion: perceptual aliasing in grid world navigation. In: TOWARDS INTELLIGENT MOBILE ROBOTS, 4., 2003, Bristol. **Proceedings...** Bristol: Institution of Electrical Engineers, 2003. p. 1–10. 77, 150, 178

CZOGALA, E.; MROZEK, A.; PAWLAK, Z. The idea of a rough fuzzy controller and its application to the stabilization of a pendulum-car system. **Fuzzy Set and Systems**, v. 72, n. 1, p. 61–73, 1995. 67

DAMPER, R. I.; FRENCH, R. L. B.; SCUTT, T. W. ARBIB: an autonomous robot based on inspirations from biology. **Robotics and Autonomous Systems**, v. 31, n. 4, p. 247–274, 2000. 35

DESOUZA, G. N.; KAK, A. C. Vision for mobile robot navigation: A survey. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 24, n. 2, p. 237–267, 2002. 37

DRIANKOV, D. A reminder of fuzzy logic. In: DRIANKOV, D.; SAFFIOTTI, A. (Ed.). **Fuzzy Logic Techniques for Autonomous Vehicle Navigation**. Berlin, Germany: Springer-Verlag, 2001, (Studies in Fuzziness and Soft Computing, v. 61). p. 25–47. ISBN 3-7908-1341-9. Disponível em: <http://www.aass.oru.se/~asaffio/Papers/flan01.html>. 63

DRIANKOV, D.; HELLENDOORN, H.; REINFRANK, M. **An introduction to fuzzy control**. Berlin: Springer-Verlag, 1996. 58, 61

DUBOIS, D.; PRADE, H. What are fuzzy rules and how to use them. **Fuzzy Sets and Systems**, v. 84, n. 2, p. 169–185, 1996. 58

ELFES, A.; BUENO, S. S.; BERGERMAN, M.; RAMOS, J. J. G.; GOMES, S. B. V. Project AURORA: development of an autonomous unmanned remote monitoring robotic airship. **Journal of the Brazilian Computer Society**, v. 4, n. 3, p. 70–78, 1998. ISSN 0104-6500. 86

ESTEVES, C. H.; GABRIELLI, L. H.; COSTA, C. H. Discretização CMAC baseada em conhecimento a priori para generalização da experiência. In: XXIV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAçãO, 24., 2004, Salvador. **Anais...** Salvador: SBC, 2004. p. 8. ISBN 85-88442-93-0. (1 CD-ROM). 41, 80

FARIA, A. O.; PAULO, D. **libhairCAPTURE**. 2005. Disponível em: <http://libhaircapture.codigolivre.org.br/>. Acesso em: 6 dez. 2005. 186

FARIA, G.; ROMERO, R. Incorporating fuzzy logic to reinforcement learning. In: IEEE INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS, 9., 2000, Texas. **Proceedings...** Texas: IEEE, 2000. p. 847–852. 81

FAUSETT, L. **Fundamentals of neural networks**. Upper Saddle River: Prentice Hall, 1994. 48, 50, 51, 54

FIGUEIREDO, M. Navegação autônoma de robôs. In: ESCOLA DE INFORMáTICA DA SBC REGIãO SUL, 7., 1999, Novo Hamburgo. **Proceedings...** Novo Hamburgo: SBC, 1999. p. 75–106. 37

FREW, E.; MCGEE, T.; KIM, Z.; XIAO, X.; JACKSON, S.; MORIMOTO, M.; RATHINAM, S.; PADIAL, J.; SENGUPTA, R. Vision-based road-following using a small autonomous aircraft. In: IEEE AEROESPACE CONFERENCE, 2004, Big Sky, USA. **Proceedings...** Big Sky, USA: IEEE, 2004. v. 5, p. 3006–3015. 87

FUJIE, H.; LIVESAY, G. A.; FUJITA, M.; WOO, S. L.-Y. Forces and moments in six-dof at the human knee joint: mathematical description for control. **Biomechanics**, v. 29, n. 12, p. 1577–1585, 1996. 35

FULLÉR, R. **Neuro fuzzy systems**. Turku, 1995. Abo Akademis Tryckeri. Lecture Notes. 68

GASKETT, C.; FLETCHER, L.; ZELINSKY, A. Reinforcement learning for a vision based mobile robot. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, 2000, Takamatsu, Japan. **Proceedings...** Takamatsu, Japan: IEEE, 2000. v. 1, p. 403–409. ISBN 0-7803-6348-5. Disponível em: <citeseer.ist.psu.edu/article/gaskett00reinforcement.html>. 78

GAUSSIER, P.; JOULAIN, C.; ZREHEN, S.; REVEL, A. Visual navigation in an open environment without map. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, 1997, Grenoble, France. **Proceedings...** Grenoble, France: IEEE, 1997. v. 2, p. 545–550. ISBN 0-7803-4119-8. 85

GLORENNEC, P.; JOUFFE, L. A reinforcement learning method for an autonomous robot. In: FOURTH EUROPEAN CONGRESS ON INTELLIGENT TECHNIQUES AND SOFT COMPUTING, 4., 1996, Aachen, Germany. **Proceedings...** Aachen, Germany: EUFIT, 1996. p. 1100–1104. 80

GOLDBERG, S. B.; MAIMONE, M. W.; MATTHIES, L. Stereo vision and rover navigation software for planetary exploration. In: IEEE AEROSPACE CONFERENCE, 2002, Big Sky, Montana, USA. **Proceedings...** Big Sky, Montana, USA: IEEE, 2002. v. 5, p. 2025–2036. 86, 87

GONZÁLEZ-GALVÁN, E. J.; CRUZ-RAMÍREZ, S. R.; SEELINGER, M. J.; CERVANTES-SÁNCHEZ, J. J. An efficient multi-camera, multi-target scheme for the three-dimensional control of robots using uncalibrated vision. **Robotics and Computer Integrated Manufacturing**, v. 19, n. 5, p. 387–400, 2003. 36

GONZALEZ, R. C.; WINTZ, P. **Digital image processing**. Reading: Addison-Wesley publishing company, 1987. 87, 89, 90

GRAHAM, R. **Digital imaging**. Latheronwheel: Whittles Publishing, 1998. 85

GREINER, G.; HORMANN, K. Efficient clipping of arbitrary polygons. **ACM Trans. Graph.**, ACM Press, v. 17, n. 2, p. 71–83, 1998. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/274363.274364>. 102

HAGEN, S. ten; KRÖSE, B. A short introduction to reinforcement learning. In: DAELEMANS, W.; FLACH, P.; BOSCH, A. van den (Ed.). **Proceedings...** Tilburg: BENELEARN, 1997. p. 7–12. Disponível em: <citeseer.ist.psu.edu/tenhagen97short.html>. 69, 70, 72, 74, 77

_____. Learning to navigate using a lazy map. In: INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS, 11., 2003, Coimbra, Portugal. **Proceedings...** Coimbra, Portugal: IEEE, 2003. p. 299–304. Disponível em: <citeseer.ist.psu.edu/585179.html>. 41, 79

HANDMANN, U.; KALINKE, T.; TZOMAKAS, C.; WERNER, M.; SEELEN, W. V. An image processing system for driver assistance. **Image and Vision Computing**, v. 18, n. 5, p. 367–376, 2000. Disponível em: <http://www.elsevier.com/locate/imavis>. 56

HARMON, M. E.; HARMON, S. S. **Reinforcement learning:** a tutorial. 1996. 17 p. Disponível em: <citeseer.ist.psu.edu/harmon96reinforcement.html>. Acesso em: 21 ago. 2004. 69, 70, 75, 77

HAYKIN, S. **Redes neurais:** princípios e prática. Porto Alegre: Bookman, 2001. 46, 49, 51, 53, 54

HEINEN, F. J. **Sistema de controle híbrido para robôs móveis autônomos**. Dissertação (Mestrado em Computação Aplicada) — Universidade do Vale do Rio dos Sinos, São Leopoldo, 2001. 37, 40

HOFFMANN, L. T.; CASTRO, A. P. A.; SILVA, J. D. S. Controle inteligente de um robô móvel utilizando imagens. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAçãO, 24., 2004, Salvador. **Anais...** Salvador: SBC, 2004. p. 1–10. (1 CD-ROM). 39, 57

_____. Image based autonomous navigation with softcomputing decision making techniques. In: SIMPóSIO BRASILEIRO DE REDES NEURAIS, 9., 2004, São Luis. **Proceedings...** São Luis: IEEE Computer Society/SBC, 2004. p. 1–6. 39

HORN, B. K. P. **Robot vision**. 3. ed. Massachusetts: The MIT Press, 1986. 85

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Networks**, v. 2, n. 5, p. 359–366, 1989. 50

HOWE, R.; MATSUOKA, Y. Robotics for surgery. **Annual Review of Biomedical Engineering**, v. 1, p. 211 – 240, 1999. 35

HUNTSBERGER, T.; CHENG, Y.; BAUMGARTNER, E. T.; ROBINSON, M.; SCHENKER, P. S. Sensory fusion for planetary surface robotic navigation, rendezvous, and manipulation operations. In: IEEE INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS, 2003, Coimbra, Portugal. **Proceedings...** Coimbra, Portugal: IEEE, 2003. p. 1417–1424. 86

IIDA, F. Biologically inspired visual odometer for navigation of a flying robot. **Robotics and Autonomous Systems**, v. 44, n. 3–4, p. 201–208, 2003. 35

JANTZEN, J. **Analysis of a pendulum problem**. 2003. Disponível em: <http://www.erudit.de/erudit/demos/cartball/>. Acesso em: 21 ago. 2003. 67

_____. **Design of fuzzy controllers**. 2003. Disponível em: <http://www.erudit.de/erudit/demos/cartball/>. Acesso em: 21 ago. 2003. 63

_____. **Tutorial on fuzzy logic**. 2003. Disponível em: <http://www.erudit.de/erudit/demos/cartball/>. Acesso em: 21 ago. 2003. 63

JOCHEM, T.; POMERLEAU, D. Life in the fast lane: The evolution of an adaptive vehicle control system. **Artificial intelligence magazine**, v. 17, n. 2, p. 11–50, 1996. 37, 86

JUNG, C. R.; KELBER, C. R. Lane following and lane departure using a linear-parabolic model. **Image and Vision Computing**, v. 23, n. 13, p. 1192–1202, 2005. 38

KAELBLING, L. P.; LITTMAN, M. K. Reinforcement learning: a survey. **Journal of Artifical Intelligence**, n. 4, p. 237–285, 1996. 69, 70, 71, 74, 75, 78

KARR, C. Adaptive control with fuzzy logic and genetic algorithms. In: YAGER, R.; ZADEH, L. A. (Ed.). **Fuzzy, sets, neural networks, and soft computing**. New York: Van Nostrand Reinhold, 1994. 67

KELLER, J. M.; KRISHNAPURAM, R. Fuzzy decision models in computer vision. In: YAGER, R.; ZADEH, L. A. (Ed.). **Fuzzy, sets, neural networks, and soft computing**. New York: Van Nostrand Reinhold, 1994. 67

KIDONO, K.; MIURA, J.; SHIRAI, Y. Autonomous visual navigation of a mobile robot using a human-guided experience. **Robotics and Autonomous Systems**, v. 40, n. 2–3, p. 121–130, 2003. 85

KIM, Y.; NOH, M.-J.; HAN, T.-D.; KIM, S.-D. Mapping of neural networks onto the memory-processor integrated architecture. **Neural Networks**, v. 11, n. 6, p. 1083–1098, 1998. 56

KIMURA, H.; MIYAZAKI, K.; KOBAYASHI, S. Reinforcement learning in POMDPs with function approximation. In: INTERNATIONAL CONFERENCE

ON MACHINE LEARNING, 14., 1997, Nashville, USA. **Proceedings...** San Francisco, USA: Morgan Kaufmann Publishers Inc, 1997. p. 152–160. 78

KOHONEN, T. Self-organized formation of topologically correct feature maps. **Biological Cybernetics**, Springer-Verlag, v. 43, n. 1, p. 59–69, 1982. 54, 76

LEE, C. C. Fuzzy logic in control systems: fuzzy logic controller - part i. **IEEE Transactions of systems, man, and cybernetics**, v. 20, n. 2, p. 404–418, 1990. 63

LI, W.; LU, G.; WANG, Y. Recognizing white line markings for vision guided vehicle navigation by fuzzy reasoning. **Pattern Recognition Letters**, v. 18, n. 8, p. 771–780, 1997. 38

LI, W.; XU, C.; XIAO, Q.; XU, X. Visual navigation of an autonomous robot using white line recognition. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2003, Taipet, Taiwan. **Proceedings...** Taipet, Taiwan: IEEE, 2003. p. 3923–3928. 38

LOBO, J.; QUEIROZ, C.; DIAS, J. World feature detection and mapping using stereovision and inertial sensors. **Robotics and Autonomous Systems**, v. 44, n. 1, p. 69–81, 2003. 87

MACEK, K.; PETROVIC, I.; PERIC, N. A reinforcement learning approach to obstacle avoidance of mobile robots. In: INTERNATIONAL WORKSHOP ON ADVANCED MOTION CONTROL, 7., 2002, Maribor, Slovenia. **Proceedings...** Maribor, Slovenia: IEEE, 2002. p. 462–466. 41, 81

MARTÍNEZ-MARŃ, T.; DUCKETT, T. Fast reinforcement learning for vision-guided mobile robots. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2005, Barcelona, Spain. **Proceedings...** Barcelona, Spain: IEEE, 2005. p. 4170–4175. 41, 80

MATA, M.; ARMINGOL, J. M.; ESCALERA, A. de la; SALICHS, M. Learning visual landmarks for mobile robot navigation. In: WORLD CONGRESS OF INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL, 15., 2002, Barcelona, Spain. **Proceedings...** Barcelona: IFAC, 2002. p. 6. 85

MATARIC, M. J. Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior. **Trends in Cognitive Sciences**, v. 2, n. 3, p. 82–87, 1998. 35

MATSUMOTO, A.; YOSHITA, G.; KIHANA, I. Teaching by showing few images for the navigation of mobile robots. In: IEEE INTERNATIONAL SYMPOSIUM ON ASSEMBLY AND TASK PLANNING, 5., 2003, Besancon, France. **Proceedings...** Besancon, France: IEEE, 2003. p. 270–275. 85

MENG, M.; KAK, A. C. Mobile robot navigation using neural networks and nonmetrical environment models. **IEEE Control systems**, p. 30–39, 1993. 57

MICHELS, J.; SAXENA, A.; NG, A. Y. High speed obstacle avoidance using monocular vision and reinforcement learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 22., 2005, Bonn, Germany. **Proceedings...** Bonn, Germany: ACM Press, 2005. p. 8. 41, 80

MICROSOFT. **Vision technology**. 2003. Disponível em: <http://research.microsoft.com/projects/VisSDK/>. Acesso em: 15 nov. 2003. 143

MOREIRA, M. A. **Fundamentos de Sensoriamento remoto e metodologias de aplicação**. São José dos Campos: INPE, 2001. 120, 121

MUSCETTOLA, N.; NAYAK, P. P.; PELL, B.; WILLIAMS, B. Remote agent: to boldly go where no AI system has gone before. **Artificial Intelligence**, v. 1-2, n. 103, p. 5–48, 1998. 38

MUSKINJA, N.; TOVORNIK, B. Controlling of real inverted pendulum by fuzzy logic. In: PORTUGUESE CONFERENCE ON AUTOMATIC CONTROL, 4., 2000, Guimarães, Portugal. **Proceedings...** Guimarães, Portugal: IPCA, 2000. p. 354–358. Disponível em: <http://www.au.feri.uni-mb.si/~nenad/Control%20of%20inverted%20pendulum.pdf>. 67

NASA. **The new Mars**. 2004. Disponível em: <http://www.nasa.gov/externalflash/m2k4/frameset.html>. Acesso em: 03 mar. 2005. 35, 36

_____. **New millennium program:** testing advanced technology in space flight. 2005. Disponível em: <http://nmp.jpl.nasa.gov>. Acesso em: 13 dez. 2005. 38

NG, K. C.; TRIVEDI, M. M. A neuro-fuzzy controller for mobile robot navigation and multirobot convoying. **IEEE Transactions on Systems, Man and Cybernetics**, v. 28, n. 6, p. 829–840, 1998. 68

NILSSON, J. **Visual landmark selection and recognition for autonomous unmanned aerial vehicle navigation**. 53 p. Dissertação (Mestrado) — Royal Institute of Technology, Stockholm, 2005. 87

NOGUEIRA, M. B.; MEDEIROS, A. A. D.; ALSINA, P. J. Development of a fast visual sensor of 3D position for a manipulator robot. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAçãO, 24., 2004, Salvador. **Anais...** Salvador: SBC, 2004. p. 10. ISBN 85-88442-93-0. (1 CD-ROM). 96

OLIVEIRA, L. R. de; COSTA, A. L. da. Visão computacional: uma abordagem para robótica móvel baseada em fusão de sensores. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAçãO, 24., 2004, Salvador. **Anais...** Salvador: SBC, 2004. p. 8. ISBN 85-88442-93-0. (1 CD-ROM). 109

OLLERO, A.; FERRUZ, J.; CABALLERO, F.; HURTADO, S.; MERINO, L. Motion compensation and object detection for autonomous helicopter visual navigation in the COMETS system. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2004, New Orleans, USA. **Proceedings...** New Orleans, USA: IEEE, 2004. v. 1, p. 19–24. 87

OSÓRIO, F. S.; HEINEN, F.; FORTES, L. Controle da tarefa de estacionamento de um veículo autônomo através do aprendizado de um autômato finito usando uma rede neural J-CC. In: SIMPóSIO BRASILEIRO DE REDES NEURAIS, 7., 2006, Porto de Galinhas, Recife. **Proceedings...** Porto de Galinhas, Recife: IEEE Computer Society, 2002. p. 1–6. 57

PARK, S. K.; KIM, M.; LEE, C. won. Mobile robot navigation based on direct depth and color-based environment modeling. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2005, Barcelona, Spain. **Proceedings...** Barcelona, Spain: IEEE, 2004. v. 5, p. 4253–4258. 87

PENG, J.; BHANU, B. Delayed reinforcement learning for adaptive image segmentation and feature extraction. **IEEE Transaction on Systems, Man, and Cybernetics**, v. 28, n. 3, p. 482–488, 1998. 80

PIO, J. L. de S. Navegação robótica aérea baseada em visão - requisitos de processamento de imagens para projeto e implementação. In: WORKSHOP EM TRATAMENTO DE IMAGENS, 3., 2002, Belo Horizonte. **Proceedings...** Belo Horizonte: UFMG, 2002. p. 9. 108

POMERLEAU, D. RALPH: Rapidly Adapting Lateral Position Handler. In: IEEE SYMPOSIUM ON INTELLIGENT VEHICLES, 1995, Detroit, USA. **Proceedings...** Detroit, USA: IEEE, 1995. p. 506–511. 37

RENHOU, L.; YI, Z. Fuzzy logic controller based on genetic algorithms. **Fuzzy Sets and Systems**, v. 83, n. 1, p. 1–10, 1996. 67

RODRIGUES, S. S.; SANTOS, T. S.; TRABASSO, L. G.; KIENITZ, K. H. Modelagem e controle do suporte das lâminas da serra inteligente guiada por um sistema robótico para assistência em cirurgias de osteotomia. In: BALTHAZAR, J. M.; SILVA, G. N.; TSUCHIDA, M.; BOAVENTURA, M.; TEIXEIRA, M. C. M.; LOPES-JR., V. (Ed.). **Proceedings...** Ilha Solteira, SP: SBMAC, 2004. v. 2, p. 1–13. 35

ROSENFELD, A.; PFALTZ, J. L. Sequential operations in digital picture processing. **Journal of the Association for Computer Machinery**, v. 13, p. 471–494, 1966. ISSN 0004-5411. 92

RUDOLPH, G. L.; MARTINEZ, T. R. A transformation strategy for implementing distributed, multilayer feedforward neural networks: backpropagation transformation. **Future Generation Computer Systems**, v. 12, n. 6, p. 547–564, 1997. 56

RUSSEL, S.; NORVIG, P. **Inteligência artificial**. Rio de Janeiro: Elsevier, 2004. 38, 41, 42, 45, 69, 72, 74, 77

SAFFIOTTI, A. **Autonomous robot navigation:** a fuzzy logic approach. Tese (Doutorado) — IRIDIA, Université Libre de Bruxelles, 1998. 67

SANDRI, S.; CORREA, C. **Lógica nebulosa**. São José dos Campos: INPE, 1999. 73-90 p. 58

SARIPALLI, S.; MONTGOMERY, J. F.; SUKHATME, G. S. Visually guided landing of an unmanned aerial vehicle. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2003, Taipet, Taiwan. **Proceedings...** Taipet, Taiwan: IEEE, 2003. v. 19, n. 3, p. 371–381. 87

SCHILLING, K.; ROTH, H.; LIEB, R. Remote controle of a ”Mars rover” via internet - to support education in control and teleoperations. **Acta Astronautica**, v. 50, n. 3, p. 173–178, 2002. 35

SCHMIDT, R.; WEISSER, H.; SCHULENBERG, P.; GOELLINGER, H. Autonomous driving on vehicle test tracks: overview, implementation and results. In: IEEE INTELLIGENT VEHICLES SYMPOSIUM, 2000, Dearborn, USA. **Proceedings...** Dearborn, USA: IEEE, 2000. p. 152–155. 38

SHALOM, M. W.; KULIKOWSKI, C. A. **Computer systems that learn**. San Mateo: Morgan Kaufmann, 1991. 45

SHAPIRO, L. G.; STOCKMAN, G. C. **Computer vision**. Upper Saddle River: Prentice-Hall, 2001. 85, 88, 90, 91, 92, 95

SHAVLIK, J. W.; DIETTERICH, T. G. Introduction. In: SHAVLIK, J. W.; DIETTERICH, T. G. (Ed.). **Readings in machine learning**. San Mateo: Morgan Kaufmann, 1990. 45

SHAW, I. S.; SIMÕES, M. G. **Controle e modelagem fuzzy**. São Paulo: Edgard Blücher, 1999. 58

SHIMABUKURO, Y. E.; SMITH, J. A. The least-square mixing models to generate fraction images derived from remote sensing multispectral data. **IEEE Transactions on Geoscience and Remote Sensing**, v. 29, n. 1, p. 16–20, 1991. 111

SILVA, J. D. S. **Uma abordagem hibrida por Dempster-Shafer e algoritmos genéticos para o problema de correspondencia em estereoscopia**. 298 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 1999. (INPE-10533-PRE/6005). Disponível em: <http://iris.sid.inpe.br:1905/rep/sid.inpe.br/deise/2000/11.06.11.28>. 37, 87

SILVA, J. D. S.; SIMONI, P. O. Dempster-shafer theory as an inference method for corresponding geometric distorted images. In: ENCONTRO NACIONAL DE INTELIGêNCIA ARTIFICIAL, 4., 2003, Campinas. **Proceedings...** Campinas: SBC, 2003. p. 6. 37

SIMON, H. A. Why should machines learn? In: MICHALSKI, R. S.; CARBORELL, J. G.; MITCHELL, T. M. (Ed.). **Machine learning: an artificial intelligence approach**. San Mateo: Morgan Kaufmann, 1983. 45

SINGH, S.; NORVIG, P.; COHN, D. **How to make software agents do the right thing:** an introduction to reinforcement learning. New York, USA, 1996. Dr. Dobbs journal. Disponível em: <citeseer.ist.psu.edu/singh96how.html>. 68, 75

SINGH, S.; SIMMONS, R.; SMITH, T.; STENTZ, A.; VERMA, V.; A., Y.; SCHWEHR, K. Recent progress in local and global traversability for planetary rovers. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2000, San Francisco, USA. **Proceedings...** San Francisco, USA: IEEE, 2000. p. 1194–1200. Disponível em: <http://www.frc.ri.cmu.edu/projects/mars/>. 35, 86, 87

SMART, W.; KAELBLING, L. P. Effective reinforcement learning for mobile robots. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2002, Washington, USA. **Proceedings...** Washington, USA: IEEE, 2002. p. 3404–3410. 79, 152

SMART, W. D.; KAELBLING, L. P. Reinforcement learning for robot control. In: MOBILE ROBOTS, 16., 2001, Newton, USA. **Proceedings...** Newton, USA: SPIE, 2001. v. 4573, p. 92–103. 79

SMITH, A. J. Applications of the self-organising map to reinforcement learning. **Neural Networks**, v. 15, n. 8-9, p. 1107 – 1124, 2002. Disponível em: <http://psycserv.mcmaster.ca/~smitha/PAGE_PAPERS/neural-networks-2002.pdf>. 76

SONG, F.; SMITH, S. M. How blind can a blind fuzzy logic controller design be? analysis of cell state space based incremental best estimate directed search algorithm. In: IEEE INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS, 2000, San Antonio, USA. **Proceedings...** San Antonio, USA: IEEE, 2000. p. 134–139. Disponível em: <http://www.oe.fau.edu/~fsong/fuzzy002.pdf>. 67

_____. A Takagi-Sugeno type fuzzy logic controller with only 3 rules for a 4 dimensional inverted pendulum system. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEM, MAN AND CYBERNETICS, 2000, Nashville, USA. **Proceedings...** Nashville, USA: IEEE, 2000. p. 3800–3805. Disponível em: <http://www.oe.fau.edu/~fsong/smc002.pdf>. 67

SONG, K. T.; SHEEN, L. H. Heuristic fuzzy-neuro network and its application to reactive navigation of a mobile robot. **Fuzzy Sets and Systems**, v. 110, n. 33, p. 331–340, 2000. 40, 57, 68

STONIER, R. J.; STACEY A. J. MESSOM, C. Learning fuzzy control laws for the inverted pendulum. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 25., 1998, Barcelona, Spain. **Proceedings...** Barcelona, Spain: ACM Press, 1998. p. 4. Disponível em: <http://www.massey.ac.nz/~chmessom/isca98e.pdf>. 67

SUNDAY, D. **The intersections for a set of 2D segments, and testing simple polygons**. Soft Surfer.com, 2001. Disponível em: <http://www.geometryalgorithms.com/Archive/algorithm_0108/algorithm_0108.htm>. Acesso em: 05 set. 2004. 208

_____. **Intersections of lines, segments and planes**. Soft Surfer.com, 2001. Disponível em: <http://www.geometryalgorithms.com/Archive/algorithm_0104/algorithm_0104B.htm>. Acesso em: 05 set. 2004. 208

SURMANN, H.; HUSER, J.; PETERS, L. A fuzzy system for indoor mobile robot navigation. In: IEEE INTERNACIONAL CONFERENCE ON FUZZY SYSTEMS, 4., 1995, Yokohama, Japan. **Proceedings...** Yokohama, Japan: IEEE, 1995. p. 83–86. 36, 40, 67

SURMANN, H.; HUSER, J.; WEHKING, J. Path planning for a fuzzy controlled autonomous mobile robot. In: IEEE INTERNACIONAL CONFERENCE ON FUZZY SYSTEMS, 5., 1996, Orlando, USA. Orlando, USA: IEEE, 1996. p. 1660–1664. 40, 67

SUTTON, R.; BARTO, A. **Reinforcement learning:** an introduction. Cambridge: MIT Press, 1998. 68, 69, 70, 71, 72, 73, 74, 76, 78, 79

SUZUKI, S.; KATO, T.; ISHIZUKA, H.; TAKAHASHI, Y.; UCHIBE, E.; ASADA, M. An application of vision-based learning in RoboCup for a real robot with an omnidirectional vision system and the team description of Osaka university "trackies". In: ROBOCUP. **Proceedings...** Springer, 1998. p. 316–325. Disponível em: <citeseer.ist.psu.edu/suzuki98application.html>. 80

TAKAGI, T.; SUGENO, M. Fuzzy identification of systems and its applications to modeling and control. **IEEE Systems, man, and cybernetics**, v. 1, n. 15, p. 116–132, 1985. 64

TANAKA, K. **An introduction to fuzzy logic for practical applications**. New York: Springer, 1997. 58

TARJAN, R. E. Efficiency of a good but not linear algebra and related architectures. **Journal of the Association for Computer Machinery**, v. 22, p. 215–225, 1975. 92

TEIXEIRA, M. C. M.; PIETROBOM, H. C.; ASSUNÇÃO, E. Novos resultados sobre a estabilidade e control de sistemas não-lineares utilizando modelos fuzzy e LMI. **Controle e Automação**, v. 11, n. 1, p. 37–48, 1998. 67

TESAURO, G. Temporal difference learning and TD-Gammon. **Communications of the ACM**, v. 28, n. 3, 1995. 76

The Robotics Institute. **No hands across America!** 1995. Disponível em: <http://www.ri.cmu.edu/projects/project_178.html>. Acesso em: 13 dez. 2005. 37

TOPPING, B. H. V.; SZIVERI, J.; BAHREINEJAD, A.; B., L. J. P.; CHENG, B. Parallel processing, neural networks and genetic algorithms. **Advances in Engineering Software**, v. 29, n. 10, p. 763–786, 1998. 56

TROLLTECH. **Trolltech - cross platform C++ GUI development, and embedded Linux solutions**. 2005. Disponível em: <http://www.trolltech.com/>. Acesso em: 19 dez. 2005. 186

TSOUKALAS, L. H.; UHRIG, R. E. **Fuzzy and neural approaches in engineering**. New York: John Wiley & Sons, 1997. 46, 62, 63

TUNSTEL, E.; JAMSHIDI, M. Fuzzy logic and behavior control strategy for autonomous mobile robot mapping. In: IEEE CONFERENCE ON FUZZY SYSTEMS, 1994, Orlando, USA. **Proceedings...** Orlando, USA: IEEE, 1994. p. 514–517. 65

VOUDOURIS, C.; CHERNETT, P.; WANG, C. J.; CALLAGHAN, V. L. Fuzzy hierarchical control for autonomous vehicle. In: INTERNATIONAL WORKSHOP

ON INTELLIGENT ROBOTIC SYSTEMS, 1994, Grenoble, France. **Proceedings...** Grenoble, France: LIFIA, 1994. p. 1–8. 67

WEBER, C.; WERMTER, S.; ZOCHIOS, A. Robot docking with neural vision and reinforcement. **Knowledge-Based Systems**, n. 17, p. 165–72, 2004. 41, 80

WEISSER, H.; SCHULENBERG, P.; GOLLINGER, H.; MICHLER, T. Autonomous driving on vehicle test tracks: overview, implementation and vehicle diagnosis. In: IEEE INTERNATIONAL CONFERENCE ON INTELLIGENT TRANSPORTATION SYSTEMS, 1999, Tokyo, Japan. **Proceedings...** Tokyo, Japan: IEEE, 1999. p. 62–67. 38

XU, X.; WANG, X.-N.; HE, H.-G. A self-learning reactive navigation method for mobile robots. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND CYBERNECTICS, 2., 2003, Xi'an, China. **Proceedings...** Xi'an, China: IEEE, 2003. p. 2384–2388. 80

YAGER, R.; ZADEH, L. A. **Fuzzy, sets, neural networks, and soft computing**. New York: Van Nostrand Reinhold, 1994. 36

YASUDA, G. Distributed autonomous control of modular robot systems using parallel programming. **Journal of Materials Processing Technology**, v. 141, n. 3, p. 357–364, 2003. 36

YEN, G.; HICKEY, T. Reinforcement learning algorithms for robotic navigation in dynamic environments. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2002, Honolulu, Hawaii. **Proceedings...** Honolulu, Hawaii: IEEE, 2002. v. 2, p. 1444 – 1449. 79

YEN, J. Logic: a modern perspective. **IEEE Transactions on Knowledge and Data Engineering**, v. 11, n. 1, p. 153–165, 1999. 58

ZHU, W.; LEVINSON, S. Vision-based reinforcement learning for robot navigation. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2001, Washington DC, USA. **Proceedings...** Washington DC, USA: IEEE, 2001. 79

ZHUANG, X.-D.; MENG, Q.-C.; WANG, H.-P.; YIN, B. Multi-scale reinforcement learning with fuzzy state. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND CYBERNETICS, 2002, Beijing, China. **Proceedings...** Beijing, China: IEEE, 2002. v. 3, p. 1523 – 1528. 81

APÊNDICE A

ALGORITMO DE BENTLEY-OTTMAN

O algoritmo *sweep line* de Bentley e Ottmann (1979) permite extrair eficientemente o conjunto de pontos de intersecção IT de um conjunto de segmentos SG. Neste algoritmo, uma linha varre o espaço bidimensional dos segmentos, controlando quais os segmentos podem se cruzar, minimizando o número de comparações.

Uma fila de eventos XQ representa a seqüência ordenada de todos pontos de extremidades (*endpoints* dos segmentos) que a linha de varredura encontrará. Em sua configuração clássica, os pontos de extremidades são ordenados pelos valores crescentes de suas coordenadas x (da esquerda para direita) e em caso de empate, pela y (de cima para baixo).

A *sweep line* é representada por uma estrutura de dados SL. Ela controla os segmentos que encontra durante o processo de varredura, adicionando em SL um segmento sg_e se seu ponto de extremidade esquerda foi encontrado em XQ, ou excluindo sg_e de SL, caso seu ponto de extremidade for da direita.

Cada elemento encontrado em XQ é retirado dessa fila de eventos. A inserção de segmentos em SL deve respeitar a ordenação vertical dos pontos (coordenada y), permitindo um controle de posicionamento vertical desses segmentos (que segmento está acima ou baixo de outro segmento). De modo a aumentar a eficiência do algoritmo, a estrutura SL pode ser implementada então por uma árvore binária balanceada.

Quando um novo segmento sg_e é adicionado em SL, deve-se identificar seus vizinhos acima sg_a e abaixo sg_b, verificando a existência de intersecção do segmento sg_e com sg_a e sg_b. Se for encontrada alguma intersecção it a mesma deve ser inserida na fila de eventos XQ, respeitando a ordenação estabelecida.

No caso de se remover um segmento sg_e de SL é também verificada a existência de intersecção entre sg_a e sg_b. Porém neste caso, a intersecção it só será inserida em XQ se ela já não foi inserida anteriormente. Deve-se então manter uma identificação do par de segmentos que cada intersecção representa.

Se alguma intersecção it foi encontrada, em algum momento a linha de varredura a extrairá da fila de eventos XQ. Este ponto representa a inversão de posição vertical

dos segmentos sg_{e1} e sg_{e2} que se cruzam em it. As posições de sg_{e1} e sg_{e2} devem ser invertidas em SL e it adicionado ao conjunto IT. Devido a operação de inversão de posições, procura-se por novas intersecções com sg_{e1} e sg_{e1}. O segmento que subiu (por exemplo sg_{e2}) verificará se existe intersecção com seu vizinho acima, e o sg_{e1} com seu vizinho abaixo.

O algoritmo termina quando não existirem mais elementos na fila de eventos XQ.

A Figura A.1 ilustra o funcionamento do algoritmo de Bentley-Ottman, identificando as 4 intersecções de 4 segmentos. O processo de varredura é feito da esquerda para a direita, que vai encontrando os 12 eventos na fila XQ. Os eventos são constituídos das extremidades dos segmentos (pontos vermelhos) e pontos de intersecções (círculos vermelhos).

O algoritmo da Figura A.2 descreve o procedimento completo para se encontrar os pontos de intersecção, extraído de Sunday (2001a). Uma implementação interessante para a função $Interseccao(sg_1, sg_2)$, que identifica o ponto de intersecção entre os segmentos sg_1 e sg_2, pode ser encontrada em Sunday (2001b).

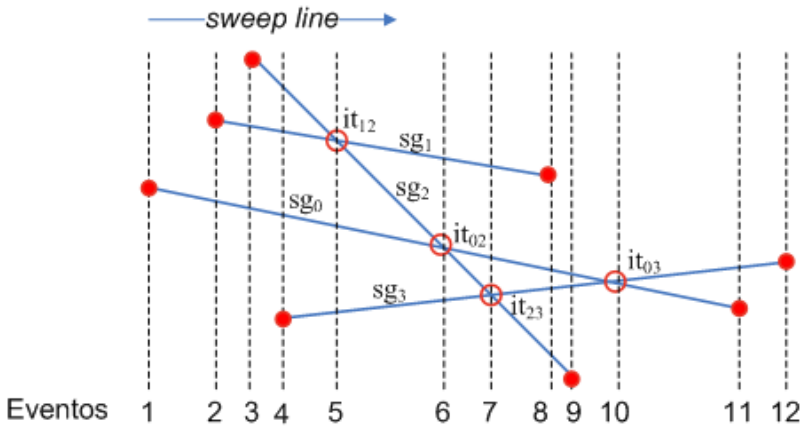


FIGURA A.1 - Ilustração da linha varrendo os eventos 1 a 12 e encontrando 4 intersecções em 4 segmentos.

Requer coleção de segmentos SG
  inicializar a fila de eventos XQ com as extremidades dos segmentos de SG
  ordenar a fila de eventos XQ por valores crescente das coordenadas x e depois y
  inicializar a *sweep line* $SL \leftarrow \emptyset$
  inicializar a lista de intersecções $IT \leftarrow \emptyset$
  enquanto $XQ \neq \emptyset$ **faça**
    $ev \leftarrow XQ_0$, onde XQ_0 é primeiro evento de XQ
    se ev é uma extremidade da esquerda **então**
      $segE \leftarrow$ segmento de ev
      adicionar $segE$ em SL
      $segA \leftarrow$ o segmento acima de $segE$ em SL
      $segB \leftarrow$ o segmento abaixo de $segE$ em SL
      se $(it \leftarrow Interseccao(segE, segA)) \neq \emptyset$ **então**
        inserir it em XQ
      fim se
      se $(it \leftarrow Interseccao(segE, segB)) \neq \emptyset$ **então**
        inserir it em XQ
      fim se
    senão se ev é uma extremidade da direita **então**
      $segE \leftarrow$ segmento de ev
      $segA \leftarrow$ o segmento acima de $segE$ em SL
      $segB \leftarrow$ o segmento abaixo de $segE$ em SL
      remover $segE$ de SL
      se $(it \leftarrow Interseccao(segA, segB)) \neq \emptyset$ **então**
        se $it \notin XQ$ então inserir it em XQ
      fim se
    senão
      adicionar ev a IT
      sejam $segE1$ e $segE2$ os segmentos interceptados por ev em SL, onde $segE1$ está acima de $segE2$
      inverter as posições de $segE1$ e $segE2$ em SL, de maneira que $segE2$ fique acima de $segE1$
      $segA \leftarrow$ o segmento acima de $segE2$ em SL
      $segB \leftarrow$ o segmento abaixo de $segE1$ em SL
      se $(it \leftarrow Interseccao(segE2, segA)) \neq \emptyset$ **então**
        se $it \notin XQ$ **então**
          inserir it em XQ
        fim se
      fim se
      se $(it \leftarrow Interseccao(segE1, segB)) \neq \emptyset$ **então**
        se $it \notin XQ$ então inserir it em XQ
      fim se
    fim se
    remover ev de XQ
  fim enquanto
  return IT

FIGURA A.2 - Algoritmo *Sweep Line* de Bentley-Ottman.

APÊNDICE B

ESQUEMÁTICO DA INTERFACE DO ATUADOR

O esquemático da placa de circuito impresso da Figura B.1 é utilizada para acionar o controle remoto do carrinho, via porta paralela. A Tabela B.1 relaciona a listagem de valores enviados a porta paralela e os respectivos acionamentos dos atuadores do robô.

TABELA B.1 - Relação de valores enviados a porta paralela e os respectivos acionamentos do robô.

Valor decimal	Valor Binário	Acionamento
0	0000	SG-PARAR
1	0001	SG-FRENTE
2	0010	SG-RE
4	0100	SG-DIREITA
5	0101	SG-FRENTE-DIREITA
6	0110	SG-RE-DIREITA
8	1000	SG-ESQUERDA
9	1001	SG-FRENTE-ESQUERDA
10	1010	SG-RE-ESQUERDA

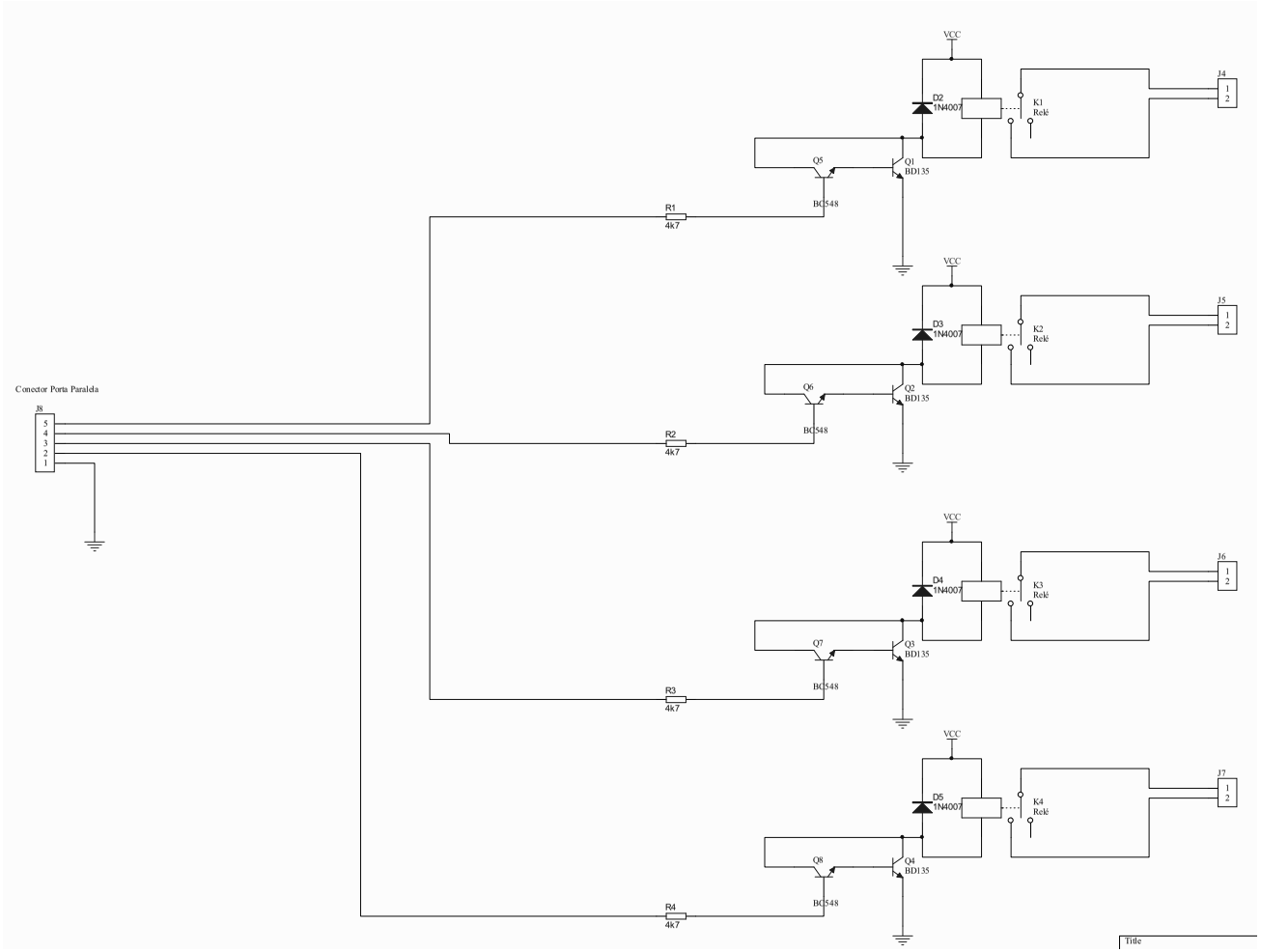


FIGURA B.1 - Esquemático da placa de circuito impresso utilizada para acionar o controle remoto do carrinho, via porta paralela.