Aplicações de Sistemas Classificadores para Robótica Autônoma Móvel com Aprendizado

por

Lubnen Name Moussi

Orientador

Prof. Dr. Marconi Kolm Madrid

Co-Orientador

Prof. Dr. Ricardo Ribeiro Gudwin

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica.

Banca examinadora:

1. Prof. Dr. Alexandre Pinto Alves da Silva - COPPE - UFRJ
2. Prof. Dr. Fernando Antônio Campos Gomide - FEEC - UNICAMP
3. Prof. Dr. Fernando José Von Zuben - FEEC - UNICAMP

7 de Novembro de 2002

FEEC

Faculdade de Engenharia Elétrica e de Computação

UNICAMP

Universidade Estadual de Campinas

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

M867a

Moussi, Lubnen Name
Aplicações de sistemas classificadores para robótica autônoma móvel com aprendizado / Lubnen Name Moussi.--Campinas, SP: [s.n.], 2002.

Orientadores: Marconi Kolm Madrid e Ricardo Ribeiro Gudwin.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes neurais (Computação). 2. Sistemas inteligentes de controle. 3. Robótica. I. Madrid, Marconi Kolm. II. Gudwin, Ricardo Ribeiro. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

As Três Leis da Robótica, segundo Isaac Asimov

- Primeira Lei: um robô não pode ferir um ser humano, ou, por omissão, permitir que um ser humano sofra algum mal.

- Segunda Lei: um robô deve obedecer às ordens que lhe sejam dadas por seres humanos, exceto nos casos em que tais ordens contrariem a Primeira Lei.

- Terceira Lei: um robô deve proteger sua própria existência enquanto tal proteção não entrar em conflito com a Primeira ou Segunda Leis.

Ref.: Isaac Asimov, "Eu, robô", Editora Expressão e Cultura, 8a. edição, pág 61, 1974.

Com muito amor dedico este trabalho a

Dna. Ditinha, Silvia e Marina

Agradecimentos

À Silvia e Marina pela confiança, entusiasmo e carinho por mim enquanto busco ideais que procuro atingir desde minha infância.

Aos meus sogros Seu Chico e Dna. Ditinha, à tia Maria, à minha cunhada Dra. Luciana e aos meus cunhados Pedro e Tinha por todo o apoio à minha família. Em especial à Dna. Ditinha, minha querida incentivadora.

À todos os meus familiares, irmãos, cunhados e sobrinhos pelo apoio. Ao Beor, à Cristiane, ao Rafael e à Silvia, pela torcida e por me fazerem sentir seu afeto mesmo distante.

Ao Prof. Dr. Marconi Kolm Madrid que me acolheu como seu orientado, e sempre me incentiva e transmite amizade, confiança e conhecimento. Além de bom companheiro nas horas difíceis.

Ao Prof. Dr. Álvaro Badan Palhares, que me aceitou como seu orientado oficial enquanto, por questões regimentais, o Prof. Madrid não pode, possibilitando meu ingresso como aluno regular.

Aos Prof. Drs. Ricardo Ribeiro Gudwin, meu co-orientador, e Fernando José Von Zuben, professor e grande amigo sempre presente neste meu trabalho, parceiros desde a formulação das primeiras idéias com relação à minha Tese, passando pela publicação de vários artigos, até a sua finalização.

Aos colegas de mestrado, de diversas áreas, Cristiane, Renato, Edílson, Valdei, Rangel, Vânio e tantos outros. Aos colegas do DSCE Mário e Fabrício pela orientação e ajuda em tantos detalhes.

À Arthur C. Clarke, Isaac Asimov e Carl Sagan, representando seres Humanos especiais que olham para a Ciência com encantamento e criatividade e a veiculam em obras maravilhosas, fontes de inspiração para aqueles que se propõem a estudar o Universo, tanto para entender e saborear os seus mistérios quanto para poderem sugerir novas formulações que concretizem um mundo bem melhor.

À UNICAMP pelo que ela representa no cenário nacional e internacional e por ter me propiciado uma infra-estrutura humana e material de pesquisa das mais qualificadas.

À CAPES por contribuir para subsidiar meus estudos.

À PRÓ-VIDA e seu idealizador e fundador Dr. Celso Charuri, que através de seus cursos, monitores, preceptores e inúmeros amigos, tem propiciado fundamentos e ambiente para eu me estruturar mental e espiritualmente na busca dos meus ideais. Em especial ao meu amigo Ariovaldo, Diretor do Departamento Científico da PRÓ-VIDA, que em nossos contatos faz renascer e fortalecer em mim meu gosto pela Ciência e pela Filosofia.

Sou imensamente grato a meus pais Name e Iesmin e ao Universo por eu estar existindo no Planeta Terra, que me fascina, e poder me manifestar, o que procuro fazer de forma verdadeira e com amor.

Obrigado Deus.

Resumo

Resumo

A Robótica Móvel tem como meta fundamental elaborar trajetórias para evitar colisões, localizar e alcançar alvos, auto-suficiência do robô em termos de suprimento de energia, transporte de objetos, etc. Uma solução que dê autonomia ao robô no aprendizado de seu comportamento se contrasta com outras abordagens mais clássicas que exigem um modelo prévio do ambiente em que o robô está inserido. Caso seja necessário colocar o robô em outro ambiente, ou caso o ambiente tenha componentes variantes no tempo, estes modelos apresentam grande ineficiência, exigindo do projetista todo um recálculo de trajetórias, ou até mesmo inviabilizando seu uso. A utilização de procedimentos de aprendizagem libera o projetista de ter que inserir no seu projeto conhecimentos detalhados do ambiente e dão ao robô a possibilidade de se comportar adequadamente em ambientes diferentes. Este trabalho se dirige à solução do problema do aprendizado do robô em tempo real de como se locomover evitando colisões. Evitar colisões é essencial para a movimentação do robô móvel e faz parte de sua estratégia mais ampla, qualquer que seja o seu objetivo. Quanto à abordagem para se resolver esse problema, é investigada, em ambiente virtual, a utilização de Redes Neurais em Sistemas Classificadores, solução não encontrada na literatura. A utilização de redes neurais pretende aumentar o poder de descrição dos sistemas classificadores, substituindo suas regras binárias, limitadas em termos de seu poder de processamento, por uma ferramenta mais poderosa. É feita também a simulação de Sistemas Classificadores em sua forma convencional, proporcionando um termo de comparação para os Sistemas Classificadores com Redes Neurais. Um resultado interessante obtido é a suavização de trajetórias proporcionadas pelas redes neurais. Várias sugestões são apresentadas para pesquisas futuras. Foi necessária a elaboração de um Simulador, que é também parte integrante deste trabalho, para se conseguir os resultados pretendidos, o qual utiliza ambiente virtual em 2 dimensões e considera algumas das características de um robô real, o mini robô Khepera.

Abstract

Mobile robotics has as its fundamental goal to elaborate trajectories avoiding collisions, locating and reaching targets, power supply self sufficiency, objects transportation, etc. A solution providing autonomy to the robot for learning its behavior is contrasting with more classical approaches that require a previous model of the environment in which the robot is inserted. In the case in which is needed to put the robot in another environment, or in which the environment has its configuration varying with time, these models present a great deficiency, demanding that the designer recalculates the trajectory, or even making its use unviable. The use of learning procedures releases the designer of inserting detailed knowledge of the environment and gives to the robot the possibility of behaving well in different environments. This work is directed to the solution of the robot's real time learning problem concerned with how to move avoiding collisions. To avoid collisions is essential for the movement of a mobile robot and is part of its wider strategy, whatever could it be its objective. Related to the approach to solve this learning problem, this work investigates, in a virtual environment, the use of Neural Networks within classifier systems, a solution not found in the literature. The use of neural networks has the intention of giving more descriptive power to the classifier systems, substituting its binary rules, limited in terms of its processing power, by a more powerful tool. It's also made a simulation of the Classifier Systems in its conventional form, which provides a comparison reference for the Neural Networks Classifier Systems. An interesting result obtained is trajectory smoothing provided by the neural networks. A number of suggestions are presented for future research. To achieve the intended results it was required to elaborate a Simulator, which is also part of this work, utilizes a 2 dimensional environment and takes into account some of the characteristics of a real robot, the mini robot Khepera.

Índice

Capítulo 1: Introdução

1.1 Prólogo

A literatura de ficção científica várias vezes tem servido para o desenvolvimento de novas tecnologias. É o que ocorre com muitas das criações de Júlio Verne em seus livros como "As 20 mil Léguas Submarinas", que na época de sua publicação pareciam completamente impossíveis, e são hoje produtos comerciais encontrados em prateleiras de lojas de produtos eletrônicos. No futuro, aquilo que hoje é considerado ficção científica poderá também ser parte de nosso dia-a-dia.

Em "Eu, Robô", Isaac Asimov [**18**], apresenta robôs com capacidade de locomoção, ações especializadas, percepção refinada do ambiente, comunicação entre si e com os humanos, dotados de cérebros "positrônicos" capazes de pensar como nós. Arthur C. Clarke não deixa por menos com seu computador HAL, em "2001 - Uma Odisséia no Espaço" [**3**].

Causa um grande fascínio a possível existência dessas criaturas artificiais. Tanto pelo que elas representariam como conquistas da ciência e da tecnologia, como em decorrência das especulações a respeito das suas implicações práticas e filosóficas, relativas ao destino do ser Humano no planeta Terra. Estas implicações podem ser positivas ou negativas, quer seja imaginando-se robôs inteligentes mas meramente serviçais que tornem realidade a volta do Paraíso Perdido ou, ao contrário, robôs inteligentes com autonomia para formularem seus objetivos e que assim, eventualmente, possam se transformar em uma nova Espécie Dominadora.

Implicações filosóficas à parte, desde as últimas décadas do século XX pesquisadores têm procurado uma maneira (por enquanto ainda não bem sucedida) de formular e implementar técnicas de Sistemas Inteligentes que elevem o comportamento das máquinas, dotando-as de Inteligência Artificial (IA). A meta de tais sistemas seria que seu processamento interno pudesse ter alguma semelhança ao do apresentado pelos seres humanos, em termos de representação do conhecimento, aprendizado, capacidade de resolver problemas, etc.

Não dá para esconder uma certa tristeza em quem assistiu, há mais ou menos trinta e cinco anos, "*2001: A Space Odyssey*", de Stanley Kubrick [**23**], pensando tratar-se de uma ficção que, mesmo parcialmente, poderia tornar-se realidade em um futuro não muito distante. O desenvolvimento acelerado de novos conhecimentos e de tecnologias para a conquista do espaço dava a entender que era só questão de um pouco mais de tempo para se obter os princípios fundamentais dessa nova tecnologia de IA , e o processo de seu desenvolvimento poderia começar antes da virada do século. Mas isso não aconteceu.

Uma máquina com capacidade de pensar, da mesma maneira que nós humanos pensamos, ainda não foi elaborada. A despeito de vários mecanismos exibirem comportamento que poderia ser entendido como inteligente, ainda que com ressalvas, o processo interior que conduz a esse comportamento nada tem a ver com o processo de pensar. Conceitos como pensamento, inteligência, mente, consciência e emoção, que apesar de tão discutidos no âmbito das ciências cognitivas, como características subjetivas próprias do ser humano (e sem nenhum consenso, diga-se de passagem), raramente aparecem na agenda de pesquisadores mais pragmáticos como engenheiros e cientistas da computação. Apesar disso, um número considerável de novas teorias têm surgido no intuito de preparar o caminho para que isso possa acontecer. Algoritmos computacionais derivados das Neuro-ciências, da Genética e da Imunologia, do estudo dos fenômenos de complexidade e do caos, têm atraído a atenção de filósofos e engenheiros, não somente por sua inspiração em fenômenos da própria natureza, mas principalmente pelos resultados

pragmáticos que tais algoritmos são capazes de demonstrar. Dentre estes algoritmos, algumas metodologias com fundamentação na Semiótica - conjunto de conceitos formulados pelo filósofo americano e fundador da Teoria dos Signos Moderna, Charles Sanders Peirce (1839-1914), apresentam-se como novos horizontes de possibilidades a serem explorados.

Uma das áreas de aplicação em que o uso de algoritmos de inteligência computacional vem sendo empregado com muito êxito é a robótica, principalmente na formulação de estratégias de planejamento de ação para os robôs. Entretanto, apesar dos inúmeros progressos, tais teorias trazem contribuições somente parciais, considerando-se que ainda não fazem parte de uma teoria mais ampla da inteligência, onde elas se apresentariam eventualmente como partes integrantes de um todo maior.

Este panorama apresentado é fonte de motivação para este trabalho - conhecer mais a respeito dessas tecnologias e aplicá-las em um contexto prático, o contexto da robótica móvel.

Esta Dissertação de Mestrado é um pequeno passo, mas significativo, nesta empreitada. Com ele se inicia o processo de ganhar competência na direção desejada, elaborando uma plataforma de trabalho e pesquisa que, espera-se, possa permitir o desenvolvimento de sistemas de controle autônomo em tempo real com capacidade de aprendizado e outras manifestações características de seres inteligentes. A idéia é iniciar estes estudos utilizando um ambiente virtual de desenvolvimento de robôs móveis (simulação). Esta estratégia é corroborada por diversos estudos encontrados na literatura que indicam a simulação como melhor maneira de se adentrar no mundo da robótica móvel. É planejada para o futuro (possivelmente em um trabalho de Doutorado), a utilização dos mesmos algoritmos de controle empregados aqui, mas aplicados a robôs móveis reais, onde questões como ruído, tecnologias de sensores, não-conformidade dos modelos, etc. poderão ser analisadas.

1.2 Breve Histórico

Esta pesquisa teve início em um trabalho prático experimental desenvolvido no âmbito da disciplina "Computação Evolutiva" do curso de pós-graduação da FEEC/UNICAMP, onde foi proposto o uso de sistemas classificadores para o controle de um robô móvel em ambiente virtual bem simples, como ilustração para a compreensão do funcionamento dos sistemas classificadores. Ainda, como projeto da disciplina "Redes Neurais", os sistemas classificadores originais foram então sofisticados com a inserção de redes neurais em sua constituição, e novos resultados interessantes foram obtidos.

Com o auxílio dos Professores Von Zuben e Gudwin (co-orientador), e do Prof. Madrid (orientador), surgiram os "Sistemas Classificadores Com Redes Neurais", cujos resultados deram origem às publicações: *"Sistemas Classificadores com Redes Neurais (NNCS) : Aplicação ao Controle de um Veículo Autônomo Simulado Computacionalmente"* [**27**] e *"Neural networks in classifier systems (NNCS): An application to autonomous navigation"* [**28**].

A partir daí ficou claro que seria necessário um investimento maior no trabalho, utilizando um ambiente computacional de simulação mais adequado à realidade, implementando características mais sofisticadas ao Sistema Classificador e verificando o desempenho do Sistema Classificador com Redes Neurais em comparação ao clássico. Seria utilizado, como inspiração, o algoritmo reativo apresentado por Richards em *"Zeroth-order Shape Optimization Utilizing a Learning Classifier System"*, [33].

Apesar da não disponibilidade de robôs móveis nas instalações da FEEC/UNICAMP durante o desenvolvimento deste trabalho, já havia sido proposta a compra de mini robôs Khepera [**21**] (ver

Apêndice A - com fotos e informações técnicas), pelo Grupo de Robótica Móvel da UNICAMP. O Khepera é um mini robô móvel muito utilizado no meio científico para o desenvolvimento de pesquisas. Assim seria interessante a utilização de um simulador que permitisse a simulação do Khepera.

O passo a seguir foi a pesquisa de simuladores para essa finalidade, a qual foi feita de forma mais ampla para proporcionar uma visão de como a simulação estava sendo abordada na robótica. Simuladores para robôs são apresentados em [**19**], [**41**], [**26**], [**11**], [**2**] e [**31**], sendo as referências [**19**] e [**41**] relativas a simuladores para o Khepera, o "Khepera Simulator v. 2.0" e o "Webots 3.0", respectivamente. As referências [**37**] e [**34**] tratam de simuladores para manipuladores de robôs. Especificamente do IEEE, foram encontradas as seguintes referências para robôs autônomos móveis com aprendizado: [**10**], [**5**], [**40**], [**4**], [**30**], [**39**], [**42**], [**6**], [**24**] e [**35**], sendo a [**4**] referente à utilização do simulador Webots [**41**]. Outros trabalhos em que são utilizados simuladores podem ser encontrados em [**15**], [**9**] e [**13**].

Nem todos esses simuladores são disponibilizados gratuitamente e alguns nem são disponibilizados. Seu exame mostra a inexistência de um simulador que seja de propósito geral, o que, de certa forma, é de se esperar, pois cada autor focaliza seu trabalho para fornecer as facilidades que lhe são mais interessantes. Existem dois simuladores que podem ser utilizados para o Khepera:

- O Khepera Simulator versão 2.0 [**19**] é um *software* antigo, permite a construção de ambientes simples e acesso ao controle do robô. Foi descontinuado em favor do Webots , comercializado pela empresa Cyberbotics. Por isso não foi considerado.
- O Webots 3.0 [**41**] é um produto comercial e é possível obter-se uma versão *trial* gratuita, para Windows e bem documentada, sendo compatível com C e C++. O ambiente, os obstáculos e os robôs são configurados em VRML, inclusive com utilização de recursos de cortar, copiar e colar. Tem modelos já elaborados para diversos robôs, além do Khepera.

Como até então o simulador tinha sido implementado em Matlab, seria interessante aproveitar as funções já desenvolvidas acoplando-as ao Webots utilizando-se recursos e bibliotecas de C/C++ do Matlab. Dentre as opções de linguagens C/C++ que tínhamos a nossa disposição, o Webots só era compatível com o DEV-C++, software gratuito [**12**] e recomendado para ser utilizado com ele. Infelizmente as bibliotecas de C/C++ do MATLAB não são compatíveis com o DEV-C++. Por esse motivo, optou-se por desistir de usar o Webots.

Em virtude de não ter sido encontrado um simulador que servisse diretamente para os propósitos desta pesquisa, tornou-se mais apropriado o desenvolvimento de nosso próprio simulador - um que permitisse controle de todas as suas particularidades, e que seria desenvolvido de tal forma a facilitar a implementação de algoritmos de IA tanto para alto como para baixo nível.

1.3 Proposição e Metodologia

1.3.1 Proposição

A Robótica Móvel tem como sua meta fundamental a elaboração de trajetórias para robôs móveis, de tal forma que eles possam atingir seus objetivos. Dentre outros objetivos, tem-se a realização de movimentos sem colisões, a localização e alcance de alvos, a auto-suficiência do robô em termos de suprimento de energia, o transporte de objetos, etc.

A procura de uma solução que dê autonomia ao robô no aprendizado de seu comportamento se contrasta com outras abordagens mais clássicas que exigem um modelo prévio do ambiente em que o robô está inserido. Nestas abordagens, apesar de ser possível o cálculo da trajetória ótima para o

robô, existe uma grande dependência do robô com seu ambiente. Caso seja necessário inserir o robô em outro ambiente, ou caso o ambiente tenha componentes variantes no tempo, estes modelos apresentam grande ineficiência, exigindo do projetista todo um recálculo de trajetórias, ou até mesmo inviabilizando seu uso. A utilização de procedimentos de aprendizagem libera o projetista de ter que inserir no seu projeto conhecimentos detalhados do ambiente e dão ao robô a possibilidade de se comportar adequadamente em ambientes diferentes. É claro que a inserção de algum conhecimento é desejável e facilita o aprendizado.

Este trabalho se dirige à solução do problema do aprendizado em tempo real de como se locomover evitando colisões. Evitar colisões é essencial para a movimentação do robô móvel e faz parte de sua estratégia mais ampla, qualquer que seja o seu objetivo.

Quanto à abordagem para se resolver esse problema, pretende-se aprofundar o conhecimento da utilização de Redes Neurais em sistemas classificadores, apresentada inicialmente em "Sistemas Classificadores com Redes Neurais (NNCS): Aplicação ao Controle de um Veículo Autônomo Simulado Computacionalmente" [**27**] e "*Neural networks in classifier systems (NNCS): An application to autonomous navigation*" [**28**], ambas utilizando ambiente virtual bastante simplificado e, posteriormente em "*A Simulator using Classifier Systems with Neural Networks for Autonomous Robot Navigation*" [**29**], utilizando ambiente virtual através de um simulador que já considera algumas das características de um robô real, o mini robô Khepera.

Em relação à utilização de Redes Neurais em Sistemas Classificadores, cuja abordagem é mostrada no Capítulo 3, não foi encontrado nada similar na literatura. Existem apenas algumas semelhanças aparentes que serão comentadas. Por isso a sua implementação tem caráter investigativo, proporcionando um ponto de partida para trabalhos futuros.

Será também simulada a utilização de Sistemas Classificadores, que neste texto passarão a ser chamados de Sistemas Classificadores Convencionais, com a finalidade inicial de proporcionar um termo de comparação para os Sistemas Classificadores com Redes Neurais, sendo que, por si só, também se constitui em um dos objetivos deste trabalho.

O Simulador é também parte integrante deste trabalho, em função da necessidade da sua elaboração para a consecução dos resultados pretendidos.

A escolha de Sistemas Classificadores para a solução em tempo real não foi feita em detrimento de qualquer outra técnica de IA para resolver este problema. Pode ser entendido que as duas aplicações a serem implementadas, mais o Simulador, se constituem em uma plataforma inicial de estudos e pesquisas para Robótica Autônoma Móvel com Aprendizado.

Este trabalho, além de abrir novas possibilidades de pesquisa para o Grupo de Robótica Móvel, para o Laboratório de Robótica do DSCE e, quem sabe, para outros interessados, se constitui em um ponto de partida para o trabalho de Doutorado do autor, onde, além de aperfeiçoamentos no Simulador para adequá-lo de forma efetiva à realidade e para o controle do Khepera, serão verificadas alternativas de controle em tempo real, inclusive com a consideração de objetivos mais amplos.

Em resumo, este trabalho se propõe a:

1. Construir um Simulador com abordagem modular que permita de forma facilitada a reutilização de recursos já desenvolvidos e a inclusão de novos recursos necessários, tanto relativos a funcionalidades do robô Móvel quanto a algoritmos de IA.
2. Implementar um Sistema Classificador Convencional como sistema de controle para o robô móvel.

3. Implementar um Sistema Classificador com Redes Neurais e investigar a sua funcionalidade, procurando fornecer subsídios para pesquisas futuras.

A base teórica e os algoritmos para os itens 2 e 3 acima são apresentadas nos Capítulos 2 e 3.

1.3.2 Metodologia

As duas aplicações de Sistemas Classificadores serão verificadas através de experimentos com o Simulador em diversos ambientes e utilizando várias sementes aleatórias para a sua obtenção.

Serão apresentados os experimentos efetuados e seus resultados.

Serão apresentadas conclusões finais comparando os dois sistemas.

Quanto ao Simulador, ele será desenvolvido e testado passo a passo. Este trabalho é sua validação.

1.4 Sobre o Conteúdo deste Documento

A seguir, no Capítulo 2, é mostrada a abordagem de Sistemas Classificadores Convencionais com os seus principais conceitos e funcionalidades, bem como um algoritmo que implementa a Atribuição de Créditos e as características do Algoritmo Genético a ser utilizado. O Sistema Classificador Com Redes Neurais é introduzido no Capítulo 3, mostrando-se o papel que as redes neurais irão desempenhar e as modificações necessárias nos procedimentos do sistema convencional para recebê-las.

O Capítulo 4 aborda o tema do Simulador, onde a pertinência de se efetuar simulações bem como as suas vantagens e desvantagens são discutidas. É mostrado também o porquê da escolha do Matlab como linguagem para desenvolvimento do *software* do simulador. Nem todos leitores estarão interessados na leitura do Capítulo 5, Detalhes do Simulador, e ele não é essencial para o entendimento desta pesquisa. Este Capítulo, pela sua natureza, foi escrito em linguagem técnica familiar para pesquisadores que utilizam o Matlab e apresenta detalhes da implementação do simulador, contendo informações para a sua utilização, manutenção e inclusão de novos recursos.

O Capítulo 6 apresenta os experimentos das aplicações Sistemas Classificadores com Redes Neurais e Sistemas Classificadores Convencionais permitindo ter-se um entendimento dos seus desempenhos. Como resultados principais neste capítulo, tem-se o projeto do Sistema Classificador Convencional funcionando a contento e a investigação sobre Sistemas Classificadores com Redes Neurais trazendo contribuições relevantes.

Por fim, no capítulo 7, encontram-se as conclusões e as perspectivas de trabalhos futuros. Quanto aos Sistemas Classificadores com Redes Neurais, eles irão exigir criatividade na elaboração de sua arquitetura em pesquisas futuras e algumas sugestões são apresentadas. A expectativa principal das redes neurais apresentarem uma aproximação mais suave para as trajetórias do robô fica confirmada.

Nos apêndices A, B e C tem-se os detalhes técnicos referentes ao robô Khepera, detalhes sobre o funcionamento do algoritmo genético e o modelo cinemático do robô utilizado nas simulações. Ao final do trabalho, é apresentada a lista de referências bibliográficas utilizadas.

Capítulo 2: Sistemas Classificadores Convencionais

Neste capítulo é apresentada a abordagem dos Sistemas Classificadores Convencionais com os seus principais conceitos e funcionalidades, bem como um algoritmo que implementa a Atribuição de Créditos e as características do Algoritmo Genético a ser utilizado. O Sistema Classificador Com Redes Neurais é introduzido no Capítulo 3, mostrando-se o papel que as redes neurais irão desempenhar e as modificações necessárias nos procedimentos do sistema convencional para recebê-las.

2.1 Introdução

Os Sistemas Classificadores, neste texto chamados de Sistemas Classificadores Convencionais (SC), foram introduzidos por Holland [**17**] em 1975. Algumas referências a SC são o artigo de Booker de 1989 [**7**], para quem os sistemas classificadores se constituem em uma metodologia para criação e atualização evolutiva de regras, o livro do Goldberg de 1989 [**14**] e a tese de Richards em 1995 [**33**], que traz uma descrição bem detalhada sobre sistemas classificadores. É uma das abordagens que se inspiram nos processos da natureza [**20**] e recebem normalmente a denominação genérica de "Computação Natural".

O que mais interessa para este trabalho são as habilidades dos SC de poderem aprender em ambientes não estacionários e em tempo real, inclusive com a presença de ruído.

Uma maneira para se introduzir o conceito de SC é fazendo uma analogia com Sistemas Especialistas (SE). Um SE simples ou básico pode ser entendido como se constituindo de um conjunto de regras proposicionais: dada uma condição tem-se, por exemplo, uma conclusão ou ação. A **condição** forma o **antecedente** da regra e a conclusão ou **ação** o seu **conseqüente**. Um projetista deste SE simplificado tem como tarefa obter todas as possíveis informações do ambiente a que os sensores do sistema a ser controlado serão expostos e codificá-las nos antecedentes das regras. Para cada um dos possíveis antecedentes o projetista deverá também produzir o conseqüente, que irá conter as informações para o sistema de ação. Se o projetista de IA não tiver o conhecimento de como colher as informações e convertê-las em regras desta natureza, ele terá de consultar um especialista, o que normalmente tem que ser feito nesta abordagem, e daí vem o seu nome.

Desta forma um SE tem que conter um conjunto de regras que resolva todas as situações que o robô, ou outro sistema que irá controlar, terá que enfrentar. Este conjunto, uma vez programado, não se modifica - não existe aprendizado, nem adaptação. Assim, se o robô encontrar uma situação diferente, ou se for colocado em outro ambiente, não terá normalmente condições de continuar sendo bem sucedido. Haverá a necessidade de re-programar o seu componente de IA, incorporando as novas regras necessárias e, quem sabe, excluindo as não mais necessárias.

Os SC também utilizam um conjunto de regras com antecedente e conseqüente, mas elas não precisam ser especificadas por um especialista, pois o SC tem condições de aprender e se adaptar.

Podem partir de um conjunto de regras aleatórias e chegar ao conjunto de regras válidas para o seu comportamento. Isto é feito valorizando-se, através de realimentação do ambiente, as boas regras, através do aumento de sua **força**, conceito associado a cada uma das regras nos SC. Regras não apropriadas têm as suas forças diminuídas. Uma regra nos SC é chamada de **classificador**. A força de um classificador representa a sua utilidade e a tendência é que classificadores com pouca força sejam excluídos. Além desse processo de seleção de regras, os SC também apresentam um mecanismo de geração evolutiva de novas regras, utilizando-se um Algoritmo Genético (AG) para a sua obtenção. O AG é normalmente aplicado em uma sub população das regras, constituída pelas melhores, utilizando como função de *fitness* a força.

Para se entender todo o processo de aprendizagem e atualização de regras do SC a sua arquitetura pode ser estudada em três níveis hierárquicos:

- Nível Básico: cuida do processamento das mensagens e preparação para os outros níveis.
- Atribuição de Créditos: faz a seleção dos classificadores alocando valores maiores para as forças dos classificadores que têm maior utilidade.
- Algoritmo Genético: utiliza os melhores classificadores do SC para, utilizando as suas forças como função de *fitness*, formular novos classificadores por um processo evolutivo.

Fica claro que a capacidade de um SC aprender e se adaptar ocorre através dos procedimentos de Atribuição de Créditos e da aplicação do Algoritmo Genético. Neste Capítulo são apresentados os 3 níveis da arquitetura dos Sistemas Classificadores Convencionais e os algoritmos que serão utilizados para a sua implementação.

2.2 Nível Básico da Hierarquia

A Fig. 2.1 ilustra o Nível Básico da arquitetura dos SC. O SC interage com o ambiente através de suas Interfaces de Entrada e de Saída.

As informações do ambiente são coletadas através de sensores e a interface de entrada as codifica em forma de mensagens que serão colocadas na Lista de Mensagens (LM). Uma mensagem na LM é um arranjo de caracteres composto de duas partes: os caracteres iniciais identificam quem enviou a mensagem e os seguintes identificam ocorrências no ambiente, detectadas pelos sensores. Por exemplo, uma mensagem pode ter a seguinte constituição:

1	0	0	1	1	0	1	0	0	1	0

No exemplo hipotético acima, para um robô móvel, os dois primeiros bits "10" identificam a Interface de Entrada como a autora da mensagem. Os 9 bits seguintes descrevem a existência de obstáculos no campo de visão do veículo, cada bit correspondendo a uma região, onde o valor 0 corresponde a não haver obstáculo e 1 indica a existência de obstáculo.

Observe que não somente a interface de entrada envia mensagens para a LM, mas também a Lista de Classificadores (LC). O porquê disso será explanado na seção em que é descrito o mecanismo de Atribuição de Créditos. Por ora esta possibilidade não será considerada.

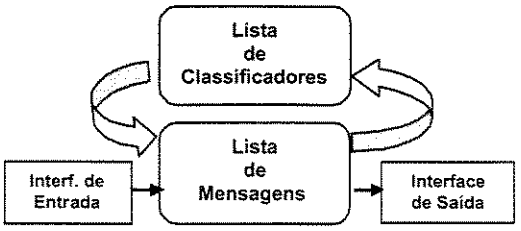


Figura 2.1: Sistema Classificador, Nível Básico

A LC é composta por um conjunto de Classificadores (regras), constituídos de um arranjo de caracteres dividido em 3 partes. Na primeira, temos os caracteres que identificam o tipo de conseqüente ou ação do Classificador - por exemplo, se ele será uma nova mensagem dirigida à LC, ou uma mensagem à interface de saída. Na segunda, temos caracteres que constituem a sua condição, também chamada de antecedente. Na terceira, estão os que correspondem ao conseqüente. Sendo o classificador escolhido para enviar a sua mensagem à LM, ela será composta pela sua primeira e terceira partes. A seguir, temos um exemplo de um classificador:

0	1	#	1	#	0	1	#	0	#	#	0	1

No exemplo acima, os dois primeiros bits, por uma convenção arbitrária, indicam que o conseqüente é uma mensagem para a interface de saída. Os 9 bits seguintes correspondem à condição, onde nota-se a presença do caractere #, significando ***don't care***, ou **tanto faz**. Se a condição de um classificador for igual à mensagem da interface de entrada, ocorreu um ***matching***. Os símbolos # valem 0 ou 1, favorecendo o *matching*. Na terceira parte, o conseqüente do classificador, pode ser de dois tipos: uma mensagem que será usada pelos classificadores no próximo ciclo, ou uma mensagem para a interface de saída providenciar uma determinada ação.

Os Classificadores que apresentarem *matching* com alguma(s) mensagem(ns) da LM irão participar de uma competição - procedimento de Atribuição de Créditos, e o vencedor coloca a sua mensagem na LM. Seguindo o exemplo, como houve *matching* entre o antecedente do classificador e a informação dos sensores codificada na mensagem de entrada, e supondo que este classificador venceu a competição, a sua mensagem será colocada na LM:

0	1	0	1

Por uma convenção arbitrária esta mensagem é dirigida à interface de saída. Isto está representado nos dois bits iniciais, que são também os dois iniciais do arranjo de caracteres do classificador. Os dois bits finais da mensagem correspondem ao conseqüente do classificador.

A interface de saída, da mesma forma que a LC, fica constantemente monitorando a LM, e uma vez que verifique a existência de mensagem para si, recolhe-a da lista, executando as instruções que nela estiverem contidas. Isto é feito decodificando a mensagem recebida e transformando o seu conteúdo em comandos para os atuadores.

Esse *loop* inicial em nada difere de um mecanismo de base de regras proposicionais. Ele pressupõe que a LC represente um conjunto de regras que possua o conhecimento necessário para processar as informações vindas do ambiente. Entretanto, ao contrário de uma base de regras usual, onde o conhecimento é adquirido de um especialista, nesta técnica as regras, ou classificadores, são introduzidas por meio de um mecanismo evolutivo.

2.2.1 Sistemas classificadores reativos e dinâmicos

Da maneira apresentada, temos um processamento de mensagens apropriado para SC Dinâmicos, nos quais, além da possibilidade de respostas imediatas aos estímulos de entrada, existe também a possibilidade de construção de uma seqüência de mensagens que irão produzir uma ação em tempo futuro. Por exemplo, seja um sensor percebendo um objeto distante. A sua mensagem na LM após ser processada pela LC tem a possibilidade de ser uma mensagem na LM dirigida novamente à LC. Esta mensagem, no próximo ciclo, será capturada pela LC podendo resultar em uma nova mensagem para a LM ou para a interface de saída.

Um algoritmo que permite a implementação da Atribuição de Créditos para um SC Dinâmico é o chamado algoritmo *Bucket Brigade* e foi apresentado por Booker *et. al.*, em 1989 [7]. Utiliza a idéia de que os classificadores podem postar mensagens que irão gerar novas mensagens para a LC, formando um encadeamento até gerar uma ação. O *Bucket Brigade* permite a existência de múltiplas mensagens na LM, sendo algumas delas para a LC e algumas para a interface de saída, a qual terá a atribuição adicional de resolver possíveis conflitos oriundos das mensagens a ela destinadas.

Assim, para estes SC dinâmicos é necessário que se faça a identificação do destino das mensagens a serem postadas na LM. Neste caso tem-se a lista de classificadores constituída por classificadores de vários tamanhos. Seguindo o exemplo usado anteriormente, eles poderão ter a parte inicial de seu arranjo de caracteres, que identifica o destinatário da mensagem, de mesmo tamanho. Os antecedentes de todos os classificadores têm o mesmo tamanho, 9 caracteres. Quanto ao conseqüente, terá dois caracteres se for para a interface de saída e terá nove caracteres se for para a LM. Assim, quando a LC coloca uma mensagem na LM dirigida à LC, ela tem a aparência da mensagem produzida pela interface de entrada.

Neste trabalho utilizamos a abordagem reativa para SC, significando que para cada valor de entrada o SC produz uma saída e a LC produz sua mensagem apenas para a interface de saída. O modelo segue, em linha geral, o especificado por Richards em [**33**].

Nos SC reativos os classificadores têm todos o mesmo tamanho, a LM contém uma mensagem por vez e não é necessário ter-se a identificação do destinatário da mensagem. O ciclo de processamento se encarrega de endereçar as mensagens. Assim, a mensagem de entrada é dirigida à LC e a mensagem da LC é destinada à interface de saída.

2.2.2 Especificidade e força

Considerando o nosso exemplo, para o SC reativo as mensagens não necessitam de identificação. Então, a mensagem da interface de entrada colocada na LM pode ser apresentada assim:

0	1	1	0	1	0	0	1	0

Supondo que a lista de classificadores contenha 5 classificadores, eles poderiam ser:

#	1	#	0	1	#	0	#	#	0	1
#	#	0	1	1	1	0	#	1	0	1
0	1	1	1	#	0	1	1	1	0	0
0	1	1	0	1	0	0	1	0	0	0
1	0	0	#	#	1	#	#	1	1	0

O primeiro e o quarto classificadores dão *matching* com a mensagem da interface de entrada. Nota-se que o quarto classificador não tem nenhum caractere #, por isso ele é um classificador **específico** para essa mensagem. O primeiro classificador, por causa dos caracteres tanto faz, é um classificador **genérico**, ele pode dar *matching* também com mensagens diferentes desta. Este é o conceito de **especificidade (E)**, que tem sua medida determinada pelo número de caracteres diferentes de tanto-faz do antecedente do classificador dividido pelo comprimento do antecedente.

$$\mathbf{E = (na - nt)/na} \qquad \textbf{(2.1)}$$

onde

na: comprimento do antecedente

nt: nº de # do antecedente

Os conseqüentes dos dois classificadores também são diferentes. Eles irão competir e o vencedor irá enviá-lo para a interface de saída. Isto é feito pela Atribuição de Créditos. Se o primeiro vencer, esta será a sua mensagem para a interface de saída:

Se o quarto ganhar, será esta:

Como o conseqüente tem dois caracteres, que podem ser 0 ou 1, é possível mapear 4 mensagens diferentes para a interface de saída.

Inicialmente os classificadores normalmente são gerados de forma aleatória e vão ter um valor associado a eles que é a sua ***strength*** ou **força**. Também normalmente todos eles recebem o mesmo valor, no início, para a sua força, a qual será alterada a cada iteração pela Atribuição de Créditos.

2.3 Níveis Superiores da Hierarquia

Os classificadores são normalmente criados inicialmente de forma aleatória. É claro que sempre que for possível introduzir algum conhecimento inicial, ou mesmo durante o aprendizado, o aprendizado é facilitado. Então, é necessário descobrir quais classificadores geram comportamentos apropriados, criar novos classificadores e descartar aqueles que sejam ou tornem-se inapropriados, considerando a variabilidade do ambiente. Esse mecanismo de aprendizagem de regras é implementado pelo algoritmo de **Atribuição de Créditos** (AC) e pelo **Algoritmo Genético** (AG), que atuam nos níveis superiores da hierarquia estrutural do sistema classificador.

2.3.1 Atribuição de Créditos

O modelo segue, em linha geral, o especificado por Richards em [**33**]. São feitas algumas alterações para adequá-lo às necessidades específicas encontradas.

Cálculo da aposta

No nível básico da hierarquia são identificados os classificadores que dão *matching* com a mensagem de entrada. O *matching* no SC é verificado observando-se cada caractere da mensagem de entrada comparado com o caractere correspondente do antecedente do classificador, sendo que o caractere # do antecedente sempre garante *matching*. Ainda, para o SC, a especificidade é calculada pela expressão 2.1.

Os classificadores com *matching* selecionados participam de uma competição efetuando uma aposta dada pela expressão 2.2. O classificador com maior aposta ganha a competição.

$$B = k0.(k1 + k2.E^{k3}).F \qquad \textbf{(2.2)}$$

onde:

B: Aposta do classificador

k0: Coeficiente de aposta, valor positivo menor ou igual a 1.

k1: Valor positivo, menor ou igual a 1, que corresponde à participação da parte não referente à especificidade na aposta..

k2: Valor positivo, menor ou igual a 1, correspondente à participação da especificidade na aposta.

k3: Parâmetro controlando a importância da especificidade para determinar a aposta (padrão = 1).

E: Especificidade.

F: Força do classificador

Seja um exemplo como o visto na seção anterior, agora apresentando para cada classificador, na sua parte final, o valor de sua força. Imaginando que esses classificadores acabaram de ser criados e receberam todos como valor inicial para a sua força o valor 50:

#	1	#	0	1	#	0	#	#	0	1	50
#	#	0	1	1	1	0	#	1	0	1	50
0	1	1	1	#	0	1	1	1	0	0	50
0	1	1	0	1	0	0	1	0	0	0	50
1	0	0	#	#	1	#	#	1	1	0	50

Lembrando que o primeiro e o quarto classificadores deram *matching*, para calcular a sua aposta é preciso antes calcular a sua especificidade utilizando a expressão (2.1). Sejam E1 e E4 as especificidades desses classificadores:

E1 = (9 – 5)/9 = 0.44

E4 = (9 – 0)/9 = 1

Se forem adotados os valores k0 = 0.1, k1 = 0, k2 = 1 e k3 =1, a expressão (2.2) resulta em:

$$B = 0.1E.F$$

e calculando as suas apostas B1 e B4 tem-se:

B1 = E1.F1 = 0.1x 0.44x50 = 2.2

B4 = E4.F4 = 0.1x1x50 = 5.0

com o quarto classificador ganhando.

#	1	#	0	1	#	0	#	#	0	1	50
#	#	0	1	1	1	0	#	1	0	1	50
0	1	1	1	#	0	1	1	1	0	0	50
0	1	1	0	1	0	0	1	0	0	0	50
1	0	0	#	#	1	#	#	1	1	0	50

Taxa de aposta

O classificador vitorioso é aquele que oferece maior aposta, e este valor é integralmente subtraído de sua força. Os outros classificadores que participaram da competição multiplicam a sua aposta pela taxa de aposta, **TaxaBid**, antes de fazerem essa subtração. Assim, a força dos classificadores após a aposta é calculada pelas expressões:

$$\mathbf{F(t+1) = F(t) - B} \qquad \text{para o classificador ganhador} \qquad \mathbf{(2.3)}$$

$$\mathbf{F(t+1) = F(t) - TaxaBid.B} \qquad \text{para os outros apostadores} \qquad \mathbf{(2.4)}$$

onde

TaxaBid: Taxa de aposta

t iteração em processamento

Continuando com o exemplo e adotando TaxaBid = 0.8, as forças dos classificadores serão obtidas por:

$F1(t+1) = 50 - 0.8x2.2 = 48.24$

$F4(t+1) = 50 - 5.0 = 45.0$

Resultando em:

#	1	#	0	1	#	0	#	#	0	1	48.24
#	#	0	1	1	1	0	#	1	0	1	50.00
0	1	1	1	#	0	1	1	1	0	0	50.00
0	1	1	0	1	0	0	1	0	0	0	45.00
1	0	0	#	#	1	#	#	1	1	0	50.00

Taxa de vida

Finalmente, em cada iteração, todos os classificadores ficam sujeitos a um decréscimo em sua força devido a uma taxa de vida determinada por:

$$TxVida = 1 - (1/2)^{(1/n)} \qquad \mathbf{(2.5)}$$

onde:

n: Vida média, medida em iterações

que diminui a sua força utilizando-se a expressão abaixo.

$$\mathbf{F = (1 - TxVida)*F(t+1)} \qquad \mathbf{(2.6)}$$

Adotando uma vida média n = 100, pela expressão (2.5) obtém-se:

TxVida = 0.0069

e utilizando-se este valor na expressão 2.6 aplicada a todos os classificadores, obtemos o novo valor para a força:

#	1	#	0	1	#	0	#	#	0	1	47.91
#	#	0	1	1	1	0	#	1	0	1	49.65
0	1	1	1	#	0	1	1	1	0	0	49.65
0	1	1	0	1	0	0	1	0	0	0	45.00
1	0	0	#	#	1	#	#	1	1	0	49.31

Recompensa ou punição

Até este ponto, tivemos a determinação por competição do classificador ganhador, os classificadores que participaram tiveram que pagar uma taxa de aposta e todos ficaram sujeitos a uma taxa de vida. A atribuição de créditos prossegue para sua etapa final, onde irá colher um *feedback* do ambiente que determinará se a ação resultante do conseqüente do classificador ganhador teve resultado bom ou ruim. O resultado bom dará uma recompensa ao classificador ganhador e o resultado ruim o penalizará utilizando um valor. No caso da recompensa, será restituída a aposta do classificador, além de ser somada a recompensa à sua força.

Assim, temos as seguintes expressões para a obtenção da força pela aplicação do mecanismo de recompensa e punição:

$$\mathbf{F = F + B + R} \quad \text{para recompensa} \qquad \textbf{(2.7)}$$

$$\mathbf{F = F - P} \quad \text{para punição} \qquad \textbf{(2.8)}$$

onde

R valor especificado para a recompensa

P valor especificado para a punição

Não ocorrendo *matching*

Quando não ocorre *matching* o algoritmo do Richards introduz um novo classificador com antecedente igual ao valor codificado pelos sensores, com o conseqüente gerado aleatoriamente e com força igual à média das forças dos classificadores na iteração em curso. Este procedimento corresponde, de certa forma, a uma introdução automatizada de conhecimento e foi implementado desta forma nos SC.

2.3.2 Principais diferenças na implementação do algoritmo

Considerando a abordagem definida por Richards, o algoritmo utilizado neste trabalho tem as seguintes diferenças.

Não ocorrendo mudança na mensagem da interface de entrada

Quando em uma dada iteração a mensagem da interface de entrada repete a da iteração anterior o mesmo classificador irá vencer e determinará a nova ação, recebendo a recompensa por isso. Esta ocorrência é comum em Robótica Móvel, por exemplo, quando existe um trecho sem obstáculos, o que faz com que o classificador responsável pelas mesmas ações sucessivas tenha sua força aumentada indevidamente, podendo ter implicações negativas caso ele seja genérico e leve o robô a uma colisão. A punição da última iteração fica quase que sem efeito em face das recompensas das anteriores. Assim não serão recompensadas ações desta natureza, considerando recompensa apenas quando for requerida mudança de atitude.

Punição

Quando ocorre uma colisão, são punidos classificadores que ganharam as últimas iterações. Isto é feito respeitando-se o parâmetro nº máximo de classificadores a serem punidos **NGP**, limitado também pelo número de iterações desde a última colisão. Estes ganhadores terão a Recompensa e a Taxa de Aposta subtraídas de sua força. Além disso será também aplicada a eles uma punição adicional subtraindo da sua força o valor obtido pela seguinte expressão:

(i/m)*P **(2.9)**

P: valor da punição

NGP: nº máximo de classificadores a serem punidos

m: número de últimos ganhadores a punir, com m <= NGP

i: valor de 2 a m

O classificador responsável pela colisão é punido independentemente , por isso i vai de 2 a m.

Nota-se que os classificadores ganhadores de iterações mais anteriores serão punidos com mais intensidade. Desta forma está sendo considerado que o resultado obtido na iteração atual foi influenciado pelas iterações anteriores (memória de aprendizagem).

Esta abordagem é necessária no caso do controle da orientação do robô pelo fato de que se não forem tomadas medidas quanto à seqüência de ações que o levaram a colidir, ou seja, se for punida somente a ultima ação, os classificadores que ganharam anteriormente estarão tendo a sua força aumentada, o que acabará por forçar este comportamento. Assim o robô fica dependendo de mudança acentuada de direção, que, eventualmente, pode estar fora dos limites permitidos. Isto é uma espécie de armadilha, e a sua eliminação aumenta o desempenho e ajuda a garantir que o aprendizado possa ocorrer.

Após a punição, caso a força do classificador atinja valor menor do que 1 ela é zerada, o que, no modelo adotado, elimina a possibilidade deste classificador participar de novas apostas.

2.3.3 Algoritmo Genético (AG)

O processo de descoberta de regras em sistemas classificadores utiliza um algoritmo genético.

Basicamente, o AG seleciona os classificadores com as maiores forças ou *strengths* como pais, gerando novos indivíduos por meio da recombinação e mutação destes. Os novos classificadores gerados tomam o lugar dos mais fracos, modificando o conjunto de classificadores do sistema [**17**].

O AG (ver Apêndice B), seleciona um determinado número 'n' de melhores classificadores e aplica o algoritmo *Roulette Wheel*, apresentado por Goldberg em 1989 [**14**] para formar os pares.

No SC é utilizado *crossover* de 1 ponto e mutação simples. No *crossover* de 1 ponto é selecionado aleatoriamente um ponto nos cromossomos dos pais, formados por classificadores, e os genes são permutados como indicado na Fig. 2.2. Na mutação simples, é selecionado um ponto de um cromossomo e o seus gene é trocado de valor, se for um passa para zero e se for zero passa para um.

Figura 2.2 Crossover de um ponto

São utilizados dois critérios para disparar o AG: número de iterações, ou número de colisões, aquele que ocorrer primeiro:

Número de colisões: muitas colisões podem indicar insuficiência de classificadores adequados, o que seria resolvido acionando-se o AG.

Número de iterações: se não há colisões não significa que o comportamento não possa ser melhorado.

2.4 Término da Aprendizagem

O mecanismo dos Sistemas Classificadores pode ter alternativas de operação considerando que irá existir uma fase de aprendizado e depois dela, com o conhecimento já adquirido, as hierarquias superiores podem ser desligadas ou terem seu funcionamento modificado. Neste caso é interessante deixar um gatilho para dispará-los de novo caso o robô volte a apresentar uma determinada taxa de colisões.

Capítulo 3: Sistemas Classificadores com Redes Neurais

No Capítulo 2 foi mostrada a abordagem dos Sistemas Classificadores Convencionais com os seus principais conceitos e funcionalidades. Neste capítulo é introduzido o Sistema Classificador Com Redes Neurais, mostrando-se o papel que as redes neurais irão desempenhar e as modificações necessárias nos procedimentos do sistema convencional para recebê-las. São também apresentadas algumas semelhanças com outras abordagens, apesar de não ter sido encontrada na literatura nenhuma abordagem equivalente à aqui apresentada

3.1 Sistemas Classificadores com Redes Neurais

O Sistema Classificador com Redes Neurais (SCRN) está inteiramente fundamentado no Sistema Classificador Convencional (SC). A modificação no SC para se obter o SCRN é feita substituindo-se o seu classificador, apresentado no Capítulo 2, por um classificador neural, mostrado na Fig. 3.1 e explicado a seguir.

Nos SCRN as regras binárias existentes nos classificadores convencionais são substituídas por redes neurais. São utilizadas duas redes neurais para substituir cada classificador convencional, uma para o antecedente e outra para o conseqüente da regra. O antecedente da regra é responsável pela avaliação da adequabilidade do classificador a uma dada situação e a rede neural que o substitui recebe o nome de **Rede de Avaliação**. O conseqüente da regra é responsável pela determinação da ação causada pelo classificador e a rede neural que o substituiu é chamada de **Rede de Ação.** A Fig. 3.1 apresenta a constituição do Classificador Neural, evidenciando as suas duas redes neurais.

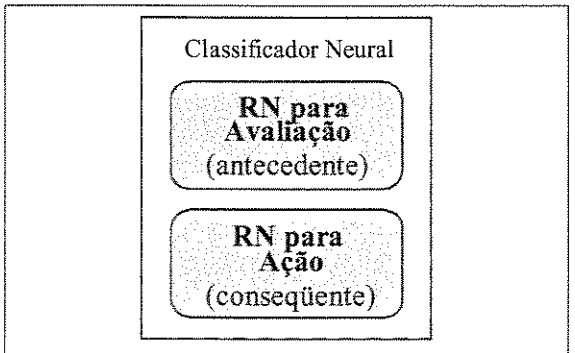


Figura 3.1: O classificador neural de um SCRN.
O classificador neural é constituído de duas redes neurais, uma para avaliação e outra para ação.

Em um SCRN, classificadores neurais serão processados de maneira equivalente à que os classificadores convencionais o são em um SC convencional, passando pelos mesmos processos nas hierarquias superiores do sistema, onde serão selecionados e evoluídos.

3.1.1 Por que usar redes neurais

A utilização de redes neurais nos SC pretende aumentar o poder de descrição dos sistemas classificadores, substituindo as regras binárias, limitadas em termos de seu poder de processamento, por uma ferramenta mais poderosa. O uso de redes neurais como classificadores de regiões contínuas vem desde os primórdios de sua proposição como ferramenta computacional.

Uma rede neural do tipo Perceptron com 3 camadas, Fig. 3.2, pode classificar com precisão determinada (dependendo-se do número de neurônios em suas camadas internas) qualquer tipo de região (inclusive regiões não conexas) [16]. Da mesma maneira, redes do tipo Perceptron com 3 camadas constituem aproximadoras universais de funções [16] (também com precisão determinada, dependendo-se do número de neurônios em suas camadas internas) o que as torna particularmente interessantes para a determinação de qualquer tipo de função de atuação.

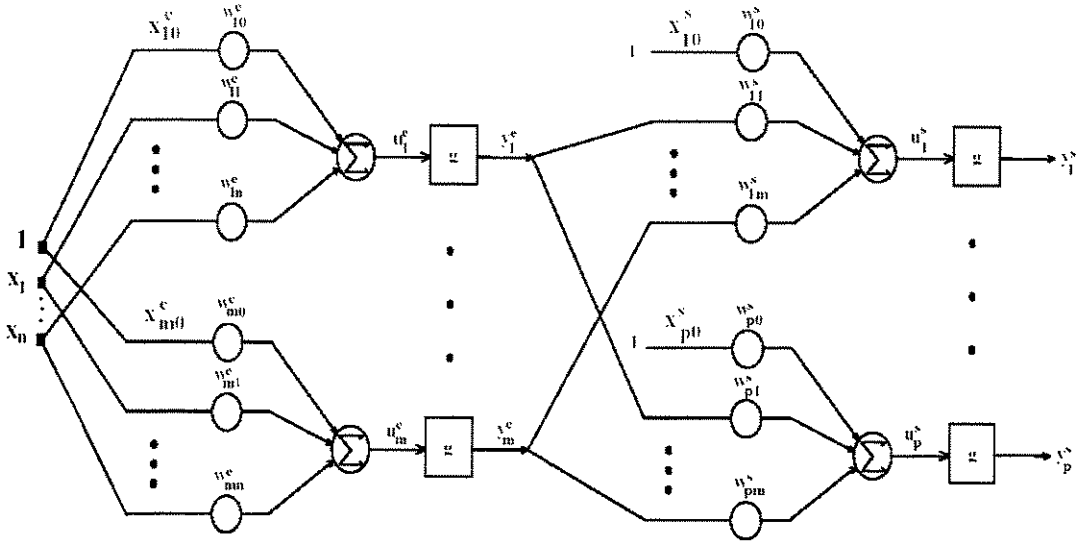


Figura 3.2: Perceptron de três camadas (uma camada intermediária)
São consideradas redes Perceptron para constituírem as redes neurais do classificador neural.

Se analisarmos com cuidado o funcionamento de um classificador, veremos que seu antecedente avalia se a regra (o classificador) deve ou não ser utilizada dada uma determinada situação. Isso é feito no classificador convencional por meio do *pattern matching* entre o antecedente a mensagem. Essa tarefa é substituída no classificador neural, por uma rede neural com a mesma finalidade. A diferença é que ao invés de limitar-se às regiões cobertas por uma regra binária, tem-se a flexibilidade de classificação que uma rede neural proporciona. O mesmo se dá com relação ao conseqüente. No classificador convencional, o conseqüente arbitra uma ação única (constante). Utilizando uma rede neural, ao contrário, pode-se fazer com que a saída do sistema seja uma função qualquer arbitrária (implementada pela rede neural) da entrada, que pode inclusive ser constante!

Assim, vê-se que o uso de um classificador neural não altera a funcionalidade básica de um classificador, mas tão somente aumenta seu poder descricional, permitindo que um maior número

de regiões (situações) seja considerado, e também modificando seu poder de atuação, permitindo que funções mais sofisticadas sejam utilizadas na atuação do sistema.

3.1.2 O Classificador Neural

As redes neurais do classificador neural são perceptrons de 3 camadas, como na Fig. 3.2. Estas redes têm tantos neurônios na camada de entrada quantas sejam as entradas, mais um neurônio para o sinal de polarização. A camada de saída tem um neurônio para cada saída. A camada intermediária deve ter uma quantidade de neurônios que permita a aproximação desejada. As funções de ativação g dos neurônios da camada intermediária são normalmente sigmodais, sendo que a dos neurônios da camada de saída usualmente são lineares. Cada classificador neural é representado no SCRN por um vetor, cujo antecedente é formado pelos pesos da rede neural de **avaliação** e cujo conseqüente é formado pelos pesos da rede neural de **ação**.

A Fig. 3.3 ilustra o papel desempenhado pelo classificador neural nos SCRN. Todos os classificadores neurais recebem nas entradas das suas redes de ação e de avaliação a mensagem de entrada. Em primeiro lugar, todas as redes de avaliação (de todos os classificadores neurais do sistema) processam essa mensagem, produzindo o que seria equivalente ao *matching* e à especificidade de cada um dos classificadores neurais. Estes valores são enviados para o processamento da Atribuição de Créditos, onde haverá competição, sendo que apenas um dos classificadores neurais vencerá.

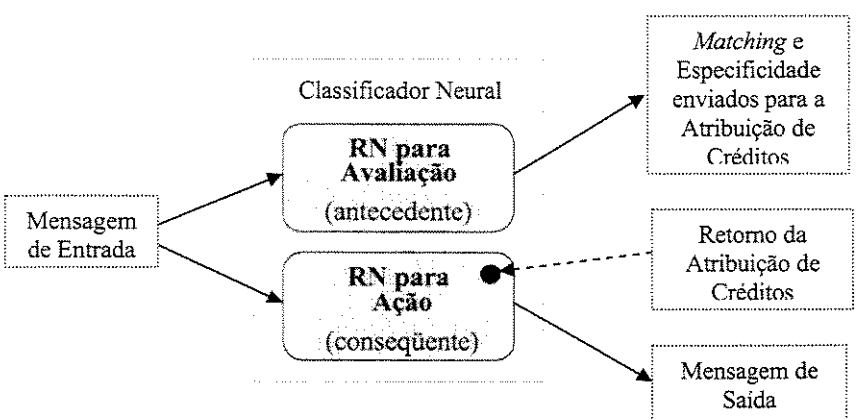


Figura 3.3: O funcionamento do classificador neural.

A rede neural de ação do classificador neural vencedor recebe um sinal de retorno da Atribuição de Créditos, tendo permissão para processar a mensagem de entrada, e, então, produz a sua mensagem de saída. Vê-se que todas as redes de avaliação processam a mesma mensagem de entrada, mas apenas uma rede neural, a rede de ação do classificador vencedor, terá permissão para processá-la e fornecer a saída.

Assim existe uma grande similaridade com o sistema classificador convencional, mas na realidade existem mudanças no comportamento, que serão examinadas posteriormente no Capítulo 6.

3.1.3 Configuração das redes neurais

As redes neurais de ação e de avaliação são do tipo perceptron com uma camada intermediária de um ou mais neurônios. A camada de saída da rede de avaliação tem um ou dois neurônios, que fornecem o *matching* e o nível de especificidade, considerados iguais no caso de apenas um neurônio. A camada de saída da rede de ação tem um neurônio, com a função de comandar a orientação do robô. A função de ativação da camada intermediária é a tangente hiperbólica, e a camada de saída tem ativação linear.

A entrada das redes é a mensagem da Interface de Entrada mais polarização, sendo, portanto, dependente do nº de sensores e de como as suas informações são codificadas.

Seja um exemplo para ilustrar a configuração de um classificador neural. A especificação da camada de entrada das redes de ação e de avaliação é dependente da mensagem de entrada. Para um sensor que tem condições de detectar 3 regiões do ambiente, a mensagem codificada pela interface de entrada será composta por um vetor com 3 elementos, cada elemento tendo um valor binário, 0 para a existência de obstáculo na região, 1 para a ausência:

{ E1 E2 E3 } mensagem de entrada

Esta mensagem determina o número de neurônios da camada de entrada, que será 4, 3 para o vetor de entrada e 1 para a polarização. A mensagem enviada pelos neurônios da camada de entrada para o neurônio da camada intermediária será acrescida de um elemento, correspondente à polarização, formando o vetor:

{ E1 E2 E3 1 } mensagem para a camada intermediária

Considerando apenas 1 neurônio na camada intermediária das redes de ação e de avaliação, este terá um peso para cada neurônio de entrada, ou seja, 4 pesos. Considerando 1 neurônio na camada de saída para a rede de avaliação e de ação, este terá 1 peso para cada neurônio da camada intermediária, mais 1 peso para polarização, resultando em 2 pesos:

Sejam os pesos do neurônio da camada intermediária da rede de avaliação representados por XI1, XI2, XI3 e XI4, e da rede de ação por YI1, YI2, YI3 e YI4. Sejam os pesos da 3ª. camada, ou camada de saída, da rede de avaliação representados por XS1 e XS2 os da rede de ação por YS1 e YS2.

O antecedente do classificador neural é representado por um vetor constituído pelos pesos da rede de avaliação, colocando-se em seqüência os pesos do neurônio da camada intermediária e os da camada de saída. De forma análoga é formado o conseqüente do classificador neural.

{ XI1 XI2 XI3 XI4 XS1 XS2 } antecedente do classificador neural

{ YI1 YI2 YI3 YI4 YS1 YS2 } conseqüente do classificador neural

Finalmente, o classificador neural tem a sua representação formada pelo vetor constituído pelos elementos do antecedente e do conseqüente:

{ XI1 XI2 XI3 XI4 XS1 XS2 YI1 YI2 YI3 YI4 YS1 YS2 } classificador neural

Se a rede de avaliação tivesse dois neurônios na camada de saída, cada um desses neurônios teria dois pesos, podendo ser identificados por XS11 e XS12 para o primeiro neurônio e XS21 e XS22 para o segundo neurônio, o que resultaria em

{ XI1 XI2 XI3 XI4 XS11 XS12 XS21 XS22 YI1 YI2 YI3 YI4 YS1 YS2 }

para o seu classificador neural.

3.2 Semelhanças com outras abordagens

Apesar de não ter sido encontrada na literatura nenhuma abordagem equivalente à aqui apresentada, existem algumas abordagens que, mesmo não sendo equivalentes, apresentam algumas semelhanças quanto aos procedimentos aqui empregados. Vale a pena portanto destacá-las aqui para efeito de comparação:

3.2.1 Evolutionary Reinforcement Learning

Ackley e Littman em [1], formulam o conceito de *Evolutionary Reinforcement Learning* (ERL). Uma população de agentes em um ambiente com predadores e alimento tem que evoluir para sobreviver. Cada agente tem seu comportamento comandado pela atuação de duas redes neurais, uma para avaliação e outra para ação. A rede de avaliação faz um mapeamento dos valores de entrada dos sensores dos agentes em um escalar real, representando o grau de *goodness* da situação atual. Os pesos desta rede são herdados e não se modificam durante a vida do indivíduo. A rede de ação faz um mapeamento das entradas dos sensores do agente em comportamento, sendo que seus pesos são herdados e podem ser modificados durante a sua vida. A modificação dos pesos é feita utilizando um algoritmo de *Reinforcement Learning*, através de um sinal de reforço da rede de avaliação e, pelo fato dos pesos serem genes e estarem sendo modificados durante a vida, Ackley e Littman acrescentaram *Evolutionary* ao *Reinforcement Learning*.

No SCRN a rede de avaliação fornece sua saída ao mecanismo de Atribuição de Créditos e não diretamente à rede de ação. Além disso, a população de classificadores do SCRN não tem comportamento isolado, a competição faz com que apenas um deles gere comportamento para uma mensagem de entrada, e não cada um deles em particular como no ERL. O principal ponto em comum é que as redes são evoluídas geneticamente nos dois casos, mas lembrando que, no ERL, a rede de ação pode ter seus pesos modificados durante a vida do agente, o que não ocorre com o SCRN, onde a rede de ação não é modificada durante a existência do classificador.

3.2.2 Adaptive Critic

O treinamento supervisionado de uma rede neural só pode ser efetuado em situações onde se conhece ou pode-se obter uma amostra dos valores de entrada com as saídas esperadas. Esta amostra é apresentada à rede neural permitindo o seu treinamento. Existem situações em que além de não se ter esta amostra, as únicas informações que podem ser fornecidas para a rede são um escalar que mede o acerto de sua saída perante uma dada entrada. Este é o caso onde se aplicam técnicas de *Reinforcemente Learning* (RL) [38]. No caso em que a informação do ambiente não pode ser utilizada imediatamente, ou seja, quando existe a necessidade de se fazer a previsão do acerto futuro, é utilizado um algoritmo de RL onde o sinal de reforço é gerado por uma rede neural denominada *Critic* [22], sendo uma de suas variações denominada *Adaptive Critic*.

A rede neural desempenhando a função de *critic* recebe informações do ambiente e da saída da rede neural de controle e fornece o sinal de reforço para a rede de controle. Dessa forma pode-se entender o *critic* desempenhando a função da rede neural de avaliação no SCRN.

De fato existe a semelhança funcional entre o *adaptive critic* e o classificador neural utilizado no SCRN, mas os procedimentos são bastante diferentes. Primeiro, em RL o que se procura é otimizar um único controlador neural responsável pelo comportamento total, utilizando-se o algoritmo de RL para fornecer as informações de avaliação, e este algoritmo é numérico, não evolutivo. No SCRN o sinal da rede de avaliação não vai diretamente para a rede de ação - é enviado para a Atribuição de Créditos, estando em evolução um conjunto de classificadores, que dividem a responsabilidade do controle do sistema. No RL o mecanismo de aprendizagem efetua modificações diretas nos pesos da rede de controle, e no SCRN isto ocorre por evolução genética.

3.3 Níveis Superiores da Hierarquia

As arquiteturas dos níveis superiores são inteiramente semelhantes às apresentadas para o SC.

3.3.1 Atribuição de Créditos

O modelo segue, em linha geral, o especificado por Richards em [**33**]. São feitas algumas alterações para adequá-lo às necessidades específicas encontradas.

Cálculo da aposta

No nível básico da hierarquia são identificados os classificadores com *matching* com a mensagem de entrada.

Com o SCRN, o *matching* e a especificidade são saídas da rede de avaliação, tendo como entrada a mensagem de entrada. Sendo estes valores números reais, é adotado o critério de considerar a ocorrência de *matching* quando forem positivos e, se a especificidade for negativa o classificador não é aceito para competir. Este procedimento permite a ocorrência de um grande nº de classificadores competindo, assim é adotado o critério adicional de aceitar apenas um determinado nº de classificadores, utilizando o seu valor de *matching* para selecioná-los, ou seja, serão aceitos um dado nº de classificadores com maior *matching*.

É utilizada a Exp. 2.2 para o cálculo da aposta.

Taxa de aposta

São utilizadas as expressões 2.3 e 2.4 para calcular-se a força dos classsificadores neurais após a aplicação da taxa de aposta.

Taxa de vida

É utilizada a Exp. 2.5 para o cálculo da taxa de vida e a Exp. 2.6 para o cálculo da força.

Recompensa ou punição

Tem-se as expressões 2.7 e 2.8 para a obtenção da força pela aplicação do mecanismo de recompensa e punição.

3.3.2 Principais Diferenças na Implementação do Algoritmo de Atribuição de Créditos

Considerando a abordagem definida por Richards, o algoritmo utilizado neste trabalho tem as seguintes diferenças.

Não ocorrendo *matching*

Quando não ocorre *matching* o algoritmo do Richards introduz um novo classificador com antecedente igual ao valor codificado pelos sensores, com o conseqüente gerado aleatoriamente e com força igual à média das forças dos classificadores na iteração em curso.

No entanto, a sua implementação no SCRN requer a obtenção de uma Rede Neural para o antecedente do novo classificador, a sua rede de Avaliação, que produza *matching*. Mas neste sistema tanto *matching* como especificidade são valores reais e, como será mostrado posteriormente, devem ser positivos, ou seja, um classificador será considerado com *matching* apenas se o seu *matching* e especificidade forem maiores que zero. Finalmente, tem-se ainda que obter a Rede Neural que produza esses valores como saída. Assim, este novo classificador é introduzido, na abordagem com Redes Neurais, de forma aleatória, isto é, seu antecedente e seu conseqüente, pesos das Redes Neurais de Avaliação e Ação, são gerados aleatoriamente. É tomado o cuidado para que sejam obtidos classificadores com *matching* e especificidade positivos, testando-se estes valores e, se eles não respeitarem tal condição, então é feita uma nova tentativa, gerando-se um novo classificador. Decidiu-se fazer até 10 repetições desse ciclo e, caso não tenha sido conseguido *matching* e /ou especificidade, repete-se a ação anterior.

Não ocorrendo mudança na mensagem da interface de entrada

Este procedimento é inteiramente semelhante ao utilizado no SC.

Punição

Este procedimento é inteiramente semelhante ao utilizado no SC.

3.3.3 Algoritmo Genético (AG)

Este procedimento é inteiramente semelhante ao utilizado no SC.

Com relação aos operadores genéticos, utiliza-se *crossover* de dois pontos, no qual são escolhidos aleatoriamente dois pontos nos cromossomos dos pais, sendo permutados os genes localizados entre eles. É usada mutação inversiva, onde são selecionados dois pontos de um cromossomo e os seus genes são permutados, como indicado na Fig. 3.4.

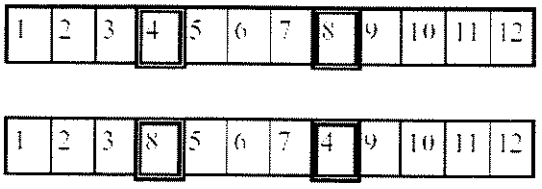


Figura 3.4 Mutação Inversiva

Capítulo 4: O Simulador

Este capítulo aborda o tema do Simulador, onde a pertinência de se efetuar simulações bem como as suas vantagens e desvantagens são discutidas. É mostrado também o porquê da escolha do Matlab como linguagem para desenvolvimento do *software* do simulador. Por fim, é apresentada uma visão geral do simulador e de seus recursos

4.1 Por que Simular ?

4.1.1 A simulação faz parte do mecanismo de inteligência

Simular é um procedimento inerente ao ser humano e pode ser visto como sendo parte de seu mecanismo de inteligência, conforme ilustrado a seguir:

Quando tem-se que tomar uma decisão, lança-se mão de simulações para verificar qual a ação mais apropriada em função do objetivo pretendido. Por exemplo, ao querer-se pegar um objeto, é feito pela mente um traçado imaginário do percurso a ser seguido e o cérebro passa a comandar o movimento da mão, procurando seguir o traçado obtido. O cálculo da trajetória pode ser entendido como uma simulação prévia para estabelecer a trajetória a ser seguida. Em situações bastante simples, sem a existência de obstáculos e com alvos estacionários, o traçado mental é apenas refeito de tempos em tempos, como realimentação para as alterações nas posições da mão. Em situações mais complexas faz-se necessário o planejamento de vários traçados para se chegar à decisão de qual o mais indicado e as realimentações serão mais freqüentes.

Ve-se que simular e calcular são termos com significados parecidos. Os cálculos de um projeto são elaborados para se obter uma solução mais adequada do que a que seria obtida sem eles. Efetuar um projeto para se obter algo, é efetuar uma simulação prévia. Atualmente as simulações ganham mais poder, ou seja, mais realidade e velocidade, com o uso de processamento digital.

Por isso é quase que natural, principalmente com os recursos computacionais existentes, pensar-se em efetuar simulações como etapa inicial do desenvolvimento de um projeto.

4.1.2 Simular ou não ?

Existem no meio científico divergências quanto à possibilidade dos resultados obtidos por simulação serem significativos quanto a representarem a realidade. A oposição ao uso de simuladores em robótica móvel em geral tem Brooks como um grande aliado em seu artigo "*Intelligence Without Reason*" [8].

É de se esperar que os resultados obtidos na simulação sejam diferentes dos obtidos utilizando-se a situação real. No entanto, devem existir maneiras de se minimizar estas diferenças. É o que relata Grefenstette em vários artigos onde justifica a utilização do simulador Samuel [15].

Schultz & Grefenstette em seu trabalho "*Using a Genetic Algorithm to Learn Behaviors for Autonomous Vehicles*" [36] e Ramsey *et al* em seu trabalho *"Simulation-assisted learning by competition: Effects of noise differences between training model and target environment"* [32] mostraram que os conhecimentos adquiridos por procedimentos de aprendizagem em simulações computacionais podem ser robustos e portanto aplicáveis ao mundo real, no caso da simulação incluir condições mais realistas a respeito do mundo real, como por exemplo se a simulação incorporar ruído e um conjunto diversificado de condições ambientais.

4.1.3 Vantagens

Agilidade nas implementações e análises

Dentre todas as vantagens em se utilizar um simulador, provavelmente a agilidade com que as implementações e análises podem ser feitas talvez seja a de maior relevância para o pesquisador. Não existem as atividades relativas ao provimento de energia, *download* de programas, medidas de segurança para se evitar danos ao robô e ao ambiente. O mesmo se pode dizer com relação à observação e obtenção dos dados para análise.

Investimento e custeio

Uma simulação normalmente exigirá investimento de capital menor do que a aquisição de um robô real. Mas, no caso de optar-se por simulações prévias à implementação real, este item deve ser considerado como um investimento adicional.

Talvez o investimento maior seja relativo ao tempo do profissional dedicado ao desenvolvimento do *software*. Cabe, no entanto, a observação de que haverá economia de tempo nas modificações e testes. Ainda, o mesmo simulador poderá ser aproveitado para diferentes técnicas experimentadas.

Uma vez que simulador e modelos sejam adequados para a representação dos ambientes reais, o custeio será mais barato, demandará apenas o uso do computador, dispensando o uso do robô e toda a infra-estrutura necessária para a fase de experimentações e treinamento.

Considerações teóricas

Uma implementação de um modelo teórico em um simulador, mesmo sendo bastante simplificada, pode se tornar um grande auxiliar na análise de implicações gerais. Pode inclusive indicar a viabilidade do modelo teórico, antes de sua implementação mais detalhada. Esta é uma aplicação muito importante do simulador, principalmente se este for construído de tal forma que a inserção de diferentes técnicas de controle seja facilitada. Daí o interesse em desenvolver o Simulador de forma modular, utilizando programação estruturada com Classes e Objetos.

Tempo de execução reduzido

As simulações podem também reduzir o tempo da experimentação, pois é possível executar o modelo em velocidade superior à real, dependendo apenas do desempenho do microcomputador.

Portabilidade.

O Simulador é um *software* - pode ser transportado, por exemplo, do Laboratório de Pesquisas, que pode ser um mero Escritório com um microcomputador, para casa, etc.

Replicação

Devido à sua portabilidade, o Simulador pode ser utilizado por vários pesquisadores ao mesmo tempo, diminuindo significativamente a demanda pelo robô.

Segurança

Evita danos ao robô, ao ambiente e ao pesquisador, em caso de funcionamento em condições incertas ou desconhecidas de operação, que podem ser processadas em simulação.

4.1.4 Inconveniências

Modelagem do robô

Esta , talvez, a maior dificuldade, principalmente na modelagem dinâmica.

Modelo do Ambiente e Obstáculos

A modelagem do ambiente com os obstáculos não é tarefa simples, mais ainda considerando um ambiente não estacionário.

Investimento, custeio e tempo adicionais

Pode ser necessária a compra de *hardware* adicional para se fazer a simulação, por exemplo, um microcomputador com o desempenho requerido. O custeio será praticamente relativo ao tempo despendido no desenvolvimento do *software* do simulador para os modelos do robô e do ambiente com seus obstáculos. Os algoritmos de controle terão que ser desenvolvidos em ambos os casos, para a simulação e para o experimento real. Todo este processo requer um tempo adicional, entre a decisão de se efetuar a pesquisa e a sua implementação.

4.2 Escolha da Linguagem de Programação

4.2.1 Decisões preliminares

Para escolher a linguagem de programação foi considerada a facilidade a partir de recursos disponíveis na linguagem, o desempenho e sua aceitação e utilização nos meios científicos e na engenharia. O uso de classes é apropriado para a abordagem estruturada e modular, o que permite extensão do simulador para vários tipos de robôs, com seus dispositivos peculiares de direção, detectores e algoritmos de navegação.

Com relação a recursos gráficos têm-se duas demandas distintas. A primeira é relativa à visualização do comportamento dinâmico do robô - sem requerer muita sofisticação. A segunda é relativa à visualização dos dados em gráficos, que, com recursos apropriados da linguagem, podem ter a sua eficiência de interpretação aumentada.

A inserção de dados dos parâmetros e a recuperação de dados para análise são fatores primordiais nas pesquisas, que, dependendo da linguagem, irão requerer o desenvolvimento de programas e rotinas específicas.

Foram consideradas três linguagens como candidatas: C++, Java e MATLAB, sendo a inclinação pessoal do autor pela terceira.

As linguagens C++ e Java atendem a quase todos os requisitos, exceto com relação à recuperação de dados e sua visualização em gráficos, que requerem o desenvolvimento de rotinas específicas, além da inexistência de facilidades intrínsecas da linguagem para o desenvolvimento dos programas do simulador, como funções para o uso de matrizes e para as técnicas de IA que serão utilizadas no futuro. Uma solução neste caso é o uso de bibliotecas, o que requer a aquisição de softwares adicionais e a verificação de sua funcionalidade. Mas não resta dúvida que com relação ao desempenho a linguagem C++ estaria em primeiro lugar, seguida de Java.

Com relação ao MATLAB, em primeiro lugar entra a familiaridade do autor com a linguagem e seu estilo pessoal. O uso de matrizes e as funções de matrizes existentes no MATLAB facilitam muito o desenvolvimento dos algoritmos e também a entrada e recuperação de dados. A existência nesta linguagem de vários recursos adicionais para programação que poderão ser utilizadas no controle com aprendizado (como, por exemplo, o *Neural Network Toolbox,* o *Control Systems Toolbox*, o *Simulink*) tornam-na bastante atraente.

O MATLAB é uma linguagem interpretada, o que usualmente significa um desempenho ruim, ou, ao menos, não muito bom em termos de velocidade de computação. No entanto as versões mais recentes apresentam um desempenho bastante melhorado e, o que é melhor, a possibilidade de transformar funções em *dynamic link libraries* (dlls), permitindo desempenho quase equivalente ao das linguagens C/C++.

4.2.2 MATLAB 6.0 Release 12 – a linguagem escolhida

É claro que a escolha da linguagem é de suma importância e não pode se basear apenas nas qualificações apresentadas. Assim, esta etapa foi concluída após o desenvolvimento do simulador básico, cujos testes iniciais comprovaram a viabilidade do uso do MATLAB. Por exemplo, um processador de 300 MHz com 128 MB de RAM é confortável para a maior parte dos testes, sendo recomendável um processador mais rápido dependendo da complexidade do ambiente e/ ou da existência de mais de um robô.

Como ilustração, utilizando-se um processador Pentium III, com 256 MB de RAM, sendo o robô configurado com dois sensores e o ambiente contendo um obstáculo, foi obtido o desempenho de 1000 iterações por minuto. Assim, alguns dos experimentos do Capítulo 6, que utilizaram 15000 iterações, podem ser executados cada um deles, com este *hardware*, em aproximadamente 15 minutos.

4.3 Visão Geral

As facilidades oferecidas pelo Simulador são de três naturezas distintas: Recursos do Simulador, o robô e suas características e os algoritmos de controle.

4.3.1 Recursos do simulador

Os recursos do simulador são os componentes do Simulador utilizados para a definição do ambiente onde o robô deverá operar, independentemente do tipo e características do robô que será utilizado. Estes recursos permitem a definição dos obstáculos e dos contornos do ambiente. Os obstáculos são implementados como objetos da classe **FormaGeometrica**. O contorno do ambiente é definido pelo método **Ambiente**. O Simulador foi desenvolvido para ambiente 2D, com contorno de qualquer formato, a ser definido pelo usuário.

4.3.2 O robô e suas características

São considerados robôs com direção por rodas diferenciais, com forma circular, como objetos da classe **RobotMovel**, representando de maneira simplificada o mini robô **Khepera** [21] que, pretende-se, será utilizado futuramente na consecução deste trabalho. O robô pode ter um número qualquer de sensores, podendo ser especificado para cada um deles a sua posição, abertura e alcance máximo. A abordagem modular permite a implementação de outras formas de direção e de outros tipos de sensores.

4.3.3 Os algoritmos de controle

A simulação é implementada utilizando um programa Controlador que

- inicializa parâmetros
- define o ambiente
- define os objetos obstáculos da classe *FormaGeometrica*
- define os objetos robôs da classe *RobotMovel*
- chama os métodos das classes
- chama os métodos referentes aos algoritmos de controle em uso (*AlgoritmoGenetico, Atuador, Classificador*)
- efetua a Recompensa ou Punição

4.3.4 Resumo das classes e métodos

Classes	Métodos Públicos	Métodos Privados
RobotMovel	RobotMovel (construtor)	DetectaAuxiliar
	DWS	Posiciona
	set	Sensor
	DadosParaPlot	
	Detecta	
FormaGeometrica	FormaGeometrica (construtor)	Posiciona
	NovasCoordenadas	
	set	
	DadosParaPlot	

Métodos dos Algoritmos de IA
AlgoritmoGenetico
Atuador
Classificador
Recompensa e Punição (dentro do controlador)
Método para Ambiente

Tabela 4.1 Resumo das Classes e Métodos do Simulador

Capítulo 5: Detalhes do Simulador

Deve ser mencionado inicialmente que nem todos os leitores estarão interessados na leitura deste capítulo e ele não é essencial para o entendimento desta pesquisa. Ele está escrito, pela sua natureza, em linguagem técnica familiar para pesquisadores que utilizam o Matlab e apresenta detalhes da implementação do simulador, contendo informações para a sua utilização, manutenção e inclusão de novos recursos.

Os leitores que desejarem ter um conhecimento apenas um pouco mais aprofundado das possibilidades do simulador sem terem que passar por todos os detalhes deste capítulo, podem se dirigir diretamente ao último item, o Controlador, onde encontrarão uma visão da dinâmica do funcionamento do simulador e também entrarão em contato mais especificamente com os parâmetros necessários para a configuração do sistema para um experimento.

5.1 Entrada, Saída de Dados e Apresentação da Simulação

A entrada de dados é feita através de parâmetros localizados no programa **Controlador**, apresentado no último item deste capítulo.

A saída de dados utiliza o **workspace** do MATLAB. Os dados a serem recuperados são guardados em matrizes do programa Controlador, ficando disponíveis no **workspace**, portanto, quando a execução finaliza, ou é interrompida. No caso de necessitar-se de informações sobre os dados referentes a algum método de controle em particular, pode-se utilizar os recursos de depuração do MATLAB para observá-los, ou então passá-los como parâmetro de saída para o programa do Controlador.

Uma vez obtidos os dados (e com os recursos gráficos do MATLAB), produz-se os gráficos necessários para a análise, interpretação e avaliação. Isto pode ser feito pelo programa **VeResultados** de forma automatizada.

A apresentação da simulação é feita em uma figura contendo o ambiente com os obstáculos e o robô. Concomitantemente são apresentados dois gráficos, um mostrando a curva de aprendizado, e outro o número de iterações transcorridas entre colisões sucessivas.

5.2 Recursos Específicos do Simulador

São os recursos elaborados especificamente para o Simulador, independentes do tipo e características do robô que será utilizado. Compreendem a criação da figura que irá conter o ambiente de simulação, os gráficos, o ambiente e os objetos a serem utilizados.

5.3 Ambiente

O ambiente é estabelecido por desenho dos pontos com suas coordenadas contidas em dois vetores linhas: **contornox** e **contornoy**.

5.3.1 Funções para Ambiente

Existem vários métodos acompanhando o Simulador que podem ser utilizados para a definição do ambiente de simulação com seu contorno e obstáculos. Eles são utilizados como parâmetro no

programa **Controlador**. A partir deles fica fácil, por edição, introduzir alterações no ambiente de simulação. Por exemplo, o método **Ambiente0**

Sintaxe

[VetorObjetos,contornox, contornoy] = Ambiente0

Parâmetros de saída

VetorObjetos	Contém os objetos criados pelo método
contornox, contornoy	Contém os pontos referentes às coordenadas cartesianas para desenho do contorno do ambiente.

5.4 Classe FormaGeometrica

A classe FormaGeometrica é usada para produzir objetos geométricos de várias formas. Foram elaborados dois tipos diferentes de objetos, com forma circular e forma retangular.

5.4.1 Construtor: FormaGeometrica

Chama o método privado **Posiciona** para determinar os pontos para desenho do objeto.

Esta classe permite a criação de objetos de forma geométrica e seu desenho no ambiente. Podem ser criados dois tipos de objetos: com forma circular e com forma retangular, que atendem as necessidades dos experimentos, com a sintaxe:

Sintaxe

h = FormaGeometrica(tipo,dadoespecifico,xcentro,ycentro)

que, para a forma circular, resulta em

ucircular = FormaGeometrica('Circulo',raio,xcentro,ycentro)

onde são fornecidos o raio e as coordenadas do centro. Para a forma retangular resulta em

uretangulo = FormaGeometrica('Retangulo',[l1 l2],xcentro,ycentro)

onde é fornecido um vetor linha com os lados e as coordenadas do centro do retângulo.

Os objetos retangulares são criados com os lados paralelos aos eixos de coordenadas x,y.

A introdução de novas formas geométricas é bastante facilitada pela modularidade do desenvolvimento do software. Inclusive poder-se-ia introduzir um novo tipo **SemForma** onde seriam fornecidos os pontos de definição de seu contorno. Eventualmente, por clareza no desenvolvimento, poder-se-ia criar uma nova classe para esse tipo de objeto.

Parâmetros de entrada

tipo	Especifica a forma do objeto (**'Circulo'** ou **'Retangulo'**)
dadoespecifico	Para **'Circulo'** é o raio, para **'Retangulo'** é um vetor com **ladohorizontal** e **ladovertical**
xcentro,ycentro	Coordenadas do centro do objeto

Parâmetros de saída

h	Objeto FormaGeometrica.

5.4.2 NovasCoordenadas

Calcula as novas coordenadas do objeto. Chama **Posiciona** para efetuar os cálculos. Aparentemente **NovasCoordenadas** poderia ser embutido em **set**, mas assim o MATLAB não forneceria os resultados desejados para a atualização dos campos do objeto. É necessário, para tanto, que os novos dados sejam passados de fora, ou seja, não de dentro de algum método do objeto. Por isso, para: posicionar o objeto em um novo lugar, alterar o registro de seus dados e exibi-lo deve ser usado **NovasCoordenadas** e **set**.

Como o método NovasCoordenadas calcula as novas propriedades do objeto em uma nova posição definida por um novo centro, ele permite também a mudança de posição do objeto, facilitando a geração de ambientes não estacionários.

Note que este método apenas calcula as novas propriedades. Para estabelecê-las e visualizar o objeto na nova posição deve-se usar o método **set**.

Sintaxe

[fgx,fgy] = NovasCoordenadas(u,xcentro,ycentro)

Parâmetros de entrada

u	Identificador do objeto. Contém a estrutura com dados do objeto.
xcentro,ycentro	Coordenadas do centro do objeto

Parâmetros de saída

fgx,fgy	Coordenadas para desenhar o contorno do objeto.

5.4.3 set

set atribui novos valores a campos do objeto e o re-exibe. Observar que a variável de retorno deve ser o próprio objeto. Este método não dá o resultado desejado se chamado por outro método do objeto. Estas observações são características da maneira como o MATLAB utiliza classes e objetos.

Lembrar que após definir novas propriedades com o método **NovasCoordenadas** temos que utilizar o método **set** para estabelecê-las como novas propriedades do objeto **FormaGeometrica** e para visualizá-lo na nova posição.

Sintaxe

u = set(u, xcentro,ycentro,fgx,fgy)

É interessante notar a semelhança na utilização de set para a classe RobotMovel e para a classe FormaGeometrica.

Parâmetros de entrada

u	Identificador do objeto. Contém a estrutura com dados do objeto.
xcentro,ycentro	Coordenadas do centro do objeto.
fgx,fgy	Coordenadas para desenhar o contorno do objeto.

Parâmetros de saída

u	Identificador do objeto.

5.4.4 DadosParaPlot

DadosParaPlot permite acesso aos dados para desenho. Foi desenvolvido para ser usado pelo método **Detecta** da classe **RobotMovel**. Mas como método público poderá ter outras finalidades.

Sintaxe

[fgx,fgy] = DadosParaPlot(u);

Parâmetros de entrada

u	Identificador do objeto.

Parâmetros de saída

fgx,fgy	Coordenadas para desenhar o contorno do objeto.

Posiciona

Posiciona calcula todos os pontos para desenhar o objeto nas coordenadas pedidas. Como se trata de um método privado, tem que ser colocado em uma pasta denominada '**private**'. Somente os métodos públicos ficam no mesmo diretório, da classe.

Deve ser método privado pois é utilizado pelo construtor, antes de definir o objeto.

Sintaxe

[fgx, fgy] = Posiciona(varargin)

Quando chamado pelo construtor utiliza a sintaxe:

[fgx fgy] = Posiciona(tipo,dadoespecifico,xcentro,ycentro);

Quando chamado por NovasCoordenadas utiliza a sintaxe:

[fgx fgy] = Posiciona(u,xcentro,ycentro);

Parâmetros de entrada

u	Identificador do objeto.
tipo	Especifica a forma do objeto ('Circulo' ou 'Retangulo')
dadoespecifico	Para 'Circulo' é o raio, para 'Retangulo' é um vetor com ladohorizontal e ladovertical
xcentro,ycentro	Coordenadas do centro.

Parâmetros de saída

fgx,fgy	Coordenadas para desenhar o contorno do objeto

5.5 Classe RobotMovel

Esta classe tem a finalidade de servir de ponto de partida para a modelagem de um mini robô Khepera, que pretende-se utilizar futuramente na seqüência deste trabalho. Assim o robô tem forma circular, seu controle de direção é feito pela velocidade diferencial em suas rodas de direção e, ainda, utiliza sensores para detectar o ambiente.

Não se pretende por enquanto uma modelagem completa e pormenorizada do robô. Desta forma, com relação ao movimento, são considerados apenas os aspectos relativos à cinemática, sem

aceleração. Os sensores são representativos de sensores de distância, podendo ser definida a sua quantidade, posicionamento e abertura. É simulada também a existência de sensores de contato ao longo de sua superfície.

A Classe RobotMovel implementa o método **DWS** para efetuar mudança de direção por velocidade diferencial, que pode ser considerado como método específico, podendo aceitar métodos para outros tipos de abordagem. Implementa também um método chamado **Detecta**, que permite ao robô detectar, através de seus sensores, os contornos do ambiente, os objetos e outros robôs.

Utiliza o método privado **Posiciona** para determinar os pontos de desenho do robô, que por sua vez utiliza o método privado **Sensor** para determinar os pontos de desenho dos contornos de cobertura dos sensores.

A sintaxe para criar um objeto RobotMovel é:

u = RobotMovel(Eixo,XIn,YIn,TetaIn,DadosSensor)

onde

Eixo: define o eixo do robô

XIn,YIn: coordenadas x e y do centro do robô

TetaIn: orientação angular

DadosSensor: vetor linha definindo os sensores

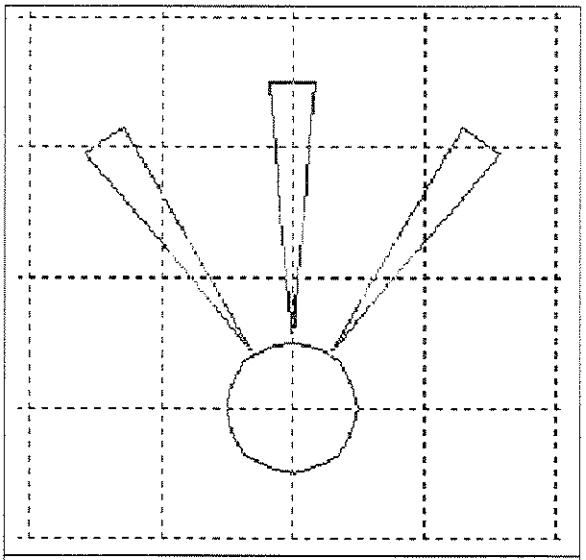


Figura 5.1 – Ilustração de um objeto RobotMovel

Seja o seguinte exemplo a definição dos sensores

DadosSensor = [35 10 100 0 10 100 -35 10 100]

de um robô como o definido na Fig. 5.1. Podemos perceber a área de percepção de três sensores, o primeiro 35 graus à esquerda do rumo do robô, o segundo no rumo do movimento e o terceiro 35 graus à direita. Todos os sensores têm uma abertura de 10 graus e um alcance de 100 pixels.

O Khepera tem 70 mm de diâmetro, correspondendo na figura a 50 pixels, o que significa uma escala de 1.4 mm/pixel. O alcance dos sensores de 100 *pixels* está representando um alcance de 140 mm.

5.5.1 Construtor: RobotMovel

Sintaxe

```
u = RobotMovel(eixo,x,y,teta,dadossensor)
```

Propriedades

Os parâmetros a serem utilizados para a criação do eixo e dos sensores do objeto RobotMovel constituem propriedades da classe. Estas propriedades foram definidas de modo a não serem alteradas, não existindo método para tanto.

Os parâmetros a serem utilizados para as coordenadas do centro do robô e sua orientação são propriedades. Elas podem ser alteradas pelo método **DWS**.

Parâmetros de entrada

eixo	Diâmetro ou tamanho do eixo diferencial.
x,y	Coordenadas do centro do robô.
teta	Orientação e sentido do robô.
dadossensor	Vetor cujos elementos são considerados de três em três, correspondendo ao deslocamento da posição do sensor, sua abertura e seu alcance.

Parâmetros de saída

u	Identificador do objeto. Contém a estrutura com dados do objeto.

5.5.2 Sensor (método privado)

A classe **RobotMovel** utiliza o método privado **Sensor** para definir o sensor, o qual pode detectar (através do método **Detecta)** objetos em três regiões: perto, meia distância e longe. Note que existe a facilidade de introduzirem-se novos modelos de sensores como novos métodos, bastando desenvolverem-se novos métodos privados.

Calcula os pontos para desenhar os sensores. É chamada por **Posiciona**. Utiliza duas funções locais **DireitaEsquerda** e **Arco**.

sintaxe

```
[sx1, sy1, sx2, sy2, sx3, sy3] = Sensor(eixo,x,y,teta,dadossensor)
```

Parâmetros de entrada

eixo	Diâmetro ou tamanho do eixo diferencial.
x,y	Coordenadas do centro do veículo.

teta	Orientação do veículo.
dadossensor	Vetor contendo dados para a definição dos sensores, organizados de três em três, correspondendo à posição, abertura e alcance.

Parâmetros de Saída

sx1, sy1	Vetores com as coordenadas da primeira seção do sensor.
sx2, sy2	Idem segunda seção.
sx3, sy3	Idem terceira seção.

5.5.3 DWS

O método **DWS** executa os cálculos para computar a mudança de direção por rodas diferenciais. Neste modelo é considerada a cinemática sem aceleração, suposição que preenche as necessidades desta abordagem. Seus parâmetros de entrada são a posição atual do robô e sua orientação, o intervalo de tempo **dt** que irá transcorrer e as velocidades das rodas direita e esquerda. Os parâmetros de saída fornecerão as novas coordenadas do robô e dados para a seu desenho.

Dadas as coordenadas atuais, a orientação atual, a velocidade atual e o intervalo de tempo **dt**, calcula as novas coordenadas do centro do veículo e sua orientação. A seguir calcula os pontos para desenho do objeto e dos sensores chamando o método privado **Posiciona**, que por sua vez chama **Sensor**, também método privado.

DWS apenas calcula os novos pontos para desenho. Para alterar o registro de seus dados e exibi-lo deve ser usado **set**.

sintaxe

[x,y,teta,vx, vy, sx, sy, SX1, SY1, SX2, SY2, SX3, SY3] = DWS(u,dt,x,y,teta,ve,vd)

Parâmetros de entrada

u	Identificador do objeto. Contém a estrutura com dados do objeto.
dt	Intervalo de tempo
x,y	Coordenadas do centro do veículo.
teta	Orientação do veículo.
ve,vd	Velocidade das rodas esquerda e direita.

Parâmetros de saída

x,y	Coordenadas do centro do veículo.
teta	Orientação do veículo.
vx,vy	Vetores com coordenadas dos pontos para desenhar o veículo.
sx,sy	Coordenadas do ponto indicativo da orientação e sentido.
SX1,SY1	Cell contendo as coordenadas da primeira seção de todos os sensores.
SX2,SY2	Idem segunda seção.
SX3,SY3	Idem terceira seção.

Ver o apêndice relativo ao desenvolvimento matemático das relações entre os vários parâmetros que efetuam a movimentação por DWS.

5.5.4 set

O método **set** é utilizado para definir novas propriedades para o objeto e atualizar a sua posição. Ele precisa ser usado após chamar **DWS**.

set atribui novos valores a campos do objeto e o re-exibe. Observar que a variável de retorno deve ser o próprio objeto. Este método não dá o resultado desejado se chamado por outro método do objeto.

É interessante notar a semelhança na utilização de set para a classe RobotMovel e para a classe FormaGeometrica.

Sintaxe

u = set(u,x,y,teta,vx,vy,sx,sy,SX1,SY1,SX2,SY2,SX3,SY3)

Parâmetros de entrada

u	Identificador do objeto.
x,y	Coordenadas do centro do veículo.
teta	Orientação do veículo.
vx,vy	Vetores com coordenadas dos pontos para desenhar o veículo.
sx,sy	Coordenadas do ponto indicativo da orientação e sentido.
SX1,SY1	Cell contendo as coordenadas da primeira seção de todos os sensores.
SX2,SY2	Idem segunda seção.
SX3,SY3	Idem terceira seção.

Parâmetros de saída

u	Identificador do objeto.

5.5.5 DadosParaPlot

DadosParaPlot dá acesso às propriedades referentes aos dados para desenho do objeto. Foi desenvolvido para ser usado pelo método Detecta. É importante notar que DadosParaPlot não precisa ser usada para obter-se os dados do objeto RobotMovel para serem usados por Detecta, pois como Detecta é um método do objeto RobotMovel, ele tem acesso aos dados diretamente. Alias, isto é válido para qualquer método RobotMovel. Isto significa que este método será útil principalmente se outros objetos precisarem dos dados de desenho do objeto RobotMovel. Assim, este método para o RobotMovel é relevante somente no caso em que tivermos mais de um robô, para um deles poder detectar o outro. O que será feito através de **Detecta**, que chamará **DadosParaPlot** do outro robô para obter os seus dados para detecção.

Outro ponto importante é que a classe FormaGeometrica tem também o método DadosParaPlot. Qual deles será usado é definido pelo objeto dado como parâmetro de entrada. Este comportamento

é característico do MATLAB - lidar com classes, o que permite uma padronização da terminologia utilizada.

Sintaxe

[vx,vy,sx,sy,SX1,SY1,SX2,SY2,SX3,SY3] = DadosParaPlot(u);

Os parâmetros correspondem aos pontos do contorno do robô, da indicação da sua orientação e das seções dos sensores.

Parâmetros de entrada

u	Identificador do objeto. Contém a estrutura com dados do objeto.

Parâmetros de saída

x,y	Coordenadas do centro do veículo.
vx,vy	Vetores com coordenadas dos pontos para desenhar o veículo.
sx,sy	Coordenadas do ponto indicativo da orientação e sentido.
SX1,SY1	Cell contendo as coordenadas da primeira seção de todos os sensores.
SX2,SY2	Idem segunda seção.
SX3,SY3	Idem terceira seção.

5.5.6 Detecta

Detecta é o método que irá verificar se o objeto RobotMovel tocou um obstáculo, outro robô ou o contorno do ambiente. Ele também obtém as leituras dos sensores. Requer como entrada todos os objetos mais o contorno do ambiente a serem detectados. Ele utiliza o método **DadosParaPlot** do objeto a ser detectado.

O método **Detecta** é responsável pelo código de maior custo computacional do simulador. A sua dificuldade aumenta em função do número de objetos. O contorno do ambiente e cada obstáculo devem ser verificados quanto às suas posições relativas ao robô e quanto a estarem na área de cobertura dos sensores e em qual de suas seções.

A técnica utilizada lança mão do método **inpolygon** do MATLAB com a seguinte sintaxe:

IN = INPOLYGON(X, Y, XV, YV)

retorna uma matriz IN do tamanho de X e Y.

IN(p,q) = 1 se o ponto (X(p,q), Y(p,q)) está estritamente dentro da região do polígono cujos vértices são especificados pelos vetores XV e YV.

IN(p,q) = 0.5 se o ponto está sobre o polígono.

IN(p,q) = 0 se o ponto está fora do polígono.

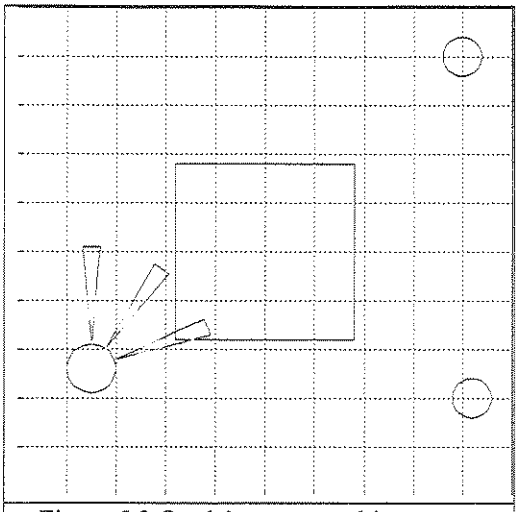


Figura 5.2 O robô em um ambiente com obstáculos.

Com relação à detecção do contorno do ambiente é verificado se o robô tem algum ponto em contato ou invadindo os limites do ambiente. A mesma abordagem é usada para cada seção dos sensores. Assim, uma vez que o ambiente tem dimensões necessariamente maiores do que as do robô e das regiões detectáveis pelos sensores, ele não precisará utilizar muitos pontos para a sua definição, apenas aqueles necessários para caracterizar a sua forma. Se fosse feito o inverso, ou seja, se fosse verificado se algum ponto do ambiente está em contato ou invadiu o robô (o mesmo para os Sensores), ele teria que utilizar uma resolução compatível com as dimensões do robô (e dos Sensores).

Já, com relação aos objetos, é utilizada a abordagem inversa, uma vez que os objetos podem inclusive ser menores que o robô (e que as áreas de detecção dos sensores). Assim é verificado se algum ponto dos objetos está em contato ou invadiu o polígono que define os contornos do robô (o mesmo para os Sensores). Isto não exige alta definição para o robô e para os Sensores, mas apenas resolução compatível para o traçado dos objetos. Desta forma, os procedimentos relativos ao cálculo e desenho dos pontos do robô e das regiões de detecção dos sensores (**Posiciona**, **Sensor** e **set**) acabam por ter seu tempo do processamento melhorado.

Adicionalmente, para melhorar o desempenho, ao invés da verificação ser feita objeto por objeto, todos eles têm os seus pontos concatenados em um único par de vetores, um para x e outro para y. É como se tivéssemos um único objeto a ser detectado.

Detecta não faz tudo sozinho, utiliza **DetectaAuxiliar**, um método privado, para fazer a maior parte dos cálculos. A idéia é permitir compilar **DetectaAuxiliar**, transformando-o em uma biblioteca dinâmica (DLL), para melhorar o desempenho. Isto porque o MATLAB ainda não compila funções que utilizam objetos, sendo previsível que suas novas versões terão essa possibilidade. Desta forma **Detecta** identifica os objetos a serem detectados e por quem, deixando os cálculos por conta de **DetectaAuxiliar**.

Sintaxe

[h, contato] = detecta(u,VetorObjADetectar,contornox,contornoy);

Parâmetros de entrada

u	Identificador do objeto RobotMovel. Contém a estrutura com dados do objeto.
VetorObjADetectar	Contém os identificadores dos objetos que serão detectados.
contornox,contornoy	Contornos do Ambiente.

Parâmetros de saída

h	Contém o vetor linha com o resultado da detecção efetuada pelos sensores.
contato	Indica se houve ou não contato.

5.5.7 DetectaAuxiliar (método privado)

Este método é chamado por **Detecta** fazendo os cálculos necessários para a detecção.

Sintaxe

[h,contato] =

DetectaAuxiliar(contato,nrsensores,nrsecoes,vx,vy,

SX1,SY1,SX2,SY2,SX3,SY3,contornox,contornoy,FGX,FGY)

Parâmetros de Entrada

contato	Indica se houve ou não contato (inicialização).
nrsensores	Número de sensores.
nrsecoes	Número de seções dos sensores.
vx,vy	Vetores com coordenadas dos pontos para desenhar o veículo.
SX1,SY1	Cell contendo as coordenadas da primeira seção de todos os sensores.
SX2,SY2	Idem segunda seção.
SX3,SY3	Idem terceira seção.
contornox,contornoy	Contornos do Ambiente.
FGX,FGY	Contém todos os pontos para desenho de todos os objetos a serem detectados.

Parâmetros de Saída

h	Contém o vetor linha com o resultado da detecção efetuada pelos sensores.
contato	Indica se houve ou não contato.

5.5.8 Posiciona (método privado)

Posiciona calcula os pontos para desenhar o robô na posição x,y com orientação teta. Utiliza o método privado **Sensor** para calcular os pontos para desenhar os campos de detecção dos sensores.

Como se trata de um método privado da classe, deve ser colocado na pasta private, pois somente os métodos públicos ficam na pasta da classe. Foi criado como método privado pois é utilizado pelo construtor fornecendo dados para definir o objeto.

Sintaxe

[vx, vy, sx, sy, SX1, SY1, SX2, SY2, SX3, SY3] = Posiciona(varargin)

Quando chamada por **RobotMovel**, na criação do objeto, utiliza a seguinte sintaxe:

[vx, vy, sx, sy, SX1, SY1, SX2, SY2, SX3, SY3] = Posiciona(eixo,x,y,teta,dadossensor);

Quando chamada por DWS utiliza a seguinte sintaxe:

[vx, vy, sx, sy, SX1, SY1, SX2, SY2, SX3, SY3] = Posiciona(u,x,y,teta);

Parâmetros de entrada

u	Identificador do objeto. Contém a estrutura com dados do objeto.
x,y	Coordenadas do centro do veículo.
teta	Orientação do veículo.
eixo	Eixo das rodas diferenciais.
dadossensor	Vetor contendo dados para a definição dos sensores, organizados de três em três, correspondendo à posição, abertura e alcance.

Parâmetros de saída

x,y	Coordenadas do centro do veículo.
teta	Orientação do veículo.
vx,vy	Vetores com coordenadas dos pontos para desenhar o veículo.
sx,sy	Coordenadas do ponto indicativo da orientação e sentido.
SX1,SY1	Cell contendo as coordenadas da primeira seção de todos os sensores.
SX2,SY2	Idem segunda seção.
SX3,SY3	Idem terceira seção.

5.6 Recursos Referentes à Técnica de Controle Utilizada

Estes recursos são referentes à aplicação que utilizará o simulador.

Serão apresentados nos itens a seguir e referem-se a métodos dos algoritmos de controle e ao Controlador específicos para cada uma das aplicações.

5.7 Métodos dos Algoritmos de Controle

Foram implementados quatro métodos: AlgoritmoGenetico, Atuador, Classificador e Recompensa e Punição. Os tópicos a seguir apresentam primeiro os métodos nos seus aspectos gerais e depois o seu detalhamento relativo às duas aplicações desenvolvidas: Sistemas Classificadores com Redes Neurais (SCRN) e Sistemas Classificadores Convencionais (SC). Para facilitar a diferenciação do uso dos métodos, os utilizados para o Sistema Classificador Convencional receberão o prefixo SC em seu nome.

5.7.1 AlgoritmoGenetico

Este método aplica um algoritmo genético básico aos classificadores com maiores forças para criar uma nova geração que irá substituir os indivíduos mais fracos na população atual. A força dos classificadores é utilizada como função de *fitness*. Os pares são formados utilizando-se o procedimento *Roulette Wheel*. Utiliza *crossover* de dois pontos e mutação inversiva para a abordagem com Redes Neurais. Para a abordagem convencional usa *crossover* de um ponto e a mutação, quando ocorre, inverte o valor do alelo para o antecedente e sorteia entre 0, 1 e -1 (tanto faz) para o conseqüente.

Com Redes Neurais

Sintaxe

[Populacao,Avaliacao,Pares,CrossOver,Mutacao] =
AlgoritmoGenetico(Populacao,PcCrossOver,PcMutacao)

Foram acrescentados alguns parâmetros de saída a mais para poder vê-los no **workspace** do programa **Controlador**. O único parâmetro de saída que será utilizado pelo **Controlador** é **Populacao**.

Parâmetros de entrada

Populacao	Subconjunto dos Classificadores com suas forças. É composto pelos classificadores com maior força e que irão ser utilizados pelo Algoritmo Genético.
PcCrossOver	Taxa de *crossover*.
PcMutacao	Taxa de mutação.

Parâmetros de saída

Populacao	Contém uma nova geração de classificadores que irá substituir os mais fracos.

Convencional

Sintaxe

[Populacao,Avaliacao,Pares,CrossOver,Mutacao] =
AlgoritmoGenetico(TCons,Populacao,PcCrossOver,PcMutacao);

Da mesma forma que para o caso de Redes Neurais, foram acrescentados alguns parâmetros de saída a mais para poder vê-los no **workspace** do programa **Controlador**. O único parâmetro de saída que será utilizado pelo Controlador é **Populacao**.

Os parâmetros têm o mesmo significado, tendo um a mais, como apresentado a seguir.

Parâmetros

TCons	Tamanho do conseqüente.

5.7.2 Atuador

No caso dos Sistemas Classificadores com Redes Neurais, o **Atuador** utiliza a Rede Neural de Ação do classificador vitorioso para processar as informações dos sensores, tomando as suas saídas para controlar a próxima movimentação do robô. É utilizada uma rede Perceptron multi camada [16]. A primeira camada tem o seu número de neurônios ajustado automaticamente de acordo com o número de sensores e do número de regiões de detecção por sensor. O número de neurônios da camada intermediária é um parâmetro de entrada. O número de neurônios na camada de saída é um dado de projeto do sistema e da rede, correspondendo ao número de variáveis requeridas para controlar o robô, por exemplo, duas variáveis, uma para controlar velocidade e outra para direção. O Atuador utiliza a Rede Neural de Ação do Classificador vitorioso para processar as informações dos sensores, tomando as suas saídas para controlar a próxima movimentação do robô.

No caso do Sistema Classificador Convencional, o **Atuador** fornecerá saídas pelo mapeamento de valores dos sensores produzidos pela interface de entrada com o conseqüente do classificador, fornecendo as variáveis necessárias para controlar a movimentação do robô.

As duas aplicações, neste trabalho, terão a incumbência de produzir um único valor como saída, denominado **difv**, entendido como um número expresso em porcentagem de velocidade. Este valor será aplicado à velocidade da roda esquerda se for negativo, ou da direita se positivo, diminuindo a velocidade da roda considerada do valor obtido. Desta forma será possível comandar as mudanças de direção do robô. Por exemplo, se **difv** for negativo ele é aplicado à velocidade da roda esquerda, obtendo-se o valor a ser subtraído desta velocidade, durante a iteração. Isto implica que a roda esquerda diminuirá a sua velocidade, girando para a esquerda.

O Sistema com Redes Neurais produz um número real para **difv**, enquanto que o Sistema Convencional irá mapear o valor do conseqüente em valores pré-definidos.

Com Redes Neurais

Sintaxe

difv = Atuador...
(h,NEntRede,NNeurCInter,NEntCSaida,NNeurCSaidaAt,TCond,TCons,ct,GV,Ganhador)

Parâmetros de entrada

h	Dados dos sensores
NEntRede	Nº de entradas da rede, contando a de polarização
NNeurCInter	Nº de neurônios da camada intermediária (o mesmo para **Classificador** e **Atuador**)
NEntCSaida	Nº de entradas da camada de saída
NNeurCSaidaAt	Nº de neurônios da camada de saída do **Atuador**
TCond	Tamanho da condição
TCons	Tamanho do conseqüente
ct	Coeficiente da tangente hiperbólica
GV	Ganho para amplificar a velocidade **difv**

Ganhador	Classificador ganhador

Parâmetros de saída

difv	Diferença de velocidade, a ser subtraída da velocidade da roda esquerda ou direita.

Convencional

Sintaxe

difv = CSAtuador(TCond,TCons,Ganhador);

Parâmetros de entrada

TCond	Tamanho da condição
TCons	Tamanho do conseqüente
Ganhador	Classificador ganhador

Parâmetros de saída

difv	Diferença de velocidade, a ser subtraída da velocidade da roda esquerda ou direita.

5.7.3 Classificador

Este método calcula o *matching*, a especificidade, faz a aposta, e, assim determina o ganhador. Subtrai a aposta da força dos classificadores e aplica a taxa de vida. O modelo criado é mostrado no capítulo das Aplicações.

No caso de redes neurais são utilizadas as redes avaliação, cujos pesos correspondem ao antecedente dos classificadores. Podem ser utilizadas redes com um ou dois neurônios na camada de saída. Com dois neurônios, um deles produzirá uma saída que será utilizada para *matching* e o outro produzirá a especificidade. Caso seja definida uma rede com apenas um neurônio na camada de saída, o método utiliza o mesmo valor para *matching* e especificidade.

Com Redes Neurais

Sintaxe

[Match,Ganhador,MC,MUGan] =
Classificador(h,NNeurCSaidaCl,NEntRede,NNeurCInter,...

NEntCSaida,NC,MC,ct,Ganhador,MUGan,nugan,NClMatch,TxVida)

A matriz **Match** não é requerida, a sua inclusão como parâmetro de saída é para poder vê-la no **workspace** do controlador.

Parâmetros de entrada

h	Dados dos sensores
NNeurCSaidaCl	N° de neurônios na camada de saída do classificador
NEntRede	N° de entradas da rede, contando a de polarização
NNeurCInter	N° de neurônios da camada intermediária (o mesmo para

	Classificador e **Atuador**)
NEntCSaida	N° de entradas da camada de saída
NC	N° de classificadores
MC	Matriz de classificadores
ct	Coeficiente da tangente hiperbólica
Ganhador	Classificador Ganhador
MUGan	Matriz dos últimos ganhadores
nugan	N° de elementos para MUGan
NNeurCSaidaAt	N° de neurônios da camada de saída do Atuador
NClMatch	N° máximo de classificadores considerados com *matching*
TxVida	Taxa de Vida.

Parâmetros de saída

Match	Matriz dos classificadores com *matching*
Ganhador	Classificador ganhador
MC	Matriz dos classificadores
MUGan	Matriz dos classificadores último ganhadores

Convencional

Sintaxe

[Match,Ganhador,MC,MUGan] = ...

CSClassificador(h,NC,MC,TCond,TCons,Ganhador,MUGan,nugan,TxVida);

A matriz **Match** não é requerida, a sua inclusão como parâmetro de saída é para poder vê-la no **workspace** do Controlador.

Parâmetros de entrada

h	Dados dos sensores
NC	N° de classificadores
MC	Matriz de classificadores
TCond	Tamanho da condição
TCons	Tamanho do conseqüente
Ganhador	Classificador ganhador
MUGan	Matriz dos últimos ganhadores
nugan	N° de elementos para MUGan
TxVida	Taxa de Vida.

Parâmetros de saída

Match	Matriz dos classificadores com *matching*
Ganhador	Classificador ganhador
MC	Matriz dos classificadores
MUGan	Matriz dos classificadores último ganhadores

5.7.4 Atribuição de Créditos

A Atribuição de Créditos é obtida através do método **Classificador** e da Rotina **Recompensa ou Punição** dentro do programa Controlador.

O método **Classificador**, citado no item anterior, faz a seleção dos classificadores que irão participar da aposta, baseado no seu *matching*, especificidade e *strength* (ou força). Faz o procedimento determinado pelo algoritmo de Atribuição de Créditos aos vencedores e determina o vencedor, aquele que irá postar a sua mensagem. No caso dos classificadores serem redes neurais, o *matching* e a especificidade serão saídas da rede neural de avaliação.

A **Recompensa ou Punição** faz a avaliação do resultado da ação determinada pelo classificador vencedor. Aplica os critérios determinados pelo algoritmo de Atribuição de Créditos em uso para recompensar ou punir os classificadores que participaram da última aposta, do vencedor e dos classificadores vencedores nas últimas iterações. A recompensa ou punição se traduz no aumento ou diminuição da força do classificador.

5.8 O Controlador

5.8.1 Apresentação

O **Controlador** é um programa MATLAB usado para controlar o processo da simulação. Seu algoritmo base está apresentado na Figura 5.3 e os seus passos detalhados a seguir. Foram desenvolvidos dois controladores, um para Sistemas Classificadores com Redes Neurais, chamado **Controlador**, e outro para Sistemas Classificadores Convencionais, chamado **CSControlador**. Neste item é descrita a funcionalidade geral, válida para os dois e nos próximos itens serão apresentados os detalhes para a utilização de cada um deles.

Inicialização de Parâmetros é relacionada a todos os tipos de parâmetros requeridos para configurar o tipo de experimento em questão.

Definição do Ambiente cria a figura que irá conter o ambiente e os objetos e define o contorno do ambiente.

Definição dos Objetos define todos os objetos que serão usados. Eles podem ser criados a qualquer momento, não apenas aqui.

Definição do RobotMovel irá configurar e colocar no ambiente todos os robôs do experimento. É claro também com relação aos robôs que eles podem ser criados e posicionados no ambiente quando necessário e a qualquer instante do processamento.

Inicialização dos Classificadores irá gerar o conjunto inicial de classificadores, usualmente através de um processo aleatório.

Atuador: com o Atuador o ciclo principal tem início, é chamado o método Atuador do objeto RobotMovel para obter as variáveis de controle do robô. Na primeira passagem o **Atuador** não tem uso, pois nada terá ocorrido.

Posicionamento do robô: o método DWS utiliza a saída do **Atuador** para determinar a nova posição do robô. Para que elas se tornem efetivas e o robô seja visualizado na nova posição é utilizado o método **set**. Estes procedimentos também não são usados no primeiro ciclo.

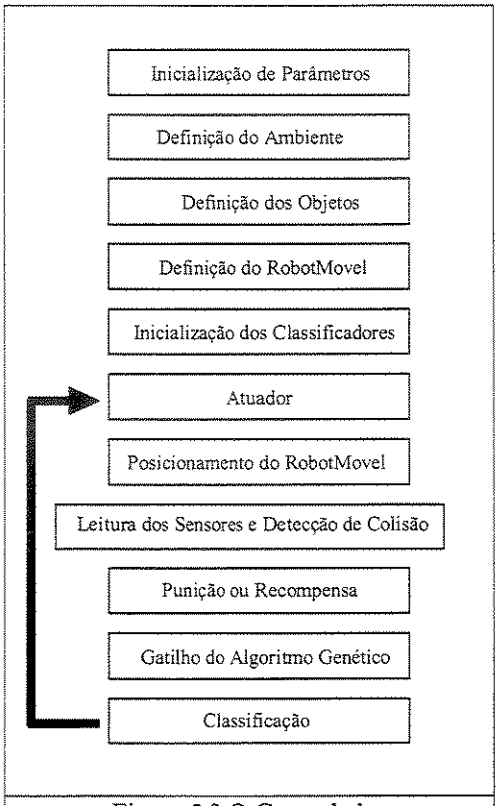


Figura 5.3 O Controlador

Leitura dos Sensores e Detecção de Colisão: agora é preciso verificar se houve colisão e a nova leitura dos sensores, o que é feito utilizando o método Detecta do objeto RobotMovel. Este procedimento terá utilidade a partir da segunda passagem.

Recompensa ou Punição: aqui acontece a realimentação do processo, referente à avaliação dos classificadores. Começando na segunda passada, a ação proposta pelo classificador vencedor será avaliada a partir dos resultados da Detecção e será efetuada a recompensa ou punição.

Gatilho do Algoritmo Genético: o Algoritmo Genético é disparado por uma das ocorrências, a que ocorrer primeiro: um dado número de colisões e um dado número de ciclos desde a última vez que ele foi chamado.

Classificação: a Classificação faz a seleção dos classificadores que irão apostar e obtém o classificador vencedor, que irá determinar a próxima ação.

5.8.2 Controlador para o Sistema Classificador com Redes Neurais

Parâmetros gerais de configuração definidos pelo usuário

Estes parâmetros se encontram logo no início do *programa*, assim fica facilitada a sua manipulação pelo usuário, estão agrupados de acordo com sua finalidade no Simulador.

Semente aleatória

rand('state',0);	Semente aleatória

Sistema Classificador

NC	N° de classificadores
NCAG	N° de classificadores a serem utilizados pelo AG
ITAG	Intervalo de iterações para se aplicar o AG
NBAT	N° de colisões para se aplicar o AG
Credito	Força alocada aos classificadores da geração inicial
Recompensa	Recompensa pela ação bem sucedida
Punicao	Punição pela ação mal sucedida
PcCrossOver	Probabilidade de *crossover*
PcMutacao	Probabilidade de mutação
nugan	Número de últimos ganhadores (define a matriz dos últimos ganhadores)
NClMatch	Número máximo de classificadores a serem considerados com *matching*
NrItAVoltar	Número máximo de iterações a voltar quando ocorre uma colisão
NrGanAPunir	Número máximo de últimos ganhadores a serem penalizados quando ocorre uma colisão.

O parâmetro **nugan** é utilizado para definir o tamanho de **MUGan**, matriz dos últimos ganhadores, que contém os classificadores que ganharam nas últimas iterações.

Quanto a **NClMatch**, trata-se de uma particularidade dos Sistemas Classificadores com Redes Neurais. No sistema convencional, um classificador em face de uma mensagem da interface de entrada, produz *matching* ou não. Com Redes Neurais o *matching* é relacionado a uma saída da Rede de Avaliação, portanto um número real, o que requer algum critério para considerar-se a existência de *matching*. Assim, neste modelo são considerados classificadores com *matching* apenas aqueles que produzirem saída positiva da rede e, adicionalmente, o seu número é limitado por **NClMatch**, sendo considerados os com maior valor.

NrItAVoltar e **NrGanAPunir** são utilizados em uma rotina disparada quando ocorre uma colisão. O robô volta no máximo **NrItAVoltar** iterações, dependendo do número de iterações desde a última colisão, acumulado em **nitbat**, variável do Controlador, não ser menor, caso em que ele volta **nitbat** iterações. **NrGanAPunir** corresponde ao número máximo de últimos ganhadores que sofrerão punição devido à colisão atual. Este número também é restringido por **nitbat**. A intenção é permitir ao sistema responsabilizar uma seqüência de classificadores que contribuíram para a colisão, e não apenas o atual.

Redes Neurais

GV	Ganho na saída da rede do atuador para amplificar a velocidade
MultPesosInic	Multiplicador para os pesos gerados inicialmente
NNeurCInter	N° de neurônios da camada intermediária (o mesmo para Classificador e Atuador)
ct	Coeficiente da tangente hiperbólica
NNeurCSaidaCl	N° de neurônios da camada de saída dos classificadores (1 ou 2)
NNeurCSaidaAt	Idem Atuadores. Cuidado: a alteração deste parâmetro implica em alterações no *software*

GV é aplicado ao valor da saída da rede neural do atuador, pois esta saída é um número normalmente pequeno, em vista da configuração da rede, para a finalidade de produzir um valor que será subtraído da velocidade de uma das rodas para proporcionar a mudança de direção.

Os pesos das redes neurais que irão compor os classificadores gerados inicialmente correspondem a valores randômicos entre -0.1 e 0.1, com distribuição uniforme. O parâmetro **MultPesosInic** é utilizado como um multiplicador desses resultados, podendo portanto alterar seus limites.

O número de neurônios da camada intermediária é utilizado para ambas as redes. É definido por **NneurCInter**, não existindo restrição quanto ao seu valor. Deve-se, contudo, observar que, como a idéia não é obter-se uma rede que resolva todo o ambiente, mas sim um conjunto de redes, cada uma especializada para soluções particulares, este valor não deve ser grande, provavelmente 1 ou 2 neurônios.

Quanto ao número de neurônios da camada de saída da rede de avaliação, que é utilizada pelo **Classificador**, ou parâmetro **NneurCSaidaCl**, este pode assumir apenas os valores 1 ou 2. Caso seja escolhido o valor 1, será entendido pelo **Classificador** que se pretende atribuir o valor único da saída da rede como valor a ser utilizado para o *matching* e para a especificidade. Com valor 2 cada um desses valores corresponderá a uma das saídas da rede.

Para a alteração do parâmetro **NneurCSaidaAt** requer-se uma mudança no Simulador. O valor que deve ser especificado é 1, que irá corresponder a utilizar a saída da rede de ação para controlar a velocidade diferencial, permitindo mudança de direção. Com mais saídas, por exemplo, com 2, ter-se-ia a possibilidade de controlar duas variáveis do robô, que poderiam ser mudança de direção e mudança de velocidade.

Preparação do ambiente e objetos usando funções Ambiente

A preparação do ambiente e dos objetos é feita através do método **Ambiente**. Já existem vários métodos preparados para diversas situações, a sua identificação é feita pela utilização de Amb ou Ambiente na parte inicial de seu nome. O método Ambiente prepara desenha a figura do Matlab que irá conter o ambiente, incluindo a definição do contorno e dos objetos. Por exemplo, na linha abaixo seria escolhida o método **Ambiente1c**.

[VetorObjetos,contornox, contornoy] = Ambiente1c;

Objeto RobotMovel

V	Velocidade inicial
dt	Intervalo de tempo de uma iteração
Eixo	Eixo das DWS do RobotMovel
Xin	Coordenada x inicial do RobotMovel
Yin	Coordenada y inicial do RobotMovel
TetaIn	Orientação inicial do RobotMovel
alcance	Alcance do sensor
Multdifv	Multiplicador para ajuste de **difv** (também pode ser ajustado por **GV**)
DadosSensor	Vetor com os dados do sensor

As coordenada são medidas em *pixels*, **Xin** na horizontal e **Yin** na vertical. Os ângulos são positivos a partir do eixo horizontal no sentido anti-horário.

O parâmetro **Multdifv** é utilizado para multiplicar o valor de **difv**, parâmetro de saída do **Atuador**, que contém o valor que será subtraído da velocidade de uma das rodas durante o intervalo de tempo **dt** da iteração para provocar a mudança de direção. Por outro lado, deve-se notar que **difv** é obtido no **Atuador** multiplicando-se a saída da rede por **GV**. Assim existem dois parâmetros considerados na obtenção do valor **difv**. Como será visto na seção referente ao Sistema Convencional, lá existe apenas o parâmetro **Multdifv** para fazer ajustes em **difv**.

DadosSensor é um vetor que contém as informações para a composição dos sensores do robô. Cada sensor requer 3 parâmetros na seguinte ordem: posição angular em graus a partir do sentido de deslocamento do robô, abertura em graus e alcance em *pixels*. O exemplo abaixo define 3 sensores, 35 graus à esquerda, em frente e 35 graus à direita, todos com abertura de 10 graus e alcance de 100 *pixels*. No caso de querer-se o mesmo alcance para todos os sensores, pode-se utilizar o parâmetro **alcance**. Cada sensor pode ser definido com características totalmente diferentes dos outros.

DadosSensor = [35 10 100 0 10 100 -35 10 100];

5.8.3 Controlador para o Sistema Classificador Convencional

Este sistema apresenta vários pontos em comum com o sistema relativo às redes neurais. Por isso neste item serão relatados apenas os aspectos específicos, ou que tenham diferenças de interpretação.

Parâmetros gerais de configuração definidos pelo usuário

Semente aleatória

rand('state',0);	Semente aleatória

Sistema Classificador

NC	N° de classificadores
NCAG	N° de classificadores a serem utilizados pelo AG
ITAG	Intervalo de iterações para aplicar o AG
NBAT	N° de colisões para aplicar o AG
Credito	Força alocada aos classificadores da geração inicial
Recompensa	Recompensa pela ação bem sucedida
Punicao	Punição pela ação mal sucedida
PcCrossOver	Probabilidade de *crossover*
PcMutacao	Probabilidade de mutação
nugan	Número de últimos ganhadores (define a matriz dos últimos ganhadores)
NrItAVoltar	Número máximo de iterações a voltar quando ocorre uma colisão
NrGanAPunir	Número máximo de últimos ganhadores a serem penalizados quando ocorre uma colisão.

Nota-se a inexistência do parâmetro **NClMatch**, específico para o Sistema Classificador com Redes Neurais.

Preparação do ambiente e objetos usando métodos Ambiente

A preparação do ambiente é feita de forma idêntica à do Sistema Classificador com Redes Neurais.

Objeto RobotMovel

V	Velocidade inicial
dT	Intervalo de tempo de uma iteração
Eixo	Eixo das DWS do RobotMovel
Xin	Coordenada x inicial do RobotMovel
Yin	Coordenada y inicial do RobotMovel
TetaIn	Orientação inicial do RobotMovel
alcance	Alcance do sensor
Multdifv	Multiplicador para ajuste de **difv** (também pode ser ajustado por **GV**)
DadosSensor	Vetor com os dados do sensor

Estes parâmetros são estabelecidos de forma idêntica aos correspondentes do Sistema Classificador com Redes Neurais. A única diferença é que, neste caso, o parâmetro **Multdifv** é o único multiplicador para definir o valor de **difv.**

Capítulo 6: Aplicações

Neste capítulo é apresentada a parte prática desta pesquisa. São mostrados experimentos efetuados utilizando o simulador para as duas aplicações de sistemas classificadores, a convencional e a utilizando redes neurais. No texto deste capítulo são utilizados alguns termos referentes a parâmetros mostrados no "Capítulo 5: Detalhes do Simulador", sendo que não é essencial o seu conhecimento para entendimento do que aqui é apresentado. No entanto, os leitores que quiserem tirar um maior proveito desta leitura devem, pelo menos, lerem o item "O Controlador" do Capítulo 5. As conclusões e perspectivas de trabalhos futuros são mostradas no Capítulo 7.

6.1 As Aplicações

As duas aplicações consideram um robô móvel com características semelhantes às do mini robô *Khepera* [**21**] (ver apêndice A), com um sistema inteligente que lhe propicie a capacidade de adquirir autonomia para se deslocar em um ambiente com obstáculos, deslocando-se com velocidade constante e aprendendo a não colidir. Assim, o único objetivo do robô nestas aplicações é não colidir.

O robô inicia o seu aprendizado, em um ambiente com obstáculos, sem a suposição de ter conhecimento inicial, inicializando seus classificadores de forma aleatória, sendo utilizadas várias sementes para a função geradora, pretendendo-se, dessa forma, verificar a independência das condições iniciais.

O sistema inteligente tem como mensagem de entrada as informações detectadas pelos sensores sobre o ambiente e seus obstáculos. Cada sensor do robô simulado divide seu campo de atuação em três regiões para detecção: perto, meia distância e longe, fornecendo através da interface de entrada um vetor de 3 elementos para identificarem os valores detectados destas regiões, 0 para a ausência de obstáculo na região e 1 para a presença. A mensagem de entrada é formada por um vetor obtido pela concatenação dos vetores dos sensores.

A saída fornecida pelo sistema inteligente é uma informação para comandar a diminuição da velocidade de uma das rodas do robô, durante a iteração de duração dt executada pelo controlador, obtendo-se assim mudança na sua direção de deslocamento.

Para se executar um experimento com o simulador, em primeiro lugar é escolhido o controlador:

Controlador.m	para SCRN, ou
CSContolador.m	para SC.

Em seguida devem ser especificados os "PARÂMETROS GERAIS DE CONFIGURAÇÃO DEFINIDOS PELO USUÁRIO", encontrados na parte inicial do controlador. Estes parâmetros configuram o simulador para o experimento e definem as características do ambiente, dos obstáculos, do robô com seus sensores e do sistema inteligente.

Uma vez definidas as características requeridas para o robô, ambiente e obstáculos, a escolha dos parâmetros para o sistema inteligente é conseguida por procedimentos empíricos, envolvendo

tentativas e erros, intuição e bom senso. Nos próximos itens são apresentados os parâmetros utilizados nos experimentos.

6.1.1 Configuração Geral

O robô

A intenção é simular, em uma primeira aproximação, o mini robô *Khepera*. Nesta primeira aproximação, especificações exatas com relação às dimensões, velocidade, aceleração e sensores do robô não são estritamente consideradas, sendo que o ponto central é atender apenas algumas características gerais de seu comportamento.

Primeiro com relação à sua forma que é aproximadamente cilíndrica, a qual foi representada no ambiente bidimensional por um círculo de 50 *pixels* de diâmetro. Em segundo lugar tem-se a maneira como é efetuada a sua mudança de direção. O *Khepera* possui duas rodas que permitem o seu deslocamento e orientação, sendo que o eixo que as une foi considerado como tendo as mesmas dimensões do diâmetro do círculo correspondente ao seu contorno. Cada uma das rodas tem um motor, e a mudança de direção é efetuada comandando-se velocidades diferentes para elas. Este tipo de comando é designado na literatura por ***Differential Wheels Steering*** (DWS), que será traduzido neste texto para **Direção por Rodas Diferenciais**. Este comportamento é atendido pela utilização do método **DWS** da classe **RobotMovel**.

Finalmente, são simulados os seus sensores de distância e proximidade. Os sensores de distância têm os valores detectados, pelo método **Detecta**, escalonados em três regiões pré-definidas, sendo possível especificar a posição, a abertura e o alcance de cada sensor. São utilizados dois sensores de distância nos experimentos. O método **Detecta** também efetua a detecção de contato (ou invasão) com os contornos do ambiente e/ou algum obstáculo.

Quanto às iterações, correspondem a ciclos de leitura dos sensores, verificação de colisão e deslocamento, que será efetuado à velocidade constante, de 50 *pixels* por unidade de tempo, exceto nas mudanças de orientação do robô, quando uma das rodas tem sua velocidade alterada durante o intervalo de tempo *dt* da iteração, para ser restabelecida depois. É utilizado o intervalo de tempo de 0.25 da unidade considerada [segundos].

Parâmetros Utilizados para o Robô

Com estas considerações, a seguir são apresentados os parâmetros do robô conforme utilizado na simulação.

V	50
dT	.25
Eixo	50
alcance	100
Multdifv	.025 c/ Redes Neurais e 1.5 Convencional
DadosSensor	[35 30 alcance -35 30 alcance]

Tabela 6.1: Parâmetros para o robô

Vê-se que o robô tem velocidade constante **V** de 50 *pixels* por unidade de tempo, cada iteração é efetuada em **dT** = 0.25 unidade de tempo, o **Eixo** das rodas diferenciais, que na simulação também corresponde ao seu diâmetro externo, é de 50 pixels. Os sensores são especificados por

DadosSensor, um vetor cujos elementos, de 3 em 3, definem a posição angular do sensor, a sua abertura e seu alcance. Assim, tem-se dois sensores de distância, iguais e simétricos com relação ao seu eixo de deslocamento, um a 35 graus à esquerda e o outro 35 à direita, tendo aberturas de 30 graus e **alcance** de 100 *pixels*.

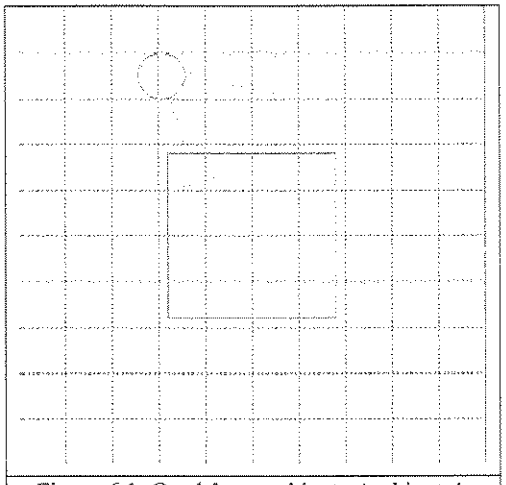


Figura 6.1: O robô no ambiente Ambiente1.
O Ambiente1 é delimitado por um quadrado com 500 *pixels* de lado, o objeto central é o obstáculo, um quadrado com 180 *pixels* de lado.

A Fig. 6.1 ilustra o robô na sua configuração para os experimentos, evidenciando a área de detecção de seus dois sensores de distância, em um dos ambientes de simulação, o **Ambiente1**.

O **Atuador** produz o valor **difv**, que é aplicado à velocidade da roda esquerda ou direita para diminuí-la e permitir a mudança de direção. No sistema com redes neurais este valor é um número real, saída da rede neural de ação. No sistema convencional ele é o obtido pelo mapeamento do conseqüente do classificador vitorioso em valores pré-definidos, mostrados na Tab. 6.2. Observar que o conseqüente é formado por três genes, com valores 0 ou 1 cada.

Conseqüente	difv
[0 0 0]	-0.8
[0 0 1]	-0.5
[0 1 0]	- 0.2
[0 1 1]	- 0.0
[1 0 0]	0.0
[1 0 1]	0.2
[1 1 0]	0.5
[1 1 1]	0.8

Tabela 6.2: Valores de difv para o SC

Limitações do Modelo do Robô

No capítulo 4 foram vistas vantagens em se utilizar um simulador. O simulador é um *software* cujos parâmetros e mecanismos de funcionamento podem ser modificados de forma mais simples e rápida do que no *hardware* real. Assim, é interessante evidenciar que o simulador pode ser utilizado na fase de desenvolvimento do sistema inteligente de controle para entendê-lo mais profundamente, de tal forma a permitir a introdução de modificações fundamentais necessárias ao modelo, até se obter o desempenho esperado. Este processo pode ser considerado como uma fase do desenvolvimento virtual de grande utilidade, e, uma vez que a funcionalidade fundamental do sistema não depende inteiramente de sua complexidade, um ganho significativo pode ser obtido utilizando-se modelos mais simplificados. Assim, a limitação na modelagem ora implementada não é acidental e nem meramente baseada em restrições de recursos, principalmente computacionais. Deve ser normalmente entendida como uma fase do desenvolvimento, que será seguida, posteriormente, pela modelagem completa, pelo menos até os limites em que isso possa realmente ser feito e/ ou que seja significativo ser feito.

As principais limitações impostas ao modelo para os experimentos aqui apresentados são:

- Não foi incluído o modelo dinâmico do robô
- Utiliza apenas dois sensores de distância
- O intervalo de tempo para as iterações é de 0.25 [s].

Estas simplificações trazem duas conseqüências imediatas de grande relevância nesta fase:

1. Baixo custo computacional, que pode tanto ser considerado na necessidade de recursos computacionais, quanto em menor tempo para o desenvolvimento dos experimentos.
2. A análise qualitativa dos resultados com menos sensores fica mais facilitada em vista da diminuição da quantidade de informação.

O Ambiente

O robô se desloca em um ambiente bidimensional em que é possível definir-se o seu contorno e os obstáculos. Existem já prontos para uso, como métodos de ambiente acompanhando o *software* do simulador, vários ambientes de simulação, com seus contornos e obstáculos. Estes métodos de ambiente são invocados pelo controlador e estão acessíveis ao usuário nos parâmetros gerais de configuração do simulador. Dentre eles temos as seguintes:

- **Ambiente0.m**: Ambiente com contorno quadrado de 500 *pixels* de lado, com um obstáculo quadrado de 40 *pixels* de lado no seu centro.
- **Ambiente1.m**: Ambiente com contorno quadrado de 500 *pixels* de lado, com um obstáculo quadrado de 180 *pixels* de lado no seu centro, Fig. 6.1.
- **AmbContornoCorredor1.m**: É um corredor situado numa figura com 800 por 800 *pixels* de superfície, Fig. 6.2.

É possível obter-se facilmente outros ambientes por edição desses métodos já existentes.

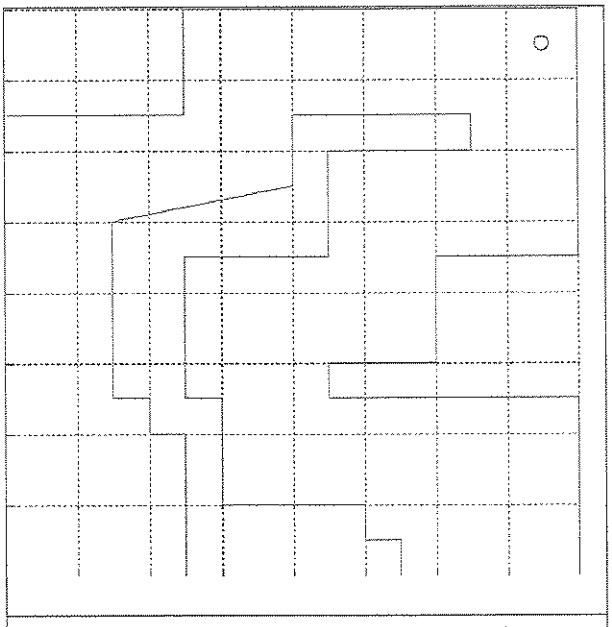


Figura 6.2: Ambiente AmbContornoCorredor1

6.1.2 Cuidados ao Interpretar os Resultados

Órbitas Locais ou Exploração de Todo o Ambiente

O robô tem o objetivo de se desviar dos obstáculos deslocando-se com velocidade constante. Assim, para um dado ambiente, após a fase de aprendizado, o robô poderá assumir inúmeros comportamentos, como soluções para este ambiente, alguns extremamente simples e outros potencialmente mais sofisticados. Por exemplo, no **Ambiente1**, Fig. 6.1, uma solução é obtida com o robô descrevendo um movimento em torno do objeto central no sentido horário ou anti-horário. Este movimento pode ser feito com o robô mais perto dos contornos externos ou do objeto central, ou ainda, alguma mistura destes comportamentos. Além disso, o robô pode simplesmente ficar em um movimento circular em qualquer parte do ambiente, em qualquer sentido, horário ou anti-horário.

Em qualquer desses resultados o robô terá atingido o comportamento de não colidir. Então qual o melhor? A rigor todos são bons pois estão de acordo com o objetivo definido, mas se o movimento resultante explorar toda a extensão do ambiente existe a possibilidade de o robô estar resolvendo um número maior de situações, obtidas a partir de seus sensores.

Por isso, é feita a tentativa de induzi-lo a este comportamento, alterando-se **Multdifv**, Tab. 6.1. Quanto maior este valor, maior a versatilidade do robô para alterar a sua direção, o que aumenta a probabilidade dele estabilizar em órbitas locais. Quanto menor, maior a probabilidade dele se deslocar por regiões mais extensas do ambiente, mas com o cuidado de não diminuí-lo demais, o que resultaria na impossibilidade de executar manobras para desviar de obstáculos. É claro que nem sempre é possível encontrar-se o valor ideal de **Multdifv**, sendo então aceita a órbita local.

Acompanhamento da Simulação

A execução do simulador permite o acompanhamento visual do desenrolar do processo. O robô é colocado em uma determinada posição inicial no ambiente e passará a se deslocar aprendendo a não colidir. O seu comportamento é apresentado em três figuras do *Matlab* que são atualizadas a cada iteração. A figura 1 do Matlab contém o ambiente de simulação, como, por exemplo, na Fig.

6.1; a figura 2 contém o gráfico do nº de iterações entre colisões versus nº de iterações, Fig. 6.3; e a figura 3 o gráfico nº de colisões dividido pelo nº de iterações versus nº de iterações, Fig. 6.4.

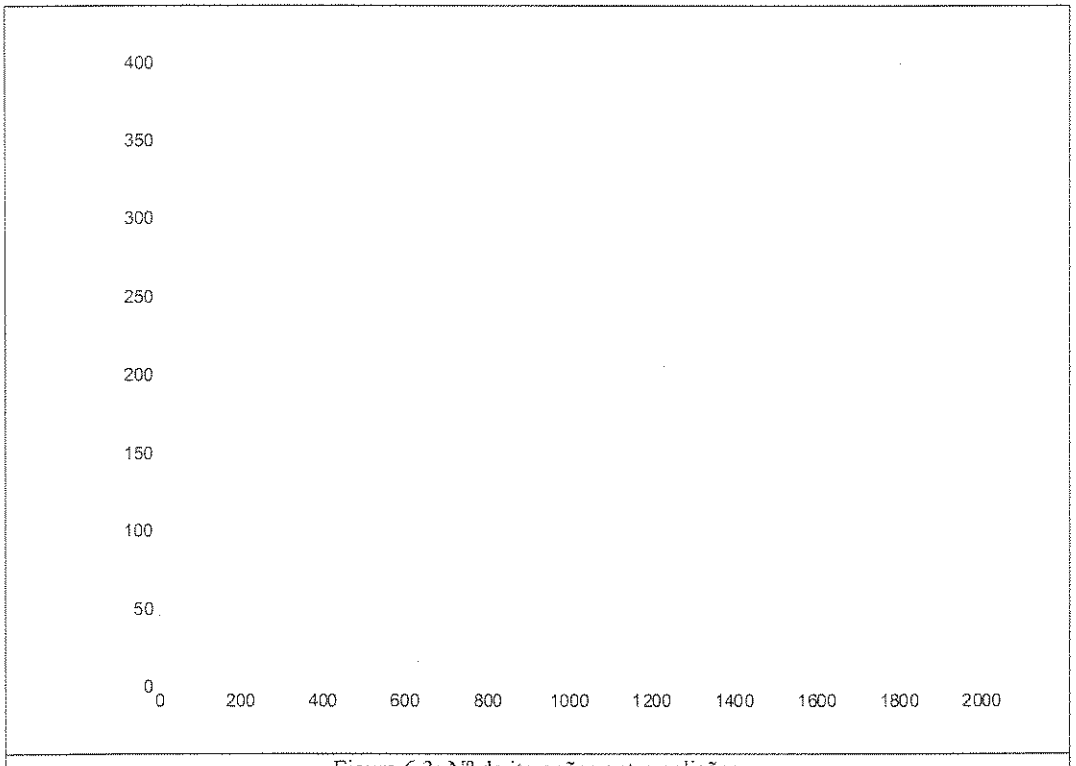


Figura 6.3: Nº de iterações entre colisões.

Eixo Vertical: nº de colisões

Eixo Horizontal: nº de iterações

Este gráfico deve ser lido apenas nos pontos de descontinuidade Observar que após a iteração um pouco acima da de número 1800 não ocorrem mais colisões até o término do experimento.

O gráfico da Fig. 6.3 é típico para gráfico de barras verticais, tendo sido utilizado traçado contínuo para melhor visualização. Por isso os valores devem ser lidos nos pontos de descontinuidade.

O gráfico da Fig. 6.4 ilustra o aprendizado do robô de forma dinâmica. Quanto menor o valor do número de colisões divido pelo número de iterações, maior o aprendizado atingido.

Quanto aos valores obtidos nos gráficos, deve-se tomar o cuidado de não entendê-los de forma absoluta, eles são sempre relativos ao experimento em questão. E, mesmo assim, deve-se considerar o fato de que o robô, após cada colisão, é re-posicionado no local em que estava em um certo número de iterações anteriores, o que pode lhe propiciar um certo número de iterações sem colidir.

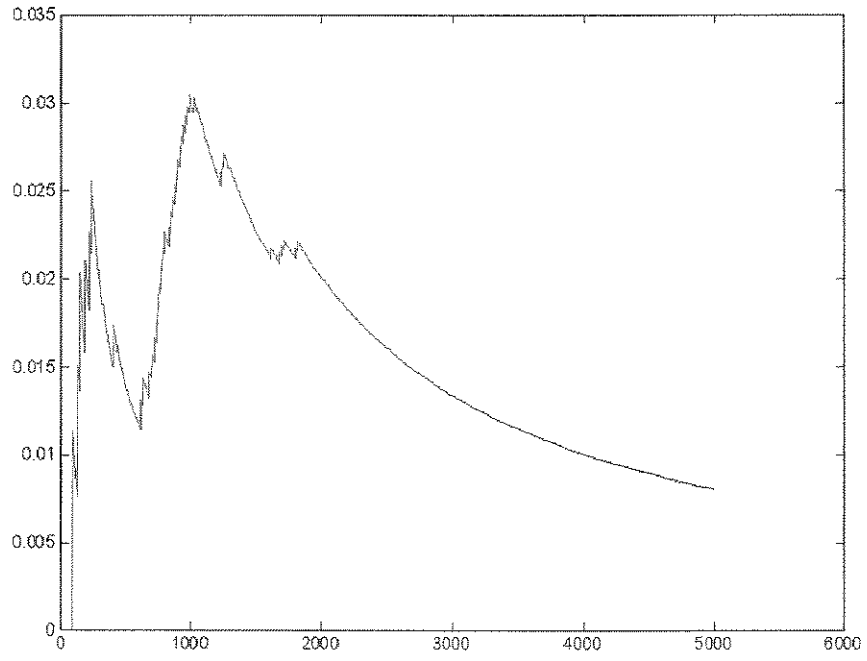


Figura 6.4: Curva de Aprendizagem
Eixo Vertical: nº de colisões / nº de iterações
Eixo Horizontal: nº de iterações

A simples observação dos gráficos das figuras 6.3 e 6.4 pode levar a interpretações errôneas, dependendo do resultado que se procura. Por exemplo, se além do objetivo de não colidir quer-se ajustar os parâmetros para forçar o robô a explorar um trecho maior do ambiente, tais gráficos podem indicar bons resultados de desempenho apesar do robô não ter atingido o objetivo esperado. Isto mostra que o acompanhamento visual pode ser de grande relevância na análise. Note que pode-se fazer uso do *Workspace* ("ver a seguir Usando o *Workspace*") do simulador para conseguir-se obter dados auxiliares no sentido de melhorar a análise e, conseqüentemente, a qualidade das conclusões.

6.1.3 Usando o *Workspace*

O ***workspace*** é a área onde o Matlab armazena as variáveis utilizadas pelos programas Após a interrupção da simulação, pode-se utilizar o *workspace* do controlador. Ele contém, além dos parâmetros, inúmeras variáveis capazes de fornecer informações que permitem analisar o comportamento do robô. Entre elas, as de maior relevância são: **MC** – Matriz dos classificadores, **MUGan** – Matriz dos últimos ganhadores e **MObVeículo** – Matriz de observação do veículo.

O exame de **MC** é útil para se verificar o estado dos classificadores e sua força. A **MUGan** permite verificar quais os classificadores que têm participado da solução e com que freqüência A **MObVeículo** registra as coordenadas, velocidade e orientação do veículo, permitido reproduzir a sua trajetória em qualquer período do experimento.

Foi desenvolvido um programa para ser executado sobre o *workspace*, por exemplo, ao finalizar um teste do simulador, ou, ainda, carregando-se para a memória um *workspace* de um experimento efetuado anteriormente e salvo em disco. Trata-se do **VeResultados**, que, basicamente, origina uma

cópia de **MC**, denominada **MCSort**, ordenada pelos genes dos classificadores, e uma cópia de **MUGan**, denominada **MUGanSort**, ordenada pelo número do classificador. **MCSort** facilita a observação da incidência de clones e **MUGanSort** evidencia quais classificadores participaram das últimas iterações. O número de linhas de **MUGan** define a quantidade de últimos ganhadores que estão sendo guardados nela e é definido por **nugan**.

Além disso o programa **VeResultados** reconstitui as figuras 1, 2 e 3, sendo que a figura 1 passa a mostrar o traçado do percurso efetuado pelo robô.

6.2 Sistema Classificador com Redes Neurais

A seguir são apresentados os parâmetros gerais e os experimentos efetuados com os módulos do sistema classificador com redes neurais utilizando o simulador.

6.2.1 Parâmetros Gerais

Parâmetros para o Cálculo do Aposta

Os parâmetros para o cálculo da aposta não estão diretamente acessíveis para o usuário. O cálculo da aposta através da expressão 2.2 utiliza os seguintes valores para **k0**, **k1**, **k2** e **k3**, especificados no método **Classificador**:

k0 = 0.1

k1 = 0

k2 = 1

k3 = 1

A taxa de aposta **TaxaBid** também é especificada no método **Classificador** recebendo o valor:

TaxaBid = 0.75

A taxa de vida é obtida pela expressão 2.3 através do programa **Controlador**, sendo que o parâmetro **n**, relativo à vida média medida em iterações, é determinado por:

$$\mathbf{n} = \mathbf{ntv.NBAT} \qquad \textbf{(6.1)}$$

onde:

ntv = **número médio de iterações sem colidir**

NBAT = **número de colisões para aplicar o AG**

O parâmetro **NBAT** é especificado diretamente pelo usuário e o seu valor utilizado é apresentado no item a seguir. Foi adotado para **ntv** o valor:

ntv = 10

Parâmetros para o Sistema Classificador e para o Algoritmo Genético

Os parâmetros para o sistema classificador são definidos no programa **Controlador** e valem:

NC	40	N° de classificadores
NCAG	8	N° de classificadores a serem utilizados pelo AG
ITAG	2000	Intervalo de iterações para se aplicar o AG
NBAT	40	N° de colisões para se aplicar o AG
Credito	50	Força alocada aos classificadores da geração inicial
Recompensa	1	Recompensa pela ação bem sucedida

Punicao	5	Punição pela ação mal sucedida
PcCrossOver	0.2	Probabilidade de *crossover*
PcMutacao	0.02	Probabilidade de mutação
nugan	200	Número de últimos ganhadores (define a matriz dos últimos ganhadores)
NClMatch	5	Número máximo de classificadores a serem considerados com *matching*
NrItAVoltar	12	Número máximo de iterações a voltar quando ocorre uma colisão
NrGanAPunir	7	Número máximo de últimos ganhadores a serem penalizados quando ocorre uma colisão.

Parâmetros para as Redes Neurais

Estes parâmetros são definidos no programa **Controlador**, assumindo os valores:

GV	120	Ganho na saída da rede do atuador para amplificar a velocidade
MultPesosInic	2	Multiplicador para os pesos gerados inicialmente
NneurCInter	1	N° de neurônios da camada intermediária (o mesmo para rede de Avaliação e rede de Ação)
ct	1	Coeficiente da tangente hiperbólica
NneurCSaidaCl	1	N° de neurônios da camada de saída da rede de Avaliação (1 ou 2)
NneurCSaidaAt	1	Idem Atuadores. Cuidado: a alteração deste parâmetro implica em alterações no *software*

Lembrando que são utilizadas redes neurais tipo perceptron para as redes de avaliação e de ação, com as definições acima temos as seguintes configurações:

- Rede de Avaliação: 1 neurônio na camada intermediária e 1 neurônio na camada de saída.
- Rede de Ação: 1 neurônio na camada intermediária e 1 neurônio na camada de saída.

O número de neurônios na camada de entrada é determinado pelo número de regiões a serem detectadas pelos sensores, apresentado no item a seguir.

Parâmetros para o objeto RobotMovel

Os parâmetros para o robô são definidos no programa **Controlador**, sendo adotados os seguintes valores:

V	50	Velocidade inicial
dT	.25	Intervalo de tempo de uma iteração
Eixo	50	Eixo das DWS do RobotMovel
Xin	50	Coordenada x inicial do RobotMovel
Yin	25	Coordenada y inicial do RobotMovel
TetaIn	90	Orientação inicial do RobotMovel
alcance	100	Alcance do sensor
Multdifv	.025	Multiplicador para ajuste de **difv** (também pode ser ajustado por **GV**)
DadosSensor	[35 30 alcance -35 30 alcance]	Vetor com os dados do sensor

Pela configuração acima, são utilizados dois sensores para o robô. Cada sensor identifica 3 regiões de detecção. Assim, teremos 6 regiões de detecção, compondo o vetor de entrada das redes neurais. Considerando uma entrada para polarização, as redes neurais de avaliação e de ação, que são do tipo perceptron, terão 7 neurônios na camada de entrada. Este resultado é calculado e adotado automaticamente pelo simulador.

6.2.2 Experimentos e Resultados

Estes experimentos utilizam várias sementes para o método gerador dos valores aleatórios.

Experimento com Ambiente0

Cada experimento vai até 2000 iterações, com ITAG = 250, o que garante, no mínimo, 3 evoluções do Algoritmo Genético, se NBAT = 40 não for atingido nenhuma vez. O número de últimos ganhadores **nugan** para definir o número de linhas de **MUGan** é de 200.

Semente	Nº total de Colisões	Nº de Ganhadores Nugan = 200	Saídas diferentes da Rede de Avaliação	Existência de Clones	Característica dominante do movimento do robô
0	27	1	3	Sim	Órbita local anti-horária.
1	39	1	7	Sim	Órbita local anti-horária
2	24	1	7	Sim	Órbita local anti-horária
3	26	1	3	Sim	Órbita local anti-horária
4	114	2	6	Sim	Órbita local anti-horária
5	1	1	2	Sim	Movimento horário em torno do objeto central
6	21	1	4	Sim	Órbita local anti-horária

Tabela 6.3: Resultados do experimento usando SCRN no Ambiente0

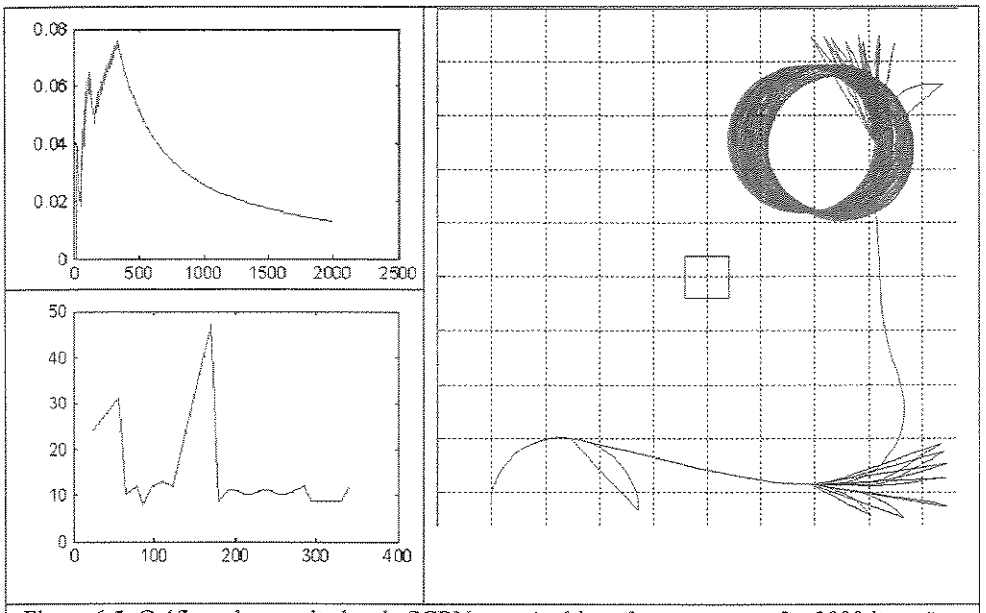


Figura 6.5: Gráficos dos resultados do SCRN para Ambiente0 com semente 3 e 2000 iterações. No sentido horário, a partir do canto superior esquerdo: "Curva de Aprendizado" (eixo vertical: nº de colisões/nº de iterações, eixo horizontal: nº de iterações), "Ambiente de Simulação" com o traçado do percurso do robô e "Nº de Iterações Entre Colisões" (eixo vertical: nº de iterações entre colisões, eixo horizontal: nº de iterações).

Na Tab. 6.3, o nº de Ganhadores corresponde ao número de classificadores diferentes que ganharam nas últimas **nugan** iterações. As saídas diferentes são aquelas produzidas pelas redes de avaliação desses classificadores.

Como resultado geral nota-se que fica mantida a diversidade para os classificadores, há uma predominância de órbitas locais, tendo ocorrido apenas um caso, para semente 5, em que o robô contorna o objeto central. O objetivo de aprender a não colidir foi atendido para todas as sementes mostrando independência das condições iniciais para este ambiente simples.

É resultado relevante destes experimentos o fato de um único classificador, exceto para semente 4, levar o robô ao comportamento desejado. Vê-se também, para esses casos, a ocorrência de um mínimo de 2 saídas diferentes das redes de avaliação e um máximo de 7, o que indica a possibilidade desse único classificador estar produzindo comportamentos diferentes.

Experimento com Ambiente1

Cada experimento vai até 10000 iterações, utiliza ITAG = 2000, garantindo no mínimo 4 evoluções do Algoritmo Genético, se NBAT = 40 não for atingido. O número de últimos ganhadores **nugan** para definir o número de linhas de **MUGan** é de 200.

Na Tab. 6.4, o nº de ganhadores corresponde ao número de classificadores diferentes que ganharam nas últimas **nugan** iterações. As saídas diferentes são aquelas produzidas pelas redes de avaliação desses classificadores.

Semente	Nº total de Colisões	Nº de ganhadores Nugan = 200	Saídas diferentes da Rede de Avaliação	Existência de Clones	Classificadores com força maior que 1	Característica dominante do movimento do robô
0	653	2	7	Muitos	10	Muitas colisões até conseguir movimento anti-horário em

						torno do objeto central
1	318	1	5	Sim	1 (o ganhador)	Movimento horário em torno do objeto central
2	46	1	6	Sim	7	Movimento horário em torno do objeto central
3	545	?	?	Muitos	0	Não atinge o objetivo
4	923	?	?	Sim	12	Não atinge o objetivo
5	26	3	4	Sim	11	Movimento horário em torno do objeto central
6	852	1	5	Sim	1 (o ganhador)	Muitas colisões até entrar em movimento horário em torno do objeto central.

Tabela 6.4: Resultados do experimento usando SCRN no Ambiente1

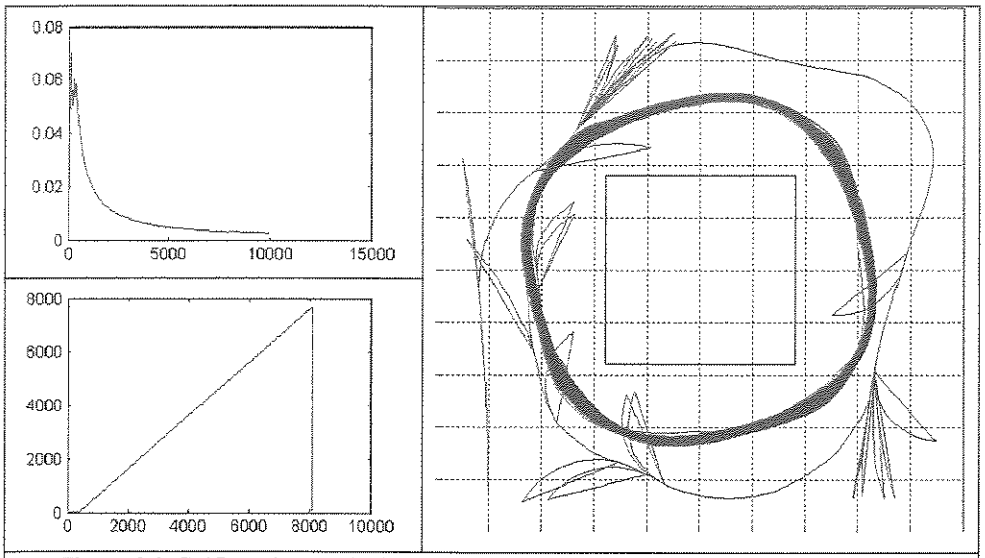


Figura 6.6: Gráficos dos resultados do SCRN para Ambiente1 com semente 5 e 10.000 iterações.

No sentido horário, a partir do canto superior esquerdo: "Curva de Aprendizado" (eixo vertical: nº de colisões/nº de iterações, eixo horizontal: nº de iterações), "Ambiente de Simulação" com o traçado do percurso do robô e "Nº de Iterações Entre Colisões" (eixo vertical: nº de iterações entre colisões, eixo horizontal: nº de iterações).

A simulação neste ambiente apresenta maior dificuldade que no caso anterior e, em geral, é mantida a diversidade para os classificadores. O tamanho do objeto central dificulta a ocorrência de órbitas locais. O objetivo de aprender a não colidir não foi atendido em dois casos, para sementes 3 e 5. É evidente uma maior dependência das condições iniciais. Por exemplo, com semente 5 foi obtido um resultado surpreendente, apenas 26 colisões nas 10000 iterações, enquanto que para as sementes 4 e 6 ocorreram 923 e 852 colisões, respectivamente, sendo que para semente 4 não foi atingido o objetivo.

É relevante também notar que em 3 casos tivemos apenas 1 classificador levando o robô ao comportamento desejado, a despeito de ser necessária, nesses casos, a ocorrência de um mínimo de 5 saídas diferentes das Redes de Avaliação e um máximo de 6.

Experimentos com Outros Ambientes

O sistema classificador com redes neurais é uma abordagem inovadora de sistemas classificadores e apresenta características de comportamento que o distingue do convencional. Para que possa ser bem sucedido em situações mais complexas é necessário fazer-se modificações, algumas das quais sugerimos neste trabalho.

6.3 Sistema Classificador Convencional

A seguir são apresentados os parâmetros gerais e os experimentos efetuados com os módulos do sistema classificador convencional utilizando o simulador.

6.3.1 Parâmetros gerais

Parâmetros para o Cálculo da Aposta

Estes parâmetros têm os seguintes valores:

k0 = 1

k1 = 0

k2 = 1

k3 = 1

TaxaBid = 0.75

ntv = 10

O significado desses parâmetros pode ser encontrado no item correspondente para o Sistema Classificador com Redes Neurais, onde poderá ser notado que foram utilizados os mesmos valores.

Parâmetros para o Sistema Classificador e para o Algoritmo Genético

Foram utilizados os seguintes valores para esses parâmetros:

NC	40	Nº de classificadores
NCAG	round(.2*NC)	Nº de classificadores a serem utilizados pelo AG
ITAG	2000	Intervalo de iterações para aplicar o AG
NBAT	100	Nº de colisões para aplicar o AG
Credito	100	Força alocada aos classificadores da geração inicial
Recompensa	1	Recompensa pela ação bem sucedida
Punicao	10	Punição pela ação mal sucedida
PcCrossOver	0.5	Probabilidade de *crossover*
PcMutacao	0.02	Probabilidade de mutação
nugan	200	Número de últimos ganhadores (define a matriz dos últimos ganhadores)
NrItAVoltar	15	Número máximo de iterações a voltar quando ocorre uma colisão
NrGanAPunir	7	Número máximo de últimos ganhadores a serem penalizados quando ocorre uma colisão.

Parâmetros para o objeto RobotMovel

São utilizados os seguintes valores:

V	50	Velocidade inicial
dT	.25	Intervalo de tempo de uma iteração
Eixo	50	Eixo das DWS do RobotMovel
Xin	50	Coordenada x inicial do RobotMovel
Yin	25	Coordenada y inicial do RobotMovel
TetaIn	90	Orientação inicial do RobotMovel
alcance	100	Alcance do sensor
Multdifv	1.5	Multiplicador para ajuste de **difv** (também pode ser ajustado por **GV**)
DadosSensor	[35 30 alcance -35 30 alcance]	Vetor com os dados do sensor

6.3.2 Experimentos e Resultados

Experimento com Ambiente0

Os experimentos vão até 1200 iterações e são interrompidos automaticamente.

É usado **ITAG** = 250 para conseguir a atuação do AG algumas vezes, mesmo se tiver sucesso rápido e não der para ser acionado por não ocorrer **NBAT** = 100. Se não ocorrer **NBAT** o último disparo do AG será em 1000, 200 iterações antes do término.

O número de últimos ganhadores **nugan** para definir o número de linhas de **MUGan** é de 200.

Semente	N° total de Colisões	N° de ganhadores Nugan = 200	Ações diferentes	Existem genéricos nos ganhadores	Existência de Clones	Classificadores com força menor que 1	Característica dominante do movimento do robô
0	25	6	5	Não	Sim	0	Circunda o ambiente em sentido anti-horário.
1	3	4	3	Todos	Sim	0	Órbita horária local
2	12	3	4	2	Sim	0	Circunda o ambiente em sentido anti-horário
3	67	?	?	Não	Sim	0	Não atinge o objetivo.
4	10	15	5	14	Sim	Alguns	Circunda o ambiente em sentido horário
5	34	2	2	1	Sim	0	Circunda o ambiente em sentido horário
6	11	6	5	Sim	Sim	1	Circunda o ambiente em sentido horário

Tabela 6.5: Resultados do experimento usando SC no Ambiente0

Na tabela 6.5, o nº de ganhadores corresponde ao número de classificadores diferentes que ganharam nas últimas **nugan** iterações. As ações diferentes são aquelas determinadas pelo conseqüente desses classificadores. A existência de genéricos se refere também aos últimos **nugan** ganhadores.

É mantida a diversidade para os classificadores e há uma tendência de circundar o ambiente, tendo ocorrido apenas um caso, para semente 3, em que o robô não atingiu o aprendizado de não colidir. Não existe grande dependência das condições iniciais.

Normalmente são necessários vários classificadores, mesmo quando existem classificadores genéricos e, exceto em 1 caso, o número de classificadores é maior que o número de ações diferentes produzidas pelo sistema.

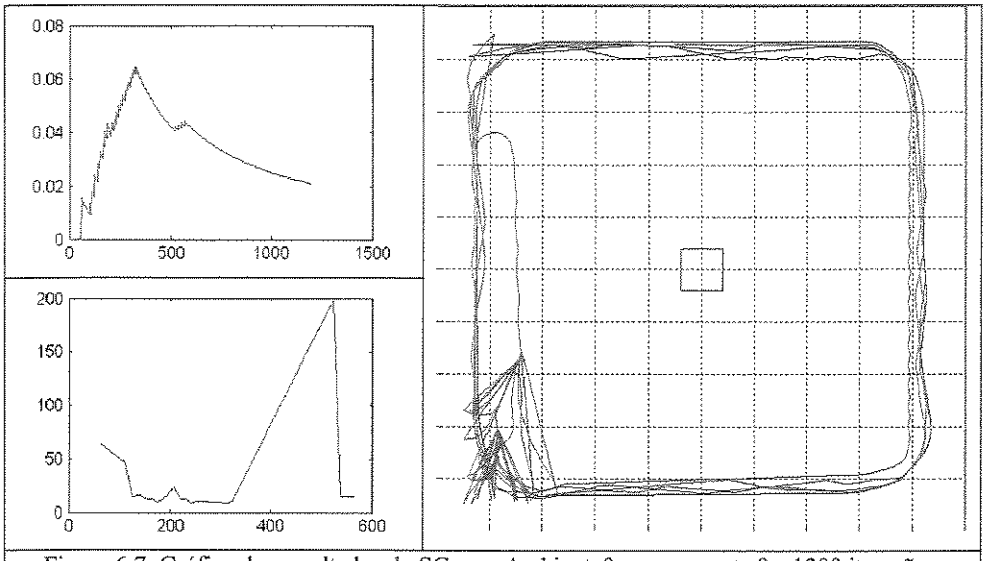


Figura 6.7: Gráfico dos resultados do SC para Ambiente0 com semente 0 e 1200 iterações.
No sentido horário, a partir do canto superior esquerdo: "Curva de Aprendizado" (eixo vertical: nº de colisões/nº de iterações, eixo horizontal: nº de iterações), "Ambiente de Simulação" com o traçado do percurso do robô e "Nº de Iterações Entre Colisões" (eixo vertical: nº de iterações entre colisões, eixo horizontal: nº de iterações).

Experimento com Ambiente1

Os experimentos vão até 5000 iterações e são interrompidos automaticamente.

É usado ITAG = 800 para que se tenha a atuação do AG algumas vezes (pelo menos 6), mesmo se convergir cedo e não der para ser utilizado NBAT = 100. Se não ocorrer NBAT o último disparo do AG será em 4800, 200 iterações antes do término.

Foi preciso ajustar **Multdifv**, que estava em 1.5, passando-o para 1.3, pois havia maior tendência para órbita local.

Estes experimentos são efetuados utilizando-se várias sementes para o método gerador dos valores aleatórios. O número de últimos ganhadores **nugan** para definir o número de linhas de **MUGan** é de 200.

Na tabela 6.6 abaixo, o nº de ganhadores corresponde ao número de classificadores diferentes que ganharam nas últimas **nugan** iterações. As ações diferentes são aquelas determinadas pelo conseqüente desses classificadores. A existência de genéricos se refere também aos últimos **nugan** ganhadores.

Semente	Nº total de Colisões	Nº de ganhadores Nugan = 200	Ações diferentes	Existem genéricos nos ganhadores	Existência de Clones	Classificadores com força menor que 1	Característica dominante do movimento do robô
0	40	5	4	Não	Sim	23	Circunda o ambiente em sentido horário.
1	21	8	5	Não	Sim	19	Circunda o ambiente em sentido horário.
2	82	6	3	Não	Sim	29	Circunda o ambiente em sentido horário.
3	22	5	3	3	Sim	27	Circunda o ambiente em sentido horário.
4	23	2	2	2	Sim	29	Circunda o ambiente em sentido horário.

Tabela 6.6: Resultados do experimento usando SC no Ambiente1

É mantida diversidade para os classificadores e há uma tendência de circundar o ambiente. Não existe dependência das condições iniciais.

Normalmente são necessários vários classificadores atuando para atingir o objetivo de não colidir, mesmo quando existem genéricos e, exceto em 1 caso, o número de classificadores é maior que o número de ações diferentes produzidas pelo sistema. Este comportamento é bem parecido com o apresentado no experimento anterior, com Ambiente 0.

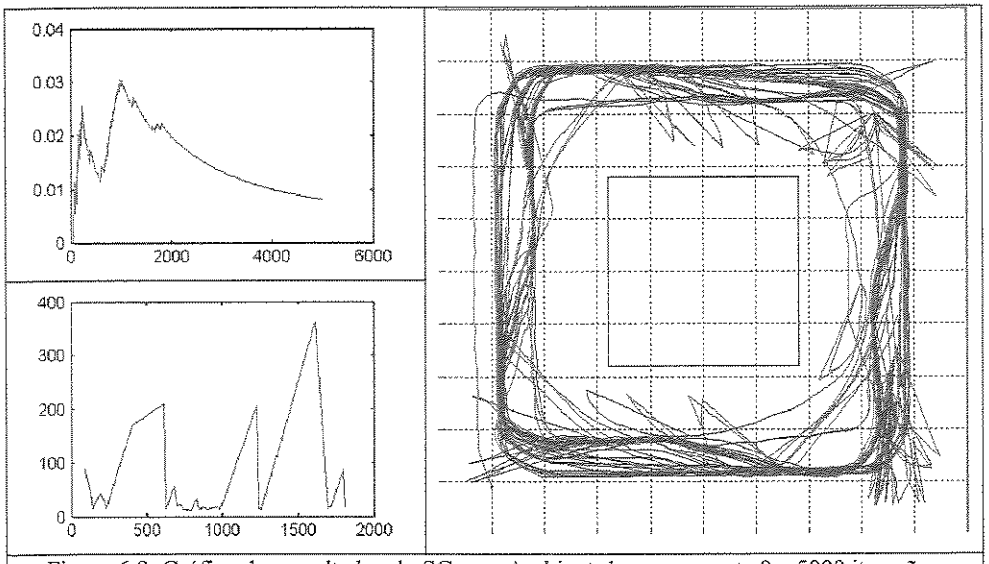


Figura 6.8: Gráfico dos resultados do SC para Ambiente1 com semente 0 e 5000 iterações.
No sentido horário, a partir do canto superior esquerdo: "Curva de Aprendizado" (eixo vertical: nº de colisões/nº de iterações, eixo horizontal: nº de iterações), "Ambiente de Simulação" com o traçado do percurso do robô e "Nº de Iterações Entre Colisões" (eixo vertical: nº de iterações entre colisões, eixo horizontal: nº de iterações).

Experimento com AmbContornoCorredor1

Os experimentos vão até 15000 iterações e são interrompidos automaticamente.

É utilizado **ITAG** = 2000, assim, se não ocorrer **NBAT** = 100, o AG será acionado pelo menos 7 vezes.

Como o ambiente é mais complexo, exigindo manobras mais rápidas, o valor de **Multdifv** foi definido como 1.5

Quanto a **nugan,** ele foi aumentado pois, no caso em que o robô consegue fazer um percurso completo de ida e volta, ocorrerão em torno de 500 iterações, assim o valor de 600 torna-se razoável.

Estes experimentos foram executados utilizando-se várias sementes para o método gerador dos valores aleatórios.

Na tabela abaixo, o nº de ganhadores corresponde ao número de classificadores diferentes que ganharam nas últimas **nugan** iterações. As ações diferentes são aquelas determinadas pelo conseqüente desses classificadores. A existência de genéricos refere-se também aos últimos **nugan** ganhadores.

Semente	Nº total de Colisões	Nº de ganhadores Nugan = 600	Ações diferentes	Existem genérico nos ganhadores	Existência de Clones	Classificadores com força menor que 1	Característica dominante do movimento do robô
0	64	6	4	2	Sim	33	Percorre todo o ambiente.
1	14	5	1	2	Sim	36	Órbita local.
2	200	?	?	?	Sim	27	Órbita local. colisões perto do final do experimento prejudicam obtenção de dados.
3	164	2	1	0	Sim	27	Órbita local.
4	75	7	4	2	Sim	30	Percorre extensões do ambiente.
5	113	7	5	1	Sim	31	Percorre todo o ambiente.
6	32	7	4	2	Sim	32	Percorre extensões do ambiente.

Tabela 6.7: Resultados do experimento usando SC no AmbContornoCorredor1

É mantida diversidade para os classificadores e há uma tendência maior de percorrer o ambiente. Não existe muita dependência das condições iniciais.

Normalmente são necessários vários classificadores atuando para atingir o objetivo de não colidir, mesmo quando existem classificadores genéricos. O número de classificadores é maior que o número de ações diferentes produzidas pelo sistema. Este comportamento é bem parecido com o apresentado nos experimentos anteriores, com Ambiente 0 e Ambiente 1.

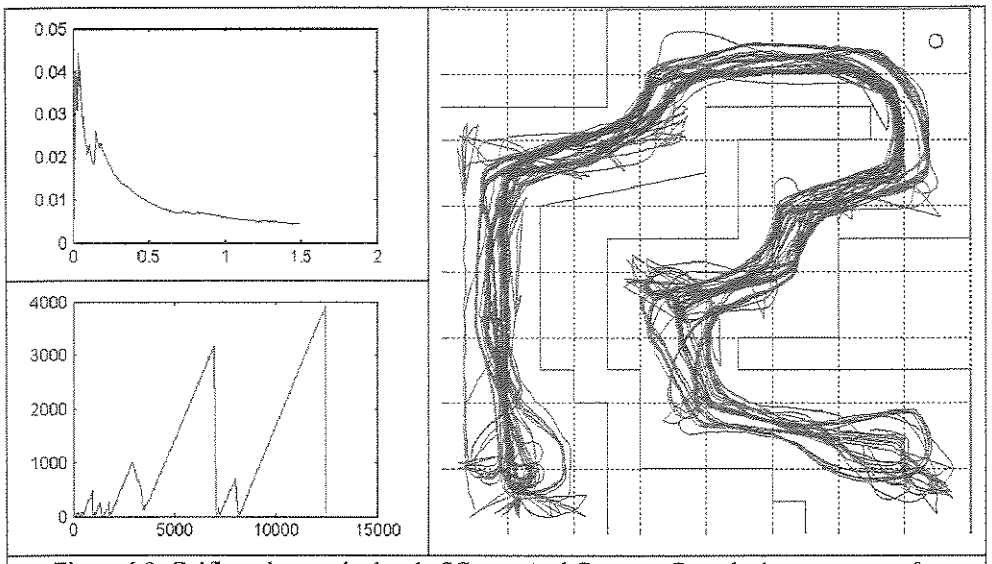


Figura 6.9: Gráficos dos resultados do SC para AmbContornoCorredor1 com semente 0 e 15000 iterações.

No sentido horário, a partir do canto superior esquerdo: "Curva de Aprendizado" (eixo vertical: nº de colisões/nº de iterações, eixo horizontal: nº de iterações), "Ambiente de Simulação" com o traçado do percurso do robô e "Nº de Iterações Entre Colisões" (eixo vertical: nº de iterações entre colisões, eixo horizontal: nº de iterações).

Experimentos com Conhecimento Inicial

Os experimentos apresentados anteriormente mostram o comportamento do robô exposto a uma situação de aprendizado, a partir da geração dos classificadores iniciais de forma aleatória. Foi visto o impacto dessas condições iniciais no seu desempenho e, de forma geral, pode-se considerar este resultado como muito bom, pois somente em poucas situações o modelo empregado não conseguiu atingir os resultados esperados.

Um aspecto bastante relevante em sistemas inteligentes, e que está presente no mecanismo dos sistemas classificadores, é relativo à possibilidade do robô aprender e, de preferência, a partir de nenhum conhecimento inicial agregado de forma explícita, pois isso reduz drasticamente o custo de projeto. Este aspecto ficou evidente no comportamento visto até aqui.

Ainda, associado com o aprendizado, o aspecto mais relevante é relativo à possibilidade de lidar com ambientes não estacionários. A abordagem apresentada a seguir trata disso, mas de uma forma peculiar. Ao invés de expor diretamente o robô a um ambiente não estacionário logo no início, isto será feito em duas etapas, considerando que, se o robô adquirir conhecimento básico de como se comportar no ambiente estático, ele terá melhores condições de aprendizado no ambiente dinâmico.

Assim, será utilizado o aprendizado obtido em um ambiente como conhecimento inicial para outro ambiente. É claro que o ideal seria introduzir modificações dinâmicas no ambiente de aprendizado após uma fase em que o conhecimento adquirido se tornasse satisfatório. Isto implicaria em fazer simulações usando modelos dinâmicos, e são motivações para a seqüência desta pesquisa.

Como exemplo, utilizou-se o conhecimento inicial adquirido em um ambiente, o **AmbContornoCorredor1**, e verificou-se o que ocorre ao usar-se tal conhecimento como conhecimento inicial nos **Ambiente0** e **Ambiente1**.

O *programa* **CSControladorMC** utiliza uma matriz de classificadores existente no **workspace** ao invés de gerá-la aleatoriamente, como é feito no **CSControlador**. Será utilizada a matriz **MC** do experimento com **AmbContornoCorredor1** realizada com semente 5.

Apesar de utilizar conhecimento prévio, os experimentos foram executados utilizando várias sementes. Isto se justifica pelo fato do algoritmo de aprendizado continuar em operação e necessitar em várias partes de seus procedimentos, como foi mostrado no capítulo 4, de geração de valores aleatórios.

Experimentos com Conhecimento Inicial para o Ambiente 0

Este experimento utiliza os mesmos parâmetros do experimento sem conhecimento inicial, inclusive é executado também durante 1200 iterações. A Tab. 6.8 apresenta um resumo comparativo dos dois experimentos, utilizando várias sementes.

Em 5 dos 7 experimentos é nítida a melhora no comportamento valendo-se do conhecimento inicial adquirido em outro ambiente com semelhanças, as quais permitiram o seu aproveitamento. Em 4 deles o aproveitamento foi total. Em 1 deles, que inclusive não havia conseguido aprendizado, o resultado pode ser considerado muito bom, com apenas 7 colisões.

Semente	Nº total de Colisões	Nº total de Colisões: com conhecimento inicial	Característica dominante do movimento do robô	Característica dominante do movimento do robô: com conhecimento inicial
0	25	0	Circunda o ambiente em sentido anti-horário.	Circunda o ambiente em sentido anti-horário.
1	3	0	Órbita horária local	Circunda o ambiente em sentido anti-horário.
2	12	0	Circunda o ambiente em sentido anti-horário	Circunda o ambiente em sentido anti-horário.
3	67	7	Não atinge o objetivo.	Circunda o ambiente em sentido anti-horário.
4	10	8	Circunda o ambiente em sentido horário	Circunda o ambiente em sentido anti-horário.
5	34	0	Circunda o ambiente em sentido horário	Circunda o ambiente em sentido anti-horário.
6	11	12	Circunda o ambiente em sentido horário	Circunda o ambiente em sentido anti-horário.

Tabela 6.8: Resultados do experimento usando SC no Ambinte0 com conhecimento inicial.

Em dois deles, com semente 4 e semente 6, não há melhora nem piora significativas. Estes resultados mostram a grande vantagem de se lançar mão de conhecimento inicial.

Experimentos com Conhecimento Inicial para o Ambiente 1

Este experimento utiliza os mesmos parâmetros do experimento sem conhecimento inicial, inclusive é executado também durante 5000 iterações. No entanto o valor de **Multdifv** foi definido como 1.5, o valor utilizado pelo experimento doador do conhecimento, e não 1.3 como usado neste experimento anteriormente. O quadro abaixo apresenta um resumo comparativo dos dois experimentos, utilizando várias sementes.

Semente	N° total de Colisões	N° total de Colisões: com conhecimento inicial	Característica dominante do movimento do robô	Característica dominante do movimento do robô: com conhecimento inicial
0	40	0	Circunda o ambiente em sentido horário.	Circunda o ambiente em sentido anti-horário.
1	21	10	Circunda o ambiente em sentido horário.	Circunda o ambiente em sentido anti-horário.
2	82	0	Circunda o ambiente em sentido horário.	Circunda o ambiente em sentido anti-horário.
3	22	50	Circunda o ambiente em sentido horário.	Circunda o ambiente em sentido anti-horário.
4	23	0	Circunda o ambiente em sentido horário.	Circunda o ambiente em sentido anti-horário.

Tabela 6.9: Resultados do experimento usando SC no Ambinte1 com conhecimento inicial.

Em 4 dos 5 experimentos é nítida a melhora no comportamento valendo-se do conhecimento inicial adquirido em outro ambiente com semelhanças que permitiram o seu aproveitamento. Em 1 deles, com semente 3, o aproveitamento foi pior. Estes resultados mostram, novamente, a grande vantagem de se lançar mão de conhecimento inicial.

Capítulo 7: Conclusões e Perspectivas

Os sistemas classificadores com redes neurais apresentados neste trabalho não têm similar encontrado na literatura, sendo que a sua investigação permite a apresentação de algumas implicações e sugestões sobre seu uso que podem nortear pesquisas futuras.

A aplicação de sistemas classificadores convencionais, baseada em um algoritmo clássico, evidencia a potencialidade dos sistemas classificadores para o aprendizado em tarefas de robótica autônoma móvel. Isto significa que nos trabalhos futuros esse modelo poderá ser aprimorado sem muitas dificuldades e receber as implementações mais detalhadas das características reais do *Khepera* e dos módulos para comandá-lo através do próprio simulador.

O simulador elaborado mostrou-se inteiramente apropriado para o desenvolvimento de técnicas de sistemas inteligentes para robótica autônoma móvel com aprendizado.

Os resultados obtidos nos experimentos confirmam aspectos do desenvolvimento teórico dos modelos de sistemas inteligentes utilizados e evidenciam pontos que requerem melhor detalhamento ou uso de abordagens mais apropriadas

É importante evidenciar que na etapa de desenvolvimento existe uma infinidade de definições, estudos, testes, decisões que devem ser tomadas. Por mais que se queira manter um registro detalhado desse processo isto se torna impraticável. Fica também impraticável pela exigüidade de tempo e de espaço para a sua exposição, dentro do escopo deste trabalho. Assim, vale a pena lembrar que na apresentação do simulador e das duas aplicações de sistemas inteligentes nos Capítulos 4 e 5 estão embutidas também idéias a partir de muitos resultados referentes a testes efetuados além dos explicitamente apresentados neste trabalho. Por exemplo, a opção de se utilizar o procedimento de fazer o robô recuar **NrItAVoltar** e penalizar **NrGanAPunir** após uma colisão, e a maneira como isso é feito, foi derivada a partir da observação detalhada do comportamento proporcionado pelo simulador e, depois de implantada, foi verificada a sua pertinência, também utilizando o simulador, o que permitiu comparar resultados.

Os comentários apresentados a seguir evidenciam diferenças essenciais no comportamento dos dois modelos de sistemas inteligentes, mostrando porque o modelo com redes neurais não pode ser simplesmente entendido como uma variante, ou um substitutivo comum para um sistema classificador, mas sim como um sistema que tem uma natureza distinta e ampliada frente ao sistema classificador convencional, agregando novas possibilidades que exigem adoção de critérios diferentes para a modelagem, aprofundamento do conhecimento, e criatividade no seu emprego. Mostram também que, quanto ao sistema classificador convencional, as expectativas se confirmam, principalmente com as modificações que foram feitas e apresentadas nos capítulos pertinentes.

7.1 Conclusões

7.1.1 O Uso de Redes Neurais Como Classificadores

A introdução das redes neurais como classificadores modifica o comportamento em direções que precisam ser entendidas para poder explorar o seu potencial e/ ou mesmo para poder incrementar o seu desempenho. Estas diferenças serão ressaltadas nos próximos itens, com comparações relativas aos resultados obtidos pelo modelo convencional.

7.1.2 Matching

Nos sistemas classificadores convencionais o *matching*, diante de uma mensagem dos sensores, ocorre ou não. Com o emprego das redes neurais, é diferente. O *matching* é representado por um valor real na saída da rede de avaliação do classificador. Assim, para verificar se houve ou não *matching*, é necessário especificar e avaliar faixas de valores.

Inicialmente foi considerado que todos os classificadores iriam dar *matching* para qualquer das mensagens, e que a saída da rede de avaliação definiria a especificidade. Esta medida permite que classificadores fracos, ou com baixas apostas, vençam se as suas especificidades forem negativas e suas forças tiverem se tornado negativas, portanto gerando apostas positivas.

Assim, em princípio ficou estabelecido que somente valores positivos podem determinar o *matching*, e que este mesmo número é utilizado para a especificidade. De qualquer, forma este assunto necessita de mais investigação, por isso o simulador apresenta a possibilidade de utilização de dois neurônios para a saída da rede de avaliação e, neste caso, um deles é utilizado para o *matching* e o outro para a especificidade.

7.1.3 *Reinforcement Learning*

No entanto, a simples utilização dos dois neurônios para a rede de avaliação não necessariamente apresenta melhoras no desempenho, como ficou evidenciado em alguns testes realizados. Uma direção que se apresenta como apropriada para os próximos passos nas pesquisas deste modelo é referente à utilização de técnicas de *Reinforcement Learning*. Estas técnicas utilizam a realimentação do ambiente para basear as alterações nos genes dos cromossomos do classificador, sendo um dos exemplos já consagrados o referente ao trabalho "Interactions Between Learning and Evolution", de Ackley e Littman [**1**]. Existe uma grande variedade de procedimentos para *Reinforcement Learning*, como pode ser visto em Sathiya Keerthi, B. Ravindran "A Tutorial Survey of Reinforcement Learning" de Sathiya Keerthi, B. Ravindran [**38**] e " An Introduction to Neural Networks" de Krose e Smagt [**22**]. Deve ser acrescentado que utilizar estas técnicas com as redes neurais como classificadores tem uma dificuldade adicional, ou seja, cada um dos classificadores terá que ser otimizado, não se tratando portanto de otimizar uma única rede, mas sim um conjunto delas.

7.1.4 Especificidade

A noção de especificidade é entendida como uma característica intrínseca do classificador. Com redes neurais este conceito necessita ser modificado, pois neste caso a especificidade do classificador é dependente também da entrada de sua rede de avaliação. Isto permite que um mesmo classificador possa ter especificidades diferentes, ou seja, a especificidade deixa de ser uma característica do classificador, representando um conceito generalizado, provavelmente significando que um classificador será mais específico para um padrão de mensagens e menos específico para outro, expressando um comportamento mais complexo.

7.1.5 Conseqüente

Um classificador pode atender diferentes mensagens, dependendo de sua especificidade. O mesmo ocorre com redes neurais, mas de uma maneira muito mais sofisticada. Agora, olhando para a ação que um classificador irá produzir, para um classificador convencional ela será sempre a mesma para todas as mensagens que ele classifique, porque a sua ação é ditada pelo seu conseqüente. No entanto, com redes neurais, ela é ditada pela saída da rede de ação, que tem entradas diferentes para mensagens diferentes, podendo, assim, produzir saídas diferentes.

7.1.6 Número de Classificadores para Resolver um Ambiente Estacionário

Considerando classificadores convencionais e supondo a existência apenas de classificadores específicos para resolver um ambiente estacionário, haverá a necessidade de um classificador para cada comando que se faça necessário para os atuadores. Mas, poderá existir um classificador genérico produzindo a mesma ação de um classificador específico, e, assim, o genérico e o específico irão competir, podendo ora um ora outro vencer sem que nenhum deles seja descartado. Portanto, fica claro que, em um sistema convencional, o número de classificadores para resolver um ambiente estacionário corresponde, no mínimo, ao número de situações diferentes para o comando dos atuadores.

Um caso que pode ser resolvido através de um único classificador convencional pode ocorrer em um movimento circular em uma região sem simetria. Neste movimento os sensores irão produzir diferentes leituras que deverão dar *matching* com um classificador genérico com força suficiente para compensar a sua baixa especificidade. Este classificador irá produzir sempre a mesma saída (por exemplo: girar 10 graus à direita) de acordo com o seu conseqüente.

Com o sistema classificador com redes neurais é teoricamente possível, e foi verificado nos experimentos, que apenas um classificador pode resolver movimentos mais complexos do que o circular, requerendo saídas diferentes da rede de ação. De certa forma, esta possibilidade aponta na direção de uma característica promissora, em acordo com as características de aproximadoras universais das redes neurais, de oferecer uma solução melhor. Mas deve-se considerar que, neste caso, existirá apenas um vencedor, diminuindo o significado do processo de atribuição de créditos assim que as redes atinjam o treinamento considerado suficiente. Provavelmente a solução é melhor com mais classificadores participando, mesmo que o seu número permaneça menor do que o requerido pelo sistema convencional, principalmente em ambientes não estacionários, onde manter uma diversidade de classificadores com força significativa pode ser determinante na geração de novos comportamentos, tanto pelo processo de atribuição de créditos como pelo algoritmo genético.

7.1.7 Número de Classificadores Versus Número de Ações para Resolver Um Ambiente

Um outro aspecto interessante no comportamento dos dois modelos de sistemas inteligentes utilizados é com relação ao número de classificadores necessários para resolver um ambiente versus o número de ações diferentes requeridas. O comportamento dos dois sistemas é bastante distinto.

Para os sistemas convencionais normalmente o número de classificadores é maior que o número de ações necessárias, uma vez que situações diferentes para os sensores podem ser mapeadas para uma mesma ação produzindo classificadores diferentes com mesmas saídas. Por exemplo, se houver apenas um obstáculo e à esquerda, independentemente de sua distância, uma solução possível seria girar um valor fixo de graus à direita, e que poderia ser obtida por vários classificadores com antecedentes diferentes e conseqüentes iguais.

Para os sistemas com redes neurais, pode ocorrer com freqüência a situação inversa, isto é, o número de ações necessárias é maior que o número de classificadores, conforme ficou evidenciado no item anterior.

7.1.8 Conhecimento Inicial

Não há dúvida de que robôs autônomos utilizando técnicas de aprendizado demandam menos esforço de projeto quando comparados com sistemas especialistas, onde todas as regras devem ser

definidas a priori, normalmente por um especialista. Mas não deixa de ser uma boa idéia, e que pode trazer bons resultados, a introdução de conhecimento disponível já na inicialização, e também durante o treinamento, principalmente quando isso não for de grande dificuldade.

Nos sistemas convencionais, é relativamente fácil fazer tal inserção de conhecimento porque o mapeamento dos dados dos sensores para o antecedente + conseqüente é direto.

7.1.9 Inserção de Conhecimento Durante o Treinamento

Os mesmos comentários acima podem ser feitos quanto à introdução de um novo classificador quando não ocorre *matching*, que foi utilizada de uma forma bem facilitada e com bons resultados no sistema classificador convencional.

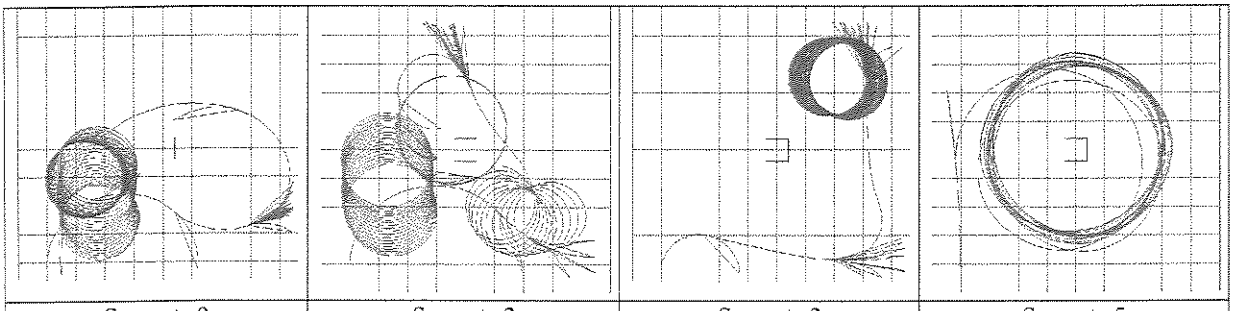
A introdução deste procedimento no sistema com redes neurais foi feita de forma ainda incompleta em vista das dificuldades mencionadas. Assim o antecedente e o conseqüente do novo classificador são gerados de forma aleatória, procurando apenas garantir a ocorrência de *matching* para a mensagem.

7.1.10 Dependência de Condições Iniciais

Os dois sistemas se mostraram dependentes de condições iniciais, mas o convencional menos.

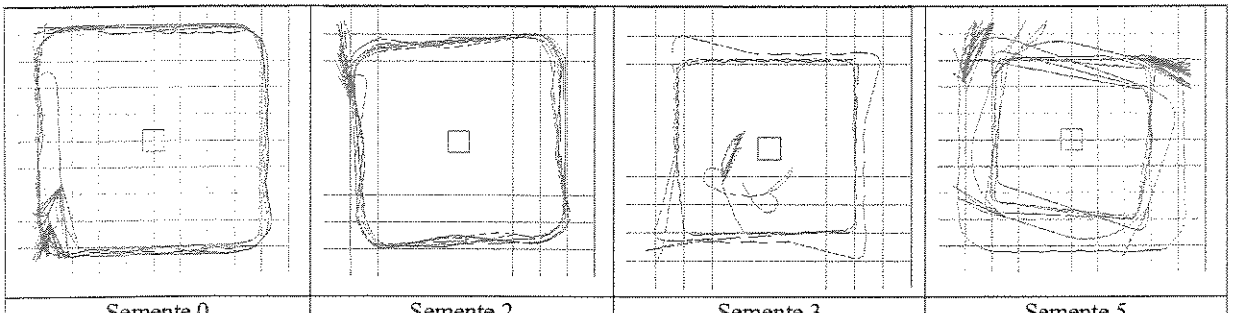
7.1.11 Suavização da Trajetória Utilizando Redes Neurais

Um fato notável é que em grande parte dos experimentos com SCRN observou-se suavização da trajetória do robô, o que, nestes casos, evidencia a eficácia da ação das redes neurais para aproximações suaves de funções. As Fig. 7.1 a 7.4 mostram este fato, comparando as trajetórias do SCRN com as do SC para vários experimentos nos Ambiente0 e Ambiente1.



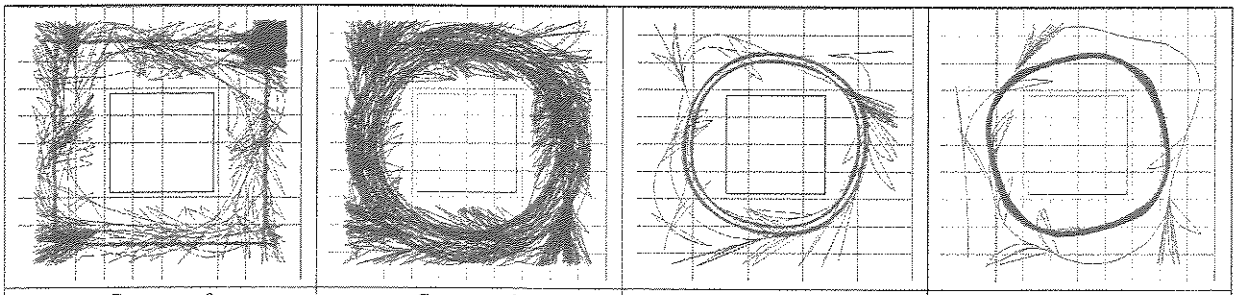
Semente 0	Semente 2	Semente 3	Semente 5

Figura 7.1: Traçados típicos de trajetória para SCRN no Ambiente0



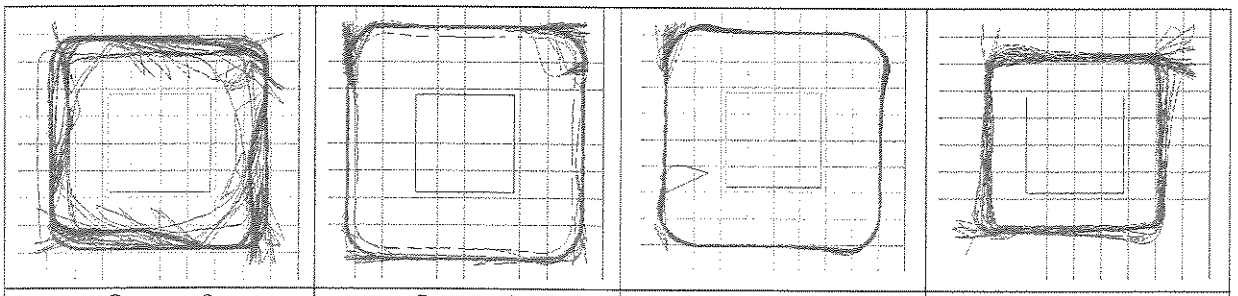
Semente 0	Semente 2	Semente 3	Semente 5

Figura 7.2: Traçados típicos de trajetória para SC no Ambiente0



Semente 0	Semente 1	Semente 2	Semente 5

Figura 7.3: Traçados típicos de trajetória para SCRN no Ambiente1



Semente 0	Semente 2	Semente 3	Semente 4

Figura 7.4: Traçados típicos de trajetória para SC no Ambiente1

7.2 Conjecturas

A partir dos resultados deste trabalho fica claro que valerá a pena fazer os experimentos com o robô real, o *Khepera*. Isto exigirá três atividades: 1) Introduzir o modelo dinâmico do *Khepera* no simulador; 2) Introduzir os modelos dos seus sensores; 3) Desenvolver os módulos de comando que irão utilizar o próprio simulador.

Existem duas possibilidades distintas para os experimentos com o *Khepera*. Desenvolver os experimentos diretamente com ele, ou desenvolvê-los em simulação e, depois do aprendizado, passar o comando do simulador ao *Khepera*.

O novo simulador com estas características poderá ser utilizado para se determinar diferenças em treinamento virtual e treinamento real, inclusive com medidas objetivas.

No SC existe um mapeamento das entradas em um pequeno conjunto de saídas. Isto não ocorre no SCRN, cuja saída é um número real. Isto origina a possibilidade de grandes alterações na orientação do robô. Seria interessante colocar um limite nestes valores? Quais as suas implicações?

Nota-se uma tendência espontânea no SCRN implementado neste trabalho a convergir para poucos ou praticamente um único classificador. Este resultado está em desacordo com a abordagem utilizada, em que a solução é a população de classificadores (entendida na literatura como abordagem *Michigan* [25]). É interessante investigar o que ocorre, mantendo a arquitetura do classificador neural utilizado e elaborando um algoritmo de atribuição de créditos que tenha como finalidade obter um único classificador (abordagem *Pittsburgh* [25]). Desta forma, o SCRN estaria se comportando como um sistema para treinar uma rede neural que seria a solução. Esta técnica teria a vantagem, comparada com outras técnicas de evoluir redes neurais para controle de robôs autônomos, de ser executada em tempo real.

Existe também a possibilidade de se caminhar na direção de melhor adequar o modelo do SCRN para a abordagem *Michigan*. Lembrando, como já mencionado no Capítulo 3, que as redes neurais introduzem mais sofisticação à solução, e talvez o modelo do algoritmo de atribuição de créditos não possa comportá-la. Uma das medidas então seria, mantendo o algoritmo de atribuição de créditos, simplificar a arquitetura da rede neural utilizada no classificador, quem sabe usando apenas um neurônio, o que poderia trazer o grau de sofisticação a um nível aceitável. A outra medida seria trabalhar no algoritmo de atribuição de créditos, procurando adequá-lo, ou, talvez melhor, elaborar um novo voltado às características novas introduzidas pelas redes neurais.

Voltando às idéias quanto a implantar a abordagem *Pittsburgh* apresentada acima, se ela der certo, lembrando da semelhança funcional do SCRN com as técnicas de *reinforcement learning*, pode-se verificar a viabilidade de utilizar uma abordagem semelhante para evoluir a rede de avaliação utilizada nestes algoritmos. No caso de *reinforcement learning* para redes neurais elas podem corresponder ao *critic*. Talvez se torne necessário, ou vantajoso, modificar a arquitetura do classificador neural para correspondê-lo melhor ao *critic*. A vantagem é a utilização de um método evolutivo, em tempo real.

Uma outra possibilidade se refere a utilizar rede neural apenas para o conseqüente, mantendo o antecedente da mesma forma utilizada no modelo convencional. Assim a especificidade e o *matching* funcionariam da mesma forma que no sistema convencional e poderia ser verificado o que ocorre com a introdução de rede neural no conseqüente.

Com relação ao algoritmo genético, seria vantajoso utilizar um modelo mais elaborado? Ainda, seria vantajosa a utilização de operadores genéticos específicos?

Este trabalho não considera um mecanismo para interromper ou modificar a atuação dos algoritmos de atribuição de créditos e genético após aprendizagem, o que resulta na ocorrência de colisões mesmo após a essa fase, como conseqüência da competição e da geração de novas regras. O modelo seguido foi o apresentado por Richards [33], o qual não considera esse efeito. No entanto, é fundamental adotar-se algum critério para a atuação dos algoritmos após a fase de aprendizagem com o objetivo de minimizar este efeito.

7.3 Limitações Deste Trabalho

Este trabalho teve o intuito de estudar diferentes abordagens de sistemas classificadores como solução ao problema da navegação autônoma investigando as implicações da introdução do classificador neural. Assim, não foram consideradas outras abordagens ao mesmo problema, como as redes neurais convencionais, sistemas baseados em regras *fuzzy*, outros sistemas baseados em computação evolutiva, além de diversas metodologias que podem ser aplicadas à resolução deste problema, que poderiam ser utilizadas como abordagens de referência ao estudo aqui efetuado.

Não que não se quisesse fazê-lo, entretanto, deve ser lembrado que este é um trabalho sob o escopo de uma dissertação de mestrado e existem limitações de tempo e recursos dedicados a este tipo de estudo que impõem restrições à sua maior abrangência.

Pela mesma razão, nesta pesquisa foram efetuadas somente simulações. Apesar do autor ter ciência de que o uso de simulações é de grande valia nesta linha de pesquisa, como argumentado no capítulo 4, gostaria também de aplicar os resultados simulados em robôs reais. Assim, na seqüência deste trabalho, com certeza deverão surgir estudos experimentais com robôs reais utilizando o simulador e as aplicações aqui desenvolvidas.

Mesmo considerando que poderiam ser encontradas outras limitações que não foram aqui apontadas, elas não invalidariam os resultados aqui obtidos. Esta dissertação teve o mérito de evidenciar o potencial dos sistemas classificadores aplicados ao problema da robótica móvel. Além disso, com a proposição dos sistemas classificadores com redes neurais, provavelmente foi aberta uma nova frente de pesquisa, com muitas oportunidades de trabalho para o futuro.

7.4 Perspectivas de Trabalhos Futuros

Como já foi mencionado neste trabalho, ele se constitui da parte inicial e fundamental de um longo trabalho a ser desenvolvido no futuro. Particularmente, o autor tem a perspectiva de dar continuidade a este estudo em um futuro Doutorado, onde vislumbra uma série de atividades que visam complementar e adicionar ao trabalho ora efetuado.

Dentre outras atividades, espera-se:

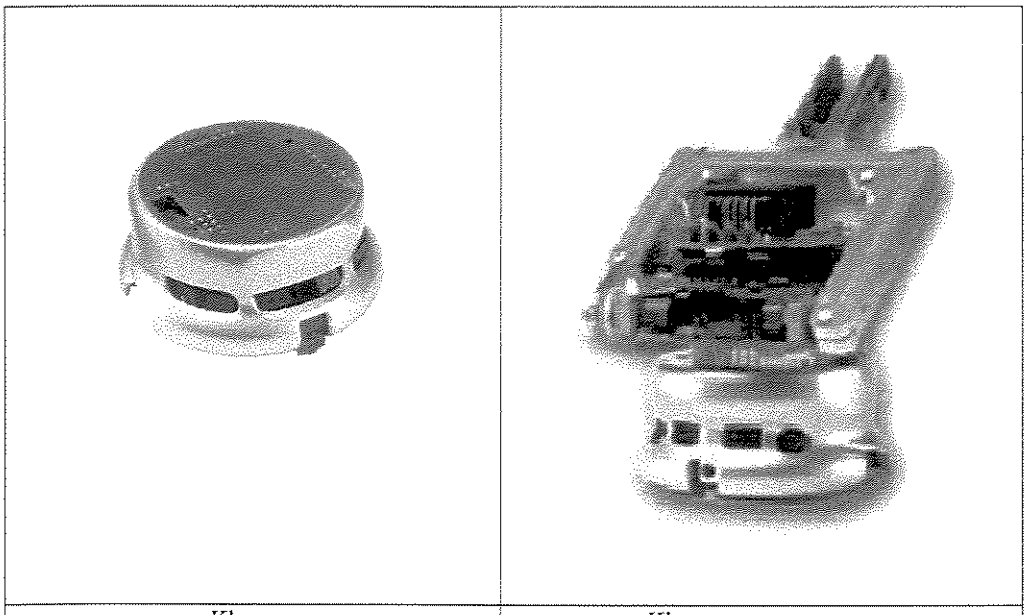
- efetuar uma comparação das abordagens aqui tratadas com outras abordagens encontradas na literatura (tais como redes neurais, sistemas de lógica *fuzzy*, computação evolutiva, etc.)
- promover uma modelagem mais realista do mini robô *Khepera*, que permita em um segundo passo a utilização do controlador aqui desenvolvido diretamente no robô real.
- aprimorar as técnicas de inteligência artificial para robótica móvel com aprendizado aqui apresentadas
- desenvolver os módulos para o comando do robô diretamente no simulador, de tal forma que possam ser utilizados indistintamente no simulador ou no robô real.

Além disso, espera-se que a proposição do classificador neural possa servir como inspiração para que o mesmo possa ser aplicado em outras áreas de atuação, onde suas qualidades possam ser também interessantes. Assim, outras áreas referentes a controle inteligente poderão se beneficiar deste novo paradigma aqui criado.

Por fim, enquanto os robôs inteligentes de Isaac Asimov e Arthur Clarke ainda não chegam, o autor fica com a sensação de ter andado um pequeno passo que seja na direção de ter contribuído para que um dia, no futuro, eles possam se tornar uma realidade.

Apêndice A: Khepera

O Khepera é produzido pela empresa K-Team, fundado em 1995 por pesquisadores e engenheiros do *Swiss Federal Institute of Technology of Lausanne*. É um mini robô largamente utilizado no meio científico devido à sua precisão e ao seu tamanho, podendo inclusive ser utilizado em cima de uma mesa de escritório.



Khepera	Khepera com sua garra

Figura A.1: Fotos do Khepera na sua configuração básica e com um de seus acessórios - a sua garra.

As suas especificações gerais atualizadas são apresentadas na Tabela A.1.

KHEPERA II SPECIFICATIONS	
Elements	**Technical Information**
Processor	Motorola 68331, 25MHz
RAM	512 Kbytes
Flash	512 Kbytes Programmable via serial port
Motion	2 DC brushed servo motors with incremental encoders (roughly 12 pulses per mm of robot motion)

Speed	Max: 60 cm/s, Min: 2 cm/s
Sensors	8 Infra-red proximity and ambient light sensors with up to 100mm range AND Power Consumption
I/O	3 Analog Inputs (0-4.3V, 8bit)
Power	Power Adapter OR Rechargeable NiMH Batteries
Autonomy	1 hour, moving continuously. Additional turrets will reduce battery life.
Communication	Standard Serial Port, up to 115kbps
Extension Bus	Expansion modules can be added to the robot using the K-Extension bus.
Size	Diameter: 70 mm Height: 30 mm
Weight	Approx 80 g
Payload	Approx 250 g
Simulators	• WEBOTS, Realistic 3D Simulator (Windows & Linux). • Freeware.
Development Environment for Autonomous Application	• KTProject, graphical interface for GNU C Cross-Compiler (Windows). • GNU C Cross-Compiler, for native on-board applications (Windows, Linux & Sun). • Freeware.
Remote control Software via tether or radio	• SysQuake® (on PC or MAC) using RS232. • LabVIEW® (on PC, MAC or SUN) using RS232. • MATLAB® (on PC, MAC, Linux or SUN) using RS232. • SysQuake® (on PC, MAC, Linux or SUN) using RS232. • Freeware. • Any other software capable of RS232 communication

Tabela A.1: Especificações gerais do Khepera

Apêndice B: Algoritmo Genético

Neste apêndice são apresentados detalhes da implementação do Algoritmo Genético.

B.1 Roulette Wheel

B.1.1 Intervalos para Roulette Wheel

Para utilizar o algoritmo Roulette Wheel, é montada uma matriz denominada **Avaliacao**, com número de linhas igual ao tamanho da população de classificadores que participará da seleção para a formação dos pares, e com 4 colunas. O ***fitness*** irá corresponder à força do classificador. As colunas têm os seguintes significados:

Índice: o índice da coluna é o mesmo utilizado na matriz que contem os classificadores em suas linhas.

Coluna 1: Força do classificador

Coluna 2: Participação percentual da força de cada classificador na soma das forças de todos os classificadores.

Coluna 3: A primeira linha vale zero. As linhas seguintes são obtidas somando-se a linha anterior da Coluna 2 com a Coluna 3.

Coluna 4: As linhas dessa coluna são preenchidas com o valor da linha seguinte da Coluna 3. A última linha vale 1.

Desta forma as Colunas 3 e 4 representam os limites inferiores e superiores da participação de cada classificador na Roulette Wheel. Veja a tabela abaixo com valores de exemplo para 4 indivíduos:

	1ª	2ª	3ª	4ª
1ª	1	.1	0	.1
2ª	4	.4	.1	.5
3ª	2	.2	.5	.7
4ª	3	.3	.7	1

Caso seja permitido valor negativo para a força, o módulo da força mínima será somado ao valor de todas as outras. Isto ocasiona um deslocamento de tal forma que a força mínima se transforme em zero. Para evitar que o pior indivíduo (aquele com força zero) seja eliminado pode-se adicionar um valor conveniente ao deslocamento.

B.1.2 Formação dos pares

A seleção de indivíduos usando *Roulette Wheel* é feita da seguinte forma: gera-se um valor aleatório, com distribuição uniforme entre 0 e 1, verifica-se o intervalo a que ele pertence, considerando as Coluna 3 e 4 de **Avaliacao**, identificando-se então a linha contendo o indivíduo a ser selecionado. Foi utilizada a seguinte expressão no programa, escrita em Matlab:

```
Pares(NrIndividuo,:) = Populacao(max(find(Avaliacao(:,3)<rand(1))),:)
```

A matriz **Populacao** contém os classificadores que irão participar da seleção para a formação dos pares para reprodução. A matriz **Pares** contém os pares de classificadores selecionados para aplicar

o operador de **crossover**. A expressão acima escolhe, conforme explicado a seguir, uma linha de **Populacao** para preencher a linha de índice **NrIndividuo** de **Pares**.

O método **find** monta um vetor com os números das linhas da terceira coluna de **Avaliacao** com conteúdo menor ou igual ao valor aleatório gerado por **rand**. A seguir o método **max** escolhe o valor máximo deste vetor, localizando-o dentro do intervalo especificado pelo *Roulette Wheel*, que corresponde então ao número da linha de **Populacao** que será copiada na linha **NrIndividuo** de **Pares**.

Nota-se que não será usada a 4ª. coluna de **Avaliacao**, a qual foi utilizada em uma abordagem anterior a esta. Ela foi mantida por facilitar a visualização dos limites e por não onerar o processamento.

Deve ser notado que o algoritmo utilizado neste método é o clássico, mas no processo dos sistemas classificadores empregado é utilizada uma sub população: um dado número de melhores indivíduos é selecionado para a evolução e a nova geração irá substituir um mesmo número de piores indivíduos. Assim ficam preservados os melhores indivíduos. Por isso no método é feita a tentativa de se evitar a formação de pares do mesmo indivíduo já que, com o *crossover* usado, resultariam cópias iguais aos pais e às já preservadas na população original. É claro que ainda existiria a possibilidade de mutação, que, no entanto, normalmente utilizará taxas muito pequenas.

B.2 Crossover

Utiliza-se o *crossover* simples. Monta-se a matriz **CrossOver**, inicialmente igual à matriz **Pares**, que passa a conter o resultado do *crossover*. Para cada par calcula-se aleatoriamente o ponto ou pontos de *crossover*. No SC é utilizado *crossover* de 1 ponto e para o SCRN *crossover* de 2 pontos.

B.3 Mutação

Monta-se a matriz **Mutação**, inicialmente igual a **Crossover**. Para SCRN, a taxa de mutação é testada para cada indivíduo. Se o indivíduo for selecionado para mutação, são escolhidos aleatoriamente dois pontos e é feita a permutação de seus genes. Para o SC a taxa de mutação é testada para cada gene do indivíduo e, se ele for selecionado, tem-se duas possibilidades. A primeira refere-se ao gene da **condição**, ele recebe aleatoriamente valor 1, 0 ou -1 ('tanto faz'). A segunda refere-se ao **conseqüente**, onde é invertido o valor do gene.

Apêndice C: Cálculos para DWS

Este apêndice apresenta os cálculos para se obter o controle de movimento por velocidades diferenciais nas duas rodas do robô.

C.1 Determinação das novas coordenadas

As expressões a serem apresentadas a seguir utilizam a convenção mostrada na Fig. C.1. Na figura, E0-D0 representa o eixo do robô na posição inicial e E1-D1 na posição final, após um intervalo de tempo dt em que as velocidades das duas rodas, embora sendo diferentes, permanecem inalteradas.

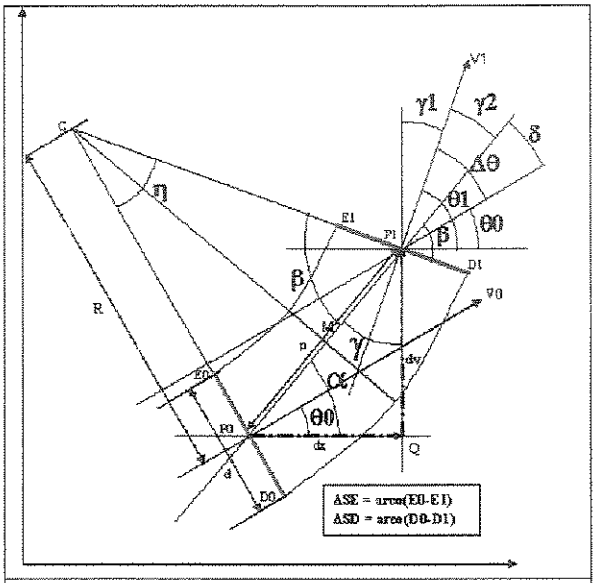


Figura C.1: Controle de orientação por DWS

C.1.1 Cálculo do raio de curvatura R

As velocidades diferenciais fazem com que as trajetórias descritas pelas rodas percorram arcos de circunferência com comprimentos diferentes.

No intervalo de tempo dt elas descreverão um arco de circunferência correspondente ao deslocamento angular η. Pode ser visto que η é igual ao deslocamento angular Δθ da direção do veículo , pois têm lados ortogonais.

$$\eta = \Delta\theta \qquad (C.1)$$

Sejam vd e ve os valores absolutos das velocidades das rodas direita e esquerda respectivamente, e ΔSD e ΔSE os arcos de circunferência descritos por elas.

$$\Delta SD = (R+d/2).\Delta\theta = vd.dt \qquad (C.2)$$

$$\Delta SE = (R- d/2).\Delta\theta = ve.dt \qquad (C.3)$$

Destas duas expressões deduz-se o valor de R

$$R = [(d/2).(vd+ve)]/(vd-ve) \qquad (C.4)$$

C.1.2 Cálculo do deslocamento angular Δθ

Da expressão (2) temos

$$\Delta\theta = (vd.dt)/(R+d/2) \qquad (C.5)$$

Substituindo na expressão acima o valor determinado para R em (4) e simplificando, obtemos o valor para Δθ

$$\Delta\theta = [(vd-ve).dt]/d \qquad (C.6)$$

C.1.3 Cálculo de gama

Este ângulo permitirá o cálculo de α no triângulo retângulo P0-Q-P1, permitindo calcular-se os deslocamentos na direção x e y.

O ângulo γ é determinado por

$$\gamma = \gamma 1+\gamma 2 \qquad (C.7)$$

Por sua vez γ1 é obtido por

$$\gamma 1 = (\Pi/2)-\beta \qquad (C.8)$$

E β é calculado através do triângulo isósceles C-P0-P1

$$\beta = (\Pi-\eta)/2 = (\Pi-\Delta\theta)/2 \qquad (C.9)$$

Usando (C.9) em (C.2) e simplificando obtemos

$$\gamma 1 = \Delta\theta/2 \qquad (C.10)$$

O ângulo γ2 é obtido por

$$\gamma 2 = (\Pi/2)-\theta 1 \qquad (C.11)$$

onde θ1 é a nova orientação do robô, que corresponde à orientação anterior mais o deslocamento angular Δθ, ou seja

$$\theta 1 = \theta 0+\Delta\theta \qquad (C.12)$$

que colocada em (11) resulta em

$$\gamma 2 = (\Pi/2)-\theta 0-\Delta\theta \qquad (C.13)$$

As expressões (C.10) e (C.13) permitem calcular o valor de γ em (C.7)

$$\gamma = \Delta\theta/2+(\Pi/2)-\theta 0-= \Pi/2-\theta 0-\Delta\theta/2 \qquad (C.14)$$

C.1.4 Cálculo de alfa

Finalmente, pode-se calcular α

$$\alpha = \Pi/2-\gamma = \Pi/2-(\Pi/2-\theta 0-\Delta\theta/2)$$

ou seja

$$\alpha = \theta 0+\Delta\theta/2 \qquad (C.15)$$

Agora é possível verificar uma propriedade interessante derivada do movimento, o valor de δ. Pela figura tem-se

$$\delta = \Delta\theta-\gamma 1 = \Delta\theta-\Delta\theta/2$$

ou seja

$$\delta = \Delta\theta/2 \qquad (C.16)$$

resultando em δ e γ1 serem iguais, o que mostra que o segmento de reta P0-P1, que une os pontos centrais do robô nas duas posições, antes e após dt, divide o deslocamento angular Δθ ao meio.

C.1.5 Cálculo dos deslocamentos dx e dy

Estes valores são obtidos através do triângulo retângulo P0-Q-P1

$$dx = p.\cos(\alpha) \qquad (C.17)$$

$$dy = p.\sin(\alpha) \qquad (C.18)$$

faltando apenas apenas conhecer a hipotenusa p, que é achada considerando-se o triângulo retângulo C-M-P0:

$$p = 2.R.\sin(\Delta\theta/2) \qquad (C.19)$$

Assim, a expressão para os deslocamentos fica:

$$dx = 2.R.\sin(\Delta\theta/2).\cos\alpha \qquad (C.20)$$

$$dy = 2.R.\sin(\Delta\theta/2).\sin\alpha \qquad (C.21)$$

Bibliografia

[1] Ackley, D., Littman, M. "Interactions Between Learning and Evolution", in Artificial Life II, SFI Studies in the Sciences of Complexity, vol. X, edited by C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen, Addison-Wesley, 1991.

[2] ARC – Autonomous Robot Controller, Xavi Màrquez Navàs, Barcelona, 2000. xmarquez@indai.com

[3] Arthur C. Clarke, "2001/Odisséia Espacial", edição de bolso, Editora Edibolso, 1975.

[4] Berlanga, A.; Isasi, P.; Sanchis, A.; Molina, J. M. "Neural networks robot controller trained with evolution strategies". Proceedings, page 419, vol. 1, Evolutionary computation, 1999.

[5] Berlanga, A.; Sanchis, A.; Isasi, P.; Molina, J. M. "A general learning co-evolution method to generalize autonomous navigation behavior". Proceedings, pages 769-776 vol. 1, Evolutionary Computation, 2000.

[6] Bianco, G.; Cassinis, R. "Multi-strategic approach for robot path planning". Proceedings, pages 108-115, Advanced Mobile Robot, 1996.

[7] Booker, L. B., Goldberg, D. E., Holland, J. H. "Classifier Systems and Genetic Algorithms" - Artificial Intelligence 40 pp. 235-282, 1989.

[8] Brooks, R. "*Intelligence Without Reason*", Proc. IJCAI, pág. 569-595, agosto, 1991.

[9] Cazangi, R. R., Figueiredo, M. "Sistema Autônomo Inteligente Baseado em Computação Evolutiva aplicado à Navegação de Robôs Móveis", Departamento de Informática, Universidade Estadual de Maringá – UEM, proceedings do V SBAI, Canela, dez/2001.

[10] Chohra, A.; Scholl, P.; Kobialka, H. U.; Hermes, J.; Bredenfeld, A. "Behavior learning to predict using neural networks (NN): towards cooperative and adversarial robot team (RoboCup)". Proceedings, pages 79-84, Robot Motion and Control, 2001

[11] Darwin2K – Simulation and Automated Synthesis for Robotics, CMU's Robotic Institute.

[12] "DEV-C++ Tutorial for CSC 161 Students", Bloodshed. www.geocities.com/uniqueness_templat

[13] Figueiredo, M. F "Redes Neurais Nebulosas Aplicadas em Problemas de Modelagem e Controle Autônomo", Tese de Doutorado, FEEC – UNICAMP, agosto, 1997.

[14] Goldberg, D. E. "Introduction to Genetics-Based Machine Learning" - Chapter 6 do livro "Genetic Algorithms in Search, Optimization and Machine Learning" - David. E. Goldberg - Addison-Wesley 1989

[15] Grefenstette, J.; Schultz, A "An Evolutionary Approach to Learning in Robots" (SAMUEL). Navy Center for Artificial Intelligence, Naval Research Laboratory, Washington, DC 20375.{gref, schultz}@aic.nrl.navy.mil

[16] Haykin, S "Neural Networks – A Comprehensive Foundation", 2nd. edition, Prentice-Hall, 1999.

[17] Holland, J. H. "Adaptation in Natural and Artificial Systems", University of Michigan Press, An Arbor, MI, 1975.

[18] Isaac Asimov, "Eu, Robô", Editora Expressão e Cultura, 8a. edição, julho, 1974.

[19] "Khepera Simulator 2.0" distribuído como *freeware* por Oliver Michel. http://diwww.epfl.ch/lami/team/michel/khep-sim/

[20] Kovacs, T., Lanzi, P. L. "A Learning Classifier Systems Bibliography", Technical Report: CSRP-99-19, University of Birmingham, United Kingdom, 1999.

[21] Khepera, produzido por K-TEAM, http://www.k-team.com/careers.html

[22] Krose, B., Smagt, P. van der chapter 7 "Reinforcment Learning" of " An Introduction to Neural Networks", eigth edition, November 1996, The Universtity of Amsterdam.

[23] Kubrick, S., produtor do filme "2001: A Space Odyssey".

[24] Lund, H. H.; Miglino, O. "From simulated to real robots". Proceedings, pages 362-365, Evolutionary Computation, 1996.

[25] Michalewicz, Z. "Genetic Algorithms + Data Structures = Evolution Programs", 3rd edition, Springer, 1996.

[26] MOBS – Mobile Robot Simulator, Horst Stolz sob a direção de Thomas Braunl, Univ. Stuttgart, Germany.

[27] Moussi, L. N., Gudwin, R. R., Von Zuben, F. J., Madrid, M. K. "Sistemas Classificadores com Redes Neurais (NNCS) : Aplicação ao Controle de um Veículo Autônomo Simulado Computacionalmente", proceedings do V SBAI, Canela, RS, dezembro, 2001.

[28] Moussi, L. N., Gudwin, R. R., Von Zuben, F. J., Madrid, M. K. "Neural networks in classifier systems (NNCS): An application to autonomous navigation" in V. V. Kluev & N. E. Mastorakis (eds.) Advances in Signal Processing, Robotics and Communications, Electrical and Computer Engineering Series, WSES Press, pp. 256-262, 2001.

[29] Moussi, L.N., Von Zuben, F. J., Gudwin, R. R., Madrid, M. K. "A Simulator using Classifier Systems with Neural Networks for Autonomous Robot Navigation". Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'2002), vol. 1, pp. 501-506, in the 2002 IEEE World Congress on Computational Intelligence (WCCI'2002), Honolulu, Hawaii, 12-17 maio, 2002.

[30] Nechyba, M. C.; Yangsheng Xu "Human control strategy: abstraction, verification, and replication" IEEE Control Systems Magazine, Volume: 17 Issue: 5, Pages: 46-61, 1997.

[31] Nolfi, S. Floreano, D. "Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines". Cambridge, MA: MIT Press/Bradford Books). Intelligent Robots and Autonomous Agents series edited by Ronald C. Arkin, novembro, 2000.

[32] Ramsey, C.L., Schultz, A.C., and Grefenstette, J.J. "Simulation-assisted learning by competition: Effects of noise differences between training model and target environment", Proceedings of the Seventh International Conference on Machine Learnig, Austin, TX, Morgan Kaufmann , pp. 211-215, 1990.

[33] Richards, R. A. "Classifier Systems & Genetic Algorithms" Robert A. Richards - Chapter 3 of Richards, Robert A.; Zeroth-order Shape Optimization Utilizing a Learning Classifier System, Ph.D. Dissertation, Mechanical Engineering Department, Stanford University, 1995.

[34] Robotica. "*Robotica Manual*", Robotica TM Software, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. ftp://ftp.csl.uiuc.edu/

[35] Schiller, I.; Draper, J. S. "Mission adaptable autonomous vehicles". Pages 143-150. Neural Networks for Ocean Engineering, 1991.

[36] Schultz, A.C., and Grefenstette, J.J. "Using a Genetic Algorithm to Learn Behaviors for Autonomous Vehicles", Navy Center for Applied Research in Artificial Intelligence, Navy Research Laboratory, Washington, DC, Proceedings of the AIAA Guidance, Navigation and Control Conference, Hilton Head, SC, August 10-12, 1992.

[37] Simderella. Patrick van der Smagt, "Simderella: a robot simulator for neuro-controller design", published in Neurocomputing, V. 6 N. 2, Elsevier Science Publishers, 1994. http://www.robotic.dlr.de

[38] S. Sathiya Keerthi, B. Ravindran "A Tutorial Survey of Reinforcement Learning", Department of Computer Science and Automation, Indian Institute of Science, Bangalore.

[39] Yung, N. H. C.; Ye, C. "EXPECTATIONS – an autonomous mobile vehicle simulator". Pages 2290-2295 vol. 3. Systems, Man, and Cybernetics, 1997.

[40] Wang, L. F.; Tan, K. C.; Prahlad, V. "Developing Khepera robot applications in a Webots environment". Proceedings, pages 71-76, Micromechatronics and Human Science, 2000.

[41] Webots Release 3.0.1 (for evaluation purpose only), from Cyberbotics Ltd., www.cyberbotics.com.

[42] Wyeth, G. "Neural mechanisms for training autonomous robots". Proceedings, pages 194-199, Mechatronics and Machine Vision Practice, 1997