UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ARQUITETURA NEURAL COGNITIVA PARA CONTROLE INTELIGENTE DE ROBÔS MÓVEIS EM LABIRINTOS DINÂMICOS

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica.

LUCIENE DE OLIVEIRA MARIN

Florianópolis, fevereiro de 2010.

ARQUITETURA NEURAL COGNITIVA PARA CONTROLE INTELIGENTE DE ROBÔS MÓVEIS EM LABIRINTOS DINÂMICOS

Luciene de Oliveira Marin

'Esta Tese foi julgada adequada para a obtenção do título de Doutor em Engenharia Elétrica, Área de Concentração em *Sistemas de Informação*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. Edson Roberto De Pieri, Dr.
Orientador

Prof. Mauro Roisenberg, Dr.
Co-orientador

Prof. Roberto de Souza Salgado, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Edson Roberto De Pieri, Dr., DAS-UFSC
Presidente

Prof. Mauro Roisenberg, Dr., INE-UFSC

Prof. Adelardo Adelino Dantas de Medeiros, Dr., DCA-UFRN

Prof. Paulo Martins Engel, Dr., INF-UFRGS

Prof. Marcelo Ricardo Stemmer, Dr., DAS-UFSC

"A natureza esconde seu segredo porque é sublime,
não por astúcia."
Albert Einstein

A meu pai Antônio Mauro, meu herói...
a minha mãe Dilma, meu amparo...
e aos meus sobrinhos José Luiz, Cássia e Rafael, meus encantos...

AGRADECIMENTOS

Primeiramente a Deus por todas as oportunidades, aprendizados e principalmente pelas pessoas que de alguma forma fizeram parte de minha história, durante esta trajetória de estudos e pesquisas.
A meus pais Dilma e Antônio Mauro pelo amor e apoio incondicionais. A meus irmãos Dulcilene, Estela, Wagner, ao cunhado Gilberto, a meus lindos sobrinhos José Luiz, Cássia e Rafael, a meus avós, toda a família e amigos, pela compreensão de minhas ausências e pela torcida sempre animada.
Ao professor e orientador Edson Roberto de Pieri, pela oportunidade a mim concedida de realizar este trabalho de tese, pelo exemplo de dedicação, pelas contribuições, paciência e pela amizade com que pude contar sempre.
Ao professor e co-orientador Mauro Roisenberg, meu grande incentivador e amigo desde os tempos do mestrado, me "iniciando" primeiro no mundo das redes neurais artificiais, e mais tarde no mundo da robótica inteligente. Pelas idéias inspiradoras e pelas inúmeras vezes que me encorajou mediante minhas fases "beco sem saída".
Aos meus amigos e irmãos do peito Alessandra Furtado e Alessandro Bovo, cujo apoio e amizade foram fundamentais nestes últimos meses de trabalho. Também aos amigos que levarei sempre em meu coração: Roberto, Denise e família, Alexandra, Netanias, Rafinha, Dona Maria, Seu Carlos, Thiago, Carol, Angela, Cristine, Diego, Léo, Neiva, Carol, Edu, Denise, Marilinha, Déia, Elisângela, Solange, Paola, Naiara e Thyago. Em especial ao meu amigo Gláucio Adriano (Xodó), a quem também devo meus primeiros contatos com a robótica e muitos momentos inesquecíveis de descontração e gargalhadas.
Ao colega André Chinvelski, pelo precioso apoio prestado no início das implementações no simulador java *WSU Khepera Robot*.
Aos secretários do programa de pós graduação em engenharia elétrica, Wilson Costa e Marcelo Siqueira, pela cordialidade, seriedade e presteza.
A todos os professores que participaram de minhas bancas de qualificação e defesa de tese, pelas importantes argumentações, sugestões e contribuições dadas. Em especial ao professor Guilherme Bittencourt (in memoriam), pela idéia do "xixi" de ratos. Ao professor Paulo Martins Engel, por aceitar o convite para realizar o parecer desta tese, e aos professores das disciplinas que cursei durante o primeiro ano de doutoramento.
À Capes e ao CNPq pelo apoio financeiro.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Doutor em Engenharia Elétrica.

ARQUITETURA NEURAL COGNITIVA PARA CONTROLE INTELIGENTE DE ROBÔS MÓVEIS EM LABIRINTOS DINÂMICOS

LUCIENE DE OLIVEIRA MARIN

Fevereiro/2010

Orientador: Edson Roberto De Pieri
Co-orientador: Mauro Roisenberg
Área de Concentração: Sistemas Computacionais
Palavras-chave: Robôs Móveis Autônomos, Navegação, Exploração, Aproveitamento, Aprendizado de Mapa, Redes Neurais Artificiais, Aprendizagem por Reforço
Número de Páginas: xx + 219

Este trabalho propõe uma arquitetura de controle inteligente para navegação de robôs móveis que foi resultante de uma série de construções prévias cuja evolução é descrita. A fim de prover maior flexibilidade e um menor custo computacional ao modelo de navegação proposto, foram estudadas técnicas de Inteligência Artificial, tais como, Redes Neurais Artificiais e Aprendizagem por Reforço, com o foco na questão do aprendizado em tempo de operação para ambos os níveis da arquitetura. A arquitetura final integra de maneira eficiente os níveis deliberativo e reativo, dispondo de um método de aprendizado adaptativo de mapa topológico no nível mais alto, suportado por um processo de aprendizado de mapeamento percepção ação no nível mais baixo. As implementações e resultados das simulações mostram o desenvolvimento progressivo do sistema de navegação proposto, capaz de prover a um robô móvel, com limitados recursos de sensores, memória e processamento, a habilidade de executar tarefas de navegação em labirintos do tipo T desconhecidos e modificáveis durante seu tempo de operação. Uma outra contribuição deste trabalho refere-se a uma nova descrição proposta para o dilema conhecido como "*exploration* versus *exploitation*", no contexto da integração entre os níveis reativo e deliberativo e como medida de desempenho.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

NEURAL COGNITIVE ARCHITECTURE FOR INTELLIGENT CONTROL OF MOBILE ROBOT IN DYNAMIC MAZES

LUCIENE DE OLIVEIRA MARIN

February/2010

Advisor: Edson Roberto De Pieri
Co-advisor: Mauro Roisenberg
Area of Concentration: Computational Systems
Key words: Autonomous Mobile Robots, Navigation, Exploration, Exploitation, Map Learning, Artificial Neural Networks, Reinforcement Learning
Number of Pages: xx + 219

This work proposes an intelligent control architecture for mobile robot navigation which was resulting from a series of previous constructions whose evolution is described. In order to provide a larger flexibility and a small computational cost to the proposed navigation model, Artificial Intelligence techniques, such as Artificial Neural Networks and Reinforcement Learning were studied, with the attention focus on the online learning issue to both levels of the architecture. The final architecture integrates in an efficient way the deliberative and reactive levels, disposing of an adaptive topological map learning method in the highest level, supported by a mapping perception action learning process in the lowest level. The implementations and simulation results show the progressive development of the proposed navigation system which is capable to provide to a mobile robot, with limited resources of sensoring, memory and processing, the ability to execute navigation tasks in unknown and changing T-mazes during its operation time. Another contribution of this work refers to a new description proposed to the known "exploration versus exploitation" dilemma, in the context of the integration between the reactive and deliberative levels and as a performance measure.

Sumário

Lista de Acrônimos

MLP-R	Rede multi-camadas de perceptrons com aprendizado por reforço
MLP-R-RR	Rede multi-camadas de perceptrons recorrente com aprendizado por reforço recorrente
ART1-MLPs-RR	Rede ART1 comutadora de redes MLPs com aprendizado por reforço recorrente
ART1-R-MLPs-RR	Rede ART1 recorrente comutadora de redes MLPs com aprendizado por reforço recorrente
ART1-R-MLPs-RR-Marcos	Rede ART1 recorrente comutadora de redes MLPs com aprendizado por reforço recorrente e baseada em marcos
Map-Tree	Mapeamento cognitivo com planejamento de caminho através de transposição para árvore binária
Map-Dijkstra	Mapeamento cognitivo com planejamento de caminho através do algoritmo de Djisktra
NeuroCog	Mapeamento cognitivo com exploração e planejamento de caminho determinado pelo ajuste do dilema *exploration* versus *exploitation*

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO 1

Introdução

1.1 Descrição

A área de conhecimento em Robótica agrega diferentes domínios científicos e tecnológicos e desde seu advento até os dias atuais, se mantém atrativa e desafiadora devido a esta característica multidisciplinar e à diversidade de aplicações. Robôs são dispositivos físicos que, através de sensores, "percebem" o mundo para interagir por meio de ações como manipulação e locomoção, sendo assim classificados em manipuladores e móveis. Os manipuladores são, em sua maioria, de uso industrial, e operam em áreas de trabalho restritas e dedicadas, por exemplo, a pintura, montagem, posicionamento de peças, etc. Já os robôs móveis podem mover-se pelo ambiente, realizar supervisão, inspeção, transporte, monitoramento, etc. Seu uso pode ser no meio industrial, agrícola, militar, doméstico, medicina, entretenimento, entre outros.

Na robótica móvel, há dois tipos de robôs: os veículos guiados automaticamente (AGVs - do acrônimo em inglês: *Automatic Guided Vehicle*) e os robôs móveis autônomos. Os primeiros operam em ambientes projetados (p.e., com caminhos induzidos[1], faróis de direção, ou outras marcas de balizamento) e executam tarefas de transporte ao longo de rotas fixas. Por esta razão se tornam inflexíveis e frágeis: alterar uma rota implica em aumento de custo e qualquer mudança inesperada (tais como objetos bloqueando o caminho) pode levar a falhas na execução da tarefa. Como alternativa a este tipo de robô, surgiram os robôs móveis autônomos (NEHMZOW, 2000).

Com o contínuo progresso tecnológico, uma ampla variedade de robôs móveis vêm sendo produzida em diferentes categorias de robôs tais como

[1] fios de direção enterrados debaixo do chão

terrestres, aéreos, aquáticos, além dos emergentes micro/nano robôs (SITTI, 2005). Apesar do aspecto hardware evoluir continuamente, resultando em diversos tipos de robôs móveis, o complexo problema da navegação autônoma está presente na maioria das aplicações e ainda está longe de ser esgotado. Consequentemente, diferentes áreas de conhecimento tais como ciências cognitivas, psicologia e filosofia também atraem a atenção dos pesquisadores em robótica que procuram por inspirações e idéias que possam contribuir na questão de como incorporar comportamentos inteligentes aos robôs móveis autônomos.

O problema do controle da navegação de um robô móvel autônomo pode ser dividido em duas partes principais. A primeira relacionada às dinâmicas e cinemáticas do sistema em questão, e a segunda à geração ou seguimento de trajetórias. Problemas de controle e cinemática de robôs estão fora do escopo de desenvolvimento deste trabalho de tese. Mais especificamente, esta tese está voltada à questão do controle em um nível mais alto de abstração, considerando-se questões de implementação de comportamentos mais complexos e orientados a tarefas, a um robô móvel com poucos recursos computacionais. No que tange a este foco, assume-se que os níveis mais básicos de controle são efetivamente fornecidos.

1.2 Breve Histórico

Enquanto nos anos 40 surgiam os incipientes braços mecânicos operados por humanos (SELIG, 1992), ao final da mesma década um neurofisiologista inspiraria mais tarde a geração de pesquisadores dos anos 80. Willian Grey Walter tornou-se o pioneiro em robótica móvel e vida artificial com a invenção de suas tartarugas mecânicas (LEBOUTHILLIER, 1999) que demonstravam determinado comportamento autônomo obtido através de simples dispositivos “neurais”.

Ainda nos anos 60 e 70, enquanto nas indústrias de montagem e automobilística, braços mecânicos operados por computador evoluíam para robôs manipuladores de objetos microscópicos, o surgimento da área de Inteligência Artificial (IA) incorporou à Robótica novas possibilidades de inovações, atraindo a atenção dos pesquisadores mais para os aspectos de software e raciocínio, do que para os aspectos de hardware (NEHMZOW, 2000). E deste modo surgiram em diversas universidades, os primeiros robôs móveis autônomos com habilidades mais complexas, tais como por exemplo a de manter um modelo de mundo global a partir de informação sensória, evitar obstáculos e objetos em movimentos em ambiente estruturado [robô Shakey (NILSSON, 1984)], exploração planetária [robô JLP Rover (NEHMZOW, 2000)], etc.

1.2.1 Arquiteturas de Controle para Robótica Inteligente

Como todo problema em robótica inteligente deve tratar o relacionamento entre as entidades ***sentir***, ***planejar*** e ***agir***, Murphy (2000) definiu três paradigmas de controle como base das arquiteturas de robôs móveis autônomos existentes na literatura: hierárquico (***sentir***-***planejar***- ***agir***), reativo (***sentir***-***agir***) e híbrido [***planejar***, (***sentir***-***agir***)]. O problema da navegação autônoma de robôs móveis primeiramente foi abordado segundo o paradigma hierárquico durante os anos de 1967 a 1990. Uma arquitetura de controle construída com base neste paradigma prioriza sobremaneira o planejamento, incorporando toda informação sensória em um modelo de mundo global, para em seguida executar ações. As desvantagens de uma arquitetura hierárquica são as dificuldades em construir modelos globais e genéricos de mundo e a sua vulnerabilidade quanto à falha de algum de seus módulos, implicando na falha do sistema como um todo.

Inspirado nos trabalhos de W. Grey Walter, Brooks (1986) foi quem propôs o inovador paradigma reativo através de sua arquitetura chamada de subsunção, a qual proporcionou um avanço promissor em robótica, sendo amplamente utilizada a partir de 1988. Arquiteturas de controle construídas sob este paradigma excluem a entidade ***planejar***, acoplando diretamente as primitivas ***sentir*** e ***agir*** para formar uma hierarquia de comportamentos onde cada um deles responde diretamente à determinada entrada sensória. Alguns trabalhos referentes a sua utilização são (MATARIC, 1992b; NILSSON, 1994; GOODRIDGE; KAY; LUO, 1996; XU; WANG; HE, 2003). A criação de arquiteturas reativas foi motivada por várias razões. Dentre elas, devido à deficiência das arquiteturas hierárquicas, ao surgimento da IA conexionista, também inspirada por ciências tais como biologia, psicologia e cognição, e pela diminuição do custo e aumento da capacidade computacional. Apesar das arquiteturas puramente reativas serem as que melhor tratam as dinâmicas de um ambiente, elas alcançam um número reduzido de tarefas porque eliminam qualquer forma de representação explícita de conhecimento, impossibilitando assim a execução de tarefas mais complexas que exijam algum tipo de planejamento.

Contudo o paradigma híbrido, que emergiu nos anos 90, tem como base a utilização conjunta de idéias dos paradigmas reativos e hierárquicos. Em linhas gerais uma arquitetura baseada no paradigma híbrido deliberativo/reativo organiza seu funcionamento da seguinte forma: primeiramente o sistema planeja (delibera) como decompor uma tarefa em sub-tarefas, que em seguida são tratadas pela implementação de comportamentos. A maneira como a informação sensória é tratada no paradigma híbrido é um misto dos

estilos hierárquico e reativo. Alguns exemplos de arquiteturas que utilizam o paradigma híbrido são (SCHöLKOPF; MALLOT, 1995; ARLEO; MILLáN; FLOREANO, 1999; ROISENBERG et al., 2004; CAPI; DOYA, 2005; KONIDARIS; HAYES, 2005). Como a criação dos paradigmas reativo e híbrido foi inspirada em ciências como etologia e psicologia cognitiva que consideram comportamentos como blocos de construção básicos da inteligência em seres vivos, uma arquitetura dita *baseada em comportamento* tem como base um destes paradigmas.

1.3 O Estado da Arte: Robótica Móvel

1.3.1 Robótica Bioinspirada

A busca incessante por idéias que possam inovar a navegação autônoma de robôs móveis, faz com que estudos principalmente na área de etologia e neurologia estejam no centro das atenções dos pesquisadores em robótica. Animais como ratos, por exemplo, são proficientes em navegação e o estudo tanto dos comportamentos quanto do cérebro destes animais já rendeu diversas teorias a respeito de como eles desempenham tal habilidade.

A teoria dos mapas cognitivos (TOLMAN, 1932; O'KEEFE; NADEL, 1978), por exemplo, é muito utilizada em diversos níveis de abstração, na implementação de sistemas de navegação. Uma importante hierarquia e análise destes modelos são encontrados em (TRULLIER et al., 1997; FRANZ; MALLOT, 2000). Trullier et al. (1997) revisaram diversos sistemas de navegação presentes na literatura que se referiam como "biologicamente inspirados", com o intuito de proporem um framework hierárquico de classificação. Os autores concluíram que os modelos revisados estavam aquém da principal capacidade dos "Mapas Cognitivos" reais que consiste da execução robusta de comportamentos como desvio e atalho, por exemplo. A hierarquia proposta pelos autores sugere quatro categorias (***orientação***, ***resposta ao disparo de reconhecimento***, ***navegação topológica*** e ***navegação métrica***) que indicam o nível de complexidade do processamento requerido, ou seja, como a informação é percebida, representada e processada. Na perspectiva dos paradigmas de controle em robótica inteligente, as duas primeiras classes podem caracterizar os sistemas reativos, enquanto as duas últimas os deliberativos ou híbridos deliberativo/reativo.

Franz e Mallot (2000) consideraram a hierarquia proposta por Trullier et al. (1997) e fizeram uma análise de sistemas miméticos que contribuíram significativamente em dois aspectos. Primeiro, modelos computacionais aplicados a situações do mundo real; segundo, os modelos que propuseram novos mecanismos de navegação disponíveis para aplicações técnicas, principalmente no campo da navegação "*indoor*". Os autores também constataram

que capacidades mais complexas, tal como a habilidade dos vertebrados de descobrir um caminho de um lugar a outro, ainda constitui um desafio para os sistemas que visam algum tipo de autonomia. Além disto, nenhum dos sistemas avaliados alcançou a flexibilidade e o desempenho da navegação de abelhas e formigas, muito menos ainda de pássaros e peixes migratórios.

Outra linha de pesquisa em robótica bioinspirada propõe a implementação de modelos CNE (do acrônimo em inglês Computational Neuroethology). Os modelos CNEs de navegação computacional são baseados em estudos de substratos neurais de cérebro de animais e suas funções envolvidas em tarefas de navegação (BROWNING, 2000). Estes modelos, os quais não são o foco de interesse deste trabalho de tese, também formam a base de construção de muitos sistemas reativos e se referem aos modelos biologicamente plausíveis.

Para o desenvolvimento deste trabalho de tese, foram considerados os modelos de navegação biologicamente inspirados porém, o foco de atenção não esteve voltado a questões de plausibilidade biológica e sim a questão de como integrar de maneira funcional e eficiente os controles deliberativos e reativos a fim de produzir um sistema de navegação facilmente aplicável a um robô real com limitados recursos computacionais e direcionado à resolução de uma tarefa de navegação complexa. Portanto, foram estudados alguns exemplos de sistema de navegação baseados na teoria dos mapas cognitivos, e/ou que utilizaram técnicas neurais para navegação *indoor* ou em labirintos, p.ex. (MATARIC, 1992b), (SCHMAJUK; THIEME, 1992), (SCHöLKOPF; MALLOT, 1995), (HAFNER, 2005), (WYETH; BROWNING, 1998), (VOICU; SCHMAJUK, 2000), (VOICU; SCHMAJUK, 2002).

Ambientes do tipo labirinto são ótimas provas de teste para o desenvolvimento de modelos de navegação computacional. Inspirando-se em experimentos com ratos, as primeiras competições em robótica móvel utilizaram labirintos com o objetivo de desafiar construtores de robôs (BRäUNL, 2003). Este tipo de ambiente pode ser ponto de partida para a construção de sistemas mais complexos. Por exemplo, Voicu e Schmajuk (2001) e Voicu (2003) realizaram navegação planejada em ambientes abertos, partindo de abordagens prévias voltadas a labirintos (SCHMAJUK; THIEME, 1992; VOICU; SCHMAJUK, 2000). Ambos Schmajuk e Thieme (1992), Voicu e Schmajuk (2000) e Voicu e Schmajuk (2002) preocuparam-se com questões de plausibilidade biológica, tanto na construção de seus modelos, onde o mapeamento cognitivo é implementado através de uma rede neural associativa, quanto nos comportamentos resultantes. Os autores utilizaram o mesmo protocolo realizado em experimentos com ratos, a fim de reproduzirem, em um agente simulado, os mesmos comportamentos apresentados por estes animais. Entretanto, a mai-

oria das abordagens para a navegação em labirintos propostas na literatura não trata, por exemplo, de questões como mudanças no ambiente em tempo de operação e ainda, necessitam de informação a priori tais como tamanho ou número de células de grid do ambiente.

1.3.2 Redes Neurais Artificiais

Redes Neurais Artificiais (RNAs) e aprendizado de máquina são técnicas de Inteligência Artificial maciçamente utilizadas como blocos básicos de construção nas arquiteturas de controle de robô móveis autônomos. Uma rede neural corresponde a um sistema adaptativo formada por unidades de processamento simples, que funcionam de forma paralela e distribuída, e armazenam conhecimento através de experiências (HAYKIN, 2001). Por apresentarem propriedades intrínsecas de generalização, tolerância a falhas, paralelismo e eficientes algoritmos de aprendizado (OMIDVAR; SMAGT, 1997), as RNAs tornam-se poderosas ferramentas para modelar comportamentos adaptativos e podem ser aplicadas em problemas de navegação como localização p. ex. (FILLIAT; MEYER, 2003a), construção de mapas, p. ex. (KRöSE; EECEN, 1994; KURZ, 1996; ZIMMER, 1996; OLIVEIRA, 2001) e planejamento de caminho (MILLáN, 2003).

Roisenberg (2000) por exemplo, propôs a utilização de técnicas da IA conexionista junto a uma arquitetura baseada em comportamentos chamada PiramidNet, cujo princípio é ser composta por RNAs organizadas em estruturas modulares e hierárquicas (ROISENBERG et al., 2004; VIEIRA, 2004). Esta abordagem inspira-se especificamente na organização global do cérebro da qual sua estrutura hierárquica em camadas produz comportamentos complexos. RNAs têm mostrado eficiência na resolução de uma série de classes de problemas, porém quando utilizadas em aplicações grandes e complexas mostram-se limitadas. Como na biologia, o processo evolutivo desenvolveu estruturas modulares nos cérebros dos seres vivos, na tentativa de imitar este processo, a arquitetura PiramidNet é composta por redes diretas atuando no nível mais baixo com a função de implementar os comportamentos mais simples e reativos, p. ex. (SILVA, 2001). Além disto, para a elaboração de comportamentos mais complexos, como os chamados comportamentos motivados, ou mesmo para tratar o controle dos comportamentos reativos e reflexivos, são inseridas nas camadas mais altas redes neurais recorrentes, p. ex. (ROISENBERG et al., 2004; VIEIRA, 2004).

1.4 Objetivo da Tese

O objetivo principal deste trabalho é desenvolver uma arquitetura de controle híbrida baseada em comportamentos, denominada **NeuroCog**, que propicie seu uso em diferentes aplicações incluindo veículos autônomos e robôs móveis com poucos recursos computacionais, direcionados a uma tarefa de navegação complexa em labirintos dinâmicos. Para alcançar o objetivo principal, os seguintes objetivos específicos foram definidos:

- Desenvolver e evoluir uma série de acoplamentos entre diferentes arquiteturas de RNAs e técnicas de aprendizado online, tais como Aprendizagem por Reforço, como arquiteturas de controle neural.
- Investigar quais tipos de comportamentos podem ser obtidos através destes arranjos neurais de controle e se de fato ocorre um aprendizado em tempo de operação eficiente, mediante o aumento progressivo da complexidade do labirinto e tarefa de navegação.
- Devido a necessidade de uma representação de conhecimento explícita do ambiente do robô, determinar uma estrutura de integração entre os níveis deliberativo e reativo para a construção de uma arquitetura de controle híbrida.
- Desenvolver um método eficiente para o aprendizado desta representação de conhecimento, buscando inspiração na teoria de mapas cognitivos.
- Dispor um processo de tomada de decisão para tratar o dilema conhecido como "*exploration* versus *exploitation*" relacionado ao ajuste entre os comportamentos deliberativos de exploração e planejamento.
- Propor como medida de desempenho da arquitetura híbrida, o custo do aprendizado em tempo de operação da camada reativa, que está relacionado ao ajuste do dilema citado na camada deliberativa.

1.5 Organização da Tese

Este trabalho de tese está dividido em oito capítulos, onde esta introdução é o primeiro deles.

O Capítulo 2 apresenta uma revisão sobre as principais arquiteturas de robôs móveis inteligentes segundo os paradigmas de controle hierárquico, reativo e híbrido.

O Capítulo 3 descreve uma hierarquia de modelos de navegação bastante utilizada na literatura de robôs móveis, para classificação de sistemas computacionais de navegação biologicamente inspirados.

O Capítulo 4 mostra como diferentes arquiteturas de redes neurais artificiais e formas de aprendizado podem ser utilizadas em um sistema de controle.

O Capítulo 5 apresenta uma descrição a respeito de como evoluíram sucessivos sistemas neurais reativos e como esta evolução resultou na arquitetura neural reativa que serve de suporte à camada deliberativa da arquitetura final **NeuroCog**.

O Capítulo 6 descreve o desenvolvimento da arquitetura híbrida **NeuroCog** como proposta de controle inteligente de robôs móveis, para navegação em labirintos estáticos e dinâmicos.

O Capítulo 7 apresenta as simulações e resultados referentes às arquiteturas neurais reativas. Em seguida, são vistas as simulações referentes as arquiteturas híbridas prévias. E por último são apresentadas as simulações e resultados referentes à arquitetura híbrida **NeuroCog**.

No Capítulo 8, são apresentadas conclusões e considerações sobre as principais contribuições deste trabalho e também são apresentadas algumas sugestões de trabalhos futuros que ficaram em aberto.

Finalmente, o Apêndice A apresenta as bases de regras utilizadas no treinamento em tempo de operação, das redes diretas que formam os arranjos neurais reativos, apresentados no Capítulo 5.

CAPÍTULO 2

Arquiteturas de Controle Inteligente

2.1 Introdução

Na literatura, existem várias definições para o termo *arquitetura* de robôs móveis. Segundo Mataric (1992a), uma arquitetura constitui um princípio de organização de um sistema de controle. O projeto de uma arquitetura de controle visa habilitar um robô móvel autônomo a operar em seu ambiente utilizando-se de seus recursos físicos e computacionais. Seu sistema de controle deve assegurar o cumprimento de suas tarefas de maneira estável e robusta.

Muitas são as arquiteturas propostas porém, não existe um paradigma definitivo que atenda a todas as funcionalidades requeridas (MEDEIROS, 1998). As arquiteturas concebidas para uma dada aplicação podem ter desempenho aquém do desejado para outras aplicações. Além disto, segundo Russell e Norvig (1995) não existe nenhuma teoria que possa ser usada para provar que um projeto é melhor do que outro. Entretanto, existem muitos projetos para a construção de um robô inteligente que se distinguem de maneira superficial quanto à questão de como o robô deve *perceber* e *agir* no mundo real.

Na literatura de robôs móveis autônomos há uma classificação bastante aceita na comunidade de pesquisa que estabelece três categorias de arquiteturas, hierárquica, reativa e híbrida, de acordo como estão organizados seus sistemas de controle e como os dados dos sensores são processados e distribuídos entre seus componentes (MURPHY, 2000).

Neste Capítulo serão revisadas as arquiteturas mais representativas referente a estas categorias, com o objetivo de identificar suas características positivas e negativas, e os aspectos que foram utilizados na concepção da arquitetura **NeuroCog** proposta neste trabalho.

2.2 Arquiteturas Hierárquicas

As arquiteturas hierárquicas são seqüenciais e organizadas: primeiramente o robô explora o ambiente, construindo um mapa global com um certo nível de detalhes pré-especificados. Em seguida o robô planeja as ações necessárias para alcançar o objetivo desejado. Posteriormente, ele age para cumprir a primeira diretiva. Depois de realizar a seqüência (***sentir***, ***planejar***, ***agir***), ele inicia o ciclo novamente: sente as conseqüências de sua ação, replaneja as diretivas e age, como ilustrado no esquema da Figura 2.1.

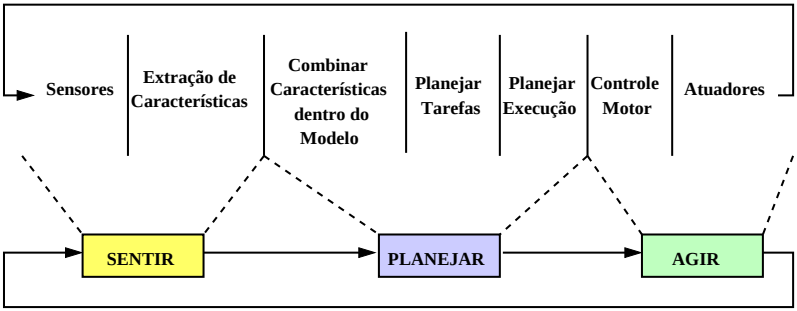
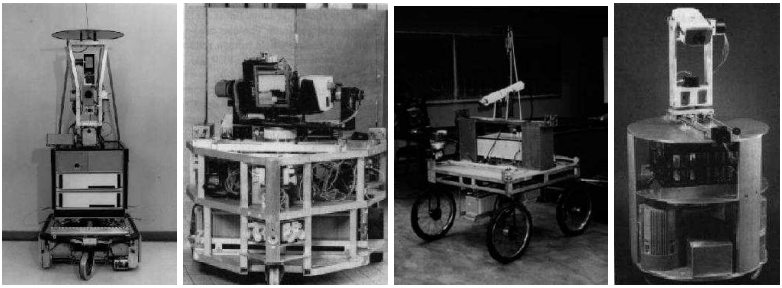


Figura 2.1: Decomposição horizontal das tarefas dentro da organização SENTIR, PLANEJAR, AGIR das arquiteturas hierárquicas. Fonte adaptada de Murphy (2000).

A estrutura de informação na arquitetura hierárquica é monolítica, isto é, é realizada a fusão de todas as observações sensórias em uma única estrutura de dados global que é acessada pelo planejador. Esta estrutura de dados global é chamada de *modelo de mundo*.

As deficiências desta abordagem tem uma série de razões. A principal delas é de que mapas são construídos com base na informação sensória que geralmente é parcial e corrompida, além da complexidade do ambiente ser bastante reduzida para simplificar o processo de modelagem (p. ex. o mundo real 3D modelado como polígonos 2D). Nas arquiteturas hierárquicas, decisões sobre a ação do robô são baseadas somente na informação de mapas. Como modelos são imprecisos, então decisões de ação também são imprecisas. Robôs que usam estas arquiteturas se movem muito lentamente para evitar colisões, e se um obstáculo for detectado em um caminho planejado, os robôs têm de parar, a fim de replanejar um novo caminho (SALICHS; MORENO, 2000).

A Figura 2.2 ilustra alguns dos primeiros robôs móveis autônomos



(a) (b) (c) (d)

Figura 2.2: Os primeiros robôs móveis autônomos com arquiteturas de controle hierárquicas. (a) Shakey de Nilsson (1984). (b) Hilare, em Tolouse de Giralt, Chatila e Vaisset (1984). (c) Stanford Cart de Moravec (1993). (d) CMU Rover de Crowley (1985).

projetados a partir de arquiteturas hierárquicas. O primeiro robô móvel autônomo denominado Shakey [Figura 2.2(a)], em uma primeira versão, integrava representações geométricas e simbólicas do mundo em uma única representação global, que era utilizada por um processo de prova automática de teoremas para gerar e executar de planos. A execução de quase todos os planos falhava, devido ao robô não monitorar problemas como deslizamento de rodas, imprecisão de medidas, etc. Em uma segunda versão de sua arquitetura, foram incorporados níveis de ação a fim de melhorar a execução de planos, porém o mundo continuou sendo representado de maneira simbólica, o que também tornava todo o processo bastante lento (RUSSELL; NORVIG, 1995).

A seguir serão apresentados dois exemplos de arquiteturas hierárquicas as quais se tornaram as mais representativas e utilizadas pelos pesquisadores de robótica inteligente na época em que foram propostas. Elas se referem à arquitetura NHC (*Nested Hierarchical Controller*) Controlador Hierárquico Aninhado e à RCS (*Real-time Control System*) Sistema de Controle em Tempo-real.

2.2.1 Arquitetura NHC

A Figura 2.3 ilustra a arquitetura NHC, cujos componentes são facilmente identificados com as diretivas (***sentir***, ***planejar***, ***agir***). O robô inicia coletando informações de seus sensores e combinando estas informações em uma estrutura de dados denominada **Modelo de Mundo,** que também pode conter um conhecimento a priori sobre o ambiente. Em seguida o robô pla-

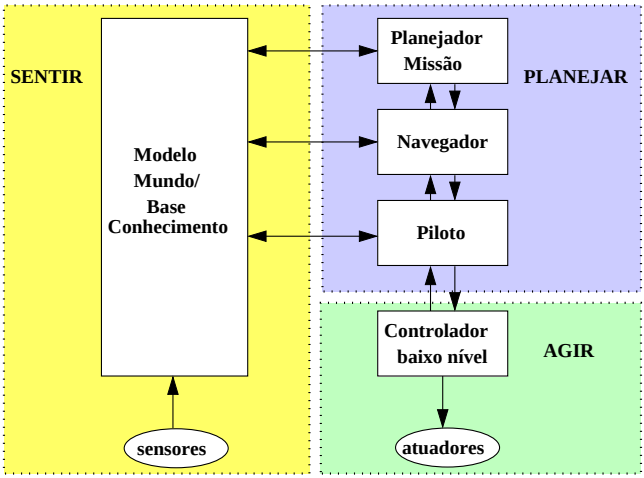


Figura 2.3: A arquitetura NHC. Fonte adaptada de Murphy (2000).

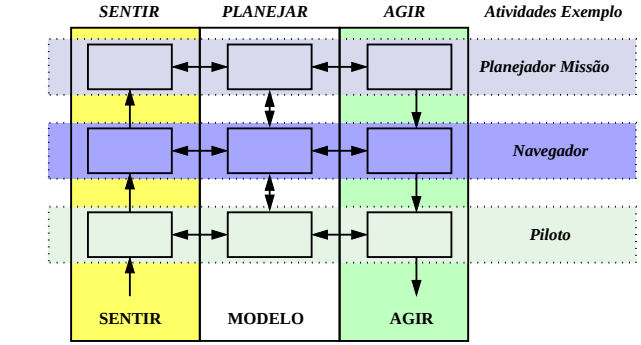
neja quais ações irá tomar. O planejamento para navegação tem um procedimento local consistindo de três passos executados pelo **Planejador de Missão, Navegador** e **Piloto**. Cada um destes módulos tem acesso ao **Modelo de Mundo** para computar sua porção de planejamento. No último passo do planejamento, o módulo **Piloto** gera ações específicas para o robô fazer (ex. virar à esquerda, virar à direita, mover-se para frente a uma dada velocidade). Estas ações são traduzidas em sinais de controle pelo **Controlador de baixo nível**. Juntos, o **Controlador de baixo nível** e atuadores formam a porção ***agir*** da arquitetura.

Uma vantagem da arquitetura NHC é sua característica de interpor planejamento e ação. O robô inicia executando um plano que pode ser alterado caso o mundo seja diferente do que ele espera. Esta decomposição é inerentemente hierárquica em inteligência e escopo. **O Planejador de Missão** é hierarquicamente superior ao **Navegador** e este é hierarquicamente superior ao **Piloto**. Ou seja, **Planejador de Missão** é responsável por um nível de abstração mais alto do que o **Navegador**, etc. A organização da arquitetura NHC foi muito utilizada por outras arquiteturas hierárquicas e também por híbridas. Uma desvantagem desta decomposição da função de planejamento NHC é que ela é apropriada apenas para tarefas de navegação. A regra de um **Piloto** para controle dos atuadores não é clara. Em seu desenvolvimento inicial, a arquitetura NHC nunca foi implementada e testada em um robô móvel real,

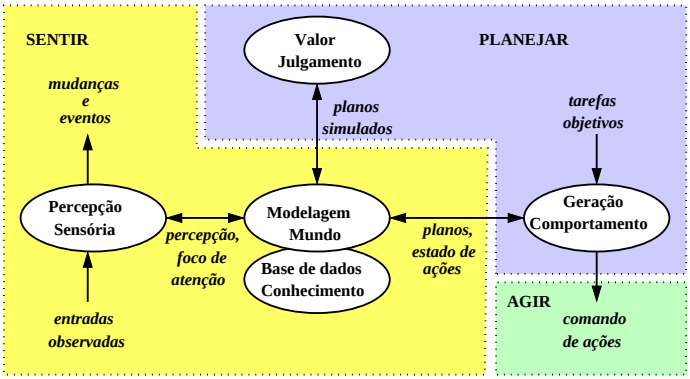
devido aos custos de hardware, a maioria dos pesquisadores a implementaram somente em simulação (MURPHY, 2000).

2.2.2 Arquitetura RCS

A arquitetura RCS (*Real-time Control System*) foi desenvolvida e detalhada com o objetivo de guiar principalmente fabricantes de manipuladores industriais, interessados em estender as capacidades desta classe de robôs.



(a)



(b)

Figura 2.4: Arquitetura RCS. (a) Hierarquia de camadas SENTIR-MODELO-AGIR. (b) Decomposição funcional. Fonte adaptada de Murphy (2000).

A arquitetura RCS, mostrada na Figura 2.4(a), utilizou a arquitetura NHC em seu projeto. Já a Figura 2.4 (b) ilustra a decomposição funcional da arquitetura onde as atividades da entidade SENTIR são agrupadas em um conjunto de módulos conforme os níveis de percepção sensória. A saída dos sensores é passada para o módulo de **Modelagem do Mundo** que constrói um mapa global usando informações de sua base de conhecimento associadas aos sensores e aos diferentes domínios de conhecimento. A principal diferença com relação à arquitetura NHC está no módulo de **Percepção Sensória**, que introduz um passo de pré-processamento entre os sensores e a fusão sensorial dentro do modelo de mundo.

O módulo **Valor de Julgamento** provê a maioria das funcionalidades associadas com a atividade PLANEJAR: além de planejar, simula também os planos para assegurar a realização da tarefa. O planejador entrega o plano para outro módulo, **Geração de Comportamento**, que converte os planos em ações que podem ser executadas pelo robô (AGIR).

O módulo **Geração de Comportamento** é similar ao Piloto na arquitetura NHC, porém ele está menos focado a tarefas de navegação. O uso do termo "comportamento" na arquitetura RCS não implica que ela tenha características de arquitetura reativa ou híbrida, o fato dela integrar toda a percepção dentro do modelo de mundo global para então planejar e agir a caracteriza como uma arquitetura hierárquica. Existe um outro módulo, chamado **Operador de interface**, não mostrado na figura, que permite a um operador analisar o que está sendo realizado pela arquitetura.

A arquitetura RCS foi adotada por muitas agências governamentais tais como *NASA* e o *US Bureau of Mines* que contrataram universidades e companhias para construírem protótipos de robôs. A vantagem desta arquitetura era sua característica de fundir vários tipos de sensores em um mapa global. Porém, a arquitetura RCS foi considerada muito detalhada e restritiva segundo pesquisadores de AI, que continuaram desenvolvendo novas arquiteturas e paradigmas de controle. A Figura 2.5 mostra quatro exemplos de robôs que usaram a arquitetura RCS.

É importante notar que as arquiteturas NHC e RCS são melhor aplicadas ao controle semi-autônomo. Desta forma, o operador humano forneceria o modelo do mundo e decidiria a missão, que seria decomposta em um plano, que por sua vez também seria decomposto em ações. E assim, o controlador de baixo nível (robô) executaria as ações. Em uma perspectiva de robótica, o robô poderia substituir mais funções e "subir" na hierarquia de autonomia, por exemplo, assumindo as responsabilidade de **Piloto**. Já em uma perspectiva de IA, o humano só poderia servir como **Planejador de Missão**. A partir deste ponto de vista, Albus, McCain e Lumia (1989) desenvolveram uma ver-

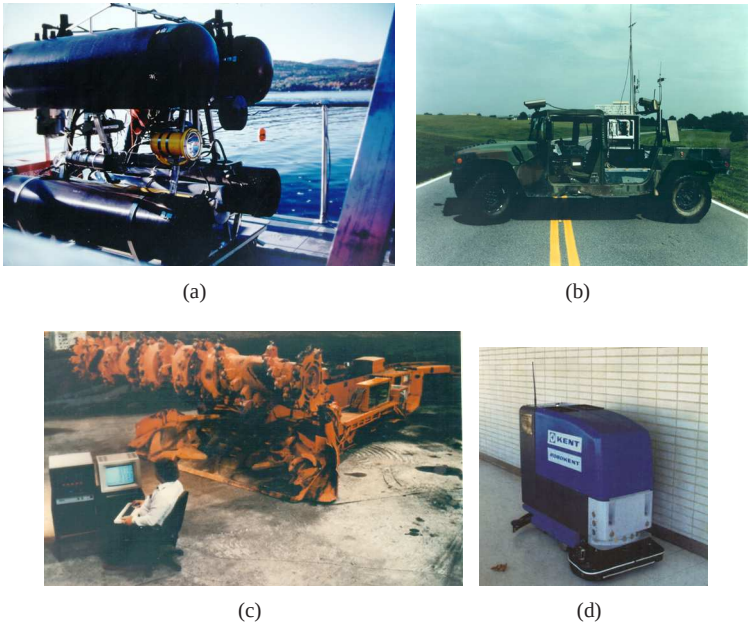


Figura 2.5: Quatro dos diversos robôs que têm usado a arquitetura RCS. (a) Um robô submarino. (b) Um veículo guiado. (c) Um robô de escavação. (c) Um robô limpador de chão comercial. Fotos cortesia do NIST (National Institute of Standards and Technology). Fonte adaptada de Murphy (2000).

são da arquitetura RCS para tele-operação de um braço do robô no espaço, que foi chamada de arquitetura NASREM, e é usada ainda hoje.

As arquiteturas NHC e RCS forneceram diretrizes em como decompor uma arquitetura de controle em módulos intuitivos. A decomposição da arquitetura NHC em **Planejador de Missão**, **Navegador** e **Piloto** esteve estritamente focada em navegação, enquanto a arquitetura RCS tornou-se mais ampla. Ambas têm sido usadas com sucesso para guiar veículos, e em especial a arquitetura RCS para controlar equipamentos de escavação, submarinos e carros, apresentando um alcance razoável de aplicações. Entretanto, a portabilidade para outros domínios não é clara ou nula. Quanto à robustez, a arquitetura RCS tenta prover alguns mecanismos explícitos, tais como o módulo **Valor Julgamento** que simula um plano para certificar-se de que ele será bem sucedido. Porém esta forma de robustez é muito limitada devido ao tempo de espera para o robô simular suas ações antes de executá-las, além de

não ser apropriada a simulação de todas as ações.

Portanto, segundo Murphy (2000), uma das principais desvantagens das arquiteturas hierárquicas é que elas foram desenvolvidas visando aplicações específicas, e não para servirem como arquiteturas genéricas para outras aplicações. Em contrapartida uma vantagem foi a ordenação do relacionamento entre *sentir*, *planejar* e *agir*. Entretanto outra desvantagem se refere ao planejamento, onde o robô deve primeiramente atualizar um modelo de mundo global, para depois planejar, o que produz um gargalo significante. Além disto, *sentir* e *agir* estão sempre desconectados, o que elimina qualquer possibilidade de ações do tipo estímulo-resposta, as quais foram a principal inovação introduzida pelas arquiteturas reativas.

2.3 Arquiteturas Reativas

As arquiteturas reativas evoluíram devido à insatisfação e limitações das arquiteturas hierárquicas e com a influência das pesquisas em etologia.[1] Em (BROOKS, 1986) e (BROOKS, 1990), com base na interpretação de que, no comportamento animal, a inteligência se estende em camadas verticais, conforme representado na Figura 2.6 (a), BROOKS criou a arquitetura de subsunção que tornou-se a base das arquiteturas reativas mais conhecidas e usadas.

Arquiteturas reativas se caracterizam pela ausência de um modelo de mundo e planejamento. Como pode ser visto na Figura 2.6 (b), *sentir* e *agir* são fortemente acoplados para a geração de comportamentos que operam em seqüência ou concorrentemente. Comportamentos simples podem ser unidos para produzir comportamentos mais complexos. Por exemplo, na Figura 2.6 (a), os comportamentos no nível mais baixo como evitar colisões, vagar e explorar atuando em conjunto, podem produzir o comportamento de mapeamento implícito do ambiente, no nível mais alto.

Portanto, a característica chave de um sistema reativo é que movimentos são consequências de reações à informação vinda diretamente dos sensores, ao invés de uma ação planejada. Como resultado houve uma mudança radical de ações baseadas em planos (arquiteturas hierárquicas) para ações baseadas em sensores, e assim os robôs começaram a mover-se mais rapidamente e com segurança em ambientes complexos, mesmo com obstáculos em movimento. Além das arquiteturas reativas do tipo subsunção, existe também as que são baseadas em campos potenciais. Ambas são apresentadas com mais detalhes nas seções seguintes.

[1] Ciência que estuda o comportamento de animais.

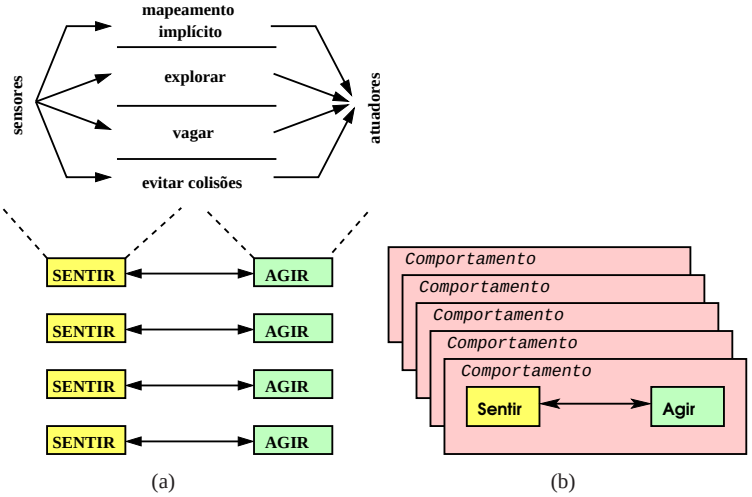


Figura 2.6: (a) Decomposição vertical das tarefas dentro da organização SENTIR-AGIR das arquiteturas reativas. (b) A organização SENTIR-AGIR com comportamentos múltiplos e concorrentes. Fonte adaptada de Murphy (2000).

2.3.1 Arquitetura de Subsunção

A arquitetura de subsunção é uma arquitetura genérica proposta por Brooks (1986) onde os comportamentos são organizados em camadas de competência. Como esquematizado na Figura 2.7, os níveis mais baixos encapsulam habilidades mais gerais, tais como evitar colisões, vagar, etc. Comportamentos são disparados de uma maneira estímulo-resposta, sem que um programa externo os coordene e controle explicitamente. Entretanto, a coordenação das camadas ou níveis na arquitetura de subsunção é feita pelas camadas mais altas, que possuem comportamentos mais específicos e orientados a objetivo, subsumindo as camadas mais baixas. As camadas de comportamento operam concorrente e independentemente, havendo a necessidade de um mecanismo para tratar conflitos potenciais. A alternativa utilizada é a do tipo vencedor-leva-tudo, onde o vencedor é sempre a camada mais alta. Os comportamentos dentro de uma camada são coordenados por um autômato de estados finitos, que podem ser diretamente implementados em hardware. Na arquitetura de subsunção, uma tarefa se cumpre pela ativação da camada apropriada, que ativa a camada inferior e assim por diante.

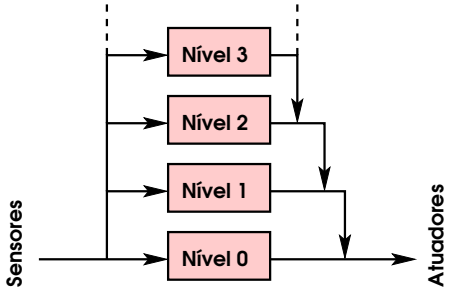


Figura 2.7: A arquitetura de Subsunção. Fonte adaptada de (BROOKS, 1986)

Através da arquitetura de subsunção são concebidos a maioria dos sistemas puramente reativos. Por exemplo, BROOKS construiu os primeiros robôs hábeis para andar, evitar colisões e subir sobre obstáculos sem as pausas "mover-pensar" do robô Shakey. Um deles é ilustrado na Figura 2.8, Genghis, um robô andador de seis pernas que aprendia a escalar obstáculos, com cada perna reagindo independentemente ao ambiente. A seguir é mostrado um exemplo de como pode ser implementada uma arquitetura de subsunção.

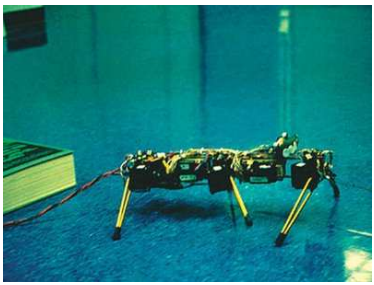


Figura 2.8: O robô "inseto" Genghis: robô andador com seis pernas construído no *MIT AI Laboratory* a partir da arquitetura de subsunção. Fonte adaptada de (MURPHY, 2000).

Exemplo de implementação de uma arquitetura de subsunção

A Figura 2.9 ilustra um exemplo de desenvolvimento do controle de um robô a partir de uma arquitetura de subsunção. Considerando-se apenas o

Nível 0 da arquitetura (excluindo o ponto de subsunção, S), o robô é capaz de se mover até que ele colida com algum obstáculo. Este robô possui sensores do tipo ultra-sônico (ou qualquer outro tipo de sensor de proximidade), cada um apontando para uma direção diferente, e dois atuadores, um para mover o robô para frente e outro para fazer o robô girar. O módulo SONAR lê as medidas dos sensores e produz uma plotagem polar que representa a faixa de leituras em coordenadas polares (r, θ), ao redor do robô. Se a leitura de

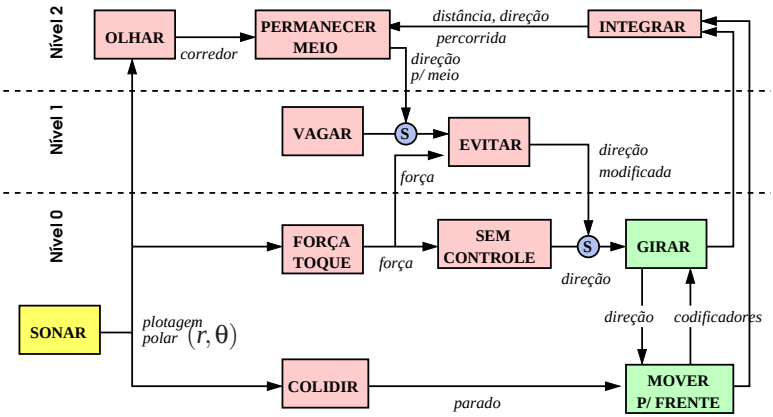


Figura 2.9: Exemplo de implementação de uma arquitetura de subsunção. Fonte adaptada de Murphy (2000).

um sensor sem atividade está abaixo de um certo limiar, o módulo COLIDIR declara uma colisão e envia um sinal *parado* para o atuador MOVER P/ FRENTE. Se o robô estava se movendo para frente, ele então pára. Enquanto isto, o módulo FORÇA TOQUE recebeu a mesma plotagem polar. Este módulo trata cada leitura de sonar como uma força repulsiva, que é representada como um vetor. Este vetor repulsivo é passado para o módulo GIRAR, que por sua vez também passa para o módulo MOVER P/ FRENTE. MOVER P/ FRENTE usa a magnitude do vetor para determinar a magnitude do próximo movimento para frente. O comportamento observado é aquele em que o robô permanece parado ainda que ele esteja em um espaço desocupado, mas até que um obstáculo se re-aproxime dele. Se o obstáculo aparecer de um lado, o robô irá girar 180^o do outro e em seguida se mover nesta direção, e neste caso ele estará sem controle. O robô reage tanto a um obstáculo que está parado como em movimento. Entretanto, se o robô estiver encurralado, ele aplica o resultado de SEM CONTROLE, onde ele pára, gira e começa a se mover novamente para frente. Paradas previnem que o robô colida com obs-

táculo enquanto ele gira e se move para frente. Portando o Nível 0 ilustra como um conjunto razoavelmente complexo de ações podem emergir a partir de módulos simples. O fluxo de dados dos sensores percorrem comportamentos concorrentes até os atuadores. Estes comportamentos primitivos (`sem controle` e `colidir`) fazem emergir o comportamento de evitar obstáculos, ou uma camada de competência.

Considerando a construção de um robô que vaga e ao mesmo tempo evita obstáculos ao invés de apenas parar na presença deles. Sob subsunção, uma segunda camada de competência, Nível 1, seria adicionada (Figura 2.9). Neste caso, o Nível 1 consiste de um módulo VAGAR que computa uma direção aleatória a cada *n* segundos, que necessita ser passada para os módulos GIRAR e MOVER P/ FRENTE. Porém esta direção não pode ser passada para o módulo GIRAR diretamente, pois isto sacrificaria o comportamento de evitar obstáculos, dado que GIRAR aceita somente uma entrada. Uma solução é adicionar outro módulo no Nível 1, EVITAR, que combina o vetor FORÇA TOQUE com o vetor VAGAR. Como a adição de um novo módulo EVITAR cria-se uma resposta mais sofisticada com relação aos obstáculos. EVITAR combina a direção da força de escape com a direção desejada. Isto resulta em uma direção mais efetiva do que fazer o robô girar 180^o, perdendo o progresso de mover-se para frente. O módulo EVITAR também é hábil para observar os componentes da camada mais baixa. A direção de saída do módulo EVITAR tem a mesma representação da saída do módulo SEM CONTROLE, e assim, GIRAR pode aceitar a direção de ambas as origens.

O problema é decidir de qual módulo ou camada aceitar o vetor de direção. A subsunção torna esta decisão simples: a saída do nível mais alto subsume a saída do nível mais baixo. Desta forma a subsunção pode ser feita de duas maneiras:

1. *inibição*. Na inibição, a saída do módulo que subsume é conectada à saída de outro módulo. Se a saída do módulo que subsume tem qualquer valor, a saída do módulo subsumido é bloqueada. Inibição atua como uma torneira, abrindo ou fechando um fluxo de saída.

2. *supressão*. Na supressão, a saída do módulo que subsume é conectada à entrada de outro módulo. Se a saída do módulo que subsume está ligada, ele substitui a entrada normal do módulo subsumido. Supressão atua como um chaveador trocando um fluxo de entrada por outro.

Neste exemplo, o módulo EVITAR elimina através de S (Figura 2.9) a saída de SEM CONTROLE, que ainda está executando. Ao invés disto, a saída de EVITAR vai para GIRAR. O uso de camadas e subsunção permite

que novas camadas sejam construídas no topo das camadas menos competentes, sem modificar as camadas mais baixas. Isto proporciona modularidade, simplificação de testes e robustez, haja visto o caso do Nível 0 permanecer intacto, ainda que alguma situação desabilite os comportamentos do Nível 1. Deste modo o robô ainda se torna hábil para preservar seu mecanismo de auto-defesa, livrando-se de obstáculos próximos.

Considerando-se ainda a adição de uma terceira camada para permitir ao robô percorrer corredores (Figura 2.9). O módulo OLHAR examina a plotagem polar do sonar e identifica um corredor. Porque identificar um corredor é computacionalmente mais custoso do que apenas extrair dados dos sensores, OLHAR pode levar mais tempo para executar do que os comportamentos das camadas mais baixas. OLHAR passa o vetor representando a direção do meio do corredor para o módulo PERMANECER MEIO. PERMANECER MEIO subsume o módulo VAGAR e entrega sua saída para o módulo EVITAR que então pode mudar a direção rapidamente ao redor de obstáculos.

Um problema normalmente encontrado é como fazer o robô manter seu curso enquanto o módulo OLHAR não computa a nova direção. Neste caso, o módulo INTEGRAR é quem observa os movimentos correntes do robô a partir dos codificadores nos atuadores. Isto dá uma estimativa de quão fora do curso o robô está percorrendo, dado a última atualização feita por OLHAR. PERMANECER MEIO pode usar dados de integração de caminho mais o curso pretendido, para computar um novo curso. INTEGRAR é um exemplo de módulo que fornece um estado interno, substituindo uma realimentação a partir do mundo real. Se por alguma razão, o módulo OLHAR não se atualizar, então o robô pode operar sem dados sensórios, ou até quebrar. Por esta razão, sistemas baseados na arquitetura de subsunção incluem constantes de tempo na supressão e inibição. Se a supressão a partir do PERMANECER MEIO executou por mais de n segundos sem uma nova atualização, a supressão cessa. Então o robô começaria a vagar, e qualquer problema (como o corredor sendo totalmente bloqueado) que tenha conduzido à perda de sinal se resolveria por si mesmo. Porém, o projeto supõe que um corredor estará sempre presente no ambiente do robô. Caso não esteja, o robô não irá se comportar como pretendido. Isto é um exemplo da conotação de que sistemas reativos são sem memória.

2.3.2 Arquiteturas Baseadas em Campos Potenciais

Outro estilo de implementação de arquiteturas reativas é a abordagem baseada em campos potenciais. Um campo potencial é uma estrutura ou campo vetorial representando uma região do espaço. Na maioria das aplica-

ções robóticas, o espaço é bi-dimensional, semelhante a um mapa. Este mapa pode ser dividido em quadrados, criando uma grade (x,y). Cada elemento da estrutura representa uma região quadrangular no espaço. Os objetos percebidos no mundo exercem um campo de força ao seu redor. Em outras palavras, campos potenciais podem ser interpretados como um campo de força agindo sobre o robô. Vetores indicariam a direção em que o robô poderia se mover pela superfície. Elevações na superfície fariam o robô se afastar ou rolar em derredor (vetores apontando para fora do centro da elevação) e as depressões atrairiam o robô (vetores apontando na direção do centro da depressão).

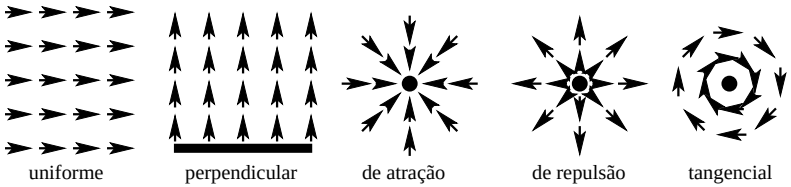


Figura 2.10: Os cinco campos potenciais primitivos.

Existem cinco campos potenciais básicos ou primitivos, como mostrado na Figura 2.10, eles podem ser combinados para resultar campos mais complexos, são eles *uniforme, perpendicular, atrativo, repulsivo* e *tangencial*. A Figura 2.11 é um exemplo de implementação do comportamento reflexivo SEM CONTROLE (Seção 2.3.1) através do campo potencial básico de repulsão.

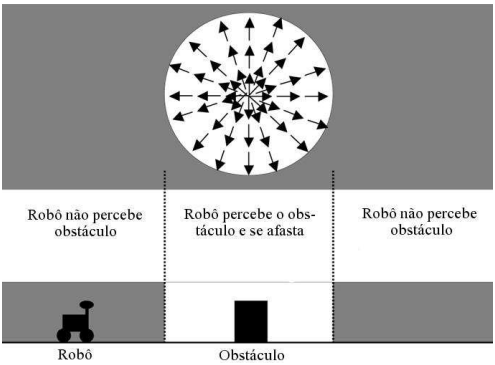


Figura 2.11: Implementação do comportamento “sem controle” via repulsão na abordagem de campos potenciais. Fonte adaptada de Murphy (2000).

O princípio de uma arquitetura baseada em campos potenciais é usar *vetores* para representar comportamentos e *soma de vetores* para combinar vetores a partir de diferentes comportamentos e produzir um comportamento resultante. Em linhas gerais, um robô terá forças de ação sobre ele vindo de múltiplos comportamentos, todos atuando de maneira concorrente.

O desenvolvimento de uma arquitetura baseada em campos potenciais torna-se vantajoso devido à sua representação ser contínua e de fácil visualização, mesmo em se tratando de uma grande região do espaço é possível se visualizar o comportamento global do robô. Outra vantagem é a fácil combinação de campos que podem ser parametrizados a fim de modificar comportamentos. E ainda, um campo bi-dimensional pode ser estendido dentro de um campo tri-dimensional onde os comportamentos desenvolvidos para 2D também funcionam para 3D. A principal desvantagem desta abordagem está relacionada ao problema do mínimo local onde múltiplos campos somam um vetor com magnitude 0. Porém existem na literatura muitas soluções dadas a este problema (MURPHY, 2000).

2.4 Arquiteturas Híbridas

As arquiteturas hierárquicas ou deliberativas vistas na Seção 2.2, apresentavam principalmente o problema do gargalo com relação ao planejamento, que é realizado somente após a atualização de um modelo de mundo global. Já as arquiteturas reativas permitiram que robôs operassem em tempo real utilizando processadores baratos, acessíveis e sem memória. Estas por sua vez não usam as características positivas do estilo deliberativo e eliminavam o planejamento. Desta forma, passou-se a pesquisar uma arquitetura que combinasse as características positivas de ambas, resultando no estilo arquitetural denominado híbrido deliberativo/reativo.

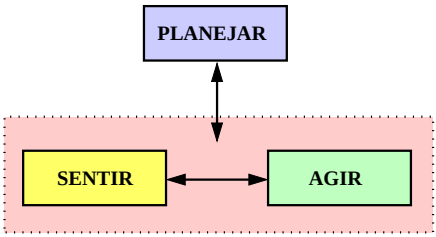


Figura 2.12: Organização básica PLANEJAR, SENTIR-AGIR das arquiteturas híbridas deliberativa/reativa. Fonte adaptada de Murphy (2000).

A organização básica de uma arquitetura híbrida deliberativa/reativa pode ser descrita como: PLANEJAR e então SENTIR-AGIR, conforme mostrada na Figura 2.12. O módulo PLANEJAR engloba toda deliberação e modelagem do mundo global, não apenas uma tarefa específica ou planejamento de caminho. O robô primeiro planeja como cumprir uma missão ou uma tarefa (usando um modelo de mundo global), e então dispara diferentes comportamentos (SENTIR-AGIR) pertencentes a um conjunto pré-definido para executar o plano (ou uma porção do plano). Os comportamentos são executados até que o plano se complete. Em seguida, o planejador gera um novo conjunto de comportamentos, e o procedimento se repete.

As arquiteturas híbridas apresentam um conjunto de componentes essenciais, geralmente apresentando os seguintes módulos ou objetos:

- ***Seqüenciador:*** é um agente que gera o conjunto de comportamentos necessários à execução de uma subtarefa, para determinar seqüências e condições de ativação. A seqüencia é usualmente representada como uma rede de dependência ou uma máquina de estados finitos, mas o seqüenciador deve ou gerar sua estrutura ou ser hábil para adaptá-la dinamicamente.

- ***Gerenciador de recursos:*** aloca recursos aos comportamentos.

- ***Cartógrafo:*** é um agente responsável pela criação, armazenamento e manutenção do mapa ou informação espacial, além de métodos para acessar os dados. O cartógrafo freqüentemente contém um modelo de mundo global e uma representação de conhecimento, ainda que isto não seja um mapa.

- ***Planejador de missão:*** é um agente que interage com o ser humano, operacionaliza comandos para o robô e constrói um plano de missão.

- ***Monitoração de desempenho e resolução de problema:*** trata-se de um agente que permite ao robô verificar se ele tem progredido ou não na execução de suas tarefas.

Como pode ser visto na Figura 2.13, a organização da percepção em uma arquitetura híbrida é mais complexa devido ao fato de que ela é em si uma ação híbrida. Para os módulos de **Comportamento**, a percepção permanece como nas arquiteturas reativas, ou seja, local e específica. Entretanto, planejamento e deliberação requerem modelos de mundo necessitando de acesso a um modelo de mundo global (módulo **Mapa Mundo/ Rep. Conhecimento**). O modelo é construído por um processo independente da per-

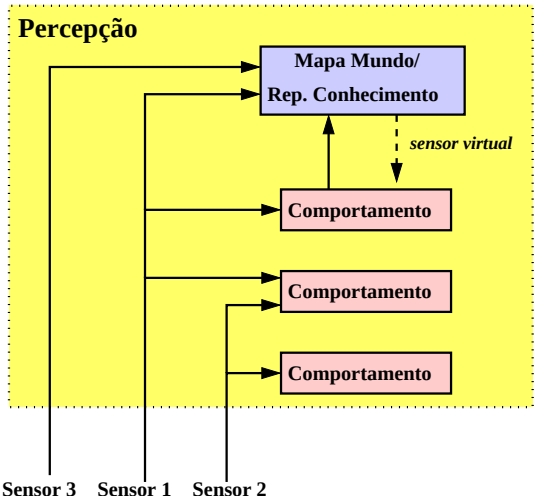


Figura 2.13: Organização da percepção nas arquiteturas híbridas: o modelo de mundo global pode ter seus próprios sensores, pode observar os sensores usados ou percepções criadas pelos comportamentos e pode atuar como um sensor virtual a ser utilizado por um comportamento. Fonte adaptada de Murphy (2000).

cepção baseada em comportamento. Todavia, ambos os esquemas de percepção baseada em comportamento ou baseada na construção de modelos podem compartilhar os mesmos dados a partir de diferentes sensores. Além disto, os processos de construção de modelos podem compartilhar as percepções criadas pelos esquemas de percepção ou podem ter sensores que são dedicados ao fornecimento de observações que são úteis para a modelagem do mundo, mas não são usados por nenhum dos módulos de **Comportamento** ativos.

As várias arquiteturas híbridas deliberativa/reativa se distinguem em diferentes aspectos. Em geral elas se dividem em três categorias de acordo com os seguintes critérios: i) a maneira como elas se dividem em modos deliberativo e reativo, ii) como são organizadas as atividades na porção deliberativa e iii) como o comportamento como um todo emerge. Assim, as arquiteturas híbridas se distinguem entre três categorias:

- arquiteturas gerenciais,
- arquiteturas de hierarquia de estados e
- arquiteturas orientadas a modelo,

cujas características e exemplos serão vistos na sequência.

2.4.1 Arquiteturas Gerenciais

O estilo gerencial está focado na subdivisão em camadas da porção deliberativa. Esta subdivisão se baseia no escopo de controle ou na responsabilidade gerencial de cada função deliberativa. As arquiteturas híbridas do tipo gerenciais são organizadas de forma similar ao gerenciamento de uma empresa. No topo estão os agentes que fazem o planejamento de alto nível e então passam o plano aos subordinados. Estes refinam o plano e reúnem recursos e passam as funções para o nível mais baixo de trabalhadores, os comportamentos reativos. O agente no nível mais alto pode supervisionar os resultados de seus agentes subordinados no nível mais baixo, podendo lhes dar instruções. Assim como na subsunção, uma camada pode somente modificar a camada inferior a ela. Nos estilos gerenciais, cada camada tenta executar suas tarefas, identificando problemas e corrigindo-os localmente. Somente quando um agente não pode corrigir seu próprio problema, ele pede ajuda a seu agente superior. Nos parágrafos seguintes alguns exemplos de arquiteturas gerenciais serão descritas:

- AuRA, uma das mais antigas e utilizadas, implementada por Arkin, Riseman e Hansen (1987) e
- PiramidNet, uma arquitetura totalmente baseada em RNAs.

Arquitetura AuRA

A Figura 2.14 ilustra a arquitetura AuRA (*Autonomous Robot Architecture*) (ARKIN; RISEMAN; HANSEN, 1987) a qual é baseada na *teoria esquema*[2] e consiste de cinco subsistemas ou módulos, equivalente a classes orientadas a objeto.

Estes subsistemas são descritos na sequência:

- Planejador. Camada deliberativa. É responsável pela missão e planejamento de tarefas e está dividido em três componentes, equivalente à arquitetura NHC:

[2] A teoria de esquema provê uma maneira de modelar a percepção dentro de comportamentos à forma de programação orientada a objeto (ARBIB, 2003). Um esquema consiste do conhecimento de como agir e/ou perceber (conhecimento, estrutura de dados, modelos) bem como o processo computacional usado para cumprir a atividade (algoritmo) (MURPHY, 2000).

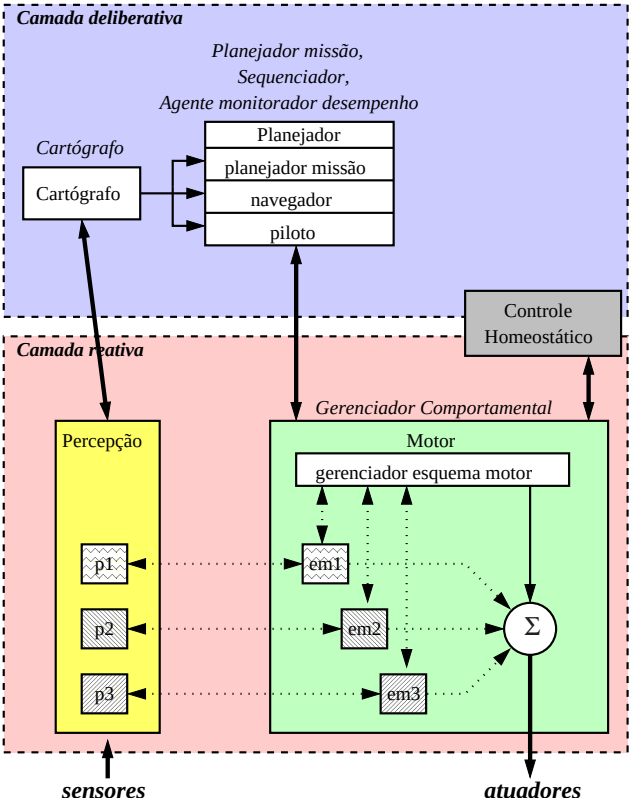


Figura 2.14: Layout da arquitetura AuRA, mostrando os cinco subsistemas. Fonte adaptada de Murphy (2000).

– Planejador de Missão: serve de interface com o operador humano, onde a implementação atual tem uma das interfaces de robôs mais abrangentes e amigáveis.

– Navegador: interage com o Cartógrafo para computar um caminho e dividi-lo em sub-tarefas.

– Piloto: toma a primeira sub-tarefa e obtém informação relevante sobre como gerar comportamentos. A porção Piloto interage com o Gerenciador de Esquema Motor (subsistema Motor), fornecendo a lista de comportamentos necessários ao cumprimento da sub-tarefa corrente.

- Cartógrafo. Camada deliberativa. Encapsula toda a construção de mapa e funções de leitura necessárias à navegação, podendo também se utilizar de um mapa a priori.

- Percepção. Camada reativa. Os subsistemas Percepção e Motor contém bibliotecas de esquemas de percepção (p1, p2, p3) e motores (em1, em2, em3), que formam o esquema comportamento. Os esquemas em si podem consistir de montagens de esquemas primitivos, coordenados por máquinas de estados finitos. Esquemas podem compartilhar informação, se necessário, através de ligações estabelecidas pelo Gerenciador de Esquema Motor. Comportamentos não são estritamente reflexivos, os esquemas podem conter comportamentos específicos a conhecimento, representações e memória. Os esquemas motores entretanto, estão restritos a campos potenciais.

- Motor. Camada reativa. O Gerenciador de Esquema Motor compõe cada comportamento examinando a biblioteca de esquemas de percepção no subsistema Percepção e esquemas motores no subsistema Motor. Os esquemas motores representam ações em campos potenciais e o comportamento global é resultante da soma de vetores.

- Controle Homeostático. Contido em ambas as porções. Situa-se na área cinza entre deliberação e reação. O propósito deste subsistema é modificar o relacionamento entre comportamentos alterando ganhos de uma função "saúde" do robô ou outras restrições. Muitos aspectos da arquitetura AuRA são motivados pela biologia, inclusive o controle homeostático. Ao invés de colocar um módulo na porção deliberativa para explicitamente raciocinar sobre como mudar o comportamento global do robô, a alteração do comportamento resultante está diretamente ligada a uma representação de campos potenciais de comportamentos, devido ao fato da biologia sugerir que animais subconscientemente modificam seus comportamentos em resposta a necessidades internas.

Arquitetura PiramidNet

A arquitetura PiramidNet (ROISENBERG, 2000) é um exemplo de arquitetura baseada em comportamentos e híbrida no estilo gerencial. Ela é composta por RNAs organizadas em estruturas modulares e hierárquicas como ilustra a pirâmide de controle da Figura 2.15. A inspiração biológica desta abordagem se baseia na estrutura hierárquica em camadas que compõe a organização global do cérebro de animais. Sua camada deliberativa envolve

sub-camadas ou subsistemas compostos por RNAs e um Módulo Simbólico no topo da pirâmide. Esta camada é responsável pelos comportamentos motivados e especializados e que também executam a tarefa de seleção e coordenação dos comportamentos das das camadas mais baixas.

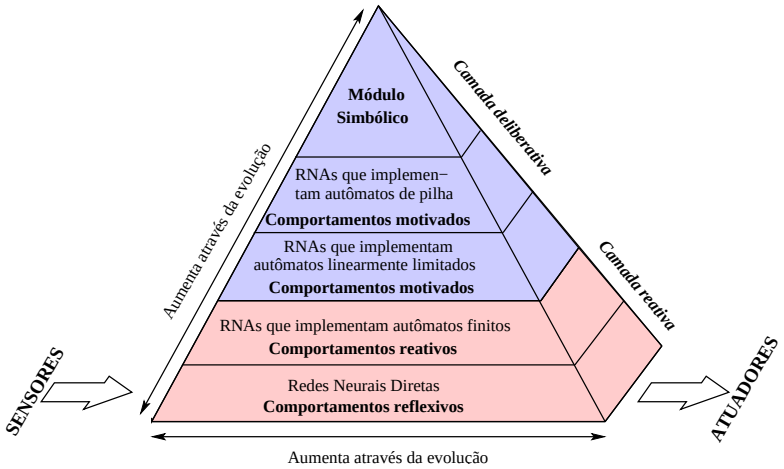


Figura 2.15: Arquitetura PiramidNet.

A camada reativa da arquitetura PiramidNet apresenta sub camadas onde comportamentos reativos e reflexivos são produzidos por RNAs que implementam autômatos finitos e RNAs diretas respectivamente, sendo que estas últimas atuam na base da pirâmide.

Em (ROISENBERG et al., 2004) apenas a camada reativa da arquitetura PiramidNet foi implementada, onde redes neurais recorrentes atuam como autômatos de estados finitos. A arquitetura possibilitou a um robô operar em um ciclo de comportamentos como seguir paredes e buscar energia, onde redes recorrentes controlam quando um comportamento ou outro deve ser executado, além de controlar a camada mais baixa (nível reflexivo).

Já em (VIEIRA, 2004) foi acrescentada à arquitetura PiramidNet mais dois níveis de controle que correspondem à camada deliberativa (Figura 2.15). Com a adição destas sub camadas, RNAs que implementam tanto autômatos de pilha quanto autômatos linearmente limitados produziram o comportamento de ir e voltar pelo mesmo caminho e ao mesmo tempo controlar os comportamentos produzidos pela camada reativa da arquitetura.

Trabalhos futuros no desenvolvimento da arquitetura PiramidNet es-

tão direcionados para a evolução do projeto de RNAs, tanto na altura quanto na largura das camadas da pirâmide, bem como desenvolver ainda o Módulo Simbólico para implementação de comportamentos de navegação mais especializados.

2.4.2 Arquiteturas de Hierarquia de Estados

Arquiteturas de hierarquia de estados usam o conhecimento do estado do robô para distinguir as atividades reativas e deliberativas. Elas também organizam as atividades através do escopo de tempo compreendido em `Presente`, `Passado`, e `Futuro`.

Os comportamentos reativos não possuem nenhum estado e funcionam somente no `Presente`. As funções deliberativas são divididas entre aquelas que requerem conhecimento sobre o estado `Passado` do robô (em que seqüência de comandos ele está executando) e as requerem conhecimento sobre o estado `Futuro` (missão e planejamento de caminho). Desta forma, as arquiteturas de hierarquia de estados possuem 3 camadas. Cada camada cumpre seu objetivo através de seus agentes de softwares ou funções. Assim como nas arquiteturas gerenciais, uma camada mais alta tem acesso às saídas das camadas inferiores e pode operar sobre a próxima camada mais baixa. Um exemplo de arquitetura deste estilo é a 3T que será apresentada a seguir.

Arquitetura 3T

A arquitetura 3T, usada pela NASA, é um exemplo de arquitetura no estilo hierarquia de estados, mostrada na Figura 2.16. Como o nome sugere, 3T se divide em 3 camadas, reativa, deliberativa e a intermediária que serve de interface entre as duas. A arquitetura 3T foi usada primeiramente para construção de robôs excursionistas planetários, veículos subaquáticos e robôs assistentes para astronautas. Na camada superior o Planejador realiza as responsabilidades do planejador de missão e cartógrafo configurando os objetivos e planos estratégicos. Os objetivos são passados à camada intermediária, chamada de Sequenciador.

O Sequenciador usa uma técnica de planejamento reativo chamada RAPs [*reactive-action packages* (FIRBY; PROKOPWICZ; SWAIN, 1995)], que seleciona um conjunto de comportamentos primitivos a partir de uma biblioteca e desenvolve uma rede de tarefas especificando a seqüência de execução dos comportamentos para o sub-objetivo particular. O Sequenciador é responsável pela seqüência e por funções de monitoramento de desempenho em uma arquitetura híbrida genérica.

A camada Sequenciador cria instâncias de um conjunto de comportamentos (chamadas habilidades) para executar o plano. Estes comportamentos formam a camada base, chamada Controlador ou Gerenciador de Habilidade. Estes comportamentos tem o mesmo escopo amplo das arquiteturas AuRA e SFX, que permitem fusão sensorial e montagens de comportamentos primitivos. O termo "habilidade" foi usado para distinguir seus comportamentos das conotações de comportamentos popularizados pela arquitetura de subsunção. Uma habilidade é uma montagem de habilidades primitivas. Um dos aspectos interessantes da arquitetura 3T é sua fundamentação como uma ferramenta para aprendizagem de montagens.

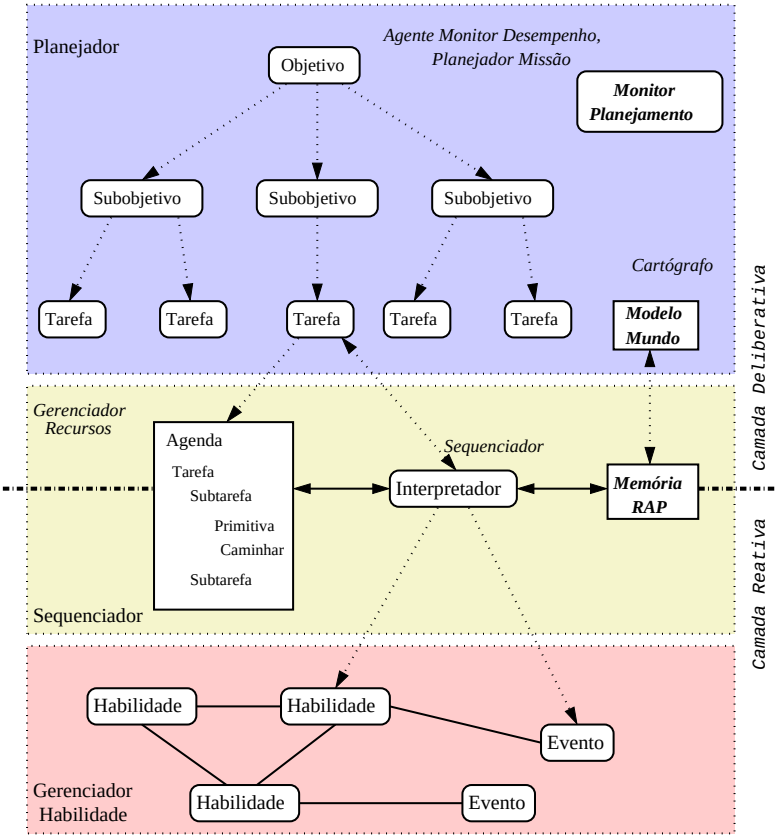


Figura 2.16: A arquitetura 3T. Fonte adaptada de Murphy (2000).

Um poderoso atributo do nível mais baixo é que as habilidades têm *eventos* associados que servem como pontos de checagem para verificar explicitamente o que uma ação fez de correto. A camada Gerenciador de Habilidade é composta por habilidades que operam somente no `Presente` (embora com algumas concessões para permitir a persistência de alguns comportamentos quando os estímulos temporariamente desaparecem).

Os componentes na camada Sequenciador operam no estado de informação refletindo memórias sobre o `Passado`, bem como o `Presente`. Seqüências de comportamentos podem ser gerenciadas pela lembrança do que o robô fez e se ele obteve sucesso ou não. Isto adiciona um grande tratamento de robustez e suporta monitoração de desempenho. A camada Planejador trabalha com informação de estado predizendo o `Futuro`. Ela pode também usar informação do `Passado` (o que o robô fez ou tentou fazer) e `Presente` (o que o robô está fazendo agora). Para planejar uma missão, o planejador necessita projetar como o ambiente estará e outros fatores. Na prática, 3T não organiza estritamente suas funções em camadas por estados (`Passado`, `Presente` e `Futuro`), ao invés disto ela freqüentemente usa o critério de *taxa de atualização*. Por exemplo, algoritmos de atualização lenta são colocados no Planejador, enquanto algoritmos rápidos vão para a camada Gerenciador de Habilidade.

2.4.3 Arquiteturas Orientadas a Modelo

As arquiteturas orientadas a modelo são caracterizadas por comportamentos que acessam porções de um modelo de mundo, os quais muitas vezes se assemelham ao modelo monolítico de mundo global, presente nas arquiteturas hierárquicas.

As arquiteturas de ambos os estilos gerencial e hierarquia de estados se desenvolveram como arquiteturas reativas, com a adição de funcionalidades cognitivas. Estes estilos enfatizam comportamentos ou habilidades como blocos básicos de construção. Já as arquiteturas orientadas a modelo são mais simbólicas, concentrando-se na manipulação simbólica do modelo de mundo global. Diferente da maioria das outras arquiteturas híbridas, que criam um modelo de mundo global em paralelo à percepção específica de comportamento, este modelo de mundo global também serve para suprir percepção a comportamentos (ou comportamentos equivalentes). Neste caso, o modelo de mundo global serve como um sensor virtual. Duas das melhores e mais conhecidas arquiteturas orientadas a modelo são a arquitetura *Saphira* e a arquitetura TCA (*Task Control Architecture*), que serão detalhadas a seguir.

Arquitetura Saphira

A arquitetura *Saphira* (KONOLIGE; MYERS, 1998), apresentada na Figura 2.17, é usada numa variedade de robôs dentre eles os descendentes diretos do robô Shakey: Flakey e Erratic. A motivação desta arquitetura se baseia

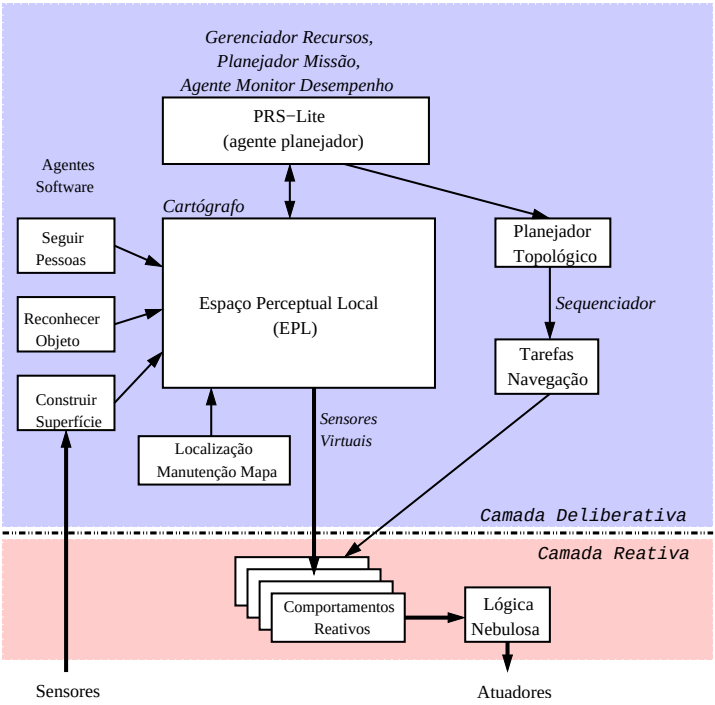


Figura 2.17: Visualização simplificada de arquitetura Saphira. Fonte adaptada de Murphy (2000).

na doutrina das três palavras chaves para o sucesso da operação de um robô no mundo aberto: *coordenação*, *coerência* e *comunicação*. Um robô deve coordenar seus atuadores e sensores (como tratado nas arquiteturas reativas), mas também deve coordenar seus objetivos sobre um período de tempo (não tratado nas arquiteturas reativas). A motivação para coordenação é compatível com reatividade, já a coerência não. Coerência é a habilidade do robô manter modelos de mundo globais, essencial para o bom desempenho comportamental. Comunicação é importante para a interação do robô com seres humanos e outros robôs.

Como pode ser visto na Figura 2.17, a camada deliberativa da arquitetura tem sua maior parte concentrada no planejamento, onde é usado um tipo de planejador reativo chamado *PRS-lite* (*Procedural Reasoning System-lite*), que consiste de um sistema de raciocínio procedural. O planejador PRS-lite é capaz de receber comandos de voz em linguagem natural e operacionalizá-los em tarefas de navegação e rotinas de reconhecimento perceptivo. Ambos, planejamento e execução, trabalham com o Espaço Perceptivo Local (EPL), que corresponde a um modelo de mundo global. A manutenção de um modelo preciso do mundo baseado nos sensores do robô e a determinação de rótulos simbólicos para as regiões requerem muito processamento. A arquitetura *Saphira* também divide as atividades de deliberação entre agentes de software (módulos Seguir Pessoas, Reconhecer Objeto e Construir Superfície). Isto proporciona um alto grau de flexibilidade, dado que agentes de software são independentes e nem sempre se executam dentro do robô que estão controlando.

Ainda na Figura 2.17, a camada reativa da arquitetura Saphira consiste de Comportamentos Reativos que extraem informações dos *Sensores Virtuais* a partir do modelo de mundo global Espaço Perceptivo Local (EPL). A saída comportamental é resultado de regras nebulosas que, por fusão a partir do módulo de Lógica Nebulosa, passa a ser comandos de velocidade e direção. Os comportamentos são gerenciados pela execução de planos das tarefas de navegação. A combinação de comportamentos realizada pelo módulo de Lógica Nebulosa essencialmente produz os mesmos resultados que uma metodologia de campos potenciais.

Além disto, o módulo Espaço Perceptivo Local pode melhorar a qualidade do comportamento geral do robô, através da suavização de erros dos sensores. Embora este processamento central introduza uma sobrecarga computacional, o aumento na potência de processadores torna o custo computacional desta arquitetura aceitável.

Arquitetura TCA

A arquitetura TCA (SIMMONS et al., 1997) é usada extensivamente por robôs projetados pela NASA, incluindo Ambler, Dante e robôs de serviço, seu leiaute é apresentado na Figura 2.18. Esta arquitetura usa estruturas de percepção dedicadas, tais como grades de evidência, que são modelo de mundo global distribuído. A informação de sensores infiltra-se nos modelos globais. O fluxo de tarefa básico é determinado pela camada de Sincronização de Tarefa (planejador PRODIGY). Esta camada interage com o usuário e determina os objetivos e ordem de execução. Por exemplo, se ao robô são dadas muitas

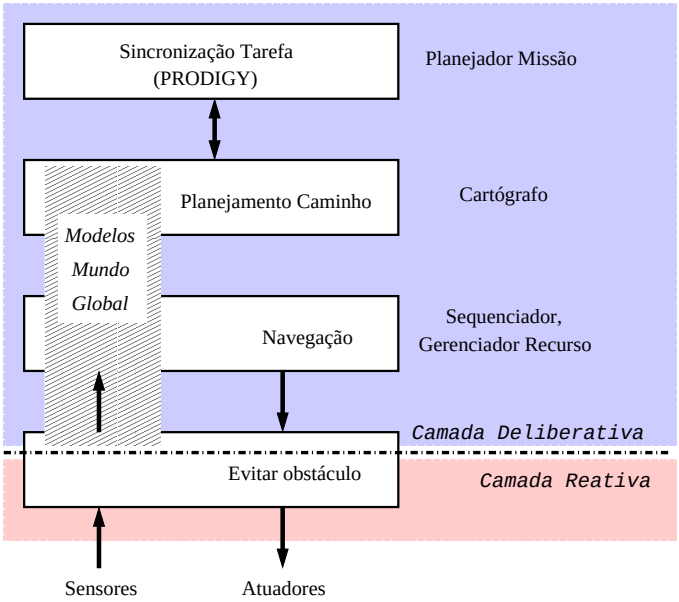


Figura 2.18: Leiaute da arquitetura TCA. Fonte adaptada de Murphy (2000).

tarefas, o planejador pode priorizar e otimizar a programação. Uma vez que a tarefa corrente foi estabelecida, a camada Planejamento de Caminho é acionada. O módulo Navegação determina o que o robô deve procurar, onde ele está e onde ele esteve. Assim como os relacionamentos entre comportamentos estratégicos e táticos na arquitetura SFX, a camada Evitar Obstáculos toma a direção desejada e a adapta aos obstáculos extraídos da grade de evidência do sensor virtual.

2.5 Conclusão

As arquiteturas hierárquicas são baseadas na abordagem (***sentir***, ***planejar***, ***agir***). A informação sensória era incorporada em mapas, que eram usados para planejar trajetórias a serem executadas pelo robô. Uma das principais limitações desta abordagem resultava do fato de que sensores forneciam informações parciais e imprecisas do mundo real. Logo, os mapas produzidos eram imprecisos o que levava a decisões de movimento também imprecisas. Um robô usando este estilo de arquitetura se movia muito lentamente devido

à demanda de seu planejamento.

Como a maneira de representar mapas junto ao acúmulo de imprecisão eram as principais causas do mal funcionamento de robôs baseados em arquiteturas hierárquicas, uma solução proposta através das arquiteturas reativas foi a não utilização de modelos de mundo. Ao invés de mapas, arquiteturas reativas utilizavam diretamente informação sensória para produzir ações, com a justificativa de que "o melhor modelo do mundo real é o próprio mundo". Desta forma, o movimento do robô era a consequência de uma reação aos dados dos sensores, ao invés de ser uma ação planejada. Para um sistema reativo o conceito de comportamento incorpora a unidade básica de controle, no contexto de ações motoras e assim, comportamentos mais complexos podem ser produzidos a partir da junção de comportamentos mais simples. Com arquiteturas puramente reativas, os robôs começaram a se mover rapidamente e de maneira segura mesmo em ambientes complexos e com obstáculos em movimento.

Contudo, a não utilização de mapas pelas arquiteturas reativas também limitava o alcance de tarefas dos robôs e assim pesquisadores buscaram unir as vantagens dos dois estilos anteriores com o desenvolvimento de arquiteturas híbridas deliberativa/reativa. Neste estilo, o nível deliberativo não é tão "rígido" como os planejadores da abordagem hierárquica, nas arquiteturas híbridas a deliberação é responsável por decisões de longo termo, enquanto o nível reativo é responsável pela execução de comportamentos motores. Entretanto, a coexistência de deliberação e reação não é um problema trivial e depende da aplicação. Uma solução encontrada pela maioria dos pesquisadores foi produzir camadas deliberativas que permitissem à camada reativa a liberdade suficiente para controlar o robô em tempo de operação.

Portanto uma arquitetura desenvolvida segundo o paradigma de controle híbrido demonstra ser mais efetiva no desenvolvimento de robôs aplicados a tarefas mais complexas. Por esta razão a arquitetura **NeuroCog** proposta neste trabalho segue este estilo arquitetural. O próximo capítulo apresenta também diferentes estratégias para a construção de modelos de navegação autônoma, através de uma hierarquia que vai de arquiteturas puramente reativas até arquiteturas híbridas, com enfoque na tentativa de imitar a flexibilidade, adaptabilidade e eficiência dos sistemas de navegação biológicos.

CAPÍTULO 3

Estratégias de Navegação

3.1 Introdução

No capítulo anterior foram apresentadas as diferentes maneiras de se organizar o controle inteligente de robôs móveis autônomos. Neste capítulo serão vistas as estratégias mais utilizadas na literatura, para abordagem do problema da navegação, sob o ponto de vista da navegação biológica.

Certos animais possuem de forma inata capacidade de navegação em diferentes tipos de ambientes, sejam eles estruturados, não estruturados, conhecidos ou não. Eles percorrem caminhos nem sempre ótimos no sentido matemático, selecionando-os com rapidez e os comportamentos resultantes são flexíveis e adaptativos (TRULLIER et al., 1997). Segundo LeBouthillier (1999), os primeiros robôs móveis autônomos surgiram na tentativa de imitar processos biológicos e desde então se constata que este é um caminho promissor na direção de construir robôs mais eficientes, incentivando pesquisadores a elaborar sistemas que tentam imitá-los.

A habilidade de navegação é uma característica bastante complexa para robôs móveis e tem merecido um grande interesse da comunidade científica nos últimos anos, p. ex. (OVERHOLT; HUDAS; CHEOK, 2005; HAFNER, 2005; MESBAHI et al., 2004; ARLEO; SMERALDI; GERSTNER, 2004; GUIVANT et al., 2004; MARQUES; LIMA, 2004). Ela envolve elementos como percepção, ação, planejamento, arquiteturas, hardwares, eficiência computacional e capacidade de resolver problemas. Localização, construção de mapas, exploração e planejamento de caminho são os principais assuntos de interesse na área de navegação de robôs móveis.

Este capítulo apresenta uma classificação de tipos de estratégias de navegação inspiradas biologicamente, que também conduziram a criação da

estratégia de navegação final proposta neste trabalho de tese. Outro objetivo é expor uma análise das vantagens de cada estratégia e apresentar uma maneira de combiná-las entre si e selecioná-las em diferentes situações. Um maior destaque é dado às estratégias que produzem comportamentos mais complexos, tais como as abordagens topológicas e métricas.

3.2 Navegação: Conceitos Básicos

A palavra navegação foi definida por Levitt e Lawton (1990) como um processo visando responder a três questões: i) ***"onde estou?"***; ii) ***"onde estão os outros lugares com relação a mim?"*** e iii) ***"como ir a outros lugares a partir daqui?"*** Esta definição, aplicada à robótica, significa usar as entradas dos sensores do robô para atualizar uma única representação global do ambiente, para que ações sejam derivadas por meio de um procedimento de inferência. Esta proposição é adotada por vários autores e forma a base de muitos sistemas de navegação de robôs móveis autônomos (FRANZ; MALLOT, 2000). A questão i) envolve o problema da localização, ou seja, o robô precisa saber onde está. A questão ii) envolve a construção de mapas e iii) envolve o problema de planejamento de caminho (MURPHY, 2000).

Esta noção clássica da navegação é muito restritiva visto que a navegação é possível sem o processamento de uma representação espacial global e sem planejamento. Por exemplo, uma capacidade mínima de navegação seria mover-se no espaço e determinar se o objetivo foi ou não encontrado. Isto implica que as características sensoriais que identificam o objetivo devem estar armazenadas em alguma forma de memória, não sendo necessário o reconhecimento da localização corrente para encontrar o objetivo.

A referência a uma localização objetivo distingue a navegação de outras formas de comportamento espacial, tais como exploração, busca por alimento, evitar obstáculos, orientação de corpo ou estabilização de curso. Por exemplo, taxia[1] nem sempre é um mecanismo de navegação. Ela se refere a uma orientação do corpo numa direção relacionada a um campo de estímulo. Desta forma, taxia está envolvida na maioria dos comportamentos de navegação (junto com evitar obstáculos e estabilização de curso), mas ela não necessariamente inclui as capacidades essenciais de locomoção e reconhecimento de objetivo.

Segundo Prescott (1994) comportamentos espaciais são divididos em dois grupos: *navegação local* e *descobrimento de caminho*. *Navegação lo-*

[1] O termo taxia se refere ao movimento de um animal direcionado, negativa ou positivamente, por um estímulo. Exemplos de tais atividades direcionadas a estímulo são quimio-taxia (taxia química), geo-taxia (gravidade), foto-taxia (luz) e fono-taxia (auditivo).

cal é o processo de movimentar-se ao redor em um ambiente imediato, i.e. um ambiente no qual somente os objetos dentro de alcance de percepção do agente são úteis. Desta forma, não é necessário nenhum tipo de representação de objetos ou lugares fora do ambiente imediato. O agente escolhe sua ação baseado em sua informação sensorial corrente. *Descobrimento de caminho* é o processo de se mover em um ambiente em grande escala, i.e., em um ambiente onde existam pistas relevantes fora do alcance de percepção do agente e o objetivo se encontra fora do ambiente imediato. *Descobrimento de caminho* pode ocorrer com ou sem planejamento e pode levar a uma trajetória feita somente por porções de caminhos conhecidos, ou a uma trajetória contendo novos caminhos. O primeiro tipo de trajetória pode ser deduzido a partir de uma representação *topológica* do ambiente, enquanto que o segundo tipo pode ser planejado somente a partir de uma representação *métrica* do ambiente.

A partir desses conceitos consideram-se ao menos quatro tipos gerais de navegação que permitem sucessivamente comportamentos mais complexos: orientação - limitada ao ambiente local (navegação local); resposta ao disparo de reconhecimento - limitada ao descobrimento de caminho sem planejamento; navegação topológica - limitada ao uso de caminhos conhecidos; e navegação métrica (TRULLIER et al., 1997).

Cada tipo de navegação requer certa informação espacial mínima do ambiente, não necessariamente originada de um *mapa cognitivo*[2]. A seguir é apresentada uma hierarquia de competências de navegação, conforme Trullier et al. (1997) e Franz e Mallot (2000).

3.3 A Hierarquia da Navegação

A hierarquia de navegação está baseada no esquema de classificação de Trullier et al. (1997), que foi modificada e estendida por Franz e Mallot (2000). Nesta hierarquia, comportamentos de navegação são classificados de acordo com a complexidade da tarefa a ser realizada, conforme pode ser visto na Tabela 3.1.

Cada nível é caracterizado por uma certa competência de navegação. Comportamentos de *navegação local* são divididos em quatro níveis: **busca, seguimento de direção, apontamento** e **orientação**; comportamentos de

[2]O conceito de mapa cognitivo foi introduzido por Tolman (1932) como uma maneira de interpretar descobertas comportamentais em ratos, onde seleção de caminho não era só mera resposta reflexiva a estímulos, mas que eles usam alguma forma de representação espacial interna. A partir deste primeiro trabalho, muitos estudos referentes à navegação têm persistido em usar o termo mapa cognitivo de forma imprecisa como uma metáfora para descrever quaisquer mecanismos usados por animais para navegar (TRULLIER et al., 1997).

(a) Comportamentos de *navegação local*.

	Pré-requisito comportamental	Competência de navegação
Busca	Reconhecimento do objetivo	Encontrar o objetivo sem uma orientação efetiva do mesmo
Seguimento de direção	Alinhamento de curso com a direção local	Encontrar o objetivo a partir de uma direção
Apontamento	Manter o objetivo à frente	Encontrar um objetivo saliente a partir de uma área de alcance
Orientação	Obter a relação espacial dos objetos ao redor	Encontrar um objetivo definido por sua relação com os arredores

(b) Comportamentos de *descobrimento de caminho*.

	Pré-requisito comportamental	Competência de navegação
Resposta ao disparo de reconhecimento	Associação sensorial padrão-ação	Seguir rotas fixas
Navegação topológica	Integração de rota, planejamento de rota	Concatenação flexível de segmentos de rotas
Navegação métrica	Embutido em uma estrutura de referência comum	Encontrar caminhos em um novo território

Tabela 3.1: A hierarquia de navegação. Os comportamentos de navegação são classificados de acordo com competências de navegação. Fonte Adaptada de (FRANZ; MALLOT, 2000).

descobrimento de caminho são divididos em três níveis: **resposta ao disparo de reconhecimento, navegação topológica** e **navegação métrica**. Um agente em um determinado nível pode apresentar todas as capacidades dos níveis mais baixos de seu respectivo grupo, mas um agente de "descobrimento de caminho" não necessariamente tem todas as habilidades de "navegação local".

Segundo a classificação realizada através dos paradigmas de controle inteligente, pode-se considerar que as estratégias do primeiro grupo, correspondente à navegação local, configuram os sistemas de navegação puramente reativos. Com relação ao segundo grupo, referente aos comportamentos de descobrimento de caminho, a estratégia de resposta ao disparo de reconhecimento caracteriza também um sistema reativo. Já as duas últimas estratégias de navegação, topológica e métrica, configurarão sistemas deliberativos ou sistemas híbridos quando além destes comportamentos, também forem incorporadas uma ou mais estratégias reativas dos níveis abaixo. A seguir cada estratégia de navegação é descrita com mais detalhes.

3.3.1 Busca

Como pode ser visto na Figura 3.1, um agente que navega apenas com a estratégia de busca não mostra nenhuma orientação efetiva em direção ao objetivo. O objetivo pode ser encontrado somente pela possibilidade do agente atingi-lo enquanto se move. Esta é a forma de navegação mais

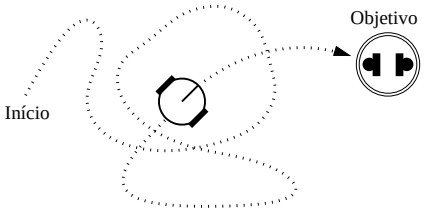


Figura 3.1: Estratégia de busca: comportamentos da navegação local onde o agente ao navegar não possui nenhuma orientação efetiva com relação ao objetivo. Fonte adaptada de (FRANZ; MALLOT, 2000).

simples e requer somente competências básicas de locomoção e detecção de objetivo, não necessitando de nenhum tipo de representação espacial. Alcançar um objetivo através de navegação por busca requer maior tempo, quando comparada a outros métodos que serão vistos nas seções seguintes. Devido à simplicidade desta estratégia, onde a direção do objetivo não necessita ser conhecida, navegação por busca torna-se uma estratégia de reserva quando outras estratégias de navegação falham ao encontrar o objetivo.

3.3.2 Seguimento de Direção e Integração de Caminho

Neste tipo de comportamento de navegação, como ilustrado na Figura 3.2, o agente deve ser hábil para alinhar seu curso com uma direção localmente disponível para encontrar o objetivo. O objetivo por si mesmo não necessita ser perceptível durante a abordagem. Um exemplo deste comportamento seria um navio ajustando seu curso mediante uma direção fixa de compasso que leva ao objetivo. A informação de direção pode ser extraída de origens *alotéticas* (baseada em referências externas) tais como campo magnético, corpos celestiais ou rastro de odor, ou a partir de origens *idiotéticas* (baseadas em uma referência interna) tais como um compasso inercial ou sinais proprioceptivos. Enquanto seguimento de direção é mais efetivo do que busca, esta estratégia de navegação leva o agente a encontrar o objetivo somente quando ele se move sobre a trilha. Se o agente for deslocado da trilha, ele errará o objetivo. Assim, este método não é muito tolerante com relação à

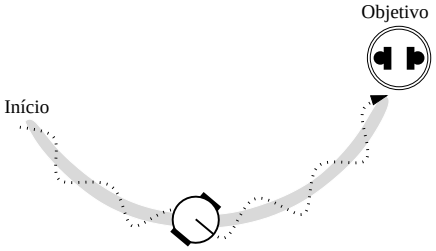


Figura 3.2: Seguimento de direção: comportamento de navegação local que permite ao agente encontrar o objetivo mediante uma trilha. Fonte adaptada de (FRANZ; MALLOT, 2000).

imprecisão da informação direcional e alinhamento. Desvios da trilha podem ser acumulados à medida que o objetivo não seja encontrado.

Se a distância do objetivo for conhecida, seguimento de direção torna-se mais efetivo, pois o agente pode escolher outra estratégia, e.g., busca, caso o objetivo passe despercebido. Novamente, existem várias fontes alo e idiotéticas para informação de distância tal como, número de passos, consumo de energia, fluxo óptico, etc. Uma estratégia para adquirir ambas a direção e a distância do objetivo é chamada *integração de caminho* em biologia, ou *hodometria* em robótica.

3.3.3 Apontamento

Como pode ser visto na Figura 3.3, um agente que utiliza a estratégia de navegação por apontamento, deve orientar o eixo de seu corpo de maneira que o objetivo esteja à sua frente. Para isto, o objetivo deve estar associado a alguma referência que esteja sempre perceptível durante a operação do robô. Em um domínio visual, esta referência é freqüentemente chamada de *beacon*.

Por exemplo, um agente pode navegar apontando sempre para um marco visível, que funciona como um beacon. Ao contrário do seguimento de direção visto anteriormente, o objetivo pode ser alcançado a partir de várias direções sem o perigo de acumular erros. A área onde a referência pode ser percebida é denominada área de alcance do objetivo.

3.3.4 Orientação

A estratégia de navegação por orientação pode ser utilizada quando o objetivo não está sinalizado por uma referência, ou está escondido. Como

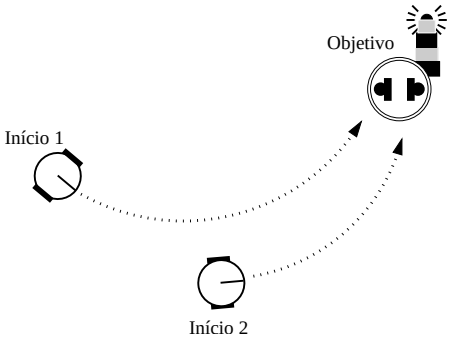


Figura 3.3: Apontamento: comportamento de navegação local que necessita de uma pista perceptível no objetivo. Fonte adaptada de (FRANZ; MALLOT, 2000).

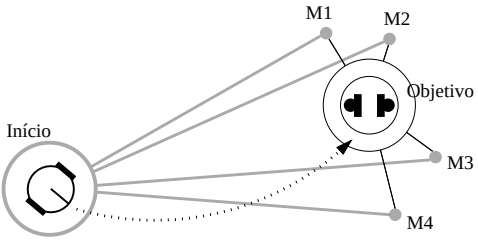


Figura 3.4: Orientação: comportamento de navegação local que habilita um agente a encontrar um objetivo definido pelo relacionamento espacial entre os marcos ao redor do objetivo. Fonte adaptada de (FRANZ; MALLOT, 2000).

pode ser visto na Figura 3.4, através do comportamento de orientação, o agente é guiado pela configuração espacial dos objetos ao redor, definidos como marcos, no caso, M1, M2, M3 e M4 na figura. Em um outro momento, o agente pode tentar se mover de forma a replicar o relacionamento entre os marcos aprendido, visando atingir novamente o Objetivo. Desta forma, a informação espacial requerida não é somente uma única direção ou orientação, mas o relacionamento espacial entre a localização corrente, o objetivo e o ambiente perceptível. A estratégia de orientação é um método de navegação local pois requer somente o processamento da informação sensorial corrente ou interna do agente. Além disto, esta estratégia não necessita de uma representação do ambiente que está fora do horizonte sensorial corrente do agente. Um outro exemplo de utilização da estratégia de orientação é quando um

agente tenta alcançar uma posição fixa entre várias marcos observados.

Portanto uma estratégia de navegação do tipo orientação aplica-se todas as vezes que for necessário ao agente otimizar algum critério relacionado a sensor, baseado na informação sensória armazenada, para alcançar seu objetivo. Ao contrário disto, os próximos níveis de estratégias de navegação, como os descritos a seguir, irão requerer algum tipo de representação espacial.

3.3.5 Resposta ao Disparo de Reconhecimento

Todos os métodos de navegação anteriores são locais pois eles conduzem a uma única localização com a ajuda de informação disponível localmente e desde que o objetivo, um marco, ou uma configuração de marcos esteja ao alcance perceptível do agente. As estratégias de navegação seguintes envolvem ambientes de grande escala e exigem a noção de ***lugar***. Ao contrário de orientação, onde lugares são pontos, um lugar em um ambiente de grande escala é definido como um conjunto de localizações adjacentes que são equivalentes quanto à seleção de ação. Em outras palavras, o agente seleciona a mesma ação locomotora dentro de um dado lugar.

Um lugar também pode ser definido como o conjunto de localizações a partir do qual um conjunto de marcos é percebido de uma maneira idêntica ou muito similar. Se o conjunto de localizações é idêntico de um ponto de vista sensorial (a partir do ponto de vista do agente), então o mesmo movimento será selecionado. A identificação de um lugar deve ser independente do ângulo de visão, i. e., da orientação do agente.

Neste tipo de navegação, **resposta ao disparo de reconhecimento**, a estratégia utilizada pelo agente para alcançar um objetivo conhecido mas não correntemente visível é feita em três fases: (a) o agente deve reconhecer o local em que está situado; (b) em seguida ele deve orientar-se dentro deste lugar; (c) e depois selecionar a direção a ser tomada para alcançar o objetivo atual. Não há planejamento de uma seqüência de movimentos subseqüentes, somente a seleção da ação mais próxima. Assim, o agente responde de uma maneira inflexível a cada situação (Figura 3.5).

Com base em explorações anteriores, o lugar atual pode estar associado com a direção memorizada do objetivo a partir deste lugar. A seleção da direção pode ser feita de duas maneiras diferentes. Primeira, esta direção pode ser definida por uma imagem armazenada. Assim, o agente terá que se rotacionar de maneira a combinar a imagem corrente com a imagem armazenada, que define a direção para o objetivo. A estratégia orientação irá re-orientar o agente na direção do objetivo atual. De outra maneira, a dire-

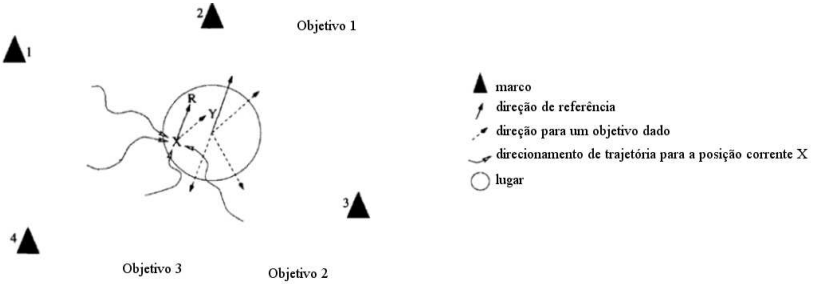


Figura 3.5: Estratégia resposta ao disparo de reconhecimento: o agente deve reconhecer sua posição e sua orientação dentro de um "lugar" (representado pelo círculo), antes da seleção da próxima direção do movimento. A configuração de marcos pode dar informação posicional, i.e. em qual lugar o agente está, sem levar em consideração a exata localização X dentro deste lugar. A configuração de marcos também pode definir um local e arbitrariamente selecionar a direção de referência (ex. R), a qual dá informação direcional. O movimento subseqüente é selecionado a partir do relacionamento entre a direção de referência R e a direção do objetivo atualmente perseguido (Y para o objetivo 1). Fonte adaptada de (TRULLIER et al., 1997).

ção memorizada para o objetivo pode ser definida com relação a uma direção de referência local e arbitrariamente selecionada. Esta direção de referência pode ser determinada pela configuração espacial de marcos. Então, o agente terá que se rotacionar até certo ângulo dado, com relação a sua direção de referência.

Tais estratégias de respostas inflexíveis baseadas em configurações de marcos locais conduzirá o agente ao próximo lugar, onde a mesma estratégia deve ser aplicada. Com o reconhecimento de qual movimento tomar em cada lugar, o agente então é hábil para alcançar o objetivo por uma série de trajetórias sucessivas.

Neste nível, o agente identifica lugares com base em marcos locais e produz uma série de respostas fixas (Figura 3.6). Mas ele não tem nenhuma representação interna das relações entre os lugares correntes e outros lugares no ambiente. O agente ainda não pode, a partir da informação disponível neste nível, planejar seu caminho, i. e. representar a trajetória completa a partir do lugar corrente a um objetivo distante. A cada passo, o conhecimento é limitado à próxima ação a executar. Se a execução da ação selecionada leva a um lugar incorreto, por causa de obstáculos, por exemplo, o agente não será hábil para alcançar o objetivo a menos que ele possa vagar, reconhecer outro

lugar, e usar a mesma estratégia de resposta local.

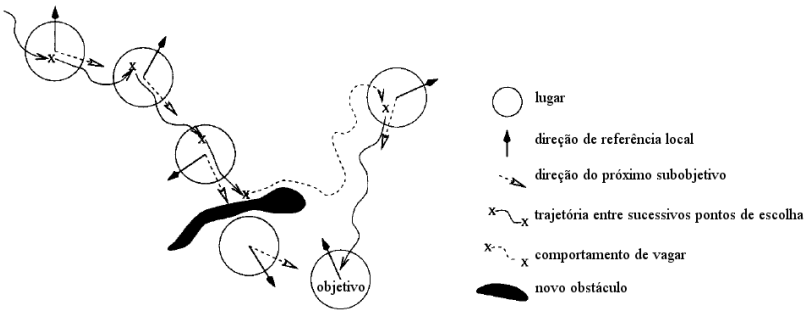


Figura 3.6: Exemplo de funcionamento. A estratégia resposta ao disparo de reconhecimento habilita o agente a se mover de um lugar para outro se ele conhecer a direção a seguir a partir de cada lugar. Quando ele se perde por causa de um obstáculo, por exemplo, ele deve vagar até chegar a um lugar conhecido novamente. Fonte adaptada de (TRULLIER et al., 1997).

3.3.6 Navegação Topológica

Quando um agente usa a estratégia de navegação **resposta ao disparo de reconhecimento**, há uma maneira dele alcançar lugares distantes sem se perder (devido a erros de movimentos ou obstáculos). Ela consiste em antecipar os marcos subseqüentes, i.e., predizer qual será o próximo lugar a chegar. Segundo O'Keefe et al. (O'KEEFE; NADEL, 1978), este é um tipo de associação estímulo-resposta-estímulo (E-R-E) e uma seqüência de tais associações é chamada de uma rota. Ser hábil para predizer o próximo marco e a próxima ação, a partir de marcos correntes, também pode ser considerado como um modelo do mundo.

Entretanto, rotas são independentes uma das outras e cada rota leva a um único objetivo. Elas são inflexíveis porque não se considera o fato de que duas diferentes rotas podem passar pelos mesmos lugares. Neste sentido, a representação de uma rota é uma simples extensão do que é usado pela estratégia de navegação resposta ao disparo de reconhecimento, i. e. associações lugar-objetivo-ação-lugar ao invés de associações lugar-objetivo-ação. A navegação seria mais adaptativa se a representação espacial fosse independente de objetivo, i. e. se a mesma representação espacial pudesse ser usada para múltiplos objetivos. Tal resultado pode ser obtido combinando associações lugar-ação-lugar derivadas da coleção de rotas (retirando a representação de

objetivo) em uma representação topológica do ambiente. Desta forma, qualquer lugar pode então tornar-se a origem ou o objetivo de um caminho e, no caso de obstáculos, caminhos alternativos podem ser tomados nas intersecções.

Neste tipo de navegação topológica, o agente pode seguir rotas ou subseqüências de rotas. Em qualquer caso, o agente percorre somente os (sub)caminhos já visitados. Uma representação topológica pode ser expressa em termos matemáticos como um grafo, onde os nós representam lugares e arestas representam adjacências, ou conectividade direta. Então, dois nós estão ligados se existe um caminho direto visitado anteriormente o qual leva de um dado lugar a outro, sem ir por um terceiro lugar intermediário conhecido.

Neste tipo de navegação, a estratégia do agente para alcançar um objetivo conhecido mas não correntemente visível é feita em quatro etapas (Figura 3.7): (a) reconhecimento do lugar onde o agente está atualmente situado; (b) localização do nó correspondente no grafo topológico; (c) buscar a seqüência de nós (lugares) que levará ao objetivo; (d) a rota resultante é uma concatenação de segmentos de rotas já percorridos, e o agente pode segui-la mesmo sem ter seguido esta seqüência particular antes.

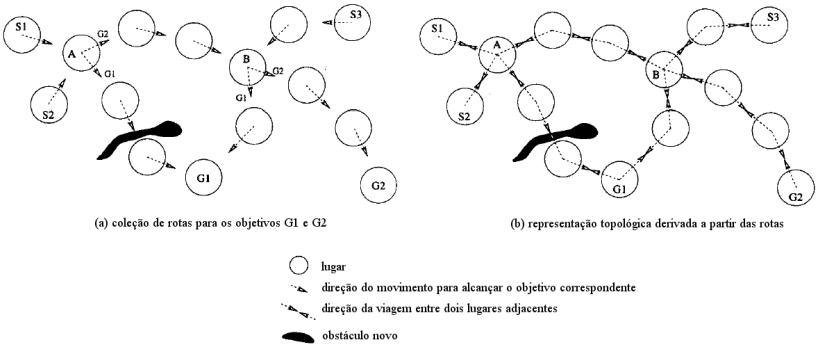


Figura 3.7: (a) Com a estratégia de resposta ao disparo de reconhecimento pode existir um grupo de intersecção de rotas. O agente é hábil para ir de S1 a G1, de S2 a G2, e de S3 a G1. Entretanto, se existir um novo obstáculo no caminho de S1 a G1, como na Figura, o agente se perderá, porque a rota de S1 a G1 é única (veja também Figura 3.6). (b) Ao contrário, se a agente unir suas representações de rotas em uma representação topológica, o agente pode voltar ao lugar A, tomar as sub-rotas entre os lugares A e B, e tomar a sub-rota do lugar B ao objetivo G1. O caminho resultante é a concatenação dos três sub-seqüencias, derivadas de três diferentes rotas. Fonte adaptada de (TRULLIER et al., 1997).

Neste nível, tem-se agora uma representação de alguns dos relacionamentos espaciais (topológicos) entre lugares. O principal resultado é que o agente planeja a seqüência completa de lugares a serem visitados. Para seguir esta seqüência, o agente pode gerar uma série de comandos pela estratégia de resposta ao disparo de reconhecimento especificando todo próximo lugar (nó) como um sub-objetivo.

3.3.7 Navegação Métrica

As estratégias de navegação dos três níveis anteriores possibilitam a um agente a habilidade de movimentar-se com sucesso em direção a localizações de objetivos, embora não necessariamente através de caminhos ótimos (i. e. mais curtos). Porém, a informação correspondente não pode ser usada para computar ou planejar uma nova trajetória ou contornos frente a obstáculos imprevisíveis, ou atalhos. A Figura 3.8 ilustra comportamentos de contornos e atalhos métricos, cujas estratégias de navegação anteriores são incapazes de realizar.

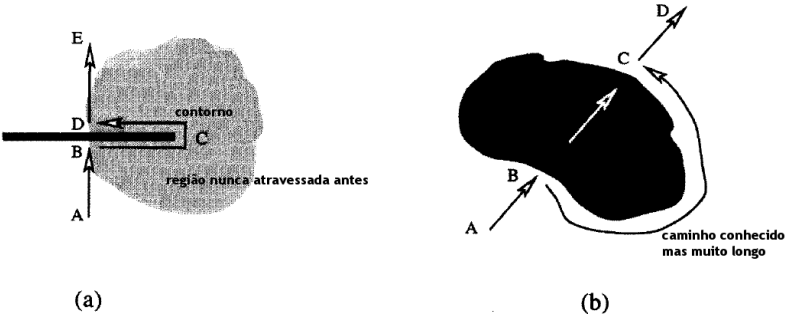


Figura 3.8: Comportamentos de contorno métrico (a) e atalho métrico (b). Em ambos os casos, o agente toma um caminho nunca percorrido antes, sem poder usar marcos conhecidos (supõe-se que a nova parede é alta e a floresta densa). Em (a), o agente poderia, em princípio, ir diretamente de C para E. Isto seria uma ilustração de um atalho métrico. Fonte adaptada de (TRULLIER et al., 1997).

Como um primeiro exemplo, considere que um agente, iniciando no lugar A, repentinamente se confronte com uma nova parede no lugar B, e que tenha planejado continuar do outro lado, chamado de lugar D para o lugar E [Figura 3.8(a)]. Ele deve dar a volta em torno do obstáculo, por exemplo seguindo a parede até chegar atrás dela (lugar C) e então voltar pelo outro

lado. A localização exata do lugar D no outro lado da parede a partir do lugar B não pode ser reconhecida por inspeção visual, porque a parede obstrui os marcos relevantes. O agente necessita saber a distância ao longo da parede - o tamanho do contorno (BCD). Neste caso, há a necessidade de integração de caminho[3]. Não considerando o caso de marcos identificando o lugar E poderem ser percebidos a partir do lugar C, pois desta forma o agente não necessitaria completar o contorno e sim poderia tomar um atalho - usando uma estratégia orientação - de C para E.

Como um segundo exemplo, considere que um agente, iniciando na localização A, necessite ir para o outro lado de uma floresta, localização C, que ele conhece, mas não pode percebê-la a partir do localização em que está, chamada localização B [Figura 3.8(b)]. O agente também conhece um caminho muito longo de B para C, ao redor da região. A partir do conhecimento deste caminho mais longo, ele tenta estimar o rumo correto do percurso por dentro da região para tomar um *atalho*. Neste caso, o agente necessita saber a orientação relativa entre as localizações e, uma vez dentro da região, ele não terá acesso a marcos da localização B ou C. Portanto, o agente necessitaria de uma nova estratégia.

Uma possibilidade para tratar este problema é a introdução de informação *métrica* - distâncias e ângulos entre lugares - em adição à informação topológica vista na seção anterior. Este é um requerimento necessário para geração de novos caminhos que consistem de contornos e atalhos métricos. A questão restante é como esta informação pode ser manipulada a fim de produzir instruções necessárias para seguir caminhos. De um ponto de vista computacional, uma distância e um ângulo com relação a uma base de referência podem ser representados por um vetor e o sistema de navegação deve ser hábil para executar operações com vetores para obter um novo caminho $\overrightarrow{BC}$ a partir dos caminhos $\overrightarrow{AB}$ e $\overrightarrow{AC}$ conhecidos [Figura 3.8(b)]. Desta forma, a seleção do movimento é o resultado de raciocínio dedutivo. De um ponto de vista funcional, o agente poderia usar internamente um mapa métrico do ambiente. Esta abordagem também é chamada de uma estratégia de análise (*survey*).

Neste sentido, considera-se que o agente utiliza uma estratégia de navegação métrica se ele navega com sucesso de um lugar a outro, eventualmente passando por novos lugares, sem levar em consideração as rotas anteriormente seguidas, i.e., realizando contornos ou atalhos métricos.

[3]Integração de caminho é uma habilidade do agente em estimar sua posição atual relativo a um ponto de início conhecido, computando o deslocamento através da integração da velocidade e da direção, i.e. através do uso exclusivo de informação adquirida. Este processo também é conhecido como "*dead-reckoning*"

3.4 Navegação Baseada em Mapa

A navegação baseada em mapa abrange as três últimas estratégias de navegação vistas anteriormente (resposta ao disparo de reconhecimento, navegação topológica e navegação métrica). Ou seja, ela está relacionada a três variedades de conhecimento em nível de cognição espacial: marcos, rotas e análise de conhecimento.

Os primeiros esforços na pesquisa em robótica sobre navegação baseada em mapa foram inspirados por processos cognitivos, principalmente depois de estudos etológicos que supuseram o uso de mapas para navegação por animais, por exemplo, a hipótese do *mapa cognitivo* de Tolman (1932). Tal hipótese ganhou maior suporte com a identificação das células lugares (*places cells*) no cérebro de roedores (O'KEEFE; NADEL, 1978). Estas células lugares são neurônios, encontrados especialmente na parte do cérebro chamada de hipocampo, que tem uma atividade correlacionada com a posição dos ratos no ambiente. Estudos experimentais mostraram que a atividade destas células dependem muito de pistas visuais, entretanto também são sensitivas ao movimento do animal, dado que elas são ativadas no escuro. Este tipo de navegação baseado em mapa é um paradigma muito mais atraente para a navegação baseado em mapa de robôs, à medida que ele não pressuponha processos de cognição de alto nível e seja hábil para trabalhar em ambientes naturais e não modificados.

Basicamente, navegação baseada em mapas invoca três processos:

- **Aprendizado de mapa**: processo de memorizar os dados adquiridos pelo robô durante exploração em uma representação adequada.

- **Localização**: processo de derivação da posição corrente do robô dentro do mapa.

- **Planejamento de caminho**: processo de escolher um curso de ação para alcançar o objetivo, dada uma posição corrente.

O terceiro processo é dependente do primeiro e segundo, ou seja, para planejar ações na direção do objetivo é necessário que haja o mapa do ambiente entre a posição corrente e objetivo. Os dois primeiros processos são inter-dependentes. Esta interdependência torna complexo o problema da localização e aprendizado de mapa simultâneos (problema SLAM, do acrônimo em inglês *Simultaneous Localization And Mapping*). Uma ampla revisão de estratégias de localização, aprendizado de mapa e planejamento de caminho é encontrada nos trabalhos (FILLIAT; MEYER, 2003a; FILLIAT; MEYER, 2003b),

neles são analisados vários métodos de navegação baseada em mapas, dentre eles muitos inspirados biologicamente.

3.5 Conclusão

Este capítulo apresentou uma hierarquia de estratégias de navegação, obtida através da análise de várias abordagens presentes na literatura, referente à navegação biologicamente inspirada com ênfase na teoria dos mapas cognitivos. Esta hierarquia é amplamente aceita pela maioria dos pesquisadores em robótica móvel e propõe uma classificação conforme i) a estrutura e o conteúdo das informações tratadas, ii) o procedimento para seleção de movimento e ii) o repertório comportamental produzido. Um estudo mais detalhado da hierarquia bem como vários exemplos de sistemas de navegação pode ser obtido em (TRULLIER et al., 1997) e (FRANZ; MALLOT, 2000).

Considerando-se os paradigmas de controle inteligente vistos no Capítulo 2, as cinco primeiras competências de navegação (**busca**, **seguimento de direção**, **apontamento**, **orientação** e **resposta ao disparo de reconhecimento**), combinadas ou não, configuram sistemas reativos, enquanto que as duas últimas, navegação **topológica** e **métrica** podem configurar um sistema híbrido deliberativo/reativo quando são incorporadas uma ou mais das estratégias reativas dos níveis anteriores.

Desta maneira, comportamentos de navegação mais complexos poderiam ser obtidos, por exemplo, integrando-se os quatro últimos tipos de estratégia de navegação, **orientação**, **resposta ao gatilho de reconhecimento**, **topológica** e **métrica**, de forma que elas interagissem entre si, ao mesmo tempo permitindo que as estratégias dos níveis mais baixos, em algumas situações, ignorassem as estratégias dos níveis mais altos, pois em certos casos isto se torna mais eficiente. Haja vista que as estratégias de níveis mais baixos são mais simples, porque requerem um mínimo de informação espacial do ambiente, que não necessariamente é originada de um mapa, em especial *mapas cognitivos*.

Um mapeamento cognitivo, no entanto, pode ser implementado em diferentes níveis de abstração e de diversos modos. As abordagens de redes neurais artificiais para mapeamento cognitivo são tão variadas quanto outros tipos de abordagens em contrapartida. Estas abordagens abrangem modelos que usam redes para tentar reproduzir características do cérebro, que são meras abstrações de alguns aspectos de comportamento originados de mapeamento cognitivo biológico (JEFFERIES; YEAP, 1995). Redes neurais artificiais também são ferramentas bastante utilizadas na implementação da percepção, navegação e controle de sistemas robóticos. Uma revisão sobre este tópico

será dada no próximo capítulo.

Portanto, o estudo desta hierarquia de estratégias de navegação, bem como os exemplos de implementações existentes na literatura, direcionou este trabalho de tese para a construção de uma abordagem que contribui para a pesquisa de navegação bioinspirada sob os seguintes aspectos. Primeiro, a união de paradigmas de aprendizado (de RNAs e por reforço) para implementação de comportamentos de navegação. Investigando até que ponto a utilização destas ferramentas permite a implementação de comportamentos mais adaptativos. Segundo, um processo de implementar mapeamento cognitivo visando uma maneira eficiente de permutar entre diferentes comportamentos de navegação, mediante situações não previstas, constitui uma importante contribuição deste trabalho.

CAPÍTULO 4

Redes Neurais Artificiais aplicadas à Navegação de Robôs Móveis

4.1 Introdução

Como visto no capítulo anterior, em essência, todo comportamento de navegação se refere à capacidade de executar um caminho de uma posição atual a uma localização desejada. Robôs móveis contam com três tipos básicos de sensores: táteis, de alcance e visão, todos eles sujeitos a respostas com ruídos. Esta incerteza, somada também à imprecisão dos atuadores e à imprevisibilidade de ambientes reais, agregam dificuldades tanto ao projeto de arquiteturas de controle, quanto ao funcionamento de sistemas de navegação de robôs móveis.

Um importante requisito, amplamente buscado por construtores de robôs, é que a navegação seja hábil para gerar comportamentos orientados a objetivo e que se adapte à complexidade da tarefa. Para isto, faz-se necessário incorporar aos robôs capacidades de aprendizado autônomo e generalização, a fim de que seus sistemas de controle se adaptem, por exemplo, a situações nunca experimentadas. Redes neurais artificiais (RNAs) são uma poderosa ferramenta para modelar comportamentos adaptativos por meio de seus algoritmos de aprendizagem. Suas capacidades de robustez e generalização diante de ruídos permitem a um robô interagir, por exemplo, com o mundo e construir mapeamentos sensório motor apropriados. Na literatura de robôs móveis, são encontrados exemplos de aplicações de RNAs em todos os subproblemas envolvidos na navegação de robôs móveis, tais como localização, construção de mapas e navegação (MILLáN, 2003).

Este capítulo apresenta uma revisão de alguns exemplos de aplicações de RNAs e algoritmos de aprendizado em robótica móvel, presentes na litera-

tura. Para um estudo detalhado sobre diversos tipos de RNAs e suas respectivas técnicas de aprendizado tem-se (HAYKIN, 2001). Para o desenvolvimento deste trabalho de tese, este capítulo direcionou principalmente a construção da camada reativa da arquitetura final **NeuroCog**, uma vez que diferentes arquiteturas de RNAs foram sendo gradualmente incorporadas, juntamente com dois processos de aprendizado em tempo de operação.

4.2 RNAs Aplicadas à Navegação Reativa

Como visto nos Capítulos 2 e 3, em sistemas de navegação planejada, construir e manter mapas globais de ambientes de maneira consistente não é um problema trivial, devido principalmente a ruídos de sensores e imprecisão de atuadores, que podem introduzir e acumular erros nos mapas. Além disto, é necessário uma boa estratégia de exploração para que o ambiente e seus relacionamentos espaciais sejam modelados com um todo. Sendo assim, a maioria das abordagens para construção de mapa conta com o comportamento de seguir paredes, juntamente com o comportamento de evitar colisão, para prevenir que o robô visite áreas abertas ou desordenadas ou venha quebrar colidindo com obstáculos. De outro modo, o robô pode usar comportamentos para alcançar seu objetivo diretamente, sem a utilização de qualquer representação de conhecimento referente a mapa do ambiente, o que significa economia de recursos computacionais e rapidez de execução.

Nas abordagens de navegação reativa (Seção 2.3), dados dos sensores são mapeados diretamente em comandos para motores, sem a construção de representação explícita do ambiente do robô e sem o uso de técnicas de planejamento. Por esta razão, a navegação reativa também é interpretada como uma ação reflexiva resultante a partir da informação dos sensores.

Segundo Kröse e van Dam (1997), sistemas reativos convencionais usam um modelo matemático explícito do sistema para encontrar o mapeamento correto entre entradas sensórias e saídas motoras. Entretanto, tal modelo é difícil e às vezes impossível de desenvolver. Como uma alternativa, RNAs são aplicadas ao controle de sistemas não lineares, por exemplo, devido a suas propriedades de aprender um modelo não linear a partir de exemplos. Uma segunda vantagem é que RNAs são adaptativas, isto é, elas podem manter o aprendizado durante a operação do robô. Isto permite melhoramentos contínuos do controlador enquanto o robô está em operação. Alguns exemplos de RNAs aplicadas à navegação reativa são (CASTELLANO et al., 1996; XU; WANG; HE, 2003; MESBAHI et al., 2004; OVERHOLT; HUDAS; CHEOK, 2005). RNAs para navegação reativa podem ser treinadas de forma supervisionada, auto-supervisionada, com técnicas de aprendizado por reforço e aprendizado

auto-organizável (OMIDVAR; SMAGT, 1997), como serão vistas a seguir.

4.2.1 Aprendizado Supervisionado

Por exemplo, um sistema de navegação, que utiliza uma rede neural com aprendizado supervisionado para controle do robô, corresponde a uma abordagem supervisionada onde a rede neural é treinada de maneira *off-line*. Na fase de treinamento, conjunto de exemplos contendo sinais de controle desejados, para as respectivas entradas de referência, são apresentados à rede neural por meio de um algoritmo de aprendizado.

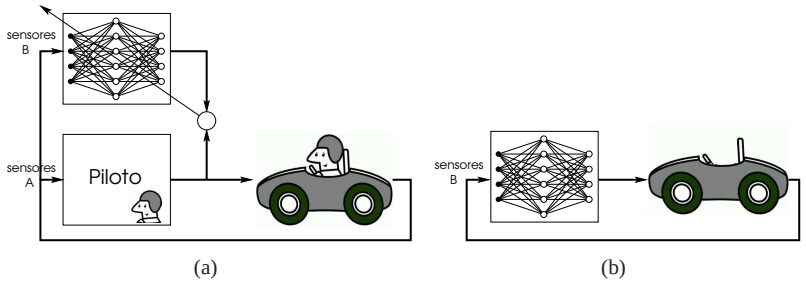


Figura 4.1: Abordagem supervisionada. (a) Fase de treinamento onde exemplos de aprendizagem são gerados através de um piloto controlando o robô veículo. (b) Operação do robô veículo sendo controlado pela rede treinada na etapa anterior. Fonte adaptada de Kröse e van Dam (1997).

Como pode ser visto no exemplo da Figura 4.1, em (a) a rede neural foi treinada com os exemplos coletados a partir da operação do piloto controlando o robô veículo. Todavia, esta rede neural treinada em (a) nem sempre será hábil para encontrar uma correta aproximação do comportamento humano em (b), pois muitas vezes o operador humano pode utilizar outras informações (sensores A) além das que são fornecidas como entrada à rede (sensores B) para gerar o sinal de navegação em (b). Por exemplo, o piloto poderia evitar uma árvore durante o percurso, ora passando pela esquerda e ora pela direita, introduzindo uma ambiguidade ao conjunto de treinamento da rede neural. Mais especificamente, na abordagem supervisionada, redes diretas são treinadas através do algoritmo de aprendizagem por retro-propagação (*backpropagation*), que depende da seqüência e da distribuição dos exemplos de treinamento. Uma outra desvantagem da aprendizagem supervisionada, para uma rede usada como controlador, é que ela não será adaptativa, se alguma coisa mudar no sistema, a rede deverá ser re-treinada para produzir o

desempenho desejado. Isto corresponde a uma das principais limitações das abordagens de navegação que utilizam aprendizado supervisionado.

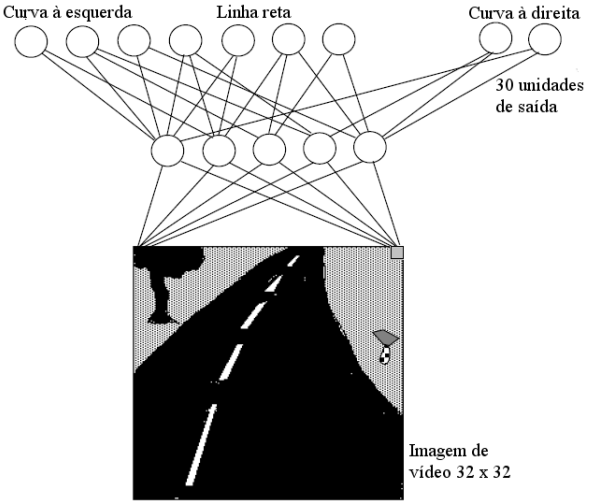


Figura 4.2: A estrutura da rede direta utilizada no robô veículo seguidor de estrada. Kröse e van Dam (1997).

Apesar de tantas objeções, são encontrados na literatura vários exemplos de utilização da abordagem supervisionada no controle de robôs móveis, tais como (POMERLEAU, 1991; NEHMZOW; MCGONIGLE, 1994). Um exemplo interessante de um mapeamento sensor-motor neural é denominado ALVINN (*Autonomous Land Vehicle in a Neural Net*), descrito por Pomerleau (1991). A rede neural controla o robô veículo autônomo Navlab, desenvolvido na Carnegie Mellon University, para uma tarefa de seguir rodovia. Diferentes de outras abordagens de navegação desenvolvidas no mesmo período na universidade, ALVINN não utiliza um modelo da rodovia. A abordagem aprende associações entre padrões visuais e comandos de direção, como ilustrado na Figura 4.2. A rede direta utilizada recebe dados de entrada a partir do sistema sensório, que corresponde a imagens de vídeo de 32 x 32 pixel a partir de uma câmera instalada no topo do veículo. Cada pixel corresponde a uma unidade de entrada da rede neural. A camada de entrada é completamente conectada com a camada intermediária que apresenta 5 unidades. A camada de saída da rede consiste de 30 unidades que correspondem a uma representação linear da direção na qual o robô veículo deve executar a fim de se manter na rodovia. Durante a operação do robô, o controle de direção é executado

considerando-se o centro de massa da "saliência" de ativação circundante da unidade de saída com o nível de ativação mais alto. Primeiramente, a rede neural foi treinada com 1200 exemplos onde foram apresentadas imagens artificiais da rodovia como entrada e os comandos de direção correspondentes, como saídas desejadas. Foi utilizado o algoritmo de retro-propagação (back-propagation) onde o conjunto de aprendizado acima foi apresentado de 30-40 vezes.

Em uma segunda etapa de experimentos, situações reais foram utilizadas para o treinamento. Enquanto o operador humano dirigia o veículo Navlab, o algoritmo backpropagation foi utilizado com a imagem da câmera corrente sendo apresentada como entrada e a direção na qual a pessoa executava como saída desejada. O problema desta abordagem é a pequena quantidade de exemplos para o conjunto aprendizado gerado através da operação do piloto. Para resolver este problema POMERLEAU adicionou imagens "transformadas", nas quais o robô veículo localiza-se a diferentes localizações relativas ao centro da estrada, correspondendo às ações de direção desejadas. Após um percurso de aprendizagem de 5 min, a rede treinada foi hábil para dirigir o Navlab a uma velocidade máxima de 32 km/h. Esta velocidade foi aproximadamente duas vezes mais rápida do que um algoritmo não neural executado no mesmo veículo.

Haja vista as desvantagens apresentadas pela abordagem de aprendizado supervisionado, muitas pesquisas são direcionadas à utilização de algoritmos de aprendizado de RNAs que são realizados em tempo de operação, cujos tipos de funcionamento serão descritos nas próximas seções.

4.2.2 Aprendizado Auto-Supervisionado

Um robô autônomo deve treinar a si mesmo de maneira *on-line* para lidar com exemplos de treinamento ambíguos e incompletos. Segundo Kröse e van Dam (1997), uma maneira é utilizar sistemas de aprendizado auto-supervisionado que podem ser construídos por meio de um modelo de referência disponível. O modelo de referência define o estado desejado do robô, que pode ser determinado a partir de dados de sensores (p. ex., em aplicações de seguimento de parede, seguimento de alvo) ou de informação do mundo (localização desejada ou planejamento de caminho). A rede neural (com conhecimento incompleto) que controla o robô normalmente leva a erros (diferença entre o estado atual e o desejado) que são usados para ajustar seus pesos. Para mapear o erro do robô (expresso no domínio de sensores ou mundo) para um sinal de erro no domínio de controle, treina-se uma outra rede neural para se identificar ao sistema a ser controlado, o que proporciona a criação de um

processo de controle adaptativo.

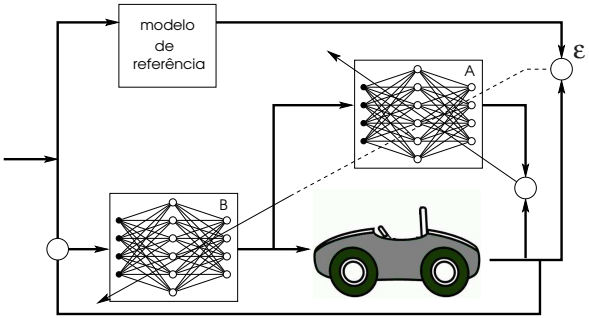


Figura 4.3: Treinamento auto-supervisionado. Fonte adaptada de Kröse e van Dam (1997).

Por exemplo, como ilustrado na Figura 4.3, através do modelo de referência, o erro no domínio de estado pode ser retro-propagado para computar o erro na saída do controlador, que por sua vez pode ser ajustado. Desta forma, o treinamento auto-supervisionado de um modelo de referência é usado para predizer uma saída desejada do sistema. Ou seja, se o modelo (neural) A do sistema for disponível, a rede controladora B pode ser adaptada por retro-propagação do erro ε através do modelo, desta forma o sistema torna-se hábil para adaptar-se a mudanças. O aprendizado desta estratégia de controle assemelha-se a uma solução de aprendizagem por reforço. No caso da aprendizagem auto-supervisionada, um modelo explícito do sistema é usado, enquanto na aprendizagem por reforço uma função de avaliação é estimada. Exemplos de RNAs com aprendizado auto-supervisionado para navegação de robôs móveis são dados em (XU; WANG; HE, 2003; ZHU; YANG, 2003).

4.2.3 Aprendizado por Reforço

Nas seções anteriores foi mostrado como uma rede neural pode ser treinada, a partir exemplos explícitos fornecidos por um supervisor ou por exemplos de aprendizado que podem ser gerados pelo próprio sistema. Entretanto, um conjunto de exemplos de aprendizado nem sempre é disponível ou pode ser derivado e a única informação fornecida é um sinal (correto/incorreto) indicando quão bem a rede neural está executando. Por exemplo, um sinal de recompensa é dado quando um objetivo desejado é alcançado ou um sinal de punição, quando o sistema falha. De fato, este sinal de reforço é menos informativo do que o conjunto de exemplos utilizado no caso super-

visionado, além de ser determinado com atraso. O sinal de sucesso ou falha aplicado em um dado momento é um resultado de uma série de sinais de controle (saídas da rede) dadas no passado. A questão é como treinar uma rede com tão pouca informação.

Diversos algoritmos foram desenvolvidos para fazer esta forma de aprendizado possível. Barto, Sutton e Anderson (1983) formularam a aprendizagem por reforço como uma estratégia de aprendizado que não necessita de um conjunto de exemplos fornecidos por um agente humano. O sistema descrito pelos autores e ilustrado na Figura 4.4, utiliza os diversos mapeamentos de entrada-saída e usa um *feedback* estimado (sinal de reforço, *r*) com relação às conseqüências do sinal de controle (saída da rede) no ambiente. Uma predição de custos associados com o estado corrente pode ser aprendida por uma rede neural "crítica" com base no sinal de reforço *r*. Esta predição é usada para adaptar os pesos da rede de controle.

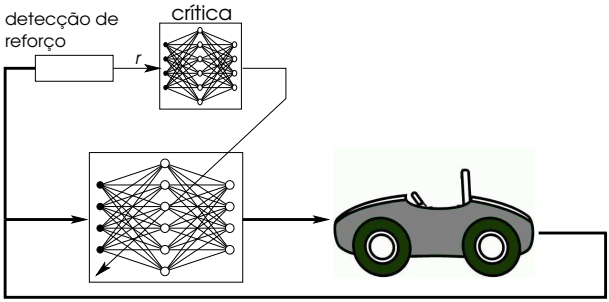


Figura 4.4: Esquema de aprendizagem por reforço. Fonte adaptada de Kröse e van Dam (1997).

Aprendizagem por reforço é usada tanto para aprendizado de navegação livre de colisão, bem como navegação orientada a objetivo. Uma das maiores deficiências desta abordagem é que ela é um processo lento (KRöSE; van Dam, 1997), dado que não é fornecido nenhum conhecimento prévio e extensivo da tarefa, e isto faz com que o sistema necessite de um determinado tempo para melhorar seu desempenho. Existem muitas extensões a partir da idéia básica de aprendizagem por reforço que consideravelmente aumentam a velocidade de convergência dos mapeamentos sensório-motor apropriados (ou políticas, como eles são chamados nos campos de controle e aprendizagem por reforço), fazendo possível a construção de uma aprendizagem prática para robôs móveis. Exemplos do uso de rede neural com aprendizagem por reforço para navegação de robôs são vistos em (SEHAD; TOUZET, 1995;

RYLATT; CZARNECKI; ROUTEN, 1997; ARLEO; SMERALDI; GERSTNER, 2004).

Em ambos os trabalhos (KRöSE; van Dam, 1992b) e (KRöSE; van Dam, 1992a), os autores descreveram um controlador neural para um robô móvel que aprende a navegar sem colisões em um ambiente complexo. O robô é equipado com 8 sensores de proximidade distribuídos em semicírculo em sua frente, mais um sensor de colisão. O robô se mantém a uma velocidade constante e o único feedback é dado somente quando ele colide, o qual corresponde a um sinal de reforço negativo. A entrada do controlador é um vetor de tamanho 8-dimensional e a saída é um sinal de direção binário. Uma representação discreta do espaço de entrada é utilizada. Mas o problema é encontrar uma quantização deste espaço. Uma solução encontrada foi utilizar um tipo de abordagem auto-organizável para quantização do espaço de entrada, como proposta em (ROSEN; GOODWIN; VIDAL, 1990). A solução proposta pelos

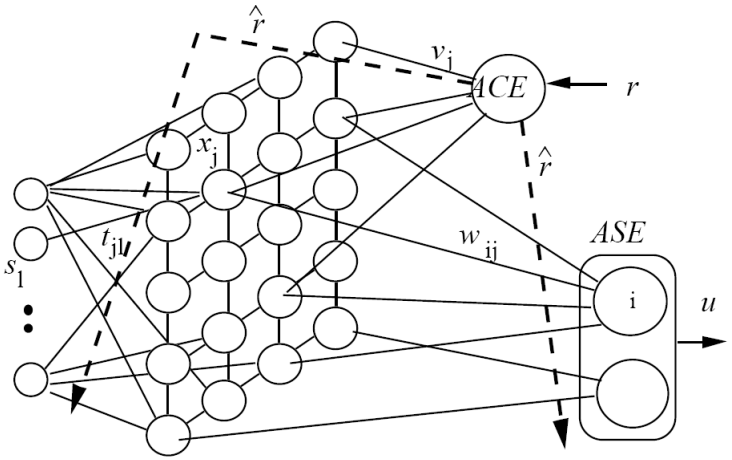


Figura 4.5: Esquema do controlador neural auto-organizável com aprendizado por reforço. Fonte adaptada de Omidvar e Smagt (1997).

autores é ilustrada na Figura 4.5. Esta abordagem apresenta uma quantização que preserva a topologia auto-organizável, introduzida por Kohonen (1988), a fim de representar o espaço de entrada (KRöSE; van Dam, 1992b). O controlador neural auto-organizável com aprendizado por reforço produz a ação ou sinal de direção *u*, gerada pela função ASE (*Associative Search Element*) que corresponde a uma função probabilística de um estado $\vec{x}$. Se o sistema está no estado x_j, uma ação u_i é selecionada e w_{ij} denota o peso entre o estado x_j e o ASE. O sinal de reforço externo *r* é igual a -1 na ocorrência de colisão e 0

caso contrário. Entretanto, o sinal de colisão é consequência não somente da última ação, mas também das ações tomadas anteriormente e em sequência nos estados prévios. O problema desta abordagem definido como "determinação de crédito" é definir um sinal de reforço externo para uma sequência de estados anteriores. Os autores utilizam regras de aprendizado de diferença temporal (SUTTON, 1984) para definir um sinal de "reforço interno" $\hat{r}$ para cada estado do sistema. A idéia básica é que para cada estado x_j um valor de avaliação tenha sido aprendido através da função ACE (*Adaptive Critic Element*). Esta função avalia a expectativa das somas descontadas de reforço externo para cada estado.

Desta maneira, esta abordagem mostrou um melhor desempenho com relação à utilização de uma discretização do espaço de entrada determinada a priori. Quando a discretização é utilizada, a distribuição de neurônios sobre o espaço de entrada mostra que muitos neurônios são utilizados para codificar regiões do ambiente que são relativamente seguras, ou seja, muitos neurônios se localizam em áreas com uma alta densidade de exemplos de treinamento. Já na abordagem auto-organizável com aprendizado por reforço, esta distribuição tem uma maior densidade de neurônios em regiões onde colisões são mais prováveis (KRöSE; van Dam, 1992a), o que torna o planejamento de caminho mais aplicável.

4.2.4 Aprendizado Auto-Organizável

Aprendizado auto-organizável é uma outra abordagem para treinamento de RNAs relacionadas a robôs móveis autônomos. Estas RNAs constroem uma representação ou mapeamento a partir de diferentes entradas sensórias que direcionam a ações de controle. Segundo Heikkonen e Koikkalainen (1997), um robô móvel autônomo equipado com um controlador (rede neural) auto organizável pode aprender comportamentos orientados a ação que são difíceis de serem implementados de forma algorítmica. Kohonen (1988) afirma que tarefas que envolvem aprendizado de mapeamentos sensório-motor em robôs autônomos são melhores tratadas via aprendizado de mapas auto-organizáveis denominados SOM (self-organizing map).

O aprendizado de mapas auto-organizáveis é difícil de ser aplicado como parte de um sistema de controle clássico, ou seja, que foi construído a partir de uma arquitetura hierárquica, devido à característica não determinística deste tipo de aprendizado. Entretanto, mapas auto-organizáveis são mais aplicáveis a sistemas reativos baseados em comportamentos (HEIKKONEN; KOIKKALAINEN, 1997). A Figura 4.6 ilustra o princípio de funcionamento de uma rede SOM cuja função é categorizar padrões de entrada de acordo

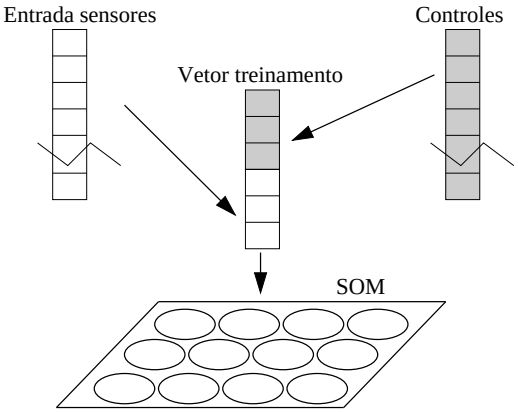


Figura 4.6: Esquema do aprendizado auto-organizável. Fonte adaptada de Heikkonen e Koikkalainen (1997).

com a distribuição destes padrões no espaço de entrada. O mapeamento resultante é uma espécie de representação livre de ruído da combinação mais geral de todas as entradas apresentadas durante o aprendizado. Desta forma, pela seleção de padrões de entrada que melhor correspondam a situações de controle significativas, associações entre a informação sensória e ações de controle podem ser construídas.

Segundo Heikkonen e Koikkalainen (1997), ao utilizar uma rede SOM como um associador, o primeiro problema não é como implementar o algoritmo e sim como selecionar exemplos de treinamento representativos. Os autores propuseram uma abordagem baseada em SOM que foi testada em 3 estudos de caso onde o robô simulado deve construir uma representação a partir de situações sensórias para ações apropriadas, com o objetivo de navegar em seu ambiente sem colidir com obstáculo. Primeiro, demonstraram como o robô móvel simulado pode aprender um eficiente comportamento de evitar obstáculos em seu ambiente a partir de vários exemplos de caminhos conhecidos fornecidos à rede. Segundo, o robô aprende a navegar entre dois pontos sem qualquer ajuda externa durante a navegação, e terceiro, o robô móvel utiliza informação visual enquanto caminha em seu ambiente.

Neste trabalho de tese foi utilizada para construção do nível reativo da arquitetura final proposta, uma rede neural da família de redes auto-organizáveis, que é chamada ART (Adaptive Resonance Theory) e será descrita no próximo capítulo.

4.3 RNAs Aplicadas à Navegação Deliberativa

Navegação deliberativa é realizada quando se tem disponível uma informação global do ambiente. Muitas abordagens convencionais para construção de representações globais durante a exploração são descritas na literatura. Uma abordagem frequentemente utilizada em aplicações de robô real é a chamada *grids de ocupação*. Ela consiste de uma representação probabilística discreta do ambiente na qual as probabilidades de presença de obstáculos são armazenadas durante exploração. Este tipo de representação foi introduzido por Elfes (1989) e exemplos de seu uso podem ser encontrados em (ELFES, 1989; IVANJKO; VASAK; PETROVIC, 2005; JARADAT; LANGARI, 2005). Um exemplo de abordagem neural para armazenar este tipo de representação é proposta em (JORGENSEN, 1987), onde uma rede Hopfield foi usada para armazenar grades de ocupação de um número de salas reservadas à operação do robô. A rede provê um endereçamento de memória de conteúdo a partir de informações parciais sobre as salas e converge para as informações dos ambientes armazenadas.

Via de regra, se uma informação global do ambiente está disponível, métodos de planejamento podem ser utilizados para navegação de um robô móvel. Muitos métodos de planejamento são propostos e todos necessitam de uma informação precisa do ambiente do robô. Vários métodos de planejamento de caminho descritos na literatura utilizam representações de RNAs do ambiente (MILLAN; ARLEO, 1997; TOLEDO et al., 2000), para planejar caminhos. Estas representações são menos complexas do que as representações convencionais, dado que não é necessário nenhum conhecimento sobre a cinemática do robô. A abordagem mais direta para o treinamento de tais redes é a utilização do conhecimento prévio do ambiente. Neste caso, o ambiente deve ser conhecido a priori a partir de todas as suas posições, verificando-se se estas posições estão ocupadas ou não por obstáculos. De maneira similar, deve-se também conhecer a partir de todas as configurações do robô, se ele colide ou não com um obstáculo. Alguns exemplos de abordagens que utilizam técnicas de RNAs para os sub-problemas da navegação de robôs tais como localização, construção de mapas e planejamento serão vistos na sequência.

4.3.1 Localização

Localização é o processo em que o robô encontra sua posição corrente em um mapa. Para isto, ele pode contar com suas entradas sensórias exteroceptivas e proprioceptivas. Em geral, localização requer reconhecimento

de lugar. Para localizar a si mesmo, o robô pode memorizar as percepções sensórias observadas durante a exploração ou pode aprender uma representação mais complexa (mapa) codificando relacionamentos espaciais entre as percepções locais experimentadas. Localização e construção de mapas são processos fortemente relacionados e inter-dependentes. Para se obter a localização do robô é necessário que exista um mapa, enquanto a construção de um mapa requer que a posição seja estimada com relação ao mapa parcial aprendido até o momento (FILLIAT; MEYER, 2003b).

No processo de localização, RNAs podem transformar dados sensórios brutos em representações mais confiáveis. Por exemplo, uma RNA direta pode ser treinada por retro-propagação para gerar uma grade de ocupação local, a partir da percepção sensória corrente. Esta grade seria uma representação discreta do espaço circundante do robô, onde cada célula apresentaria um valor estimado da probabilidade de estar ocupada e corresponderia a uma determinada área do mundo. Após a exploração, o algoritmo de localização busca a grade previamente armazenada que melhor combina com o mapa local corrente. Uma vantagem de usar RNAs para interpretar dados de sensores e construir grades de ocupação locais é que elas são tolerantes a ruídos. A desvantagem é seu custo computacional, pois a construção de uma grade $n * n$ requereria $n * n$ chamadas a uma RNA (MILLáN, 2003).

Outra maneira diferente é treinar uma RNA direta para aprender quais características do ambiente são marcos relevantes para localização. Este treinamento pode ser realizado através de exemplos coletados durante uma fase de exploração, cada exemplo consistindo de uma percepção sensória e sua localização. Durante a operação, o robô calcula a média das respostas da RNA para k exemplos vizinhos mais próximos de sua localização estimada.

Filliat e Meyer (2003a) dão exemplos de abordagens que utilizam RNAs projetadas para emular *place cells* no hipocampo de ratos. A entrada para as redes são fornecidas a partir de um conjunto de neurônios cuja atividade depende da distância e da direção de marcos específicos. As atividades destes neurônios são alimentadas por camadas sucessivas de neurônios onde mecanismos competitivos resultam na ativação de poucos nós na camada de saída que correspondem à posição do robô.

4.3.2 Construção de Mapas

Como visto no Capítulo 3, Seções 3.3.6 e 3.3.7, os dois tipos principais de representações que robôs móveis podem aprender são chamadas métricas e topológicas. Mapas métricos são mapas quantitativos que reproduzem características geométricas e espaciais do ambiente. As dificuldades desta abor-

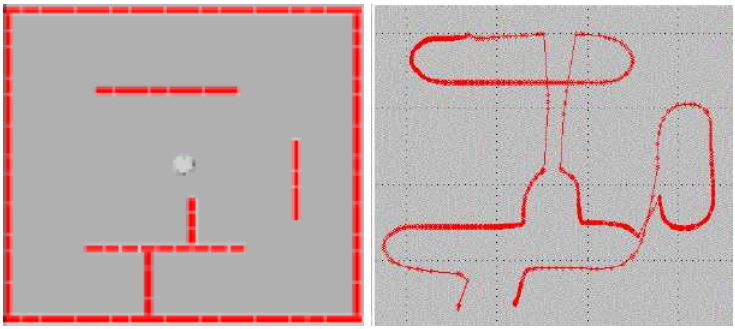
dagem se deve ao seu custo computacional e sua vulnerabilidade a erros em medidas dos sensores que acabam afetando as informações métricas. Mapas topológicos são mais qualitativos e consistem de um grafo, onde nós representam lugares distintos em nível de percepção (marcos) e arestas indicam relações espaciais entre eles. A abordagem topológica é menos vulnerável a erros sensórios e habilita um rápido planejamento de caminho que se reduz a um simples processo de busca. Todavia, representações topológicas contam com a existência de marcos sempre reconhecíveis (FILLIAT; MEYER, 2003b). A seguir serão vistos como RNAs podem ser aplicadas ao aprendizado e utilização de mapas para navegação deliberativa.

Mapas Globais Geométricos

As abordagens que usam grades de ocupação para construção de mapas são as mais populares. Em (MILLáN, 2003) é mostrado um exemplo onde uma RNA direta foi treinada para criar uma grade de ocupação local, modelando o espaço ao redor do robô. Sucessivas grades locais geradas à medida que o robô explora o ambiente são combinadas para produzir um preciso mapa métrico global. A partir do mapa métrico global, um grafo topológico é gerado de maneira *off-line*, para reduzir o custo do planejamento de caminhos entre diferentes localizações do ambiente. Esta abordagem (em conjunto com os processos de localização discutidos anteriormente) foram implementados em robôs para guiar turistas em museus.

Uma alternativa, apresentada no trabalho de Arleo, Millán e Floreano (1999), utiliza a mesma interpretação neural de dados de sensores, mas constrói de maneira *on-line* um particionamento de resolução variável do ambiente. Ou seja, o ambiente é discretizado em células de tamanhos diferentes, com alta resolução somente em áreas críticas (ao redor de obstáculos). O mapa resultante combina aspectos geométricos e topológicos que são aprendidos simultaneamente.

Já o trabalho de Oliveira (2001) apresenta uma outra alternativa para RNAs aplicadas a representação em grid. O autor utiliza um mapa auto-organizável de Kohonen para criar um mapa geométrico do ambiente através dos pesos da rede. Numa primeira fase de exploração randômica do ambiente, por exemplo Figura 4.7(a), o robô ao encontrar um obstáculo, assume o comportamento de seguir parede, para realizar o contorno do mesmo (SILVA, 2001) e coletar as posições em coordenadas (x,y). Os dados coletados são passados para uma rede de Kohonen cujo treinamento é *off-line*. Em seguida, o mapa do ambiente é gerado a partir da plotagem dos pesos da rede em um espaço bi-dimensional como mostrado na Figura 4.7(b). Após o treinamento, o robô



(a) (b)

Figura 4.7: Rede de Kohonen como mapa geométrico do ambiente: (a) Robô simulado e seu ambiente. (b) A distribuição de pesos na rede de Kohonen que representa o mapeamento do ambiente em (a). Fonte adaptada de (OLIVEIRA, 2001)

então pode navegar no ambiente consultando o mapa construído, sem necessitar da informação dos sensores. Nesta abordagem observa-se uma maneira diferente de aplicação das redes de Kohonen quanto a forma de visualização, que em geral é aplicada à classificação de espaços multi-dimensionais que são visualizados conforme a disposição dos neurônios na camada de saída.

Mapas Globais no Domínio de Sensores

Ao invés de uma caracterização do estado do robô na configuração do ambiente e a construção de uma representação global neste domínio, o estado do robô também pode ser dado no *domínio de sensor*. O domínio de sensor é definido como o conjunto $S \subset \mathbb{R}^n$ de todos os vetores de informação sensória possíveis $s = (s_1, \ldots, s_n)^T$, onde s_i são as medidas de sensores (sonar, alcance, visão, etc) do robô ou características derivadas a partir destas medidas, que são obtidas na fase de exploração, enquanto o robô vaga pelo ambiente sem colidir com obstáculo. Sendo assim, esta representação global do ambiente agora consiste de todos os vetores de medida no domínio de sensor. É importante observar que (para ambientes estáticos) a *dimensionalidade intrínseca* dos dados é igual ao número de graus de liberdade do robô. Desta forma, a representação do ambiente pode ser dada através de um "*manifold*" de baixa dimensão, bi ou tri-dimensional com relação ao domínio de sensor de alta dimensão.

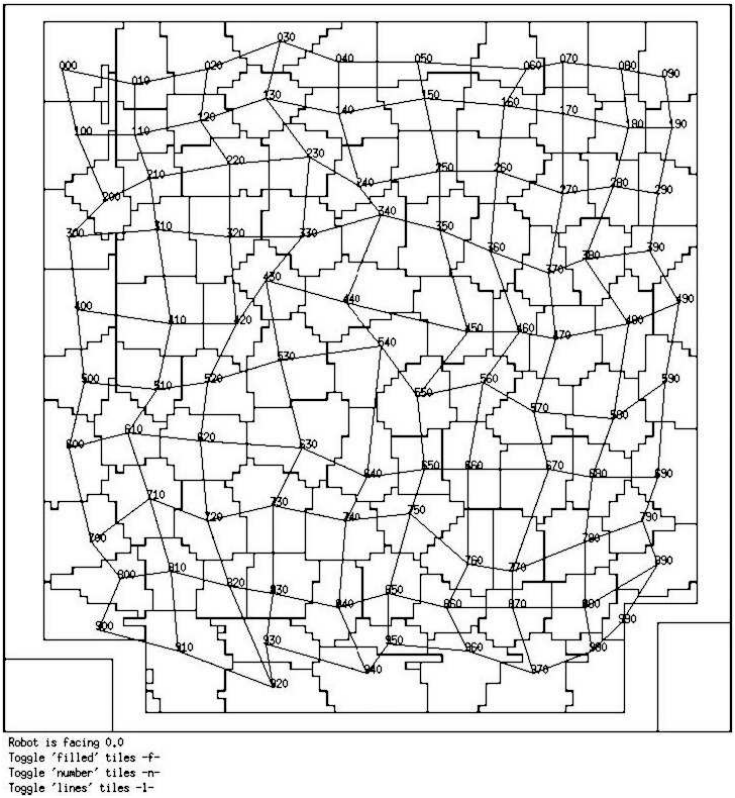


Figura 4.8: Neurônios vencedores em uma rede de Kohonen tri-dimensional como função de localização no ambiente. Fonte adaptada de (KRöSE; EECEN, 1994).

Por exemplo, no trabalho de Kröse e Eecen (1994), uma rede de Kohonen é utilizada para aprender uma representação no domínio de sensor do espaço livre de obstáculos, através da parametrização do manifold. A regra de aprendizado assegura que neurônios vizinhos na rede correspondem à áreas vizinhas no domínio de sensor. Sendo assim, o robô percorre seu ambiente sem colisão em uma fase de exploração, enquanto dados dos sensores são coletados e utilizados para treinar a rede de Kohonen, a qual representa um vetor de quantização do manifold no domínio de sensor. Como ilustrado na Figura 4.8, os autores usaram uma rede de Kohonen tri-dimensional, que relaciona pontos no espaço livre e os neurônios da rede. Sendo assim, esta topologia

produzida pela rede pode ser usada na fase de planejamento subsequente. Dado que a posição do robô também é especificada no domínio de sensor e a tarefa consiste de mover-se na direção de uma configuração cujas medidas de sensores **s** correspondam as medidas de sensor desejadas $\mathbf{s}^*$, caminhos puderam ser planejados usando as conexões da rede de Kohonen proposta.

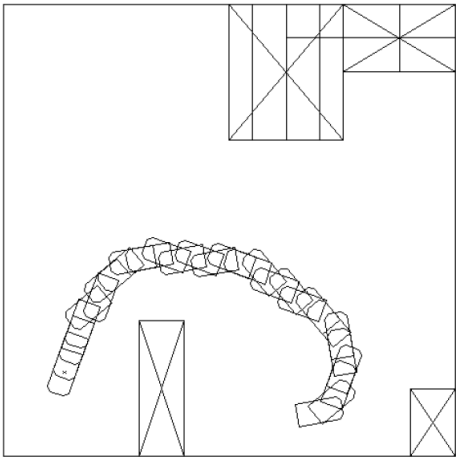


Figura 4.9: Exemplo do caminho executado pelo robô veículo controlado a partir do domínio de sensor. Fonte adaptada de (KRöSE; EECEN, 1994).

Segundo Kröse e Eecen (1994), restrições de movimento do robô (não-holonômico) podem tornar impossível a utilização de relações de vizinhança entre neurônios. Desta forma, as conexões da rede de Kohonen são descartadas logo que a representação no domínio de sensor é formada e novas conexões entre os neurônios são formados em uma segunda fase de aprendizado. Cada uma destas conexões representam a probabilidade de que o robô irá "transitar" de um neurônio para o outro para uma dada ação. O planejamento de caminho pode agora ser visto como um problema de decisão de Markov que, neste caso, foi resolvido com sucesso através de programação dinâmica. A Figura 4.9 ilustra o resultado deste tipo de planejamento sendo executado com sucesso para estacionamento de um robô móvel. O estado objetivo (no domínio de sensor) corresponde à posição base-esquerda com a frente do robô voltada para o canto base-esquerdo. Para este estado objetivo, é computada de forma iterativa uma função de custo e a navegação torna-se uma política gulosa sobre esta função de custo.

Mapas Topológicos

Segundo Kröse e van Dam (1997), abordagens de representação do ambiente no domínio de sensor podem ser consideradas como abordagens topológicas porque ambas constroem uma representação do ambiente a partir do aprendizado local de marcos perceptíveis. Esta representação corresponde a um grafo onde cada nó representa um marco pré-definido. Representações espaciais entre marcos são codificadas por ligações entre nós vizinhos no grafo, produzindo uma estrutura isomórfica da topologia do ambiente. O tratamento de ambiguidades entre padrões sensórios similares pode ser feito por discriminação contextual ou pela adição de informação métrica.

Como visto anteriormente, mapas auto-organizáveis ou de Kohonen são uma maneira alternativa para construir mapas topológicos, outros exemplos de abordagens são apresentadas em (KURZ, 1996; ZIMMER, 1996). Mas ao invés de usar um mapa auto-organizável com uma estrutura fixa (dimensionalidade ou número de unidades), também é possível aprender o mapa topológico do ambiente por meio de um mapa auto-organizável dinâmico (MILLáN, 1997; ZIMMER, 1996). Este método adiciona uma nova unidade se a percepção sensória corrente é diferente o suficiente de qualquer outra unidade existente. Neste tipo de rede a topologia do ambiente se dá pelas ligações entre suas unidades. A Teoria da Ressonância Adaptativa (redes ART) pode produzir quantizações do ambiente similar a um mapa auto-organizável dinâmico, mas o mapa resultante não exibe relacionamentos topológicos entre as unidades.

Buscando uma maior robustez, flexibilidade e adaptabilidade, muitas pesquisas em navegação de robôs autônomos tem se inspirado em descobertas neurofisiológicas que deram origem por exemplo à teoria dos mapas cognitivos e à hipótese de que a memória espacial de mamíferos é formada por neurônios sensíveis à localização (*place cells*) presentes no hipocampo.Métodos de aprendizado em tempo de operação são bastante utilizados nas propostas biologicamente inspiradas, tais como o trabalho de Arleo e Gerstner (2000), onde é proposto um modelo do hipocampo, que através do aprendizado Hebbiano não supervisionado, se obtém um mapa espacial do ambiente de maneira incremental e *on-line*.

Já a Figura 4.10 (a) apresenta o sistema de navegação proposto por Schölkopf e Mallot (1995) que consiste de uma abordagem de rede neural para aprendizado de mapa cognitivo de um labirinto hexagonal [Figura 4.10(b)], a partir de uma sequência de perspectivas e decisões de movimento. O processo de aprendizado se baseia em uma representação intermediária chamada *grafo de perspectiva* [Figura 4.10(c)], cujos nós correspondem às

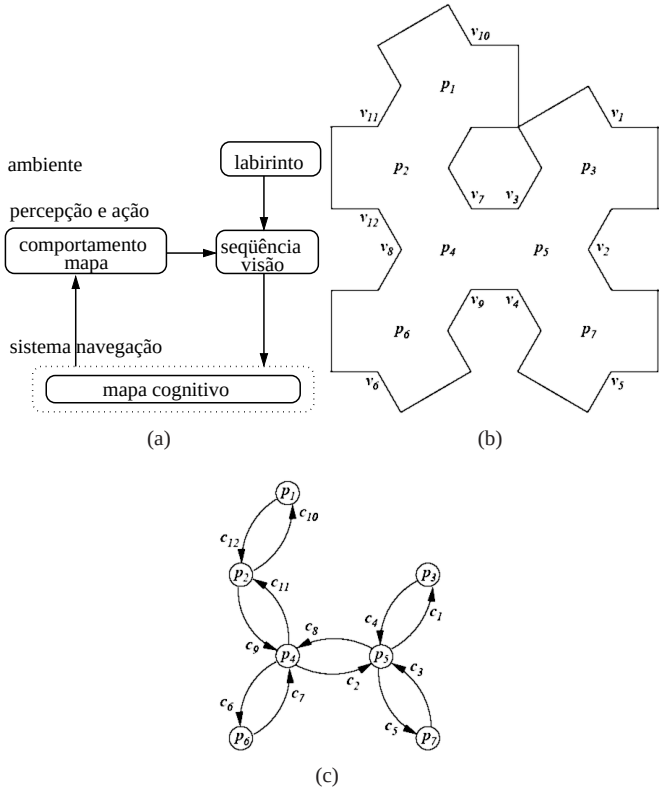


Figura 4.10: Aprendizado de mapa cognitivo de Schölkopf e Mallot (1995). (a) Ciclo percepção-ação do sistema de navegação. (b) Esquema do labirinto hexagonal com 7 lugares p_i ($i = 1, \ldots, 7$) e 12 perspectivas v_j ($j = 1, \ldots, 12$). (c) Grafo dirigido de lugares do labirinto com corredores c_j correspondendo às perspectivas v_j. Fonte adaptada de Mallot et al. (1995).

perspectivas e as arestas representam os movimentos que levam de uma perspectiva a outra. Através de um método teórico de reconstrução de grafos, o grafo de perspectiva provê uma informação completa sobre a estrutura topológica e direcional do labirinto. Planejamento de caminho pode ser executado diretamente no grafo de perspectiva não sendo necessária executar a reconstrução do grafo. A rede neural é implementada para aprender o grafo de perspectiva durante a exploração randômica do labirinto. O treinamento da rede neural se baseia em uma regra de aprendizado competitivo e não su-

pervisionado que traduz sequências de perspectivas temporais (ao invés de similares) em conectividade na rede. Como consequência, a rede utiliza o conhecimento da estrutura topológica e direcional do labirinto aprendida, para gerar suposições sobre quais as perspectivas seguintes que são mais prováveis de serem encontradas, melhorando assim o desempenho de reconhecimento das perspectivas do labirinto.

O trabalho de Hafner (2005) originalmente inspirou-se no algoritmo de mapeamento cognitivo anterior, (MALLOT et al., 1995), mas estendeu a restrição de labirintos para ambientes abertos e reais onde movimentos arbitrários são possíveis. A abordagem proposta apresenta um modelo de mapa cognitivo que cria uma representação espacial interna do ambiente. O modelo é inspirado por células lugar (*place cells*) presentes no hipocampo de ratos e reproduzindo algumas propriedades destas células quanto à forma e ao tamanho dos (*place fields*). As células lugar são representadas como nós de um grafo topológico e o algoritmo resulta em um mapa topológico esparso que pode facilmente ser estendido para informação métrica.

4.3.3 Navegação Planejada em Ambientes Conhecidos

De maneira geral, métodos de planejamento de caminho para robôs móveis em ambientes conhecidos tipicamente requerem uma decomposição do *espaço livre* do robô. Esta decomposição consiste de um número de áreas onde o robô pode mover-se livremente sem o perigo de colidir com obstáculos. Além disto, áreas serão conectadas se o robô pode executar um movimento entre elas (KRöSE; van Dam, 1997).

Nas abordagens neurais para planejamento de caminho em ambiente conhecido, *métodos de quantização de vetor* neurais [1] são usados para decompor o espaço livre do robô. Estes métodos distribuem *vetores protótipos* $\mathbf{w}_i$ sobre o espaço livre resultando em uma decomposição deste espaço em regiões disjuntas S_i centralizadas em torno de vetores protótipos: $S_i = \{\mathbf{x} : d(\mathbf{x},\mathbf{w}_i) \leq d(\mathbf{x},\mathbf{w}_j), \forall j\}$, onde $\mathbf{x}$ é uma posição dentro de uma região S_i e $d(\mathbf{x},\mathbf{w}_i)$ é a distância entre o vetor $\mathbf{x}$ e o vetor protótipo $\mathbf{w}_i$). Isto é chamado de diagrama de Voronoi do espaço livre. Cada neurônio na rede corresponde a um vetor protótipo. Um exemplo de tal decomposição é dado na Figura 4.11. Muitas vezes, os vetores protótipos de regiões vizinhas estão conectados. Estas conexões são usadas ou para indicar que um caminho livre

[1] Modelos de redes neurais aplicadas a problemas de reconhecimento de padrão estatístico onde a distribuição de classes de vetores padrões geralmente se sobrepõe e ao mesmo tempo se deve determinar a localização ótima das fronteiras de decisão. Os algoritmos de aprendizagem de quatização de vetores definem uma eficiente aproximação para as fronteiras de decisão ótimas.

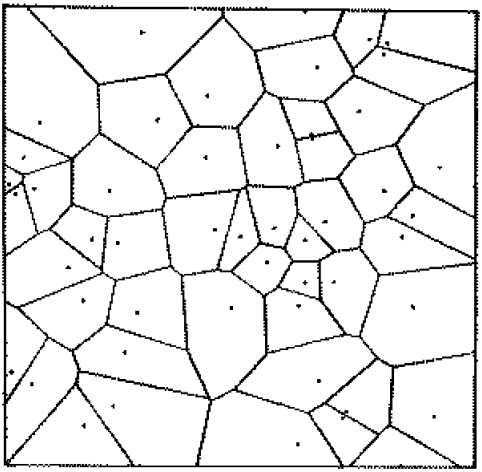


Figura 4.11: Exemplo de um diagrama Voronoi. Os pontos correspondem aos vetores protótipos $\mathbf{w}_i$ e as arestas são as fronteiras entre as áreas correspondentes de S_i. Fonte adaptada de (KRöSE; EECEN, 1994).

de colisão existe entre regiões, ou para adquirir uma decomposição específica do espaço livre mais adequada ao método de planejamento de caminho utilizado. Exemplos de como tais conexões podem ser usadas são vistos em (NAJAND; LO; BAVARIAN, 1992; GLASIUS; KOMODA; GIELEN, 1995).

Exemplos de Abordagens de RNAs

No trabalho de Najand, Lo e Bavarian (1992), uma rede neural é usada para aprender apenas o diagrama Voronoi do espaço livre de maneira mais eficiente. O diagrama Voronoi é aprendido por uma rede neural, que distribui seus vetores protótipos usando quantização de vetor padrão. Um conhecimento a priori é utilizado para assegurar que os neurônios estão distribuídos somente sobre o espaço livre. Uma árvore de dispersão (*spanning tree*) mínima é então criada para conectar neurônios vizinhos. Na Figura 4.12 é mostrado um exemplo de uma representação do espaço livre gerado por uma rede neural que corresponde a uma árvore de dispersão mínima de conexões entre neurônios. O algoritmo de planejamento de caminho planeja um caminho via vetores protótipos no ambiente do robô, seguindo as conexões na árvore. Dado que o método de quantização de vetores empilha vetores protótipos distantes de todos os obstáculos, este caminho tem a propriedade de ser distante

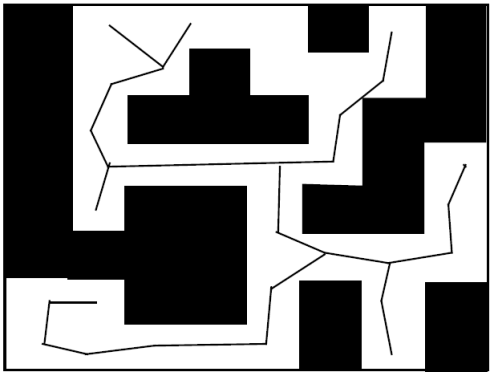


Figura 4.12: Exemplo de uma representação do espaço livre gerada pela rede descrita em Najand, Lo e Bavarian (1992). Fonte adaptada de (KRöSE; EECEN, 1994).

ao máximo de todos estes obstáculos. A partir deste trabalho, um grande número de artigos tem sido apresentados na literatura que descrevem diferentes algoritmos de planejamento de caminho que utilizam técnicas mais avançadas de quantização de vetor para aprender representações do ambiente.

Morasso, Vercelli e Zaccaria (1992) propuseram um algoritmo de planejamento de caminho, onde o robô móvel se move sempre em linha reta na direção de seu objetivo, enquanto está no espaço livre. Se o robô detecta a presença de um obstáculo em seu caminho, ele se move ao redor do obstáculo até poder reiniciar o caminho na direção do objetivo. Este algoritmo de planejamento utiliza uma representação do ambiente do robô que precisamente diferencia o espaço livre a partir do espaço ocupado. Para tal, Morasso, Vercelli e Zaccaria (1992) descreveram uma rede neural que aprende esta tal representação. Como ilustrado na Figura 4.13, um número de neurônios da rede é distribuído uniformemente sobre o ambiente do robô, classificando as áreas como *seguras* ou *inseguras*. Um método de quantização de vetor é utilizado para alterar a posição dos vetores protótipos e para adicionar dinamicamente novos protótipos onde necessário. Uma heurística é utilizada para que os neurônios se movam na direção das fronteiras com obstáculos, para que esta representação se torne aplicável ao método de planejamento de caminho. A Figura 4.13 é um exemplo de uma representação resultante do aprendizado da rede neural. É importante notar que as conexões entre os neurônios não são usadas para planejamento de caminho. A representação é usada apenas para decidir se o robô está no espaço livre, neste caso ele executa a linha reta

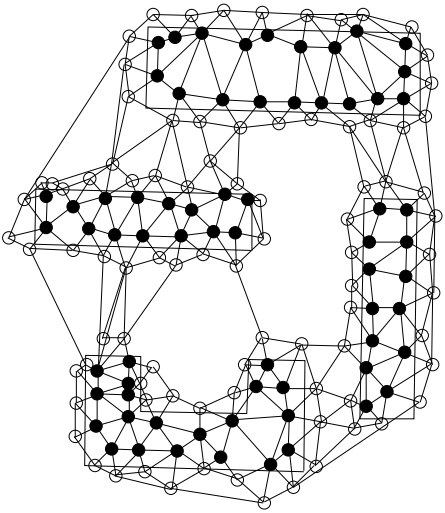


Figura 4.13: Exemplo de uma representação gerada pela rede neural descrita em (MORASSO; VERCELLI; ZACCARIA, 1992). Os círculos preenchidos representam espaço ocupado e círculos abertos representam espaço livre. A figura mostra claramente que os neurônios representam precisamente as bordas entre o espaço livre e ocupado. Fonte adaptada de (KRöSE; EECEN, 1994).

até o objetivo, caso contrário, ele se move ao redor do obstáculo.

4.4 Conclusão

Este capítulo apresentou uma revisão sobre diferentes abordagens de RNAs aplicadas ao problema da navegação de robôs móveis autônomos. Elas são amplamente utilizadas porque permitem a robôs móveis a possibilidade de aprender a agir de maneira adaptativa, principalmente em domínios do mundo real e mediante a imprecisão de sensores e atuadores.

Como visto no capítulo, RNAs foram utilizadas em tarefas de aprendizagem através da análise de interações percepção-ação aplicando-se à navegação reativa, localização, construção de mapa e planejamento de caminho. Entretanto não existe um sistema de navegação completo que seja puramente desenvolvido por componentes de redes neurais (MILLáN, 2003). Além disto, do ponto de vista de engenharia, um robô móvel completo deve incorporar outros tipos de técnicas de aprendizado para gerar modelos mais abstratos de suas percepções, ações e regras sensor-motor, combinando capacidades de

aprendizado com técnicas alternativas, tais como outras abordagens de inteligência artificial e aprendizado de máquina.

Há também uma área de pesquisa que busca inspiração em ciências biológicas tais como neuroetologia, para desenvolvimento de componentes neurais necessários para um completo sistema de navegação, cuja modelagem difere das arquiteturas de RNAs tradicionais. Os inúmeros esforços nesta direção caracterizam a pesquisa bio-inspirada, que tem a aprendizagem por reforço e os modelos de mapas cognitivos e células hipocampais como pedras angulares na construção de robôs móveis inteligentes.

O estudo realizado neste capítulo direcionou principalmente a construção das arquiteturas neurais reativas propostas neste trabalho de tese. A abordagem de RNAs proposta conta com uma rede ART que recebe como padrão de entrada a percepção obtida através dos dados dos sensores de proximidade, junto com a realimentação da ação executada no ciclo de controle anterior. A rede ART então reconhece o estado do ambiente e aciona uma rede MLP para o aprendizado da melhor ação a ser executada. Para o aprendizado *on-line* destas redes é utilizado um algoritmo aleatório baseado na técnica de aprendizado por reforço.

Como o mapa resultante do aprendizado auto-organizável da rede ART não produz relacionamentos topológicos do ambiente, foram propostas camadas deliberativas para tratar da representação topológica de labirintos, através de um método de mapeamento cognitivo proposto. Todo o processo de evolução que resultou nas camadas reativas e deliberativas, até se chegar na arquitetura final NeuroGog será descrito ao longo dos próximos capítulos.

CAPÍTULO 5

Evolução das Abordagens Reativas

5.1 Introdução

O Capítulo 4 apresentou formas de aprendizado e utilização de RNAs aplicadas a diferentes problemas envolvidos na construção de sistemas de navegação para robôs móveis autônomos, os quais serviram de guia para o desenvolvimento das abordagens propostas neste trabalho de tese. Este capítulo descreve o desenvolvimento e a evolução de sucessivas arquiteturas baseadas no paradigma de controle reativo e subsunção (Seção 2.3). Foram utilizados diferentes tipos de RNAs, juntamente com uma técnica de aprendizado em tempo de operação, baseada na AR, com o objetivo de incorporar a um robô móvel capacidades de aprendizado autônomo e adaptativo para navegação em labirintos.

Através do aumento progressivo da complexidade do labirinto e da tarefa de navegação, tornou-se necessária uma junção envolvendo dois tipos de RNAs e diferentes formas de ligação entre os processos do nível mais baixo das arquiteturas, a fim de propiciar um efetivo sistema de controle final. Desta maneira foram investigados quais tipos de comportamentos podem ser obtidos através destes sucessivos arranjos neurais e se de fato ocorre um aprendizado efetivo em tempo de operação.

Como resultado esta evolução arquitetural deu origem a uma arquitetura neural reativa, que provê um sistema de navegação cuja estratégia corresponde à **Resposta ao Disparo de Reconhecimento** (Seção 3.3.5). Esta arquitetura neural posteriormente foi incorporada à camada reativa da arquitetura final **NeuroCog**, como proposta de uma arquitetura híbrida de subsunção e baseada em comportamentos, para navegação em labirintos dinâmicos.

5.2 Forma Geral das Arquiteturas Reativas

Esta seção apresenta os ensaios de construção onde progressivamente foram integrados dois tipos de arquiteturas de RNAs, visando a construção de um processo de aprendizado de mapeamentos percepção-ação em tempo de operação. A estrutura básica das arquiteturas reativas é ilustrada na Figura 5.1(a). Os módulos **Percepção** e **Sistema motor** configuram o nível mais baixo das arquiteturas, recebendo informação dos sensores e fornecendo comando aos atuadores respectivamente, por intermédio do controle neural implementado pela camada reativa no nível acima. O robô não possui nenhuma informação a priori do ambiente e conta apenas com seus sensores de proximidade ($S_0, \ldots, S_7$), ângulo de orientação (θ) e posição [Figura 5.1(b)]. A informação de posição do robô não é utilizada pelas abordagens reativas propostas.

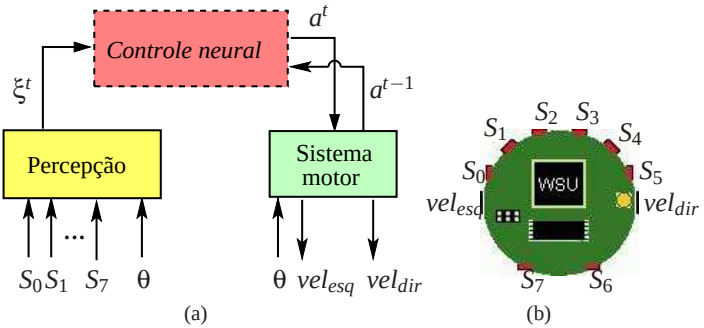


Figura 5.1: (a) Estrutura geral das arquiteturas reativas propostas. (b) O robô simulado.

5.2.1 Os Módulos Percepção e Sistema motor

Os módulos **Percepção** e **Sistema motor** constituem os processos que formam o nível abaixo da camada reativa das arquiteturas desenvolvidas e serão apresentadas na sequência. O módulo **Percepção** [Figura 5.1(a)], recebe como entrada dois tipos de informação: a primeira alotética, provinda dos sensores de proximidade $(S_0, \ldots, S_7)$ e a segunda, idiotética, referente ao ângulo de orientação do robô, θ. A informação de posição do robô não é utilizada. Através da informação de sensores e ângulo de orientação, o módulo **Percepção** define o estado do ambiente, em um instante de tempo t, como ξ^t,

que corresponde à configuração de paredes do labirinto.

Para efeito de simplificação da construção, bem como para um melhor entendimento do princípio básico de funcionamento das arquiteturas de controle propostas nesta tese, considera-se que o robô executa, a cada passo de controle, uma ação no tempo t, cujas direções possíveis são apenas: oeste, norte, leste e sul.

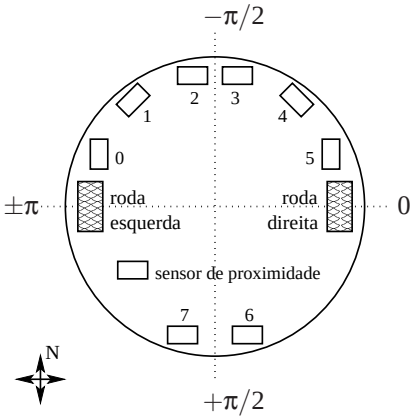


Figura 5.2: Robô Khepera e as direções globais definidas como oeste ($\theta = \pm\pi$), norte ($\theta = -\pi/2$), leste ($\theta = 0$) e sul ($\theta = -\pi/2$).

Como pode ser visto na Figura 5.2, com base no sistema de coordenadas angulares utilizado pelo sensor de compasso do robô, as direções oeste, norte, leste e sul se referem a quatro diferentes poses relacionadas à frente do robô, cujo ângulo de direção é determinado através da variável θ. Sendo assim, a direção oeste corresponde à frente do robô estar situada a um ângulo θ igual a $\pm\pi$, a direção norte referente à frente do robô equivaler a θ igual a $-\pi/2$, a direção leste a θ igual a 0 e finalmente a direção sul a θ igual a $+\pi/2$.

Desta maneira, uma ação enviada ao módulo **Sistema motor** é representada pela quádrupla $a^t = (a_o, a_n, a_l, a_s)$, onde os valores para a^t podem ser: $(1,0,0,0)$ que equivale a um movimento na direção oeste ou $(0,1,0,0)$ na direção norte ou $(0,0,1,0)$ na direção leste ou $(0,0,0,1)$ na direção sul.

Levando isto em consideração, o módulo **Percepção** determina o estado do ambiente através da quádrupla ξ^t :

$\xi^t \leftarrow$ **Percepcao**$(S_0,\ldots,S_7,\theta) = (\xi_o,\xi_n,\xi_l,\xi_s)$, onde:

$$\xi_i = \begin{cases} 1 & \text{se existe obstáculo na direção da ação } a_i, \\ & \text{onde } i = o\ (oeste),\ n\ (norte),\ l\ (leste),\ s\ (sul) \\ 0 & \text{caso contrário} \end{cases} \tag{5.1}$$

Como visto na Figura 5.1(a), o módulo **Sistema motor** atua a cada passo de controle ao receber como entrada a ação corrente a ser executada, $a^t = (a_o, a_n, a_l, a_s)$. Este módulo também efetua a realimentação do controle neural fornecendo a ação executada no passo de controle anterior, a^{t-1}. A necessidade desta realimentação e seus efeitos serão descritos na sequência deste capítulo.

O módulo **Sistema motor** utiliza o ângulo de orientação, θ, para posicionar a frente do robô na direção da ação a^t a ser executada, e também para calcular se é mais eficiente o robô girar no sentido horário ou anti-horário, caso haja a necessidade de mudança de direção. Em seguida, o módulo **Sistema motor** fornece comandos aos atuadores para que o robô execute um movimento, em linha reta, na direção da ação a^t até o próximo passo de controle.

Coordenadas relativas à orientação do robô

É importante observar que o esquema de direções visto anteriormente o qual associa a direção oeste a $\theta = \pm\pi$, a direção norte a $\theta = -\pi/2$, a direção leste a $\theta = 0$ e a sul a $\theta = +\pi/2$ pode ser diretamente substituído por um esquema de direções relativas ao corpo do robô, considerando-se por exemplo as direções esquerda (E), frente (F), direita (D) e volta (V).

Como pode ser visto na Figura 5.3, a direção frente se refere ao ângulo θ onde a parte frontal do robô está situada. A partir daí, a direção esquerda corresponde à direção frente mais 90^0 no sentido anti-horário. A direção direita equivale à direção frente mais 90^0 no sentido horário. Finalmente a direção volta corresponde a direção frente mais 180^0 em quaisquer sentido, horário ou anti-horário.

Um aspecto importante deste esquema de coordenadas relativas à pose do robô é que se elimina a condição dos movimentos serem executados apenas nas direções dos ângulos $\pm\pi$ (oeste), $-\pi/2$ (norte), 0 (leste) e $+\pi/2$ (sul). Ou seja, com o esquema de direções esquerda, frente, direita, volta, o robô pode iniciar seu movimento para frente na direção de qualquer ângulo inicial θ, desde que este movimento esteja alinhado com as paredes do labirinto, permitindo que ele execute uma trajetória reta nesta direção.

Finalmente outra consideração válida é o fato de que estas direções

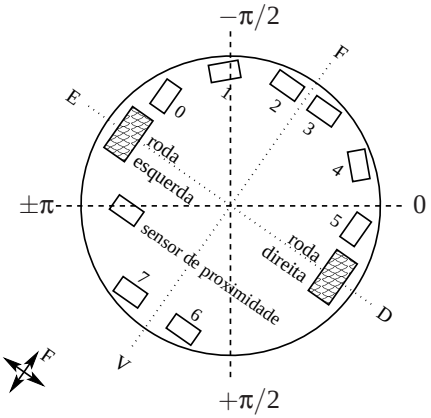
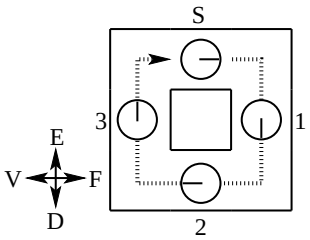


Figura 5.3: Robô Khepera e um exemplo do esquema de coordenadas relativas à pose do robô. Direções definidas como esquerda, E, igual a $-(|\theta|+|-\pi/2|)$, frente, F, igual a θ, direita, D, igual a $\theta+\pi/2$ e volta igual a $\theta+\pi$.

configuradas a partir deste esquema de pose inicial do robô vai determinar a configuração de todos os estados do ambiente e ações executadas nos passos de controle subsequentes. Por exemplo, seja a execução da trajetória descrita na Figura 5.4, onde o robô se move a partir da pose inicial S, passando pelas poses 1, 2 e 3 e retornando a S. As respectivas configurações de estados do ambiente $\xi^t = (\xi^{esquerda}, \xi^{frente}, \xi^{direita}, \xi^{volta})$ e ações executadas $a^t = (a^{esquerda}, a^{frente}, a^{direita}, a^{volta})$ são determinadas a partir da pose inicial do robô em S, cujos valores podem ser visualizados na tabela à direita.



$\xi^t = (\xi_e, \xi_f, \xi_d, \xi_v)$	$a^t = (a_e, a_f, a_d, a_v)$
$\xi^S = (1,0,1,0)$	$a^S = (0,1,0,0)$
$\xi^1 = (0,1,0,1)$	$a^1 = (0,0,1,0)$
$\xi^2 = (1,0,1,0)$	$a^2 = (0,0,0,1)$
$\xi^3 = (0,1,0,1)$	$a^3 = (1,0,0,0)$

Figura 5.4: Exemplo de execução de uma trajetória com base no esquema de direções relativas à pose inicial do robô: esquerda, frente, direita e volta.

5.3 Primeiro Ensaio: Rede MLP e a Aprendizagem por Reforço

Primeiramente, para desenvolver a camada de *Controle neural* das arquiteturas reativas [Figura 5.1(a)], o foco de interesse esteve voltado a uma rede neural do tipo direta denominada MLP (do acrônimo em inglês *multilayer perceptrons*), haja vista sua simplicidade de criação e utilização como um sistema de controle simples. Usualmente este tipo de rede é utilizada após seu treinamento off-line através do algoritmo *backpropagation*, aplicado a um conjunto de exemplos de treinamento e teste. Desta maneira, como esta forma de treinamento não provê a um sistema de controle a propriedade de adaptabilidade, diante de exemplos não previstos, um outro foco de atenção foi a vantagem prática de um aprendizado não supervisionado, fornecido pela teoria da aprendizagem por reforço, cujas técnicas são bastante aplicadas aos problemas de navegação de robô móveis. Por esta razão, os conceitos de redes MLPs bem como aprendizado por reforço são descritos nas seções seguintes, a fim de contextualizar a aplicação destas técnicas nas arquiteturas de controle reativo desenvolvidas neste trabalho de tese.

5.3.1 Redes *Multilayer Perceptrons*

Uma rede MLP consiste de múltiplas camadas de neurônios (nós computacionais) onde o sinal de entrada se propaga pela *camada de entrada* passando por uma ou mais *camadas intermediárias* até chegar à *camada de saída* (Figura 5.5). Cada neurônio da rede inclui uma *função de ativação não-linear* (diferenciável em qualquer ponto). Uma função que normalmente é utilizada, por satisfazer a questão da não linearidade e ter motivação biológica (pois mo-

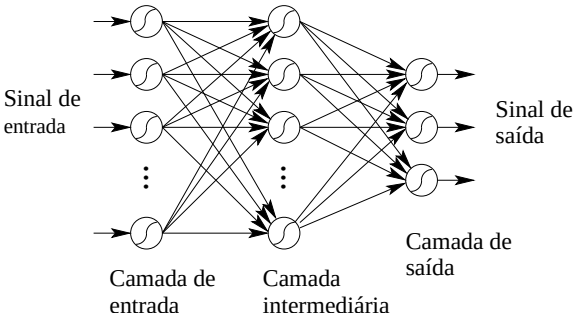


Figura 5.5: Grafo arquitetural de uma rede MLP com uma camada intermediária. Fonte adaptada de Haykin (2001).

dela a fase refratária de neurônios reais) é a função sigmoidal. Os neurônios de uma ou mais camadas intermediárias capacitam a rede a aprender tarefas complexas, extraindo progressivamente as características mais significativas dos padrões de entrada. A rede também exibe um alto grau de conectividade, determinado pelas sinapses da rede. Uma modificação na conectividade da rede requer uma mudança na população das conexões sinápticas ou de seus pesos (HAYKIN, 2001).

Historicamente o grande sucesso das redes MLPs se deve ao algoritmo de aprendizagem supervisionada, *back-propagation* (Rumelhart et al. 1986). Porém, este paradigma de aprendizado é de uso limitado no desenvolvimento de robôs móveis por muitas razões. Como já visto no Capítulo 4, Seção 4.2.1, uma delas seria a dificuldade para prever e construir os conjuntos de treinamento e validação e à necessidade de uma fase de treinamento prévia onde, após a aprendizagem, a rede não pode facilmente ser modificada de forma incremental. Pois, se algo imprevisto ocorre no sistema, uma rede MLP deixa de ser adaptativa quando utilizada como um controlador, porque necessita ser treinada novamente. Por esta razão neste trabalho objetivou-se desenvolver um método para o aprendizado on-line de redes MLPs, com base no paradigma da aprendizagem por reforço. Além disto, as redes MLPs utilizadas nesta tese podem receber como entrada o estado do ambiente ξ^t, ou ξ^t em conjunto com a ação executada no passo de controle anterior, a^{t-1}, ou apenas um sinal de ativação fornecido por outra rede. Todos estes casos serão vistos ao longo deste capítulo.

5.3.2 Aprendizagem por Reforço

A aprendizagem por reforço (AR) (SUTTON; BARTO, 1998) é uma técnica alternativa para o aprendizado de máquina, com respaldo em teorias psicológicas de aprendizado (PEARL, 2000). Sua função é garantir que um agente aprenda determinado comportamento mediante interações de tentativa-e-erro em ambientes dinâmicos. A AR é um paradigma de aprendizagem baseado em comportamento, onde as interações entre o aprendiz e seu ambiente procuram alcançar um objetivo específico, apesar da presença de incertezas. O fato de que esta interação é realizada de maneira não supervisionada torna a AR particularmente atrativa para situações dinâmicas. Há duas abordagens de AR:

Abordagem clássica: onde a aprendizagem ocorre através de um processo de punição e recompensa com o objetivo de alcançar um comportamento global altamente qualificado e

Abordagem moderna: que se fundamenta na técnica matemática conhecida como programação dinâmica (HAYKIN, 2001), a qual não é assunto de interesse neste trabalho.

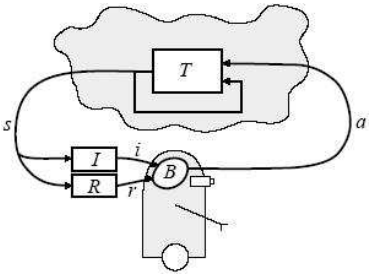


Figura 5.6: Modelo padrão da Aprendizagem por Reforço. Adaptado de Kaelbling, Littman e Moore (1996)

Por exemplo, seja o modelo padrão da aprendizagem por reforço clássica, onde um agente está conectado a seu ambiente via percepção e ação, como mostrado na Figura 5.6. A cada passo de interação o agente recebe como entrada, i, algumas indicações do estado atual, s, do ambiente; o agente então escolhe uma ação, a, para ser gerada como saída. A ação altera o estado do ambiente, e o valor desta transição de estado é comunicado ao agente através de um sinal de reforço escalar, r. O comportamento do agente, B, deve escolher ações que tendam a incrementar ao longo da execução a soma de valores do sinal de reforço. Ele pode aprender a fazer isto com o tempo através da sistemática de tentativa-e-erro, guiado por uma ampla variedade de algoritmos (KAELBLING; LITTMAN; MOORE, 1996).

Formalmente, o modelo consiste de

- um conjunto discreto de estados do ambiente, $\mathcal{S}$
- um conjunto discreto de ações do agente, $\mathcal{A}$
- um conjunto de sinais escalares de reforço; tipicamente $\{0,1\}$, ou número real.

A figura também inclui uma função de entrada I, que determina como o agente visualiza o estado do ambiente. Uma maneira intuitiva de entender a relação entre o agente e seu ambiente pode ser representada através do diálogo exemplo mostrado a seguir.

Ambiente:	Você está no estado 65. Você tem 4 ações possíveis.
Agente:	Eu tomarei a ação 2.
Ambiente:	Você recebeu um reforço de 7 unidades. Você agora está no estado 15. Você tem 2 ações possíveis.
Agente:	Eu tomarei a ação 1.
Ambiente:	Você recebeu um reforço de -4 unidades. Você agora está no estado 65. Você tem 4 ações possíveis.
Agente:	Eu tomarei a ação 2.
Ambiente:	Você recebeu um reforço de 5 unidades. Você está agora no estado 44. Você tem 5 ações possíveis.
⋮	⋮

Portanto a tarefa do agente é encontrar uma política π que mapeie estados a ações, a fim de maximizar algumas medidas de reforço ao longo da execução de sua tarefa.

Nas seções seguintes deste capítulo, serão desenvolvidas algumas formas de integração entre os paradigmas conexionista e por reforço, para a construção de arquiteturas reativas cujo aprendizado seja tanto efetivo quanto adaptativo.

5.3.3 Controle Neural: Rede MLP com Aprendizado por Reforço

Na tentativa de unir as vantagens de dois paradigmas de IA para aprendizado de máquina, o primeiro controle neural a ser implementado consistiu de uma rede MLP com aprendizado on-line baseado na abordagem clássica da AR, denominado controle neural **MLP-R** [Figura 5.7(a)]. Para validar o funcionamento da arquitetura, como primeira tarefa de navegação foi estabelecido que o robô, utilizando apenas seus sensores de proximidade e ângulo de orientação, sendo controlado através do arranjo neural **MLP-R**, percorresse um labirinto simples, executando uma trajetória livre de colisões [Figura 5.7(b)].

O controle **MLP-R** apresenta uma rede MLP com 4 neurônios na camada de entrada, para receber o padrão de entrada ξ^t (Eq. 5.1), 3 neurônios na camada intermediária e 4 neurônios na camada de saída, para produzir a ação corrente a^t. A função sigmoidal constitui a função de ativação de cada neurônio e os pesos das sinapses são inicializados de forma aleatória no intervalo de $[-0.5, +0.5]$. Como dito anteriormente, este tipo de rede foi escolhida para implementar o controle neural devido ao sucesso de suas aplicações por meio do aprendizado *backpropagation*. Porém, como neste trabalho se buscou desenvolver um algoritmo de aprendizado em tempo de operação, visando um aprendizado adaptativo, foi criado um processo de punição e recompensa baseado na modelo clássico da AR, representado pelo módulo **Políticas de aprendizado** [Figura 5.7(a)], onde a rede MLP recebe

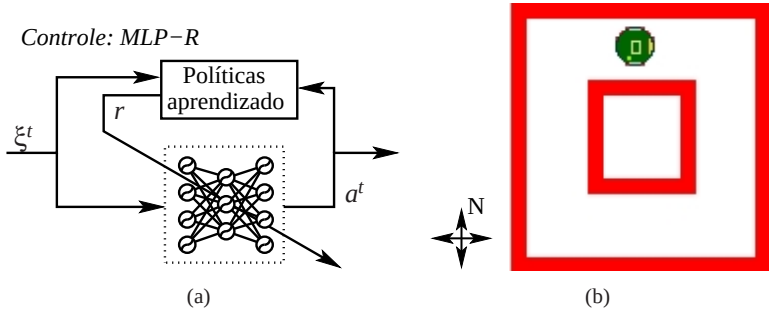


Figura 5.7: (a)Controle MLP-R: rede *multi layer perceptrons* com aprendizado por reforço. (b) Primeira tarefa de navegação.

um reforço, r, o qual corresponde a punição ou recompensa, de acordo com os efeitos da resposta da rede, a^t, na execução da tarefa de navegação do robô.

O Algoritmo 5.1 (linhas 1 - 12) descreve o controle neural **MLP-R**. Primeiramente o processo de aprendizado on-line (linhas 13 - 33) foi modelado da seguinte maneira, dada uma configuração de paredes, ξ^t, uma ação, a^t, considerada correta seria aquela que não levasse à colisão. Por exemplo, seja a configuração de paredes $\xi^t = (1,0,1,0)$, paredes a oeste e a leste, segundo a regra acima, seriam corretas duas ações $a^t = (0,1,0,0)$ e $a^t = (0,0,0,1)$ (mover-se tanto a norte como ao sul respectivamente). Quando a rede MLP responde com uma ação incorreta, o processo de punição consiste em sortear um peso para ser alterado com um valor aleatório, a fim de que a rede seja novamente executada. O índice deste peso sorteado é armazenado em uma fila do tipo FIFO (do acrônimo em inglês: *first in, first out*), que contém os índices de todos os pesos já selecionados até momento, os quais não podem mais ser sorteados numa próxima etapa, se existirem ainda outros pesos nunca selecionados no ciclo de aprendizado corrente. O processo de punição é repetido até que a rede forneça uma ação correta (Algoritmo 5.1, linha 4). Quando a ação a^t for correta, o sinal de reforço será recompensa, o que corresponde a manter os pesos da rede MLP inalterados, finalizando assim seu processo de aprendizado no passo de controle corrente.

O arranjo neural **MLP-R** ao ser aplicado para o controle inteligente do robô, resulta tanto um comportamento inadequado, tal como movimentos "vai e vem", quanto no não aprendizado on-line da rede MLP. Via de regra, uma rede MLP constitui um aproximador de função, que realiza um mapeamento não linear de entrada-saída de natureza geral (HAYKIN, 2001), todavia uma função não pode ser caracterizada pelo mapeamento de uma entrada a

Algoritmo 5.1 Controle **MLP-R**

```
1: procedure CONTROLEMLP-R
2:     while ( ; ; ) do
3:         ξ^t ← Percepcao(S_0, ..., S_7, θ)
4:         a^t ← MLP(ξ^t)
5:         r ← POLITICASAPRENDIZADO(ξ^t, a^t, MLP)
6:         if r = punição then
7:             Alterar peso da rede MLP.
8:             goto 4
9:         end if
10:        Sistema motor(a^t)
11:    end while
12: end procedure

13: procedure POLITICASAPRENDIZADO(ξ^t, a^t, MLP)
14:    if a^t ∈ {(1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)} then
15:        if ξ^t = (ξ_o, ξ_n, ξ_l, ξ_s) = (0,0,0,0) then
16:            r ← recompensa                          ▷ preservar pesos MLP
17:        end if
18:        if ξ^t = (ξ_o, ξ_n, ξ_l, ξ_s) = (1,0,1,0) then
19:            if a^t = (1,0,0,0) or a^t = (0,0,1,0) then
20:                r ← punição                                  ▷ colisão
21:            else
22:                r ← recompensa                      ▷ preservar pesos MLP
23:            end if
24:        end if
25:        if ξ^t = (ξ_o, ξ_n, ξ_l, ξ_s) = (0,1,0,1) then
               ...                                  ▷ similar à política (18 - 24)
26:        end if
27:        if ξ^t = (ξ_o, ξ_n, ξ_l, ξ_s) = (0,1,1,0) then
               ...                                  ▷ similar à política (18 - 24)
28:        end if
           ...
29:    else
30:        r ← punição                              ▷ código incorreto de ação
31:    end if
32:    return r
33: end procedure
```

duas saídas. Como resultado, para o contexto desta tarefa de navegação onde uma única entrada (estado do ambiente) pode estar relacionado a duas saídas (ações corretas), inviabiliza qualquer processo de aprendizado para uma rede MLP, inclusive através do algoritmo *backpropagation*.

Uma maneira plausível de resolver este problema é incorporar a propriedade de realimentação ao sistema de controle, característica esta presente em todo sistema nervoso biológico. Porque deseja-se que o robô ande para frente independente da direção inicial escolhida, ele precisa "lembrar" da ação executada anteriormente, para assim percorrer todo o labirinto evitando obstáculos. A realimentação da ação executada no passo de controle anterior, tanto à entrada da rede MLP, quanto ao processo de aprendizado on-line, modificando o controle neural para **MLP-R-RR**, rede MLP recorrente com aprendizado por reforço recorrente [Figura 5.8(a)], possibilitou esta memória e fez emergir o comportamento desejado [Figura 5.8(b)]. As respectivas mo-

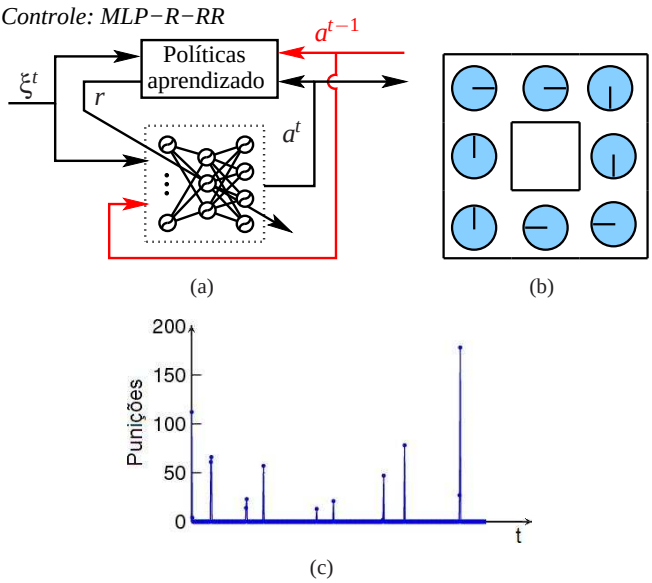


Figura 5.8: (a) Arquitetura do controle MLP-R-RR. (b) Comportamento de navegação resultante. (c) Execução contínua de punições durante aprendizado on-line da rede MLP.

dificações da linha 4 por $a^t \leftarrow MLP(\xi^t, a^{t-1})$ e da linha 5 por $r \leftarrow$ POLITICASAPRENDIZADO*$(\xi^t, a^t, a^{t-1}, MLP)$ no Algoritmo 5.1 são necessárias para

a implementação do controle neural **MLP-R-RR**. Enquanto as modificações no módulo **Políticas de aprendizado** recorrente são descritas no Algoritmo 5.2.

Algoritmo 5.2 Políticas de aprendizado com re-alimentação

1: **procedure** POLITICASAPRENDIZADO*(ξ^t, a^t, a^{t-1}, MLP)
2: **if** $a^t \in \{(1,0,0,0),\ (0,1,0,0),\ (0,0,1,0),\ (0,0,0,1)\}$ **then**
3: **if** $\xi^t = (\xi_o, \xi_n, \xi_l, \xi_s) = (0,0,0,0)$ **then**
4: **if** $a^t \neq a^{t-1}$ **then**
5: $r \leftarrow$ punição
6: **else**
7: $r \leftarrow$ recompensa ▷ preservar pesos MLP
8: **end if**
9: **end if**
10: **if** $\xi^t = (\xi_o, \xi_n, \xi_l, \xi_s) = (1,0,1,0)$ **then**
11: **if** $a^t = (1,0,0,0)$ **or** $a^t = (0,0,1,0)$ **then**
12: $r \leftarrow$ punição ▷ colisão
13: **else if** $\xi_n^t = 0$ **and** $a_n^{t-1} = 1$ **and** $a_n^t \neq a_n^{t-1}$ **then**
14: $r \leftarrow$ punição ▷ comportamento vai-e-vem
15: **else if** $\xi_s^t = 0$ **and** $a_s^{t-1} = 1$ **and** $a_s^t \neq a_s^{t-1}$ **then**
16: $r \leftarrow$ punição ▷ comportamento vai-e-vem
17: **else**
18: $r \leftarrow$ recompensa ▷ preservar pesos MLP
19: **end if**
20: **end if**
21: **if** $\xi^t = (\xi_o, \xi_n, \xi_l, \xi_s) = (0,1,0,1)$ **then**
$\cdots$ ▷ similar à política (10 - 20)
22: **end if**
23: **if** $\xi^t = (\xi_o, \xi_n, \xi_l, \xi_s) = (0,1,1,0)$ **then**
24: **if** $a^t = (0,1,0,0)$ **or** $a^t = (0,0,1,0)$ **then**
25: $r \leftarrow$ punição ▷ colisão
26: **else if** $a_n^{t-1} = 1$ **or** a_l^{t-1} **then**
27: **if** $(a_n^{t-1} = 1$ **and** $a_o^t \neq 1)$ **or** $(a_l^{t-1} = 1$ **and** $a_s^t \neq 1)$ **then**
28: $r \leftarrow$ punição ▷ comportamento vai-e-vem

Uma desvantagem desta abordagem porém é que mesmo o robô executando o comportamento correto, por meio das **Políticas de aprendizado** recorrente, o aprendizado on-line de uma única rede MLP não se realiza devido à própria arquitetura da rede e devido à característica aleatória do processo de treinamento em tempo de operação. Neste caso, a rede MLP continuamente

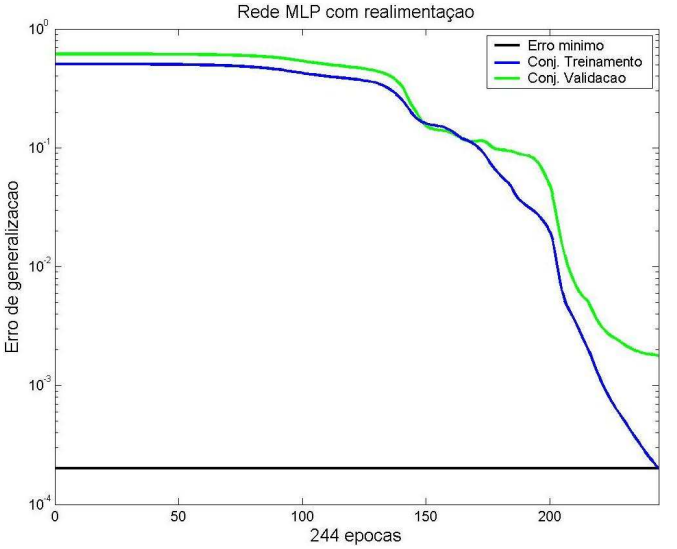
receberá punições, caracterizando seu não aprendizado, como ilustrado no gráfico da Figura 5.8(c).

Por outro lado, isto não ocorre, por exemplo, no caso em que são coletados, durante a execução do controle **MLP-R-RR**, os padrões de estados do ambiente junto com as ações corretas executadas pelo controle neural e apresentados para o treinamento off-line por *backpropagation* da mesma rede MLP, utilizada no arranjo. Neste caso se constata a convergência do aprendizado da rede na Figura 5.9(a). Já o gráfico da Figura 5.9(b) mostra como a ausência da realimentação da ação anterior à entrada da rede impede que a mesma aprenda um mapeamento correto.

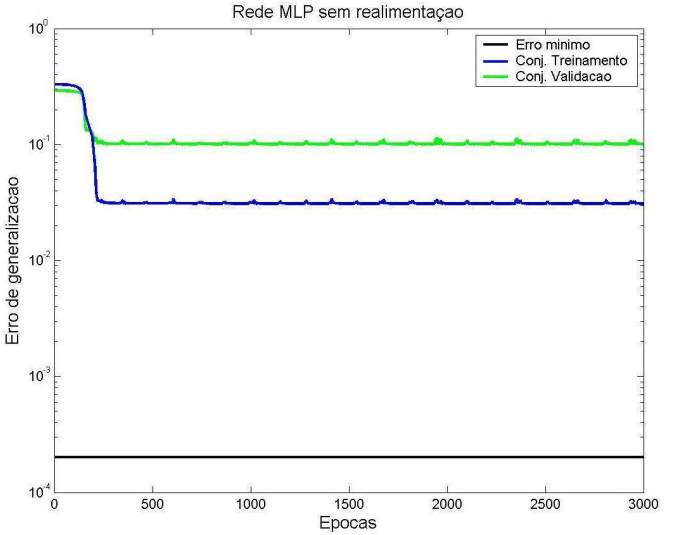
Algoritmo 5.2 Políticas de aprendizado com re-alimentação (cont.)

```
29:                 else
30:                     r ← recompensa                    ▷ preservar pesos MLP
31:                 end if
32:             else if a_o^{t-1} = 1 and a_o^t ≠ 1 then
33:                 r ← punição                           ▷ comportamento vai-e-vem
34:             else if a_s^{t-1} = 1 and a_s^t ≠ 1 then
35:                 r ← punição                           ▷ comportamento vai-e-vem
36:             else
37:                 r ← recompensa                        ▷ preservar pesos MLP
38:             end if
39:         end if
40:         if ξ^t = (ξ_o, ξ_n, ξ_l, ξ_s) = (1,0,0,1) then
                ...                                       ▷ similar à política (23 - 39)
41:         end if
            ...                                           ▷ Seção A.1
42:     else
43:         r ← punição                                   ▷ código incorreto de ação
44:     end if
45:     return r
46: end procedure
```

Portanto, a fim de resolver o problema do "esquecimento" do aprendizado em tempo de operação de uma rede MLP como controle neural, uma solução foi incorporar ao arranjo uma rede do tipo ART, dado que estas arquiteturas de RNAs formam sistemas neurais capazes de resolver o dilema conhecido como *plasticidade* $\times$ *estabilidade*.



(a)



(b)

Figura 5.9: Treinamento off-line da rede MLP por *backpropagation*. (a) Com realimentação da ação anterior à entrada da rede. (b) Sem a realimentação.

5.4 Segundo Ensaio: Redes ART e o Dilema da Plasticidade × Estabilidade

O dilema da plasticidade × estabilidade compreende a seguinte questão: "*Como um sistema de aprendizagem pode permanecer plástico, ou adaptativo, em resposta a eventos significantes, e ainda permanecer estável em respostas a eventos irrelevantes? Como alcançar estabilidade sem rigidez e plasticidade sem caos? Como preservar conhecimentos aprendidos anteriormente, enquanto continuam-se aprendendo coisas novas?*"(CARPENTER; GROSSBERG, 1988). Inúmeras topologias de redes neurais têm falhado ao tentar transpor esse dilema, ao passo que aprender um novo padrão significa esquecer o aprendizado anterior, como no caso do arranjo neural **MLP-R-RR** proposto como controle de um robô móvel na seção anterior. Todavia, as redes baseadas na teoria da ressonância adaptativa (*Adaptive Resonance Theory*) (GROSSBERG, 1976a; GROSSBERG, 1976b) dão suporte a esse problema (FREEMAN; SKAPURA, 1992).

A teoria da ressonância adaptativa se refere à teoria sobre o processamento de informação cognitiva humano que proporcionou o desenvolvimento de uma série de modelos de redes neurais de tempo real, com ambos os aprendizados não supervisionado e supervisionado, aplicadas tanto em áreas de reconhecimento de padrão como de predição (CARPENTER; GROSSBERG, 2003). Dentre os muitos tipos de redes ART existentes (CARPENTER; GROSSBERG; ROSEN, 1991a; CARPENTER; GROSSBERG; ROSEN, 1991b; CARPENTER; GROSSBER; REYNOLDS, 1991), neste trabalho de tese optou-se por utilizar a rede ART1 (CARPENTER; GROSSBERG, 1988), por sua forma de tratar os dados de entrada e saída como binários, visto que as implementações realizadas neste trabalho seguem este mesmo padrão de tratamento dos dados.

5.4.1 O Aprendizado ART1

Como ilustrado na Figura 5.10, a rede ART1 incorporada ao sistema de controle **MLP-R-RR** recebe como padrão de entrada a concatenação das informações sobre o estado do ambiente, ξ^t e a ação executada no passo de controle anterior, a^{t-1}. O aprendizado auto supervisionado da rede ART1 ocorre da seguinte maneira. Quando um padrão de entrada é apresentado, um ciclo de comparação de padrões entre as camadas F1 e F2 se inicia, para determinar se o padrão de entrada está entre os padrões previamente armazenados na rede.

O aprendizado significativo ocorre quando este ciclo termina em um estado ressonante, daí o termo "teoria da ressonância adaptativa". Este estado

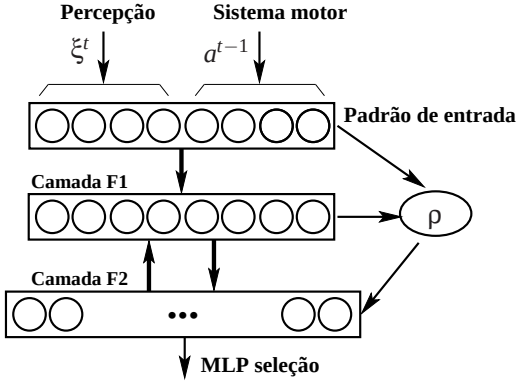


Figura 5.10: Rede ART1 incorporada ao controle **MLP-R-RR**.

ressonante implica que o sistema permitiu alterações em uma de suas classes de padrões já aprendidas, devido ao padrão de entrada ser suficientemente similar ao que ele já conhece, refinando desta maneira seu conhecimento. A finalização do ciclo de comparação de padrões também pode ocorrer com a seleção de um neurônio em F2 ainda não comprometido. Então os pesos adaptativos bottom-up e top-down ligados a este nó aprendem o padrão de ativação da camada F1, que é gerado diretamente pelo padrão de entrada, criando assim uma nova classe. Caso a capacidade total de memória acabe e nenhuma combinação adequada exista, o aprendizado é automaticamente inibido. O parâmetro de vigilância, ρ, determina o grau de similaridade permitido entre o padrão de entrada e a expectativa gerada por ele. Valores altos para ρ força o sistema a buscar por novas classes em resposta a pequenas diferenças entre os padrões. Desta forma, o sistema aprende uma classificação refinada dos padrões de entrada, com uma maior quantidade de classes. Valores baixos para ρ habilitam o sistema a tolerar grandes desigualdades e assim agrupar padrões de entrada de acordo com uma medida grosseira de similaridade.

Portanto, o aprendizado de uma rede ART1 consistirá em refinar sua codificação já aprendida, acomodando na mesma novas informações de forma segura via combinações aproximadas, ou selecionar novos neurônios para a inicialização do aprendizado de classes de reconhecimento novas, ou ainda proteger a capacidade de memória já comprometida, evitando que a mesma seja apagada por fluxos de novos padrões de entrada (CARPENTER; GROSSBERG, 1988). Devido a estas características de aprendizado, o próximo passo

na evolução da construção de uma arquitetura de controle inteligente buscada neste trabalho, foi incorporar uma rede do tipo ART1 para compor o arranjo neural de controle descrito na seção seguinte.

5.4.2 Controle Neural ART1-R-MLPs-RR

Controle neural reativo **ART1-R-MLPs-RR** é assim definido por corresponder a uma rede ART1 recorrente comutadora de redes MLPs com aprendizado por reforço recorrente. Como ilustrado na Figura 5.11(a) a rede ART1

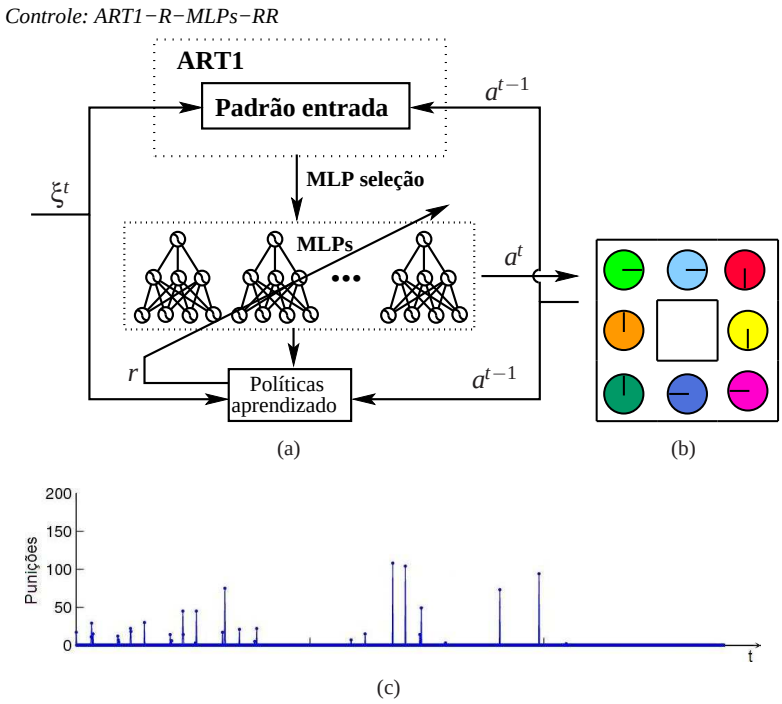


Figura 5.11: (a) Arquitetura do controle ART1-R-MLPs-RR. (b) Comportamento de navegação resultante. (c) Execução de punições durante aprendizado on-line do conjunto de redes MLPs.

recorrente foi inserida para o reconhecimento das diferentes percepções do robô, permitindo através de seu aprendizado auto-organizável a criação de uma memória de acionamento de redes MLPs que devem ser especializadas

a cada situação. Desta forma, elimina-se o constante re-aprendizado de uma mesma rede MLP, para situações de entrada diferentes, como era realizado no controle anteriormente apresentado.

Logo, a rede ART1 funciona como uma "chaveadora" de redes MLPs e a razão da utilização deste tipo de rede ao invés de uma porção *case* ou *if-then-else's* se justifica principalmente por duas razões. A primeira delas se refere às vantagens de seu aprendizado auto-organizável realizado em tempo de operação, junto com a importante característica de resolver o dilema da plasticidade e estabilidade, a partir de uma estrutura computacional mínima. Em segundo, pode-se considerar uma outra vantagem, referente a propriedade de generalização do aprendizado ART, o qual permite que o sistema de controle em questão seja aprendido através de padrões de percepção, que podem ser genéricos. Isto resulta em uma classificação ou categorização otimizada de ações, sem a necessidade de um conhecimento a priori sobre todas as padrões alternativos possíveis, que são provenientes da percepção do robô.

No novo arranjo neural proposto como controle, o padrão de entrada da rede ART1 possui 8 neurônios, os quais representam a percepção formada pelo estado do ambiente, ξ^t, mais a realimentação da ação de saída, atrasada em uma unidade de tempo a^{t-1}. À medida que o robô percorre corredores e reconhece diferentes percepções, a rede ART1 recorrente seleciona uma rede MLP indexada por seu neurônio vencedor na camada F2. Caso este índice seja fornecido pela primeira vez, uma nova rede MLP é alocada para corresponder ao novo índice. Deste modo, o conjunto de redes MLPs cresce à medida que novas classes (percepções) são detectadas ou pela característica de auto-organização do aprendizado da rede ART1. A rede MLP selecionada deve então responder a ação corrente a^t. Esta resposta apenas será enviada ao módulo **Sistema motor** caso seja correta e esta decisão depende do algoritmo de aprendizado on-line (Algoritmo 5.2) que é aplicada à MLP selecionada, através do módulo **Políticas de aprendizado**.

Como no controle anteriormente proposto, MLP-R-RR, a realimentação influencia em parte, a seleção da classe da rede MLP que deve responder a ação corrente a^t. A introdução da realimentação ao padrão de entrada da rede ART1 insere uma memória de curto termo que permite o aprendizado adaptativo de comportamentos cuja principal característica é fazer o robô andar para frente, de forma a manter o sentido de sua direção [Figura 5.11(b)]. Como a rede ART1 recorrente seleciona uma rede MLP específica a cada situação sendo que a rede MLP já não necessita mais ser re-alimentada, o aprendizado de todas as rede MLPs alocadas tendem a convergir no sentido da diminuição ou ausência de reforço como punição. Caso esta realimentação da ação anterior fosse inexistente não seria possível uma estabilização do aprendizado

das redes MLPs como no caso da Figura 5.11(c), dado que redes MLPs iguais seriam acessadas em situações diferentes como no caso da Figura 5.12.

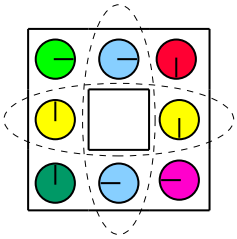


Figura 5.12: Comportamento de navegação resultante.

A implementação do controle **ART1-R-MLPs-RR** é apresentada no Algoritmo 5.3. A cada ciclo de controle, os módulos **Percepção** e **Sistema motor** fornecem as informações para a formação do padrão de entrada da rede ART1 recorrente, atribuindo-se a j a classe indicada pelo neurônio vencedor da camada F2 (linha 5). Esta classe j corresponde ao índice de uma rede MLP existente ou não no conjunto de redes MLPs. A seguir, a rede MLP invocada, MLP_j, produz a saída a^t (linha 6) a ser verificada pelo processo de treinamento online por reforço na linha 7, que corresponde ao Algoritmo 5.2. Se o sinal de reforço r retornado for punição (linhas 8 - 11), a rede MLP_j selecionada sofrerá a alteração de um de seus pesos, escolhido de forma aleatória. O índice deste peso sorteado é armazenado em uma fila do tipo FIFO (do acrônimo em inglês: *first in, first out*), que contém os índices de todos os pesos já selecionados, os quais não podem mais ser sorteados numa próxima etapa, se existirem ainda outros pesos nunca selecionados no ciclo de aprendizado corrente. Em seguida, uma nova simulação da rede MLP é realizada (linha 6). Por fim, dado que o sinal de reforço seja recompensa, devido a todas as condições da política de aprendizado serem satisfeitas, os pesos da rede MLP_j são mantidos e a ação a^t é então enviada ao módulo **Sistema motor** (linha 12) para ser executada pelo robô.

5.4.3 Considerações

A partir do arranjo neural **ART1-R-MLPs-RR** foram obtidos dois objetivos principais relacionados a um controle inteligente. O primeiro corresponde à convergência do aprendizado de todas as redes que compõe o arranjo neural, e segundo, um correto comportamento de navegação emergente (seguir paredes evitando obstáculos). A seguir são aumentadas as complexidades

Algoritmo 5.3 Controle **ART1-R-MLPs-RR**

1: **procedure** CONTROLEART1-R-MLPS-RR
2: **while** (; ;) **do**
3: $\xi^t \leftarrow$ **Percepcao**$(S_0, \ldots, S_7, \theta)$
4: $a^{t-1} \leftarrow$ **Sistema motor**
5: $j \leftarrow ART1(\xi^t + a^{t-1})$
6: $a^t \leftarrow MLP_j$
7: $r \leftarrow$ POLITICASAPRENDIZADO*$(\xi^t, a^t, a^{t-1}, MLP_j)$ ▷ Alg. 5.2
8: **if** $r =$ punição **then**
9: **Alterar peso da rede** MLP_j**.**
10: **goto** 6
11: **end if**
12: **Sistema motor**(a^t)
13: **end while**
14: **end procedure**

de ambos o labirinto e tarefa de navegação como um desafio para o controle neural apresentado nesta seção.

5.5 Terceiro Ensaio: Bifurcações e Becos sem Saída como Marcos

Esta seção apresenta uma estratégia para incorporar ao sistema **ART1-R-MLPs-RR** o aprendizado de um comportamento mais complexo, juntamente com o aumento da complexidade do labirinto através da inclusão de bifurcações e becos sem saída. No desenvolvimento desta estratégia alguns assuntos foram considerados. Primeiro, com a modificação do labirinto, o comportamento desejado é de que o robô, além de seguir paredes evitando obstáculos, aprenda uma trajetória livre de becos sem saída. A segunda questão seria investigar se através do aprendizado deste arranjo neural pudesse emergir a construção de um mapa cognitivo que resultasse em um sistema de navegação topológico, por exemplo.

Como o conceito de marco é um elemento fundamental na construção de sistemas de navegação mais complexos, a estratégia utilizada neste trabalho inicialmente foi relacionar bifurcações e becos sem saída como estados do ambiente que indicam marcos. E uma vez que um marco é alcançado, o sistema necessita de um processo de tomada de decisão dado a existência de mais de uma opção de ação correta, nas bifurcações. Deste modo seriam necessárias algumas modificações na arquitetura **ART1-R-MLPs-RR** para o tratamento de bifurcações e becos sem saída no labirinto como marcos e

assim incorporar, ao aprendizado do mapeamento de percepções-ações, um sistema de tomada de decisão implícito. Uma nova arquitetura proposta foi definida como **ART1-R-MLPs-RR-Marcos** por apresentar o mesmo arranjo neural da abordagem anterior **ART1-R-MLPs-RR** (Seção 5.4.2). As modificações necessárias à arquitetura para incorporar o conceito de marcos ao controle inteligente englobaram o módulo **Percepção**, o padrão de entrada da rede **ART1 recorrente** e o módulo de **Políticas de aprendizado** que serão vistos na sequência.

5.5.1 Acréscimos ao Módulo Percepção

Como visto no início da Seção 5.2.1, o módulo **Percepção**, através dos dados dos sensores de proximidade e ângulo de orientação, fornece a informação do estado do ambiente ξ^t, que corresponde à configuração de paredes no instante t. Com a inclusão de bifurcações no labirinto, o módulo **Percepção** deve reconhecer um padrão de posicionamento que centralize o robô a cada ocorrência de bifurcação em T, caracterizada pela junção de três corredores (p. ex. Figura 5.13). Este posicionamento, localizado ao centro de cada junção (círculos pretos) garante ao robô a possibilidade de mudar sua direção, sem colidir com paredes.

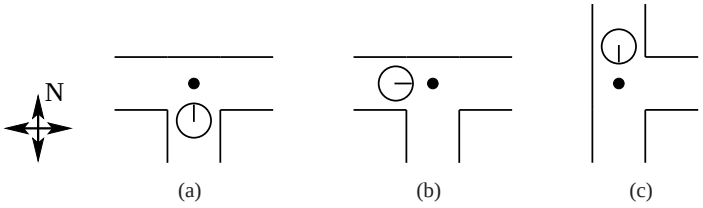


Figura 5.13: Exemplos de detecção de bifurcações.

Para detectar estes "pontos" de posicionamento seguro, o módulo **Percepção** primeiramente necessita diferenciar estados do ambiente entre três tipos de padrões: corredor, bifurcação e beco-sem-saída. A Tabela 5.1 relaciona cada tipo de padrão a seus respectivos valores possíveis como estado do ambiente, ξ^t. Como pode ser visto na tabela, todos os valores que podem representar um padrão bifurcação também estão incluídos na representação de um padrão corredor. A maneira como o módulo **Percepção** diferencia um padrão bifurcação é descrito a seguir. Em linhas gerais, após o robô sair de um estado do ambiente do tipo corredor com paredes paralelas, $\xi^t = \{(0,1,0,1),(1,0,1,0)\}$, podem acontecer dois casos. Primeiro,

Padrão	$\xi^t =$	Indicadores
corredor	$\{(0,0,0,0),(0,0,1,1),(0,1,1,0),$ $(1,0,0,1),(1,1,0,0),(0,1,0,1),$ $(1,0,1,0),(0,0,0,1),(0,0,1,0),$ $(0,1,0,0),(1,0,0,0)\}$	**bifur** $=$ **beco** $= F$
bifurcação T	$\{(0,0,0,1),(0,0,1,0),(0,1,0,0),$ $(1,0,0,0)\}$	**bifurc** $= V$
beco sem saída	$\{(0,1,1,1),(1,0,1,1),(1,1,0,1),$ $(1,1,1,0)\}$	**beco** $= V$

Tabela 5.1: Padrões de estado do ambiente diferenciados pelo módulo Percepção.

os próximos estados podem ser aqueles onde nenhuma parede é detectada, $\xi^t = \{(0,0,0,0)\}$, seguidos por um estado onde é detectada uma parede à frente do robô, como na Figura 5.13 (a), logo este estado é definido como uma bifurcação. Ao invés disto, no segundo caso, se o próximo estado for um daqueles onde é percebida parede apenas em uma das laterais do robô, Figuras 5.13 (b) e (c), estabelece-se uma contagem de "*n*" próximos estados cujo enésimo estado será uma bifurcação, caso não seja detectada parede à frente do robô.

Portanto, durante a execução do robô as bifurcações em T e os becos sem saídas são detectados a fim de serem associados a marcos e tratados pelo controle neural **ART1-R-MLPs-RR-Marcos**. Por esta razão, o módulo **Percepção** gerencia os indicadores **bifurc** e **beco** inicializando-os como falsos F, quando ocorre um estado do ambiente de padrão corredor. Caso contrário, assim que é detectada uma bifurcação ou um beco sem saída, os respectivos indicadores são inicializados com o valor verdadeiro V. Estes indicadores são fornecidos juntamente com o estado do ambiente ξ^t para o controle neural que será descrito a seguir.

5.5.2 Controle Neural ART1-R-MLPs-RR-Marcos

O controle executado pelo arranjo neural **ART1-R-MLPs-RR** ao ser aplicado em um labirinto como o da Figura 5.14(c), torna-se ineficiente para o aprendizado do comportamento esperado, que corresponde ao robô executar uma trajetória livre de obstáculos e becos sem saída. Isto ocorre porque em bifurcações não é suficiente para o robô apenas manter a direção de seu movimento anterior ou, aprender a tomar outra direção correta qualquer.

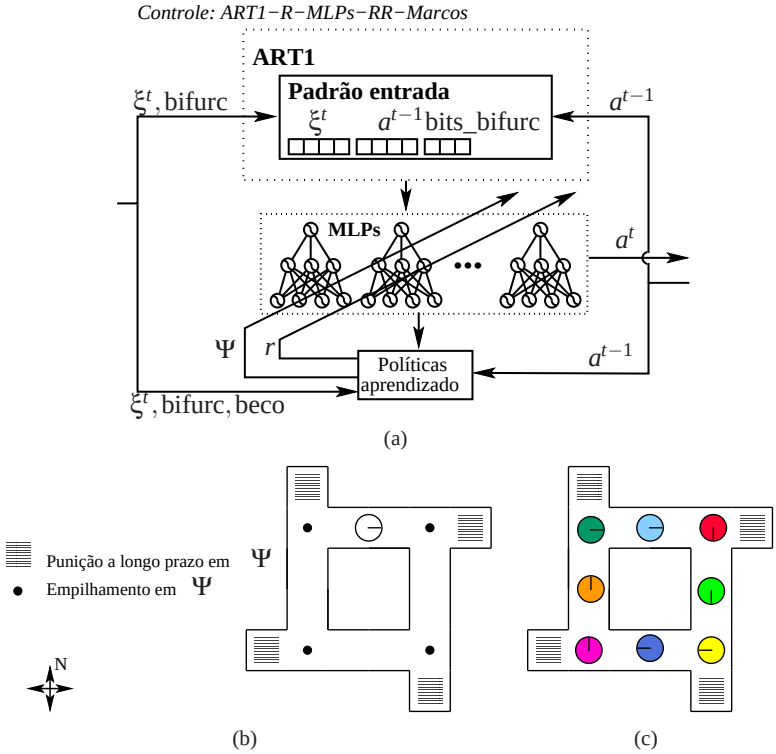


Figura 5.14: (a) Arquitetura do controle neural ART1-R-MLPs-RR-Marcos. (b) Atuação do módulo **Políticas de aprendizado**. (c) Comportamento de navegação resultante.

A complexidade reside no fato de que em uma bifurcação, o robô necessita aprender a tomar decisões mediante duas ou mais opções corretas possíveis.A Figura 5.14(a) ilustra o controle neural **ART1-R-MLPs-RR-Marcos** que em essência constitui o mesmo arranjo proposto anteriormente, **ART1-R-MLPs-RR**, porém com modificações necessárias para tratar bifurcações e becos sem saídas como marcos. Estes marcos caracterizam lugares no labirinto onde o controle neural deve reagir de maneira diferente de como reage na execução de corredores. Como resultado, a abordagem proposta nesta seção permite obter tanto o aprendizado da trajetória desejada, quanto o aprendizado de todas as redes presentes no arranjo neural. A Figura 5.14(c) ilustra um exemplo

de trajetória obtida onde as diferentes cores do robô indicam redes MLPs distintas, alocadas pela rede ART1 recorrente e treinadas em tempo de operação.

Algoritmo 5.4 Controle **ART1-R-MLPs-RR-Marcos**

```
1:  procedure CONTROLEART1-R-MLPS-RR-MARCOS
2:      while ( ; ; ) do
3:          ξ^t, birfurc, beco ← Percepcao(S_0, ..., S_7, θ)
4:          a^{t-1} ← Sistema motor
5:          if birfurc then
6:              bits_bifurc ← (1,1,1)
7:          else
8:              bits_bifurc ← (0,0,0)
9:          end if
10:         j ← ART1(ξ^t + a^{t-1} + bits_bifurc)
11:         a^t ← MLP_j
12:         r ← P_A_MARCOS(ξ^t, birfurc, beco, a^t, a^{t-1}, j, MLP) ▷ Alg. 5.5
13:         if r = punição then
14:             Alterar peso da rede MLP_j.
15:             goto 11
16:         end if
17:         Sistema motor(a^t)
18:     end while
19: end procedure
```

O Algoritmo 5.4 descreve o controle neural proposto. A primeira modificação se refere ao aumento de mais três bits, denominados "**bits_bifurc**", ao padrão de entrada da rede ART1 recorrente, para indicar a ocorrência de uma bifurcação de acordo com a sinalização fornecida pelo módulo **Percepção**. Caso o indicador de bifurcação seja verdadeiro, estes bits recebem a sequência "$(1,1,1)$", caso contrário recebem "$(0,0,0)$". Este acréscimo de bits ao padrão da rede ART provê uma diferenciação eficiente dos estados do ambiente tais como $\xi^t = \{(0,0,0,1),(0,0,1,0),(0,1,0,0),(1,0,0,0)\}$ que podem ser classificados tanto como corredores ou bifurcações. Como em uma bifurcação o robô deve apresentar um comportamento de tomada de decisão, a rede ART1 deve selecionar uma rede MLP específica para este aprendizado.

Já o módulo de **Políticas de aprendizado** implementa um processo de tomada de decisão como treinamento on-line das redes MLPs na ocorrência de marcos (bifurcação e beco sem saída), como indicado na Figura 5.14(b). Durante a execução de corredores, o módulo **Políticas de aprendizado** monitora as respostas das redes MLPs "localmente" como na abordagem anterior,

Algoritmo 5.5 Políticas de aprendizado com tratamento de marcos

```
1:  procedure P_A_MARCOS(ξ^t, birfurc, beco, a^t, a^{t-1}, j, MLP)
2:      if a^t ∈ {(1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)} then
3:          if ξ^t = (ξ_o, ξ_n, ξ_l, ξ_s) = (1,0,0,0) then
4:              if a^t = (1,0,0,0) then
5:                  r ← punição
6:              else if ¬bifurc and a_o^{t-1} = 0 and a^{t-1} ≠ a^t then
7:                  r ← punição
8:              else if ¬bifurc and a_o^{t-1} = 1 and a_l^t = 1 then
9:                  r ← punição
10:             else if bifurc then
11:                 if a_o^{t-1} = 1 and a_l^t = 1 then
12:                     r ← punição
13:                 else if a_n^{t-1} = 1 and a_s^t = 1 then
14:                     r ← punição
15:                 else if a_s^{t-1} = 1 and a_n^t = 1 then
16:                     r ← punição
17:                 else
18:                     r ← recompensa
19:                     Ψ^indice ← j                          ▷ Empilhamento
20:                     Ψ^acao ← a^t
21:                     Ψ^{acao−alternativa} ← a^alternativa
22:                 end if
23:             else    r ← recompensa
24:             end if
25:         end if
    ...                                                       ▷ Seção A.2
26:         if beco then                                      ▷ Punição de longo prazo
27:             a^aux ← MLP_{Ψ^indice}
28:             while a^aux ≠ Ψ^{acao−alternativa} do
29:                 Alterar um peso qualquer da rede MLP_{Ψ^indice}.
30:                 a^aux ← MLP_{Ψ^indice}
31:             end while
32:         end if
33:     else
34:         r ← punição                                       ▷ código incorreto de ação
35:     end if
36:     return r
37: end procedure
```

através de regras de aprendizado e aplicação do reforço r. De outro modo, quando o robô se encontra em uma bifurcação, o índice da rede MLP selecionada é empilhado em Ψ, juntamente com a ação corrente a ser executada, dada como resposta por esta rede, a^t, além da outra ação correta alternativa. Isto é realizado visto que o robô, ao alcançar um beco sem saída, necessita que sua rede MLP de índice em Ψ leve punição até aprender a outra ação alternativa correta, a fim de evitar novamente seguir na direção do beco sem saída corrente, em uma próxima ocasião.

Uma descrição do módulo **Políticas de aprendizado** com tratamento de marcos é dada no Algoritmo 5.5. O processo definido como punição de longo prazo (linhas 26 a 32) é aquele que atuará apenas na ocorrência de becos sem saída, punindo a rede MLP que em uma bifurcação anterior tomou uma ação responsável por dirigir o robô até lá. A princípio, em uma bifurcação, uma rede MLP recebe reforço recompensa localmente por responder quaisquer das duas ações corretas (linha 18). Porém é possível que uma delas possa levar à direção de um beco sem saída. Desta forma Ψ armazena estas duas opções corretas e qual foi a ação escolhida pela rede MLP na bifurcação (linhas 19 a 21), para que o processo de punição de longo prazo possa fazer com que esta rede MLP responda a outra ação alternativa, evitando assim que o robô venha novamente alcançar o beco sem saída corrente.

5.5.3 Considerações

O sistema de controle neural **ART1-R-MLPs-RR-Marcos** mostrou-se eficiente no aprendizado de trajetórias livres de obstáculos e becos sem saída, porém os labirintos utilizados devem apresentar um determinado padrão conforme descrito na sequência. Os perímetros dos labirintos devem ser formados por sucessivas bifurcações em T onde cada bifurcação deve estar associada a um beco sem saída. Além disto, a disposição espacial de cada associação "bifurcação-beco" deve ser mantida. Por exemplo, os labirintos (a) e (b) da Figura 5.15 satisfazem esta condição onde cada associação espacial "bifurcação-beco" ocorre duas vezes em ambos os labirintos. Cada símbolo marcando bifurcações corresponde a uma bifurcação que se distingue por sua configuração de paredes. Já o labirinto (c) satisfaz a condição do perímetro ser formado por bifurcações contendo becos sem saída, porém a disposição espacial destas associações são contraditórias tornando inviável o aprendizado por reforço das redes MLPs.

A contradição por exemplo está relacionada à bifurcação "$\times$" que no canto superior esquerdo do labirinto (c) está associada a um beco sem saída na direção oeste, enquanto que no canto direito acima a mesma bifurcação

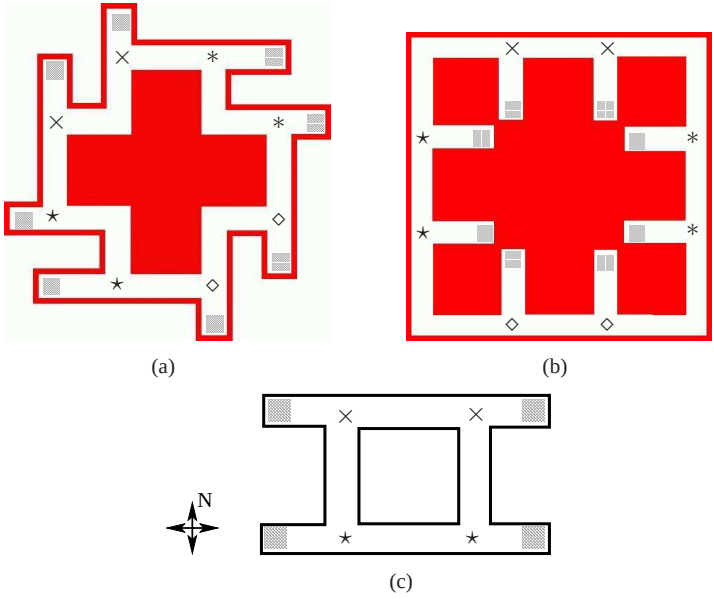


Figura 5.15: Bifurcações e becos sem saída e o controle neural **ART1-R-MLPs-RR-Marcos**. Cada símbolo ($\times$, $*$, $\diamond$, $\star$) nos labirintos correspondem a uma bifurcação distinta. (a) e (b) Exemplos de labirintos onde o sistema funciona. (c) Um exemplo de caso onde o sistema falha.

ocorre, porém com um beco sem saída localizado a leste. Isto caracteriza o problema do erro de percepção onde duas bifurcações em lugares diferentes parecem ser idênticas para o controle neural, segundo a discriminação realizada pela rede ART1. Ou seja, o problema ocorre quando o robô em momentos distintos alcança as bifurcações "$\times$" através da direção norte, por exemplo. Pois o rede MLP selecionada pela rede ART1 recorrente nestas bifurcações será a mesma e as ações corretas para evitar beco sem saída serão contrárias, considerando-se as direções distintas dos becos sem saída em questão. Por esta razão o sistema **ART1-R-MLPs-RR-Marcos** neste caso falha tanto no aspecto de não conseguir realizar o aprendizado online das redes MLPs, quanto por não obter a trajetória desejada. E assim não foi possível fazer da abordagem **ART1-R-MLPs-RR-Marcos** um sistema neural de controle flexível e adaptativo, apenas através da detecção e tratamento de marcos incorporado a seu processo de aprendizado de mapeamentos percepção-ação.

5.6 Conclusão

Este capítulo apresentou o desenvolvimento de uma série de arquiteturas reativas, onde foram utilizados dois tipos de RNAs (MLPs e ART1) com algoritmos de aprendizado em tempo de operação. Diferentes acoplamentos destas redes foram dispostos como componentes principais dos sistemas de navegação propostos. O principal objetivo foi investigar quais tipos de comportamento de navegação poderiam ser implementados, e se de fato ocorre um aprendizado efetivo de todas as redes incorporadas ao sistema neural de controle. Investigou-se também o funcionamento destas arquiteturas por meio do aumento da complexidade do labirinto, através da inclusão de bifurcações cujo um dos caminhos alternativos leva a um beco sem saída.

A primeira estratégia de controle inteligente foi implementada através de uma única rede MLP com treinamento online por reforço, que não apresentou um aprendizado satisfatório por não resolver o dilema da plasticidade e estabilidade. A segunda estratégia foi incorporar ao sistema neural uma rede ART1 que, através de seu aprendizado auto-organizável, permitiu a um robô móvel autônomo navegar por um labirinto simples evitando obstáculos, ao mesmo tempo memorizando o caminho percorrido através de um aprendizado eficiente de mapeamentos de percepção-ação. Assim foi obtido a arquitetura reativa **ART1-R-MLPs-RR**, classificada como uma estratégia de navegação biologicamente inspirada do tipo **Resposta ao Disparo de Reconhecimento**, pertencente ao grupo de comportamentos espaciais *descobrimento de caminho* sem planejamento (Seção 3.3.5).

Com o aumento da complexidade do labirinto, uma terceira estratégia foi construir um sistema neural que utilizasse de forma eficiente o conceito de marcos para obter o aprendizado de um comportamento de navegação mais complexo, tal como o aprendizado de desviar de becos sem saída a cada bifurcação encontrada. E assim foi desenvolvida a abordagem **ART1-R-MLPs-RR-Marcos**. Entretanto, a desvantagem apresentada por esta arquitetura foi seu funcionamento apenas em labirintos com um determinado padrão espacial, tornando-se assim pouco flexível e não adaptativa.

Desta maneira, como o acréscimo de bifurcações no labirinto induz a necessidade de comportamentos mais complexos, tais como planejamento de caminho, a próxima estratégia foi construir uma camada deliberativa, buscando inspiração biológica através da teoria dos mapas cognitivos, cujo desenvolvimento é descrito no próximo capítulo. Esta camada deliberativa a ser incorporada ao controle inteligente, cuja base reativa é implementada pelo arranjo neural **ART1-R-MLPs-RR**, formam a arquitetura final proposta neste trabalho de tese, denominada **NeuroCog**.

CAPÍTULO 6

Arquitetura Neural Cognitiva: NeuroCog

6.1 Introdução

O Capítulo 5 apresentou uma investigação sobre quais tipos de comportamentos podem ser obtidos através do desenvolvimento e evolução de sucessivos arranjos de redes neurais com aprendizado adaptativo e em tempo de operação, para navegação em labirintos. Esta investigação deu origem a uma arquitetura neural reativa denominada **ART1-R-MLPs-RR**, que constitui de uma rede ART1 recorrente comutadora de redes MLPs cujo aprendizado por reforço também é recorrente. Esta arquitetura reativa provê a um robô móvel autônomo os comportamentos de percorrer corredores e evitar obstáculos em labirintos simples, sem bifurcações.

Este capítulo descreve o desenvolvimento de uma arquitetura híbrida como proposta de controle inteligente de um robô móvel para navegação em labirintos dinâmicos. A arquitetura se baseia no estilo híbrido de construção de arquiteturas de controle, dado que este paradigma tem suas raízes na etologia e provê um arcabouço teórico para a exploração de ciências cognitivas (MURPHY, 2000).

A abordagem proposta é denominada **NeuroCog** devido à sua camada reativa ser implementada através do arranjo de rede neurais **ART1-R-MLPs-RR**, e à sua camada deliberativa ser formada por um processo de mapeamento cognitivo adaptativo. A maneira como são integradas ambas as camadas reativa e deliberativa configura uma das contribuições deste trabalho de tese. Esta integração é realizada através do sistema de percepção do robô o qual também representa um nível intermediário na arquitetura proposta.

Modelos de navegação híbridos deliberativo e reativo, propostos como mapeamentos cognitivos e ou que utilizaram técnicas neurais para navegação

em labirintos, foram estudados para obtenção de idéias na criação e integração da camada deliberativa da arquitetura **NeuroCog**. Ambientes do tipo labirinto, tão populares no contexto da navegação de robôs, são boas plataformas de teste para o desenvolvimento de soluções estratégicas. Haja vista este tipo de ambiente constituir o ponto de partida na construção de sistemas mais complexos, tais como (VOICU; SCHMAJUK, 2001; VOICU, 2003; JAN; CHANG; PARBERRY, 2003; LUMELSKY, 1991). O problema da navegação de um robô em labirinto também pode ser facilmente estendido para outros cenários, tais como planejar movimentos diante de obstáculos, procurar por vítimas em escombros, vistoriar tubulações de esgoto, entre outras.

Entretanto, o objetivo deste trabalho de tese não esteve voltado a questões de plausibilidade biológica, embora o princípio de funcionamento da arquitetura **NeuroCog** seja inspirado biologicamente, tanto em nível deliberativo, através do mapeamento cognitivo, como em nível reativo, através do controle por meio do arranjo neural proposto. O mapeamento cognitivo no sistema **NeuroCog** é uma metáfora do comportamento de um rato que deixa marcas de odor (BURES et al., 1998) em localidades onde encontra pistas salientes no labirinto, para em seguida relacioná-las e armazená-las, como forma de representação do ambiente explícita, em seu mapa cognitivo biológico.

Outra questão importante a ser considerada foi a maneira de como integrar de maneira funcional e eficiente ambos controles reativo e deliberativo, a fim de produzir um sistema de navegação computacional híbrido, facilmente aplicável a um robô real com limitados recursos sensórios e computacionais, e ao mesmo tempo direcionado à resolução de uma tarefa de navegação complexa.

6.2 Aspectos Gerais

Como ilustrada na Figura 6.1(a), a arquitetura **NeuroCog** é baseada em comportamentos e segue o estilo híbrido de construção de arquiteturas de controle inteligente, segundo o modelo de subsunção. A abordagem proposta apresenta três camadas onde a camada deliberativa, implementada pelo sistema de **Mapeamento cognitivo**, é responsável pelos comportamentos de exploração e planejamento de caminho. A camada reativa provê os comportamentos reativos através do arranjo neural **ART1-R-MLPs-RR** (Seção 5.4.2). E finalmente a camada mais baixa é composta pelos módulos **Percepção** e **Sistema motor**. As camadas deliberativa e reativa são competitivas, o módulo **Percepção** realiza a interface entre deliberação e reação e decide sobre qual das camadas acima irá determinar a ação a ser executada. Através da subsunção **S**, a ação de saída da camada deliberativa tem prioridade sobre a

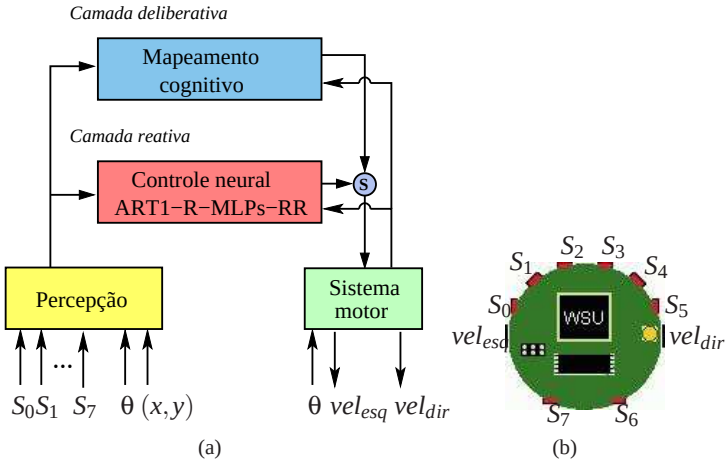


Figura 6.1: (a) A arquitetura **NeuroCog** e (b) o robô simulado.

saída da camada reativa, a ser enviada ao módulo **Sistema motor**, que ao final do ciclo de controle, computa as velocidades das rodas esquerda e direita (vel_{esq}, vel_{dir}), que são fornecidas aos atuadores, para fazer com que o robô ande em linha reta, em uma das direções oeste, norte, leste ou sul.

Assim como visto na Seção 5.2, o robô simulado, Figura 6.1(b), não possui informação a priori e conta apenas com seus sensores de proximidade ($S_0, \ldots, S_7$), ângulo de orientação (θ) e posição (x,y).

A arquitetura **NeuroCog** habilita a navegação autônoma de um robô móvel em ambientes do tipo labirinto, cujas bifurcações apresentam formato em T. Os labirintos podem ser modificados durante a execução da tarefa de navegação executada pelo robô. Por exemplo, seja o labirinto da Figura 6.2, à medida que o robô alcança becos sem saída e bifurcações (círculos azuis), o módulo **Percepção** estabelece o conceito de marcos e invoca o sistema de **Mapeamento cognitivo**, na camada deliberativa. O reconhecimento de bifurcações no labirinto é realizado assim como descrito na Seção 5.5.1. Ao passo que, durante a execução dos corredores (círculos de cor rosa) o robô está sob o controle neural reativo **ART1-R-MLPs-RR**.

Um das principais funções do sistema de **Mapeamento Cognitivo** é construir uma representação de conhecimento explícita do ambiente, através de informação sensória. Esta representação corresponde a um mapa topológico, como por exemplo, representado pelo grafo da Figura 6.2(b), após uma

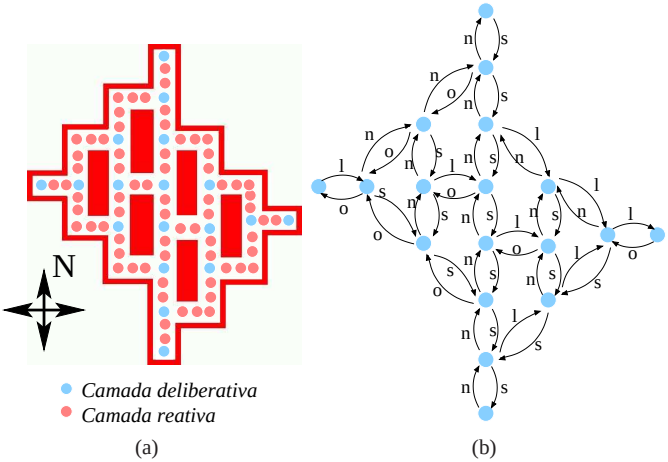


Figura 6.2: Arquitetura **NeuroCog**: exemplo de ambiente e representação de conhecimento. (a) Um exemplo de labirinto do tipo T e a atuação das camadas deliberativa e reativa ativadas pelo módulo **Percepção** durante execução do robô. (b) O grafo representando o mapa cognitivo do labirinto (b) aprendido após uma exploração completa. Os nós indicam os becos sem saída ou bifurcações visitados e as arestas as respectivas ações: o (oeste), n (norte), l (leste) e s (sul), para sair de um nó e alcançar o nó vizinho.

exploração completa do labirinto em (a). Os nós no grafo representam os marcos encontrados durante a exploração, e as arestas armazenam dois tipos de informação:

1. a ação correspondente para sair de um nó e alcançar o nó vizinho, que pode ser o (oeste), n (norte), l (leste) ou s (sul) e

2. uma medida de distância obtida pela contagem de ciclos de controle executados entre um nó e seu nó vizinho, para o planejamento de caminho.

6.2.1 Labirintos Dinâmicos

O processo de aprendizado adaptativo de mapa topológico realizado pelo sistema de **Mapeamento Cognitivo** trata com a possibilidade de ocorrerem modificações no labirinto, ao mesmo tempo que equilibra o dilema *exploration versus exploitation*, durante a realização da tarefa de navegação.

Este dilema é analisado sob dois aspectos a serem vistos a seguir. As modificações incluem bloqueios de caminhos conhecidos e anteriormente livres e novas aberturas anteriormente bloqueadas ou inexistentes. Estes tipos de modificações no labirinto durante a operação do robô visam demonstrar a flexibilidade e a adaptabilidade da abordagem **NeuroCog**, mediante a dinâmica do ambiente, mesmo após uma fase de aprendizado do sistema.

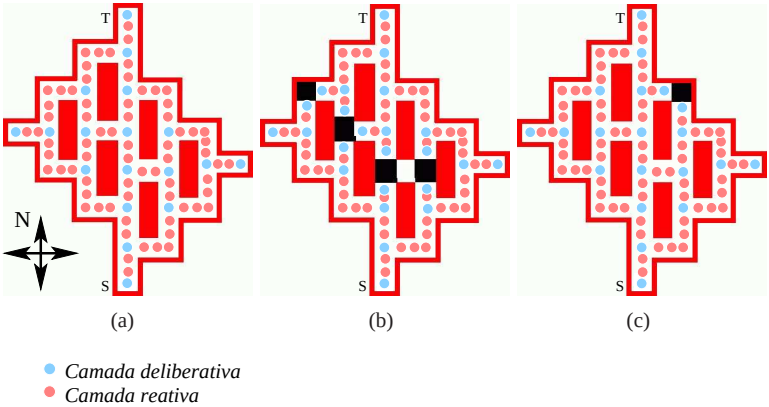


Figura 6.3: Modificações no ambiente e a atuação das camadas de controle da arquitetura **NeuroCog**. (a) O labirinto em uma fase inicial: diversos caminhos disponíveis entre o lugar origem **S** e o objetivo **T**. (b) Primeira modificação, apenas um caminho disponível entre S e T. (c) Segunda modificação, o único caminho entre S e T é bloqueado e os caminhos anteriores desbloqueados.

Por exemplo, a Fig. 6.3 ilustra sucessivas modificações que podem ocorrer no labirinto (a), enquanto o robô deve sair de S, alcançar T e retornar a S. Em (a) pode-se visualizar a representação topológica do labirinto aprendida após uma exploração completa, onde os círculos azuis representam os nós do mapa topológico. Após a primeira modificação ocorrida em (b), o mapa topológico incorpora novos nós, à medida que o robô busca caminhos entre S e T, e os encontra bloqueados. Em (c), uma nova modificação bloqueia o único caminho livre entre S e T, e desbloqueia os caminhos anteriormente interceptados. Com isto, as ligações entre nós primeiramente identificados no mapa topológico em (a) são refeitas e os nós incorporados em (b) são ignorados devido à mudança nas configurações de parede do labirinto.

As mudanças no ambiente fazem com que o robô modifique as direções das sub-trajetórias que compõem o caminho até o objetivo. Sob o aspecto da camada reativa, isto implica em alterações na seqüência de padrões

de percepção apresentados ao arranjo neural **ART1-R-MLPs-RR**, o que requer uma readaptação do aprendizado do mapeamento percepção-ação. As seções a seguir apresentam as etapas de construção e funcionamento de cada módulo que compõe a arquitetura híbrida **NeuroCog**.

6.3 Percepção Híbrida

O módulo **Percepção** na arquitetura **NeuroCog** utiliza o conceito de "marco" o qual é um elemento fundamental na construção de mapas cognitivos e navegação topológica (Seções 3.3.5 e 3.3.6). Por esta razão, este módulo reconhece e identifica marcos como becos-sem-saída e bifurcações em T, para que a camada deliberativa possa construir e manter o mapa cognitivo do ambiente, bem como decidir sobre a ação a ser tomada de maneira eficiente. A estratégia de relacionar estes tipos de padrões de estado do ambiente a marcos identificáveis foi inspirada no comportamento de ratos, que segundo os estudos de Bures et al. (1998), deixam marcas de odor em localidades onde encontram pistas salientes no labirinto, para então relacioná-las e armazená-las em seu mapa cognitivo biológico.

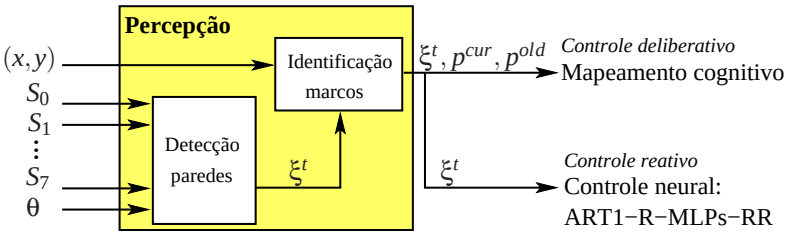


Figura 6.4: O módulo **Percepção** na arquitetura híbrida **NeuroCog**.

O módulo **Percepção** híbrida, como ilustrado na Figura 6.4, funciona assim como descrito na Seção 5.2.1. A cada ciclo de controle, o processo **Detecção de paredes** recebe como entrada dois tipos de informação: a primeira alotética, fornecida pelos sensores de proximidade $(S_0,\ldots,S_7)$ e a segunda, idiotética, referente ao ângulo de orientação, θ, para detectar a presença de paredes nas direções oeste, norte, leste e sul, definindo assim o estado do ambiente $\xi^t = (\xi_o, \xi_n, \xi_l, \xi_s)$ corrente. Também como discutido na Seção 5.2.1, estas direções poderiam se referir da mesma forma às direções relativas à pose inicial do robô: esquerda, frente, direita e volta.

Já o processo **Identificação marcos** classifica o padrão ξ^t em "corredor", "bifurcação" ou "beco sem saída" como ilustrado na Tabela 6.1 e

descrito na Seção 5.5.1. Na ocorrência destes dois últimos tipos de padrão,

Padrão	$\xi^t =$	Controle ativo
corredor	$\{(0,0,0,0),(0,0,1,1),$ $(0,1,1,0),(1,0,0,1),$ $(1,1,0,0),(0,1,0,1),$ $(1,0,1,0),(0,0,0,1),$ $(0,0,1,0),(0,1,0,0),$ $(1,0,0,0)\}$	**ART1-R-MLPs-RR**
bifurcação T beco sem saída	$\{(0,0,0,1),(0,0,1,0),$ $(0,1,0,0),(1,0,0,0)\}$ $\{(0,1,1,1),(1,0,1,1)$ $,(1,1,0,1),(1,1,1,0)\}$	**Mapeamento Cognitivo**

Tabela 6.1: Padrões de estado do ambiente diferenciados pelo módulo **Percepção**.

a informação idiotética (x,y), referente à posição do robô, é utilizada para a identificação de marcos que equivalem a locais específicos no labirinto. A cada marco alcançado, estabelece-se a informação referente a p^{cur}, que corresponde ao lugar corrente onde o robô se encontra, ao passo que p^{old} equivale ao lugar anteriormente visitado, com relação a p^{cur}. Ambas identificações são enviadas à camada deliberativa, juntamente com a informação do estado do ambiente (configuração de paredes).

Portanto, a partir da classificação de estados do ambiente, estabelece-se uma forma de integrar deliberação e reação (Tabela 6.1). Ou seja, quando o tipo de ξ^t é reconhecido como corredor, a camada reativa **ART1-R-MLPs-RR** é invocada pelo módulo **Percepção** híbrida para assumir o controle do robô. Ao contrário, quando o tipo de ξ^t é reconhecido como beco sem saída ou bifurcação, estabelece-se o conceito de marcos identificáveis, onde o módulo de **Mapeamento cognitivo** deve ser invocado para o controle do robô. A seguir, o módulo **Sistema motor** híbrido é descrito com mais detalhes.

6.4 Sistema Motor Híbrido

A Figura 6.5 ilustra o módulo **Sistema motor** híbrido que atua a cada ciclo de controle, onde recebe como entrada, a ação corrente a ser executada, $a^t = (a_o, a_n, a_l, a_s)$. Esta ação pode ser determinada ou pela camada reativa através do arranjo neural **ART1-R-MLPs-RR**, ou pela camada deliberativa

através do módulo de **Mapeamento cognitivo**, de acordo com o reconhecimento do estado do ambiente, ξ^t, realizado pelo módulo **Percepção**. Através da subsunção **S**, a camada deliberativa tem prioridade quanto à decisão da ação corrente a^t, quando ξ^t é reconhecido como um padrão bifurcação ou beco sem saída. Ao passo que a camada reativa decide sobre a ação corrente a^t, quando ξ^t é reconhecido como um padrão corredor.

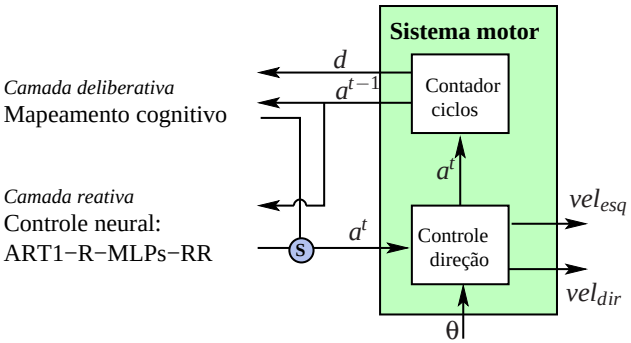


Figura 6.5: Arquitetura **NeuroCog**: o módulo **Sistema Motor**.

Além disto, o módulo **Sistema motor** também é responsável por fornecer, através do processo **Contador ciclos**, a ação executada no ciclo de controle atrasado em uma unidade de tempo, a^{t-1}, para ambos os módulos **Mapeamento Cognitivo** e **ART1-R-MLPs-RR**. No módulo reativo **ART1-R-MLPs-RR**, a ação executada anteriormente, a^{t-1}, é utilizada tanto na formação do padrão de entrada da rede ART1, quanto na busca aplicada à base de regras, para o treinamento online das redes MLPs (Seções 5.4.2 e 5.3.3). O processo **Contador ciclos** ainda fornece para o módulo **Mapeamento cognitivo**, além de a^{t-1}, uma medida de distância, d, que equivale ao número de ciclos executados entre dois lugares respectivamente visitados. Esta distância é utilizada tanto na construção e manutenção do mapa cognitivo, quanto no planejamento de caminho, que serão vistos adiante.

O processo de **Controle direção** mediante a ação corrente a^t a ser executada, calcula a partir do ângulo de orientação do robô, θ, se é mais efetivo girar no sentido horário ou anti-horário, para posicionar a frente do robô na direção da ação a^t. Em seguida, as velocidades esquerda e direita, vel_{esq} e vel_{dir} respectivamente, são fornecidas aos atuadores, para que o robô se mova, em linha reta, na direção de a^t.

6.5 Controle Deliberativo NeuroCog

Como visto na Seção 6.3, o robô não usa visão e conta apenas com a informação de seus sensores de proximidade, ângulo de orientação e posição. A partir destes dados, o módulo **Percepção** detecta tipos de padrões de estado do ambiente desviando o controle, ora para a camada reativa (quando o tipo identificado é corredor), ora para a camada deliberativa (quando o tipo identificado é beco sem saída ou bifurcação). Como o conceito de marco é um elemento fundamental na construção de mapas cognitivos e navegação topológica, o módulo **Percepção** utiliza a identificação de padrões de estado do ambiente do tipo becos-sem-saída e bifurcações em T, como marcos identificáveis para construção do mapa cognitivo.

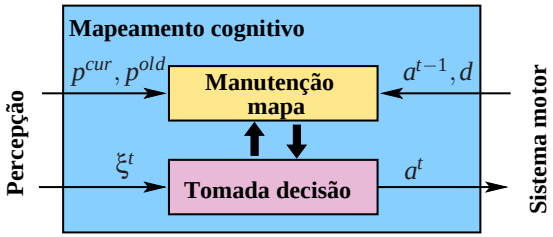


Figura 6.6: Arquitetura **NeuroCog**: camada deliberativa

A Figura 6.6 mostra a interface do módulo de **Mapeamento cognitivo** na camada deliberativa, com os módulos **Percepção** e **Sistema motor**. A cada detecção de marco, o módulo **Percepção** executa as seguintes ações:

i) relaciona a posição (x,y) do robô a uma identificação que corresponde ao lugar corrente, p^{cur};

ii) captura o estado do ambiente em p^{cur}, $\xi^t = (\xi_o, \xi_n, \xi_l, \xi_s)$ (Equação 5.1) e

iii) obtém p^{old}, que equivale ao lugar anteriormente visitado, com relação a p^{cur}.

O valor inicial de p^{cur} é 0, sendo incrementado de um a cada novo lugar detectado. Assim, $p^{cur}, p^{old} \in \mathbb{Z}$, $(0 \leq p^{cur}, p^{old} \leq \mathcal{L} - 1)$, onde $\mathcal{L}$ equivale ao total de lugares encontrados. Com estas informações, o módulo **Percepção** então desvia o controle para o módulo **Mapeamento cognitivo**.

O controle deliberativo ainda recebe do módulo **Sistema motor** as informações referentes à ação executada no passo de controle anterior, a^{t-1} e

a distância, d, percorrida entre ambos os lugares p^{old} e p^{cur}. Esta medida de distância equivale à quantia de ciclos de controle executados pelo sistema **NeuroCog** entre os respectivos lugares. A partir das informações obtidas, o módulo **Mapeamento cognitivo** constrói e mantém o mapa cognitivo através do subsistema de **Manutenção de mapa**, e em seguida decide sobre a próxima ação, a^t, a ser executada, através do subsistema de **Tomada de decisão**. Ambos os subsistemas são apresentados com mais detalhes nas subseções seguintes.

6.5.1 Manutenção de Mapa

O subsistema **Manutenção mapa** é responsável por gerenciar a construção e manutenção do mapa cognitivo, que corresponde a um mapa topológico obtido através de informações do tipo alotética e idiotética. Como pode ser visto na Figura 6.7, o mapa cognitivo, $\mathcal{M}$, é representado por um grafo, onde os nós representam lugares (becos sem saída e bifurcações no labirinto) e as arestas armazenam informações referentes a ações de saída e chegada entre os nós, mais uma medida de distância percorrida. O subsistema de **Manutenção de mapa** recebe as seguintes informações de entrada, através dos módulos:

i) **Percepção**: p^{cur} e p^{old}, ambas do tipo idiotética e alotética,

ii) **Sistema motor**: a^{t-1} e d ambas do tipo idiotética, e

iii) **Tomada de decisão**: $\mathcal{A}$, de origem idiotética e alotética.

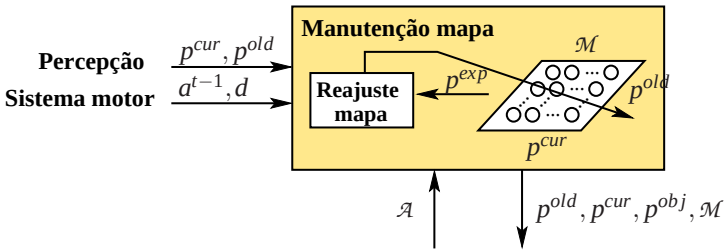


Figura 6.7: Controle deliberativo: o subsistema **Manutenção de mapa**.

A implementação do subsistema de **Manutenção mapa** que corresponde à primeira parte do módulo de **Mapeamento cognitivo**, é descrita no Algoritmo 6.1. A continuação do algoritmo, que implementa o subsistema **Tomada de decisão**, é apresentado na seção seguinte.

Para a construção do mapa cognitivo, $\mathcal{M}$, são definidas $out_{p^{old}}$ e $in_{p^{cur}}$, que correspondem as ações de saída em p^{old} e chegada em p^{cur}, respectivamente. A ação de saída em p^{old} é obtida a partir de $\mathcal{A}$ (Algoritmo 6.1, linha 2), $\mathcal{A}$ é definida e mantida pelo subsistema **Tomada de decisão** e será vista na seção seguinte. Já a ação oposta a de chegada em p^{cur} é obtida diretamente através da inversão de a^{t-1} (Algoritmo 6.1, linha 3). Porém, antes de inserir $out_{p^{old}}$, $in_{p^{cur}}$ e d em $\mathcal{M}$, o subsistema **Manutenção mapa** verifica a ocorrência de possíveis modificações no ambiente. Para tal, defini-se p^{exp}, que indica o lugar esperado segundo as informações contidas em $\mathcal{M}$, caso existentes (Algoritmo 6.1, linha 4). A verificação testa duas condições para constatar que o labirinto não sofreu alteração, onde apenas uma delas necessita ser verdadeira.

Algoritmo 6.1 Parte 1: **Mapeamento cognitivo**

1: **procedure** MANUTENCAOMAPA(p^{old}, p^{cur}, a^{t-1}, d, $\mathcal{M}$, $A^{explorer}$, $\mathcal{A}$)
2: $out_{p^{old}} \leftarrow \nabla(\mathcal{A}(p^{old}))$
3: $in_{p^{cur}} \leftarrow inversa(a^{t-1})$
4: $p^{exp} \leftarrow \cdot(\mathcal{M}, p^{old}, out_{p^{old}})$
5: **if** $\neg(p^{exp} = \emptyset$ **or** $p^{exp} = p^{cur})$ **then** ▷ Reajuste de mapa
6: $\mathcal{M}_{p^{old},p^{exp}} \leftarrow \mathcal{M}_{p^{exp},p^{old}} \leftarrow \emptyset$
7: $A^{explorer}_{\mathcal{M}_{p^{exp},p^{old}}}(p^{exp}) \leftarrow 1$
8: **end if**
9: $\mathcal{M}_{p^{old},p^{cur}} \leftarrow out_{p^{old}}$
10: $\mathcal{M}_{p^{cur},p^{old}} \leftarrow in_{p^{cur}}$
11: $\mathcal{M}_{p^{old},p^{cur}} \leftarrow \mathcal{M}_{p^{cur},p^{old}} \leftarrow d$
12: **end procedure**

O primeiro teste compara p^{exp} a vazio, para atestar se p^{cur} é visitado pela primeira vez. Já o segundo teste compara p^{exp} a p^{cur}, para atestar se p^{cur} já existe em $\mathcal{M}$, sendo neste caso re-visitado (Algoritmo 6.1, linha 5). Portanto, sem a ocorrência de modificações no ambiente, as informações pertinentes a $\mathcal{M}$ são diretamente inseridas (Algoritmo 6.1, da linha 9 a 11).

Caso contrário, o subsistema de **Manutenção mapa** (Figura 6.7) efetua as correções necessárias, onde $\mathcal{M}$ é atualizado para contemplar a modificação no ambiente. Esta correção é realizada em dois passos. Primeiro, desfazendo as ligações entre p^{old} e p^{exp} (Algoritmo 6.1, linha 6) e segundo, re-inicializando o peso do comportamento de exploração em p^{exp} para o valor máximo (Algoritmo 6.1, linha 7). Esta re-inicialização é necessária pois permite ao lugar p^{exp}, caso re-visitado, ter um peso maior na direção da ação

$\mathcal{M}_{p^{exp},p^{old}}$, cuja direção leva a uma região do ambiente modificada. A variável $A^{explorer}$, cujos valores representam pesos para o comportamento de exploração, será explicitada na Seção 6.5.2. Realizadas as correções, o subsistema **Reajuste mapa** então insere as novas informações em $\mathcal{M}$ (Algoritmo 6.1, da linha 9 a 11).

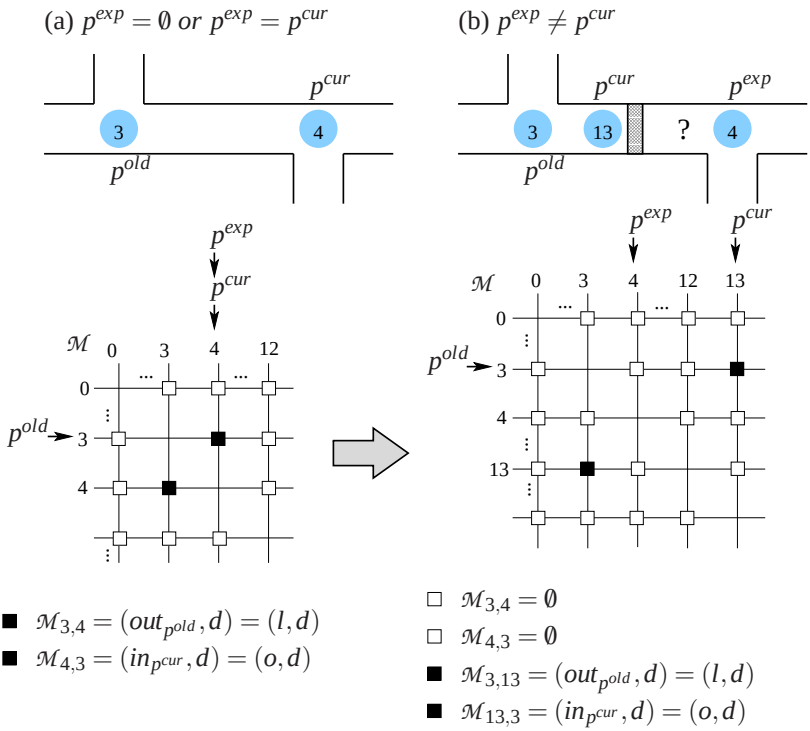


Figura 6.8: Um exemplo de modificação no ambiente. Em (a) nenhuma mudança é detectada. Em (b) uma mudança é detectada e portanto $\mathcal{M}$ deve ser corrigido.

A Figura 6.8 exemplifica um caso onde, em um primeiro momento, (a), o subsistema de **Manutenção mapa** não detecta mudança no ambiente, mas em um instante posterior, (b), uma modificação é detectada. Na Figura 6.8(a) o robô sai do lugar anterior, p^{old}, igual a 3 e ao alcançar o lugar corrente, p^{cur}, igual a 4, **Manutenção mapa** atesta ser verdadeiro que p^{cur} já existe em $\mathcal{M}$ e portanto é re-visitado ($p^{cur} = p^{exp} = 4$). Desta forma, **Manutenção mapa** insere diretamente as informações em $\mathcal{M}$, a fim de reforçar

em especial a medida de distância d, executada entre os lugares anterior e corrente, a qual pode sofrer constantes variações devido à imprecisão de atuadores.

Já a Figura 6.8(b) mostra o momento em que a modificação do ambiente é detectada. Segundo as informações existentes em $\mathcal{M}$, p^{exp} é retornado como 4, quando o robô alcança p^{cur} igual a 13. A criação deste novo lugar 13 ocorre devido ao obstáculo encontrado no caminho entre os lugares 3 e 4, e ao total de lugares encontrados até o momento ser $\mathcal{L} = 14$. Sendo assim, o subsistema **Reajuste mapa** é acionado para efetuar as devidas atualizações em $\mathcal{M}$, a fim de que os efeitos desta modificação no ambiente não interfiram na resolução da tarefa de navegação.

6.5.2 Tomada de Decisão

Como visto anteriormente, o módulo **Mapeamento cognitivo** na camada deliberativa da arquitetura **NeuroCog** está dividido em dois subsistemas, **Manutenção mapa** e **Tomada de Decisão**. O controle deliberativo atua quando o robô se encontra posicionado em uma bifurcação ou beco-sem-saída no labirinto. A Figura 6.9 ilustra o subsistema **Tomada de decisão** que determina a ação corrente, a^t, baseada no ajuste entre os comportamentos de explorar o ambiente e planejar caminho. Os comportamentos de explorar e planejar são implementados pelos subsistemas **"Explorer"** e **"Planner"** respectivamente. O subsistema **Explorer** tem por objetivo agregar informações

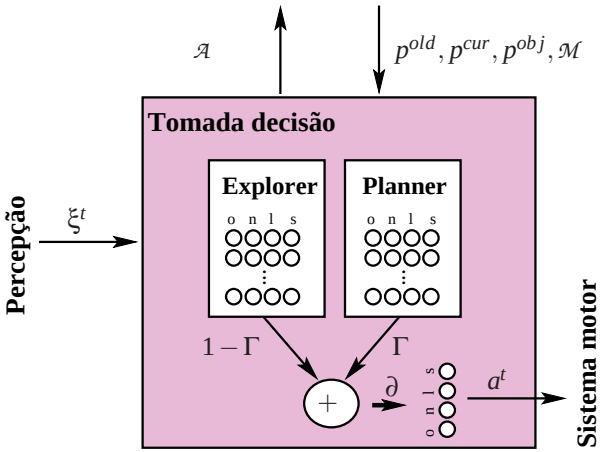


Figura 6.9: Controle deliberativo: o subsistema **Tomada de decisão**.

ao mapa cognitivo de forma a mantê-lo atualizado o quanto possível, priorizando assim áreas do ambiente ainda não exploradas e áreas que tenham sofrido modificações. Já o subsistema **Planner** utiliza-se do mapa cognitivo visando obter o menor caminho entre o lugar corrente e o lugar objetivo, caso este tenha sido descoberto.

Algoritmo 6.2 Parte 2: **Mapeamento cognitivo**

procedure TOMADADECISÃO(ξ^t, p^{old}, p^{cur}, $\mathcal{M}$, $A^{explorer}$, $A^{planner}$, $\mathcal{A}$)
  if $\neg(p^{cur} \in [0, L-1])$ **then**
    $A^{explorer}(p^{cur}) \leftarrow \overline{\xi^t}$ ▷ Explorer
  end if
  $A^{explorer}_{in_{p^{cur}}}(p^{cur}) \leftarrow \gamma \cdot A^{explorer}_{in_{p^{cur}}}(p^{cur})$
  $A^{explorer}_{out_{p^{old}}}(p^{old}) \leftarrow \gamma \cdot A^{explorer}_{out_{p^{old}}}(p^{old})$
  $\alpha \leftarrow 0$
  if $p^{obj} \neq \emptyset$ **then** ▷ Planner
    $A^{planner}(p^{cur}) \leftarrow$ **dijkstra**$(\mathcal{M}, p^{cur}, p^{obj})$
    for all $A^{explorer}_i(p^{cur}) = 1$ **do**
      $A^{planner}_i(p^{cur}) \leftarrow 1 - \Gamma$
    end for
    $\alpha \leftarrow 1$
  end if
  $\mathcal{A}(p^{cur}) \leftarrow (1-\Gamma) \cdot A^{explorer}(p^{cur}) + \alpha \cdot \Gamma \cdot A^{planner}(p^{cur})$
  $a^t \leftarrow \partial(\mathcal{A}(p^{cur}))$
  Sistema motor(a^t)
end procedure

Os subsistemas **Explorer** e **Planner** competem entre si e se utilizam de conhecimentos adquiridos previamente. O primeiro se utiliza da própria experiência de exploração e o segundo se utiliza do mapa cognitivo, $\mathcal{M}$. Ambos os comportamentos de explorar e planejar são conflitantes, pois à medida que o primeiro predomina, o desempenho do segundo pode ser comprometido, devido ao desperdício de tempo na exploração de áreas do ambiente sem relevância ou já exploradas. Por outro lado, pouca exploração pode comprometer o planejamento em obter caminhos mais curtos e alternativos até o lugar objetivo. O dilema consiste em quando cessar a exploração e priorizar o planejamento, dado que o ambiente pode ser alterado a todo momento.

Como pode ser visto na Figura 6.9, o subsistema **Tomada de decisão**, através das informações recebidas a partir do subsistema **Manutenção mapa** ($p^{old}, p^{cur}, p^{obj}, \mathcal{M}$), implementa uma maneira de equilibrar os dois

comportamentos a fim obter um bom desempenho para a tarefa de navegação. A estratégia utilizada baseia-se em uma das técnicas propostas por Thrun (1992), que trata o dilema *exploration versus exploitation* no contexto da aprendizagem por reforço para navegação reativa. Enquanto que na abordagem proposta neste trabalho de tese, o dilema é tratado no nível deliberativo, comparando-se a repercussão que o ajuste do dilema desempenha no nível reativo da arquitetura **NeuroCog**. Ambos os assuntos são analisados através de simulações que serão apresentadas no próximo capítulo. É importante observar que a camada deliberativa da abordagem proposta neste trabalho não foi construída baseando-se na técnica da AR, como ocorre em parte do aprendizado do arranjo neural na camada reativa. De fato, como será visto na sequência, o subsistema **Tomada Decisão** implementa o comportamento de exploração com base apenas em descontos de pesos de ações alternativas, em lugares específicos e reconhecidos do labirinto, enquanto o comportamento de planejamento calcula o menor caminho, utilizando-se de um algoritmo de busca sobre o mapa aprendido.

O Algoritmo 6.2 descreve o subsistema **Tomada Decisão**, que corresponde à segunda parte do módulo **Mapeamento cognitivo**. O subsistema **Explorer** define e mantém a quádrupla $A^{explorer}$ a cada lugar alcançado no ambiente. O conteúdo de $A^{explorer}$ armazena pesos para cada ação alternativa, onde a ação com maior peso equivale à ação menos executada no respectivo lugar. Isto permite que o comportamento de exploração priorize as regiões do ambiente ainda desconhecidas ou menos exploradas. Assim, quando um lugar corrente, p^{cur}, é visitado pela primeira vez, $A^{explorer}$ é inicializada de forma:

$$A^{explorer}(p^{cur}) \quad \leftarrow \quad \overline{\xi^t} \text{ (Algoritmo 6.2, linha 3), onde:}$$

$$A_i^{explorer}(p^{cur}) = \begin{cases} 1 & \text{se não existe obstáculo na direção } i, \\ 0 & \text{caso contrário,} \\ & \text{para } i = o,\ n,\ l,\ s. \end{cases} \tag{6.1}$$

Isto implica que para todas as ações alternativas em p^{cur} é atribuído o peso máximo àquelas que não estão bloqueadas por obstáculo, no contexto do comportamento de exploração. Em seguida, o subsistema **Explorer** atualiza os valores dos pesos das ações executadas tanto no lugar anterior, ação $out_{p^{old}}$, como no lugar corrente, ação $in_{p^{cur}}$, utilizando um fator de desconto fixo $\gamma \lessapprox 1$ (Algoritmo 6.2, linhas 5 e 6):

$$A_i^{explorer}(p^{cur}) \leftarrow \gamma \cdot A_i^{explorer}(p^{cur}) \text{ onde } i = in_{p^{cur}} \quad (6.2)$$
$$A_i^{explorer}(p^{old}) \leftarrow \gamma \cdot A_i^{explorer}(p^{old}) \text{ onde } i = out_{p^{old}}$$

Isto faz com que o comportamento de exploração seja guiado pela experiência adquirida no passado, onde as ações mais recentemente executadas tem seus pesos descontados, para que as outras ações alternativas possam ser priorizadas posteriormente.

Como visto na seção 6.5.1, o subsistema **Manutenção mapa** detecta uma alteração no ambiente, através da informação de que o lugar corrente, p^{cur}, é diferente do lugar esperado, p^{exp}. Então o mapa cognitivo é reajustado e o peso da ação que liga p^{exp} ao lugar anterior p^{old} é inicializado (Algoritmo 6.1, linha 7):

$$A^{explorer}_{\mathcal{M}_{p^{exp},p^{old}}}(p^{exp}) \leftarrow 1$$

Esta inicialização do peso da ação $\mathcal{M}_{p^{exp},p^{old}}$ em $A^{explorer}$, para o lugar esperado, p^{exp}, permite que uma região do ambiente anteriormente modificada, seja novamente priorizada no contexto do comportamento de exploração.

Para competir com o subsistema **Explorer**, o subsistema **Planner** define e mantém a quádrupla $A^{planner}$ a cada lugar alcançado no ambiente. Ele atua somente depois que o lugar objetivo, p^{obj}, é descoberto (Algoritmo 6.2, linha 8). Para tal, a variável α será zero enquanto o lugar objetivo não for encontrado, caso contrário, α será 1 (Algoritmo 6.2, linhas 7 e 13). Para cada lugar corrente alcançado, o conteúdo de $A^{planner}$ é calculado a partir do algoritmo de Dijkstra (1959) sobre o mapa cognitivo (Algoritmo 6.2, linha 9):

$$A^{planner}(p^{cur}) \leftarrow \textbf{dijkstra}(\mathcal{M}, p^{cur}, p^{obj}) \quad (6.3)$$

O algoritmo de Dijkstra utiliza a informação de distância, d, presente em $\mathcal{M}$ (Algoritmo 6.1, linha 11), que equivale à quantia de ciclos de controle executados, a cada par de lugares p^{old} e p^{cur} respectivamente visitados. Como resultado, o algoritmo de Dijkstra retorna o peso máximo de 1 para aquela ação em $A^{planner}(p^{cur})$ que conduz a p^{obj} pelo menor caminho, conforme a soma de d's. Para todas as outras ações alternativas é atribuído o peso igual a zero. Além disto, para que haja um equilíbrio entre os comportamentos de explorar e planejar, o subsistema **Planner** atribui àquelas ações em p^{cur} ainda

não exploradas, o seguinte peso: (Algoritmo 6.2, da linha 10 a 12):

$$A_i^{planner}(p^{cur}) \quad \leftarrow \quad 1-\Gamma \quad \forall\, A_i^{explorer}(p^{cur}) = 1$$

Isto faz com que estas ações, nunca exploradas em p^{cur}, recebam um reforço no contexto do comportamento de planejar caminho, além de já estarem com o peso máximo relativo ao comportamento de explorar o ambiente.

Calculados $A^{explorer}(p^{cur})$ e $A^{planner}(p^{cur})$, o subsistema **Tomada de decisão** decide sobre a próxima ação a^t através da quádrupla $\mathcal{A}(p^{cur}) = (\mathcal{A}_o, \mathcal{A}_n, \mathcal{A}_l, \mathcal{A}_s)$, que combina os comportamentos de explorar ambiente e planejar caminho, através do parâmetro Γ $(0 < \Gamma < 1)$ (Algoritmo 6.2, linha 15):

$$\mathcal{A}(p^{cur}) \quad \leftarrow \quad (1-\Gamma)\cdot A^{explorer}(p^{cur}) + \alpha\cdot\Gamma\cdot A^{planner}(p^{cur}) \qquad (6.4)$$

Como visto anteriormente, a variável α executa o chaveamento onde a quádrupla $A^{planner}(p^{cur})$ apenas contribui na escolha da próxima ação caso o lugar objetivo tenha sido encontrado. Em seguida, a função ∂ determina a ação a^t, considerando a componente de maior peso em $\mathcal{A}(p^{cur})$ (Algoritmo 6.2, linha 16):

$$a^t \leftarrow \partial(\mathcal{A}(p^{cur})) \qquad (6.5)$$

Caso ocorra empate dentre os valores dos pesos das ações alternativas, o critério de desempate priorizará uma ação conforme a seqüência oeste, norte, leste e sul. Em seguida a ação a^t é enviada ao módulo **Sistema motor** como resposta da camada deliberativa da arquitetura **NeuroCog** (Algoritmo 6.2, linha 17).

Exemplo

A Figura 6.10 ilustra um exemplo de funcionamento do subsistema **Tomada decisão** no instante em que o robô encontra-se em p^{cur}=4, tendo p^{old}=3 e p^{obj}=5, durante execução do labirinto (a). Em (b), os subsistemas **Explorer** e **Planner** efetuam as operações, a partir do mapa cognitivo, $\mathcal{M}$, necessárias à decisão da ação a^t, (quadrados vermelhos e azul respectivamente). O subsistema **Explorer**, através da Equação 6.2, produz $A^{explorer}(p^{old}) = A^{explorer}(3) = (0.8, 0.7, 0.7, 0)$ e $A^{explorer}(p^{cur}) = A^{explorer}(4) = (0.7, 0, 0.8, 0.8)$, com fator de desconto $\gamma = 0.999$. Estes valores indicam que o robô já visitara antes estes locais, inclusive detectando o lugar 5 como objetivo. E ainda, segundo o subsistema **Explorer**, as ações alternativas "leste" e "sul" em p^{cur} concorrem com pesos iguais. O subsistema **Planner** utiliza distâncias

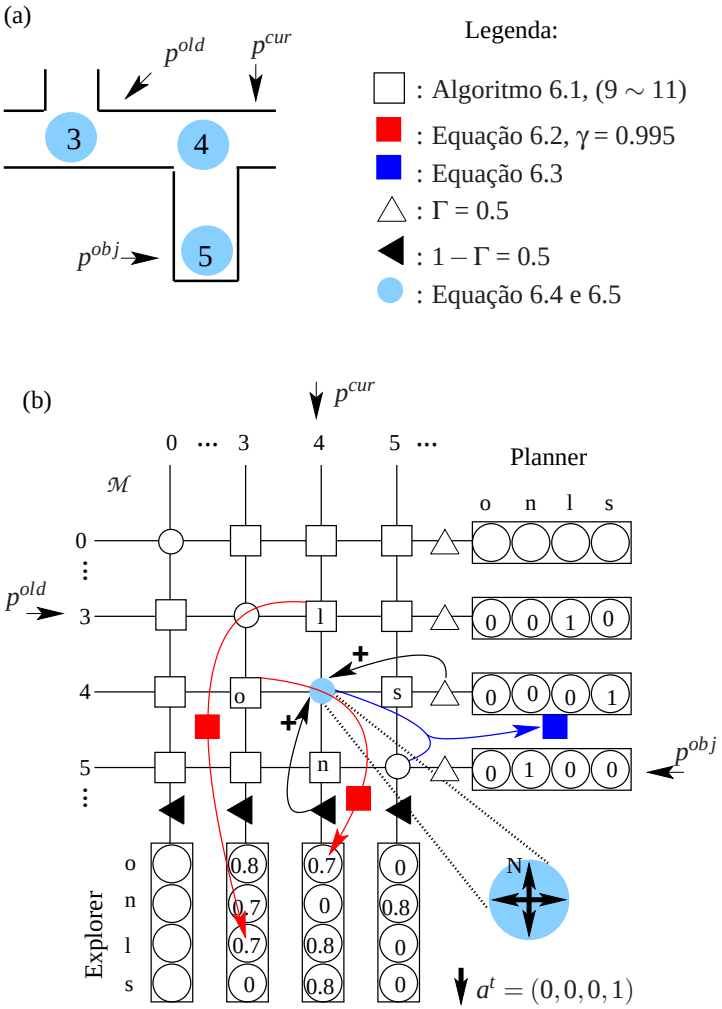


Figura 6.10: Exemplo de funcionamento do subsistema **Tomada de decisão**: o momento em que o subsistema provê a ação a^t = sul, como resposta do controle deliberativo da arquitetura **NeuroCog**.

entre lugares envolvidos nos possíveis caminhos entre p^{cur} e p^{obj}, armazenadas em $\mathcal{M}$ (quadrados brancos não vazios), para calcular, através do algoritmo Dijsktra, Equação 6.3, a ação que compõe o caminho mais curto até o lugar p^{obj}. Assim, $A^{planner}(p^{cur}) = A^{planner}(4) = (0,0,0,1)$, indica que a ação "sul" é a ação que levará o robô ao objetivo de maneira mais eficiente, com relação à soma de d's. Calculados os valores de atração das ações alternativas em p^{cur}, para ambos os comportamentos de explorar ambiente e planejar caminho, a seguir, no círculo azul em p^{cur}, o valor do parâmetro Γ na Equação 6.4 (triângulos) determinará os valores de atração finais para as ações alternativas. Em seguida, a Equação 6.5 irá estabelecer a ação resultante como $a^t = (0,0,0,1)$, por possuir o maior peso, devido a $\Gamma = 0.5$.

Tratando o Dilema "*Exploration*" versus "*Exploitation*"

O parâmetro Γ na Equação 6.4 estabelece o comportamento global do robô. No sistema **NeuroCog**, o valor de Γ é estático, permanecendo constante durante toda a execução da tarefa de navegação. Quando Γ é igual a 0, o robô exclusivamente explora o ambiente, sem apresentar comportamento de planejar caminho até o lugar objetivo, ainda que o mesmo seja encontrado. Ao contrário, quando Γ é igual a 1, ele planeja um caminho apenas entre os lugares de origem e o próximo encontrado, pois o comportamento de exploração é inexistente. O mapa cognitivo neste caso torna-se reduzido em apenas dois lugares, devido à ausência do comportamento de exploração.

Portanto, valores limites para Γ tornam-se ineficazes devido ao desempenho insatisfatório do robô para a tarefa de navegação proposta. Por outro lado, quando Γ é igual a 0.5, o comportamento do robô apresenta um equilíbrio entre ambos os comportamentos de explorar novas ações e considerar o conhecimento do mapa cognitivo para o planejar caminhos mais curtos até o objetivo. Sendo assim, o desempenho do sistema **NeuroCog** mostra-se eficiente, ainda que haja constantes mudanças no ambiente. O valor do parâmetro Γ igual a 0.5, faz com que o robô, em um primeiro momento, priorize o comportamento de exploração. Mas, à medida que ele não encontra mais novidades durante o percurso (tanto modificações no ambiente, quanto ações inexploradas), planejar caminho torna-se o comportamento predominante. Uma análise do desempenho do sistema **NeuroCog**, mediante diferentes valores de Γ será mostrada no Capítulo 7.

6.6 Trabalhos Relacionados

Como exposto anteriormente, para a construção da arquitetura **NeuroCog**, foram estudados alguns modelos de navegação presentes na literatura de robôs móveis autônomos. Estes modelos caracterizam-se por serem biologicamente inspirados, sendo propostos como sistemas de mapeamento cognitivo e ou baseados em técnicas de redes neurais, para navegação *indoor* ou em labirintos. Alguns exemplos das propostas estudadas foram (MATARIC, 1992b), (SCHMAJUK; THIEME, 1992), (SCHöLKOPF; MALLOT, 1995), (HAFNER, 2005), (ARLEO; MILLáN; FLOREANO, 1999), (WYETH; BROWNING, 1998), (VOICU; SCHMAJUK, 2000), (VOICU; SCHMAJUK, 2002), entre outras. A seguir são apresentados com mais detalhes as principais características arquiteturais e funcionais de alguns destes modelos propostos, com o objetivo de destacar as diferenças e similaridades relacionadas à arquitetura e sistema **NeuroCog**.

6.6.1 Mapa Cognitivo Neural e Plausibilidade Biológica

As abordagens desenvolvidas para navegação em labirinto tais como (SCHMAJUK; THIEME, 1992) e (VOICU; SCHMAJUK, 2000), tiveram como foco de atenção questões de plausibilidade biológica, tanto relacionado à construção dos modelos, onde o mapeamento cognitivo é implementado através de uma rede neural associativa, quanto aos comportamentos resultantes. Desta forma estes autores utilizaram o mesmo protocolo realizado em experimentos com ratos, a fim de reproduzirem, em um agente simulado, os mesmos comportamentos apresentados por estes animais.

Voicu e Schmajuk (2002) além de demonstrarem comportamento animal, tais como aprendizado latente, desvios e atalhos[1], também preocuparam-se com questões de desempenho. Os autores estenderam o modelo proposto por Schmajuk e Thieme (1992), o qual era biologicamente plausível, mas lento em ambos os processos de exploração, por ser aleatório, e tomada de decisão, por imitar um processo biológico. Voicu e Schmajuk (2002) propuseram melhorias à abordagem, fazendo com que o modelo dispusesse de

[1] Aprendizado latente se refere a um protocolo onde animais são introduzidos em um labirinto primeiramente sem recompensa. Em um momento posterior, ao ser apresentada a recompensa, os animais demonstram conhecimento do arranjo espacial do labirinto o qual presume-se permanecer latente no mapa cognitivo. O problema de desvio se refere a um protocolo no qual animais podem escolher um entre um ou mais desvios alternativos, somente se eles dispõem de um mapa cognitivo, onde podem ser integradas peças de informação aprendidas separadamente. E por fim o problema de atalho em um campo aberto, se refere a um protocolo no qual animais selecionam o caminho mais curto para um objetivo, atravessando regiões previamente inexploradas do ambiente.

uma representação a priori do ambiente em forma de grid. As modificações incluíram: o processo de exploração ser guiado pelo mapa cognitivo e o processo de decisão utilizar uma técnica de espalhamento de ativação, similar à proposta por Mataric (1991).

Em especial, nas abordagens apresentadas em Voicu e Schmajuk (2000) e Voicu e Schmajuk (2002), os autores não especificam como as entradas dos sensores são tratadas e como o agente simulado "percebe" seu(s) objetivo(s) correntes. A cada ciclo de controle, um sistema de motivação determina um ou mais objetivos, fazendo com que localizações (lugares) sejam ativadas. Esta ativação de objetivo(s) se propaga pela rede neural (mapa cognitivo), até que o lugar onde o agente está localizado seja também ativado. Em seguida, guiado por uma regra de gradiente ascendente, o agente se move até o objetivo, escolhendo o lugar vizinho com a ativação mais forte.

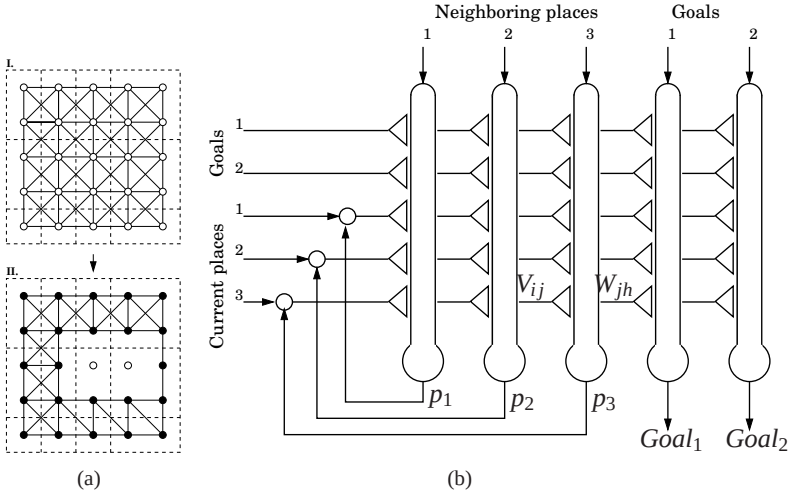


Figura 6.11: O sistema cognitivo de Voicu e Schmajuk (2000). (a) I. O quadro vazio é uma representação em grade da continuidade potencial do espaço a ser explorado. II. O mapa final é uma grade representando a continuidade corrente do espaço explorado. (b) O mapa cognitivo é implementado por uma rede neural que armazena as ligações entre um lugar e seus lugares vizinhos.

Conforme pode ser visto na Figura 6.11 (a)I, a representação do ambiente é baseada em grid e é fornecida a priori. Através desta representação, estabelece-se a criação de uma rede neural hetero-associativa, [Figura 6.11(b)], que corresponde ao mapa cognitivo. Cada célula no grid equivale

a uma posição no labirinto que corresponde a uma entrada na rede, tanto na categoria de "**Current places**", quanto na de "**Neighboring places**". Isto caracteriza um grande consumo de memória e tempo, devido ao aumento no número de iterações do processo de espalhamento, durante o processo de tomada de decisão. Além disto, esta modalidade de representação se torna problemática à medida que o tamanho do ambiente aumenta. Lugares no labirinto tem o tamanho da pegada do agente. Dado que o agente sai de um lugar e imediatamente alcança outro lugar, constata-se a ausência de um módulo reativo no sistema.

Mais especificamente em Voicu e Schmajuk (2000), os autores não comentam a respeito de ocorrerem mudanças no labirinto, após o aprendizado da rede neural hetero-associativa. Já em Voicu e Schmajuk (2002), são tratados os problemas de desvio e atalho, apenas segundo um protocolo experimental realizado com ratos. Além disto, Voicu e Schmajuk (2001) e Voicu (2003), realizaram navegação planejada em ambientes abertos, a partir das abordagens desenvolvidas para navegação em labirinto de (SCHMAJUK; THIEME, 1992) e (VOICU; SCHMAJUK, 2000).

6.6.2 Rato Artificial para Competição

Já Wyeth e Browning (1998) buscaram ir além do comportamento de ratos, porém inspirando-se biologicamente, para construir um modelo capaz de vencer competições. Os autores apresentaram uma plataforma para um robô real realizar uma tarefa de navegação complexa, que compreende a exploração e a solução de um labirinto grande. Uma estrutura de grid também é utilizada como representação do labirinto. Buscando inspiração biológica, os autores fizeram uma análise de modelos cognitivos biologicamente plausíveis, na literatura, para navegação em labirintos. A verificação foi de que estes modelos eram incompletos, segundo uma perspectiva de construtores de robôs, e ainda afirmaram que modelos cognitivos apenas são precisos e úteis, quando desenvolvidos visando um robô (real ou realisticamente simulado), junto com suas interfaces entre sensores e atuadores (reais ou realisticamente virtuais).

A arquitetura cognitiva de Wyeth e Browning (1998), apresentada na Figura 6.12(a), possui três níveis de competência e foi proposta como uma plataforma de robô para resolução de labirintos complexos, tal como o da Figura 6.12(b), utilizado em uma competição oficial. Os três níveis de competência compreendem um nível esquema (reativo), um nível cognitivo e um nível motivacional. Ambos os níveis cognitivo e motivacional caracterizam o nível deliberativo da arquitetura. A leitura dos sensores é fornecida aos três

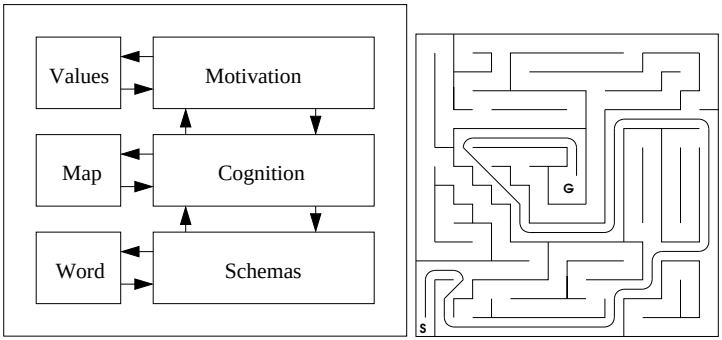


Figura 6.12: O modelo cognitivo de Wyeth e Browning (1998). (a) A arquitetura cognitiva. (b) Labirinto usado na competição "Australian micromouse" em 1996. Adaptado de (WYETH; BROWNING, 1998).

níveis. O nível reativo está limitado somente a interpretar dados dos sensores e determinar ações, sem gerar ou acessar memória. Sua função consiste em receber comandos dos níveis mais altos, tal como, "desça o corredor por três quadrados", mantendo o robô centralizado no corredor, e observando valores cinemáticos para determinar quando o cumprimento da ação terminou.

Na arquitetura **NeuroCog**, a leitura dos sensores também é fornecida aos níveis reativo e deliberativo. Porém, ambos os níveis funcionam independentemente, onde os aspectos em comum são que ambos necessitam da informação do estado do ambiente (informação de parede) e da ação executada no ciclo de controle anterior. E ainda, o nível reativo da arquitetura **NeuroCog** gera e acessa memória, referente ao mapeamento de percepção-ação produzido pelo arranjo neural, diferente do nível reativo da arquitetura de Wyeth e Browning (1998).

Os níveis cognitivo e motivacional da arquitetura de Wyeth e Browning (1998), utilizam a leitura dos sensores para produzir informação de paredes. Enquanto o nível cognitivo gerencia assuntos de representação, localização e planejamento, o nível motivacional é responsável pela estratégia a ser seguida pelo robô, tal como a definição do objetivo e a velocidade a qual o robô deve executar. Este nível também é responsável por decidir quando cessar a exploração e gerar uma rápida resolução do labirinto, proporcionando ao robô o comportamento geral de ambos se mover e resolver o labirinto.

Sob este aspecto, o nível deliberativo da arquitetura **NeuroCog** também utiliza a leitura dos sensores para produzir informação de paredes, so-

mente nos marcos encontrados. A informação da posição do robô é utilizada exclusivamente para diferenciar estes marcos, e assim produzir uma representação topológica do labirinto. Além disto, a camada deliberativa, também decidi quando cessar a exploração e priorizar o planejamento. A característica geral do comportamento de um robô utilizando a arquitetura **NeuroCog**, é a de que o robô pára, toda vez que ele necessita mudar a direção de sua trajetória, girando ao redor do próprio eixo, até encontrar o ângulo de direção da próxima ação a ser executada.

Quanto ao assunto de representação de conhecimento, a abordagem de Wyeth e Browning (1998), assim como a de Voicu e Schmajuk (2002), tira vantagens da representação em grid do ambiente, dado que o tamanho do labirinto é informado anteriormente à realização da competição. Por exemplo, o labirinto da Figura 6.12(b) é representado em um grid de 16×16 células. A abordagem de WYETH; BROWNING então descreve o mapa como um array de 16×16 entradas, onde cada entrada possui 8 bits: 4 para representar presença ou ausência de paredes e 4 para indicar quais das 4 paredes foram visitadas. Caminhos até o objetivo são encontrados através da utilização de um algoritmo baseado no cálculo do melhor tempo para a execução destes caminhos.

Wyeth e Browning (1998) identificaram problemas encontrados pela sua arquitetura, na ocasião de mudanças no labirinto, após a fase de aprendizagem do robô. Por exemplo, quando os corredores são alongados ou encurtados, seu comportamento resulta em erro. Além disto, quando corredores são bloqueados, o robô tenta escalar o bloqueio, não considerando os dados dos sensores. Da mesma forma, quando surge uma nova abertura em um corredor, ela é ignorada pelo robô. Os autores justificam estes problemas, alegando que estes comportamentos também foram obtidos em resultados experimentais com ratos, submetidos às mesmas condições.

A Figura 6.13 ilustra um exemplo de atuação da arquitetura **NeuroCog** no labirinto apresentado por Wyeth e Browning (1998). Em (a) a enumeração representa a configuração dos marcos encontrados e identificados pelo robô, após uma exploração completa do labirinto. A linha pontilhada em vermelho descreve a trajetória do robô, que compreende o caminho mais curto entre o lugar 0 até o lugar objetivo 11 (que foi adaptado na figura para se tornar um beco sem saída). Em (b) tem-se o grafo representando o mapa topológico aprendido. As arestas em vermelho indicam as ações que constituem o plano para partir do lugar origem S e alcançar o lugar objetivo G, pelo caminho mais curto. Neste caso, o mapa cognitivo é descrito pelo sistema **NeuroCog** como um array de 30×8 entradas, que compreendem os 30 lugares do labirinto, onde, para cada lugar são necessárias as informações das ligações à oeste, norte, leste e sul, com seus lugares vizinhos, e os respectivos valores

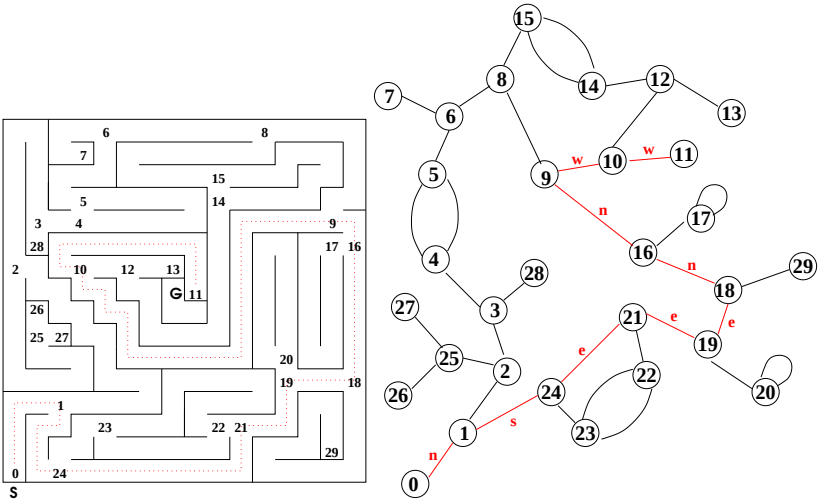


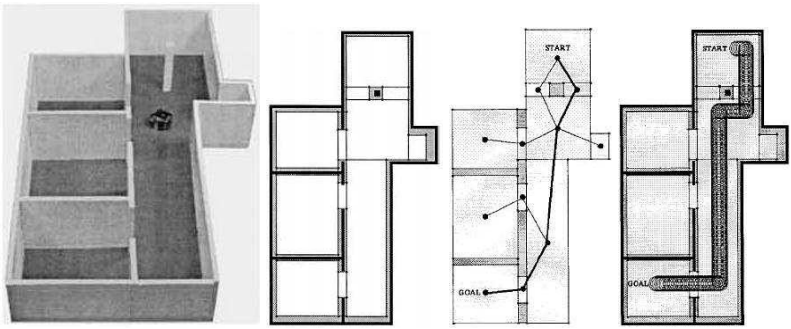
Figura 6.13: Mapeamento cognitivo realizado pelo sistema **NeuroCog**, no labirinto apresentado por (WYETH; BROWNING, 1998). (a) Detecção de lugares no labirinto após uma exploração completa. (b) O grafo do mapa cognitivo e o planejamento de caminho de S a G.

da distância percorrida até eles. Portanto, a abordagem **NeuroCog** considera apenas locais que formam becos sem saída e bifurcações como principais localizações a serem representadas e tratadas pelo nível deliberativo, porém ela conta sempre com a estrutura ortogonal de labirintos. Deixando a cargo do nível reativo a navegação em corredores que não necessitam ser discretizados, como no caso da representação em grid. De fato, esta característica torna a abordagem **NeuroCog** mais adaptativa e flexível à mudanças que podem ocorrer em corredores, mesmo após o aprendizado de ambos módulos reativo e deliberativo da arquitetura.

6.6.3 Mapa Cognitivo Com Resolução Variável

Arleo, Millán e Floreano (1999) também desenvolveram um sistema de navegação em ambientes internos para robôs reais. A abordagem dos autores assemelha-se à navegação em labirintos pelo fato das fronteiras dos obstáculos serem sempre paralelas aos eixos x e y e pelo fato do robô executar apenas trajetórias retas, e não usar visão. O modelo proposto pelos autores

consiste de um método de aprendizado adaptativo (incremental e on-line) de mapas cognitivos, onde a modelagem do ambiente incorpora ambas representações geométrica e topológica. O mapa topológico é gerado a partir de um particionamento variável de representação geométrica, construída de forma compacta, através da utilização de uma rede neural feed forward no nível reativo, que interpreta a informação sensória. Caminhos ótimos são determinados pelo nível planejador da arquitetura, através do mapa topológico.



(a) (b) (c) (d)

Figura 6.14: Resultados experimentais da abordagem de Arleo, Millán e Floreano (1999). (a) O ambiente usado e o robô Khepera dentro dele. (b) O particionamento de resolução variável aprendido. (c) O grafo topológico correspondente ao mapa em (b). (d) A trajetória do robô para ir de START até GOAL. Adaptado de (ARLEO; MILLáN; FLOREANO, 1999)

Deste modo, a arquitetura modular de Arleo, Millán e Floreano (1999) resulta em um método de aprendizado de mapa geométrico e topológico de um ambiente fechado, possuindo apenas obstáculos ortogonais, como o caso da Figura 6.14(a). Em um processo cíclico, onde predominam os comportamentos de exploração e atualização de ambos os mapas, o robô está continuamente explorando o ambiente, dirigido pelo conhecimento adquirido. O robô somente interrompe a exploração para incorporar obstáculos desconhecidos ao modelo e atualizar a resolução do particionamento do mapa geométrico. A arquitetura modular proposta pelos autores é composta por cinco módulos principais. O primeiro módulo, que constitui o nível reativo, interpreta dados sensórios através de uma rede neural feed forward. Esta rede produz um grid de ocupação local, para identificar a fronteira do obstáculo e fazer o robô se alinhar a ela. O segundo módulo faz o robô seguir as arestas do obstáculo encontrado até completar seu perímetro. O terceiro módulo atu-

aliza o particionamento do mapa geométrico, incrementando sua resolução. O quarto módulo, responsável pela exploração, faz com que o robô sempre explore o ambiente para melhorar seu mapa geométrico corrente. Dado um particionamento corrente, este módulo seleciona como partição alvo uma região do ambiente menos conhecida. O quinto e último módulo é responsável pelo planejamento e ação, e funciona da maneira a seguir. Após a atualização da resolução de um particionamento corrente, uma memória de longo termo armazena relacionamentos espaciais entre partições, para derivar um grafo topológico utilizado no planejamento de caminho. Dada uma partição selecionada como alvo, o planejador computa o caminho ótimo, através das partições livres de obstáculos, fornecendo aos controladores de baixo nível os comandos que levam o robô até lá.

Portanto, este sistema de aprendizado de mapa métrico, por exemplo a Figura 6.14(b), deriva o mapa topológico, Figura 6.14(c) que tem como nós, os centros das partições que correspondem às porções livres de obstáculos. A Figura 6.14(d) descreve uma trajetória do robô, executada pelo controlador reativo, entre partições livres e adjacentes, que compreendem o caminho entre START e GOAL.

Similar à abordagem **NeuroCog**, na abordagem proposta por Arleo, Millán e Floreano (1999) o ambiente é desconhecido, porém ele deve ser ortogonal. Os robôs utilizados pelos autores não usam visão e o conceito de marcos está implicitamente contido no mapa relacional (topológico), derivado a partir do particionamento de resolução variável. Neste particionamento, apenas os centros das partições livres de obstáculos especificam marcos, gerando um compacto mapa topológico. Assim como o mapa topológico produzido pelo sistema **NeuroCog**, que considera como marcos apenas bifurcações e becos sem saída em um labirinto. Outra diferença entre ambas abordagens relaciona-se com a implementação e atuação do controle reativo. Na proposta de Arleo, Millán e Floreano (1999), uma rede neural direta é utilizada no módulo reativo e seu treinamento é realizado de maneira off-line. Enquanto na abordagem **NeuroCog**, várias redes neurais diretas são treinadas de maneira on-line e permutadas por uma rede ART1 recorrente. A atividade reativa da arquitetura de Arleo, Millán e Floreano (1999) consiste em alinhar o robô com a fronteira de um obstáculo encontrado, durante a fase de exploração, e executar a trajetória até um alvo, determinada pelo módulo de planejamento. Os autores não comentam a respeito do ambiente sofrer modificações, após a conclusão da fase de aprendizado do mapa geométrico.

6.6.4 Considerações

As abordagens apresentadas propuseram diferentes maneiras de integrar, em um nível arquitetural, as atividades reativas e deliberativas. Em nível de implementação, foram vistos diferentes formas de utilização de métodos geométricos e topológicos, bem como técnicas de RNAs. Em contraste com as arquiteturas avaliadas, a arquitetura **NeuroCog** provê um sistema de navegação que não necessita de informação a priori do labirinto, com relação a tamanho ou número de células de grid ou número de bifurcações ou becos sem saída.

O sistema **NeuroCog** se aplica a labirintos de qualquer tamanho e que possam ser modificados. Porém, a ortogonalidade das paredes do labirinto é uma condição necessário ao seu funcionamento, bem como o formato em "T" das bifurcações. A ortogonalidade proporciona simplicidade na implementação do aprendizado adaptativo do mapa topológico, que delimita lugares através da detecção de bifurcações e becos sem saída. Para que o sistema **NeuroCog** possa atuar, por exemplo, em um labirinto hexagonal, como o utilizado em (MALLOT et al., 1995), apenas uma mudança no processo de detecção de bifurcações seria necessária.

A ortogonalidade também permite que o robô mova-se apenas ao longo de trajetórias retas. Para que a abordagem proposta possa ser aplicada a ambientes abertos, esta suposição deve ser removida e com isto, o custo da construção do mapa cognitivo tende a aumentar, à medida que se cresce o grau de liberdade do robô, ou seja, o número de direções que ele pode se mover.

Outro aspecto do sistema de navegação proposto nesta tese é que ele necessita da capacidade do robô de auto-localização e orientação. Posição e orientação de robôs são determinados pela técnica de integração de caminho (informação idiotética), que em um robô físico está sujeita à imprecisão. O sistema **NeuroCog** porém, não necessita de posição precisa em termos métricos, mas apenas para diferenciar localizações. A utilização de informação idiotética, onde becos sem saída e bifurcações são encontrados, elimina o problema de percepção onde lugares diferentes no ambiente são percebidos de maneira idêntica. Porém, o sistema **NeuroCog** necessita de uma capacidade de orientação precisa.

6.7 Conclusão

Em acréscimo às abordagens de navegação existentes na literatura, a abordagem **NeuroCog** apresentou soluções aos seguintes problemas relacionados à navegação em labirintos e baseada em mapas. Primeiro, a aborda-

gem propõe uma integração de deliberação e reação em uma arquitetura de controle híbrida, capaz de produzir um sistema de navegação flexível e adaptativo. Este sistema combina de maneira simplificada e eficiente, um método de aprendizado de mapa topológico, com um processo de aprendizado reativo implementado através de um arranjo de RNAs. Com isto, este sistema permite a um robô, contando apenas com seus sensores de proximidade, resolver uma tarefa de navegação complexa.

Segundo, a abordagem proposta pode ser aplicada a labirintos dinâmicos. Neste trabalho determinou-se como característica dinâmica, o fato do ambiente poder sofrer modificações, por exemplo, quanto a bloqueios (novos becos sem saída) ou aberturas (novas bifurcações), após o aprendizado de ambos a representação do labirinto (pela camada deliberativa) e do mapeamento de percepção-ação (pela camada reativa).

Em terceiro, para tratar as dinâmicas de um ambiente foi implementado um método de ajuste entre os comportamentos de exploração e planejamento, na camada deliberativa da arquitetura proposta. O dilema conhecido como *exploration* versus *exploitation* foi descrito e estendido sob dois contextos: i) o primeiro relacionado à camada deliberativa, que implementa ambos os comportamentos de explorar e planejar, bem como um método que ajuste a competição inerente entre os mesmos e ii) o segundo contexto está relacionado ao custo do controle neural reativo, que é influenciado pelo método de ajuste na camada deliberativa. O controle deliberativo do sistema **NeuroCog** foi implementado através do aprendizado e utilização de memórias de longo termo (mapa topológico ou cognitivo) e de curto termo (experiências de exploração e planejamento).

E finalmente a utilização do arranjo neural **ART1-R-MLPs-RR** para a implementação da camada reativa da arquitetura **NeuroCog**, propicia a criação de um mapeamento de percepção-ação também flexível e adaptativo. Devido ao ambiente de atuação da abordagem ser do tipo labirinto, onde os marcos tratados pela porção deliberativa são considerados apenas como bifurcações e becos sem saída, a atuação da porção reativa da arquitetura predomina sobre a porção deliberativa. Isto implica que o eficiente aprendizado de mapeamento, feito pelo arranjo neural, garante à arquitetura um desempenho aceitável, com relação à utilização dos recursos de memória e tempo.

CAPÍTULO 7

Simulações e Resultados

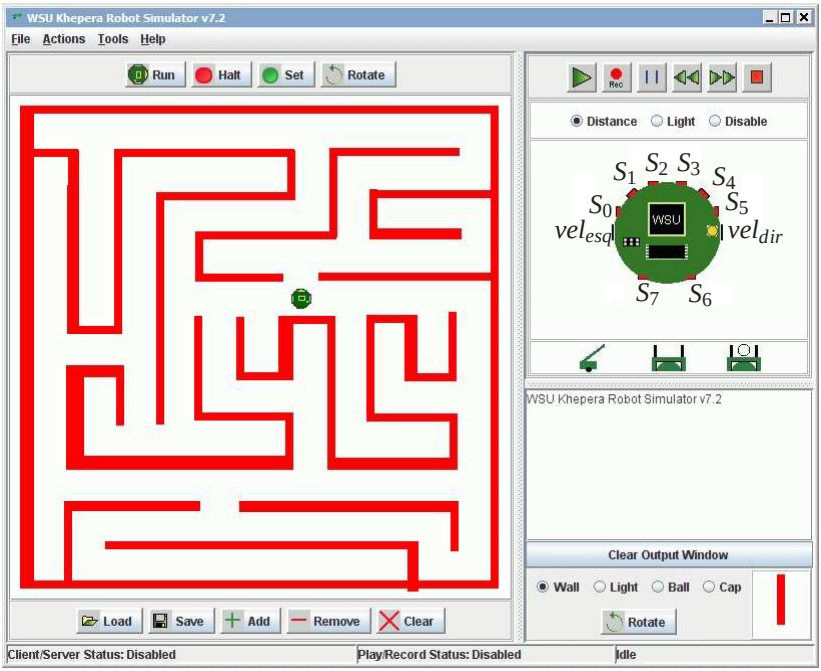
7.1 Introdução

Neste capítulo serão apresentadas primeiramente simulações e resultados referentes a arquiteturas neurais reativas propostas no Capítulo 5, as quais são aplicadas à navegação em labirintos simples. Em segundo serão vistas simulações referentes a arquiteturas híbridas prévias à arquitetura **NeuroCog**, as quais são aplicadas na sequência, a labirintos com bifurcações sem ciclos e estáticos e posteriormente a labirintos com ciclos e modificáveis, após a fase de exploração do robô. E finalmente será apresentada simulações e resultados da arquitetura híbrida **NeuroCog**, proposta no Capítulo 6, a qual é aplicada a labirintos que podem ser modificados durante a operação do robô, cuja fase de exploração e planejamento são tratadas de maneira dinâmica. Este capítulo tem por objetivo validar as arquiteturas propostas, avaliando questões de desempenho e aprendizado efetivo, através do aumento gradual da complexidade do labirinto e da tarefa de navegação.

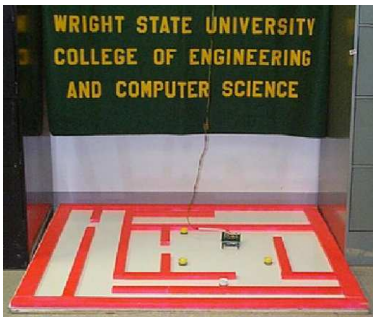
7.2 Ambiente de teste

As simulações foram realizadas no software *WSU Java Khepera Simulator*[1] [Figura 7.1(a)]. Este programa simula o robô Khepera® (MONDADA; E.FRANZI; LENNE, 1993) que constitui uma ferramenta de pesquisa usada para

[1] O simulador *WSU Khepera Robot* foi desenvolvido pela *Wright State University* e *Ohio Board of Regents*. Seu uso é dirigido pela Licença Pública da KSIM versão 1.0. O código fonte, documentação, e o texto da licença encontram-se juntos com esta versão do programa. Na falta desta, uma distribuição completa pode ser obtida em: http://gozer.cs.wright.edu/classes/ceg499/resources/



(a)



(b)

Figura 7.1: (a) WSU Khepera Robot Simulator v7.2. (b) O robô físico e seu ambiente no Laboratório de Dinâmica Neural e Computação da WSU (*Wright State University*).

implementação e teste de vários tipos de controladores robóticos. O simulador *WSU KSIM* é uma plataforma para o desenvolvimento prévio de um controlador a ser testado em um robô Khepera real. Este simulador foi desenvolvido a partir do kit de desenvolvimento Java versão 1.4.1, oferecido gratuitamente pela Sun MicroSystems. Logo, todos os algoritmos para implementação das arquiteturas propostas foram escritos nesta linguagem, a fim de serem carregados no simulador.

Mais especificamente, o robô simulado e seu ambiente foram projetados para imitar o hardware robótico e seu respectivo ambiente de teste [Figura 7.1(b)] utilizados no Laboratório de Dinâmica Neural e Computação da WSU (*Wright State University*). O robô físico consiste de um robô Khepera padrão que possui uma torre braço-garras e ocupa uma área de aproximadamente 10 cm^2. Além disto, ele apresenta 8 sensores infravermelhos ($S_0, \ldots, S_7$), os quais fornecem dois tipos de dados - proximidade do objeto e intensidade de luz, e os codificadores de roda para determinar a posição e a direção do robô. Os valores fornecidos aos atuadores incluem as velocidades para os motores das rodas esquerda e direita, vel_{esq} e vel_{dir} respectivamente.

O ambiente do robô consiste de uma arena que pode conter paredes, luzes, botões e/ou bolas. Admite-se que paredes e luzes são objetos estáticos de modo que o robô não pode atingi-los. Caso isto ocorra, o robô se tornará emperrado. Botões e bolas são considerados objetos dinâmicos e caso o robô faça um contato não intencional nestes objetos, eles se moverão. Além disto, o robô pode ainda manipular diretamente botões e bolas através de suas garras.

Na construção dos ambientes de teste para as abordagens de navegação propostas neste trabalho de tese, foram utilizados somente objetos do tipo parede, que podem ser dispostos apenas paralelamente aos eixos x e y do plano cartesiano. Além disto, para simplificar a construção das arquiteturas de controle propostas, o robô conta, durante as simulações, com a estruturação em labirinto tipo T do ambiente. E ainda, considera-se que o robô pode executar um movimento, a cada passo de controle, cujas direções possíveis são apenas: oeste, norte, leste e sul, e este movimento equivale a uma ação no intervalo de tempo t.

7.3 Arquiteturas Neurais Reativas

Esta seção apresenta as simulações computacionais realizadas a partir das arquiteturas neurais reativas **ART1-R-MLPs-RR** e **ART1-R-MLPs-RR-Marcos** respectivamente. Como descrito no Capítulo 5, estas arquiteturas integram dois tipos de arquiteturas de RNAs, que permite a um robô móvel

autônomo aprender a executar uma trajetória livre de obstáculos, por meio de um processo de aprendizado de mapeamentos percepção-ação, em tempo de operação.

Para as simulações apresentadas nesta seção, o robô não utilizou nenhuma informação a priori do ambiente e contou apenas com seus sensores de proximidade ($S_0, \ldots, S_7$) e ângulo de orientação (θ). A informação de posição do robô não é utilizada pelas abordagens reativas propostas.

7.3.1 Controle Neural ART1-R-MLPs-RR

A configuração da arquitetura **ART1-R-MLPs-RR** para a simulação apresentada na sequência é ilustrada na Figura 7.2. A rede ART1 recorrente

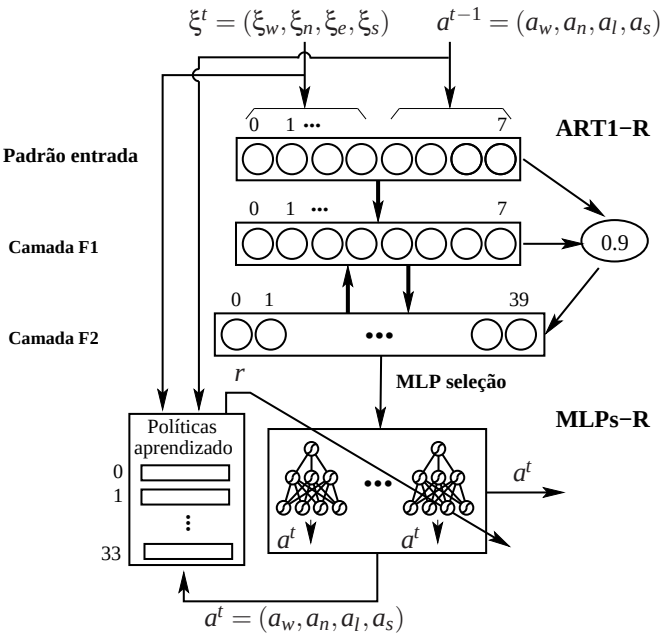


Figura 7.2: Configuração da arquitetura neural reativa **ART1-R-MLPs-RR**.

(ART1-R) é disposta com 8 neurônios na camada F1, para receber o padrão de entrada equivalente ao estado do ambiente, ξ^t, mais a recorrência da ação anterior, a^{t-1}. Ambas informações fornecidas pelos módulos **Percepção** e **Sistema motor** respectivamente. A camada F2 disposta com no máximo 40

neurônios, se mostra suficiente para categorizar as redes MLPs, para labirintos simples, sem bifurcações e becos sem saída. O parâmetro de vigilância ρ é estabelecido como 0.9, por produzir uma categorização efetiva, medida pela quantidade de punições aplicadas às redes MLPs do subsistema **MLPs-R**. Cada rede MLP possue 1 neurônio na camada de entrada, 3 neurônios na camada intermediária e 4 neurônios na camada de saída, para determinar a ação corrente a^t. A base de regras para as políticas de aprendizado online das redes MLPs somam 34 regras (Seção A.1).

As simulações realizadas nesta seção tem por objetivo comparar o desempenho da arquitetura neural reativa **ART1-R-MLPs-RR** para o controle do robô simulado em três labirintos distintos, como ilustrados na Figura 7.3.

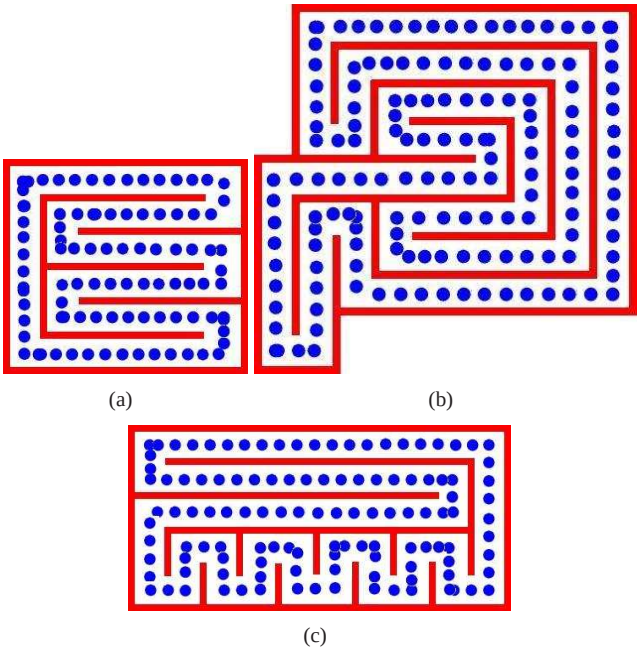


Figura 7.3: Aprendizado de caminho do controle **ART1-R-MLPs-RR**.

O robô é introduzido nos labirintos (a), (b) e (c) em posições arbitrárias. Em cada labirinto, durante a fase de aprendizado do sistema neural **ART1-R-MLPs-RR**, apenas a informação da configuração de paredes e a realimentação da ação anterior são utilizadas. E ainda, o robô executa trajetórias até que nenhuma nova classe seja criada pela rede ART1-R. Ao fim da fase de apren-

dizado, o robô torna-se apto a percorrer todo o labirinto a partir da memorização da trajetória completa (mapeamento percepção-ação) até sua posição inicial, não mais criando novas classes (ART1-R) e sem executar punições às redes MLPs, através das políticas de aprendizado.

A Tabela 7.1 mostra a quantos passos de controle ocorreu a finalização do aprendizado efetivo do sistema neural **ART1-R-MLPs-RR**, para os três labirintos analisados (Figura 7.3). A finalização do aprendizado efetivo é verificada no passo de controle (coluna "Passos") onde ocorre a última punição, executada pelo treinamento online das redes MLPs. Neste passo são computados respectivamente o número de classes obtidas pela rede ART1-R e a porcentagem média de ações corretas, que é calculada através da fórmula:

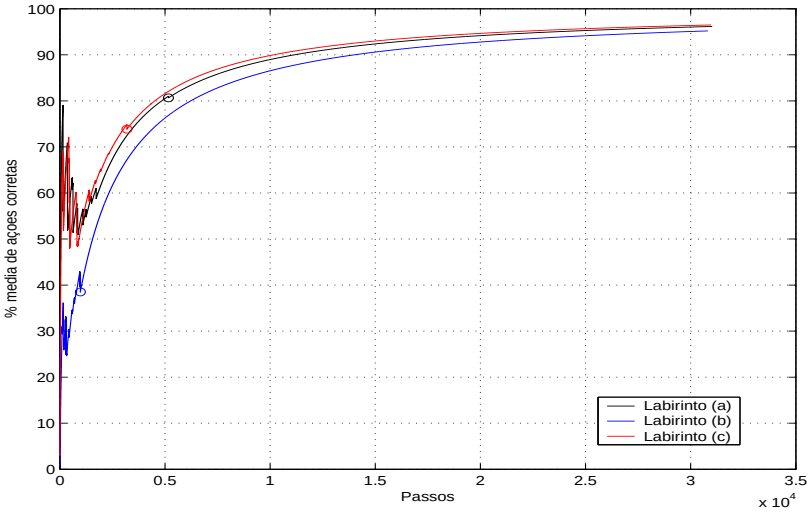
$$\%\ \text{média de ações corretas} \quad = \quad \frac{\text{recompensas}}{\text{recompensas} + \text{punições}} \times 100\% \quad (7.1)$$

É importante notar que o número de classes obtidas pelo sistema **ART1-R-MLPs-RR** é proporcional à complexidade do labirinto. Esta complexidade está relacionada ao número de sub-trajetórias que faz com que o robô mude de direção, quando alcança um canto, por exemplo, ao final de corredores. E isto implica na mudança de padrões de entrada apresentados à rede ART1-R, levando a um aumento do numero de novas classes (MLPs).

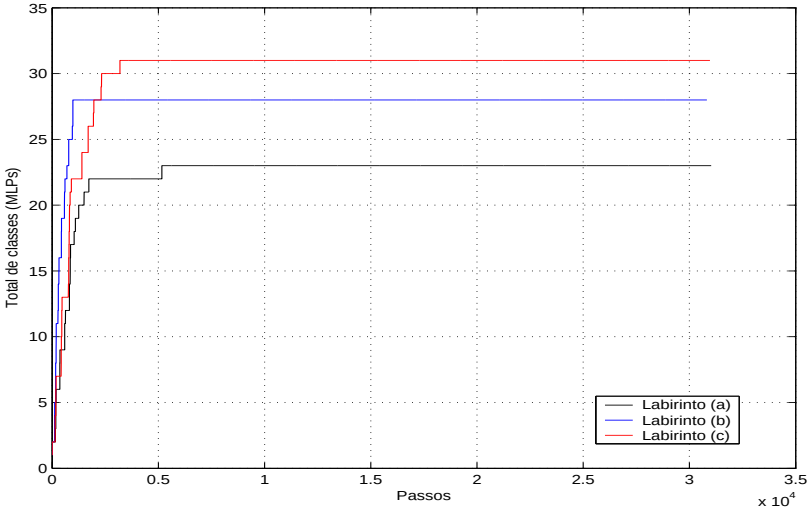
Labirinto (Fig.7.3)	Complexidade (sub-trajetórias)	Classes ART1-R	% média de ações corretas	Passos
(a)	12	23	80,6%	5165
(b)	22	28	38,5%	975
(c)	24	31	73,9%	3193

Tabela 7.1: Desempenho do sistema neural reativo **ART1-R-MLPs-RR**: Passo de controle onde ocorre o aprendizado efetivo do mapeamento percepção ação, juntamente com o número de classes (MLPs) criadas pela rede ART1-R e a porcentagem média de ações corretas executadas, para os labirintos (a), (b) e (c) da Figura 7.3.

Por outro lado, a partir dos resultados das simulações, verifica-se que dentre os três casos analisados, o aprendizado efetivo ocorreu mais rapidamente no labirinto (b), com somente 975 passos (Tabela 7.1). Isto se deve ao fato de que este labirinto possui corredores mais extensos com relação aos outros dois e isto aumenta o número de vezes com que o mesmo padrão de estado do ambiente é apresentado à entrada da rede ART1-R. Esta caracterís-



(a)



(b)

Figura 7.4: Desempenho da arquitetura neural reativa **ART1-R-MLPs-RR**. (a)Porcentagem média de ações corretas. (b) Categorização do espaço de entrada durante o processo de aprendizado da rede ART1-R.

tica causa menos variação dos padrões de percepção e como consequência, a porcentagem média de ações corretas é a menor (no passo de controle 975), devido ao fato da estabilidade do aprendizado ocorrer mais rápido do que nos labirintos (a) e (c). Portanto, o número de passos apontados na tabela indicam a quantidade de iterações necessárias para que a arquitetura neural reativa realize a construção do mapeamento percepção-ação, para o aprendizado das trajetórias.

A ocorrência da finalização do aprendizado efetivo do arranjo neural para os três labirintos (a) (b) (c) respectivamente nos passos 5165, 975 e 3193 podem ser visualizados através de círculos nas curvas do gráfico Figura 7.4(a). Ou seja, para as três simulações descritas acima, foram computadas até o número de passos alcançar a aproximadamente 3.5×10^4, os seguintes desempenhos. Primeiro, a porcentagem média de ações corretas (executadas pelas redes MLPs) e segundo, o número de classes, criadas pela rede ART1, Figura 7.4 (b), que representa o particionamento do espaço de entrada da rede ART, que leva a um mapeamento percepção-ação eficiente. Portanto, a arquitetura neural reativa **ART1-R-MLPs-RR** representa um sistema de aprendizado de trajetórias adaptativo, que permite a um robô móvel explorar um labirinto tantas vezes quanto necessário, evitando obstáculos, para então construir uma memória percepção-ação, através da aplicação de ambos aprendizados auto-supervisionado e por reforço das redes neurais utilizadas.

7.3.2 Controle Neural ART1-R-MLPs-RR-Marcos

O desenvolvimento da arquitetura neural **ART1-R-MLPs-RR-Marcos** foi uma tentativa de incorporar ao sistema reativo **ART1-R-MLPs-RR** o aprendizado de uma tarefa de navegação mais complexa, através da inclusão de bifurcações e becos sem saída em labirintos. O objetivo foi investigar se esta arquitetura proveria o aprendizado de um mapeamento cognitivo que possibilitasse navegação do tipo disparo de reconhecimento de lugar ou topológica (Capítulo 3, Seções 3.3.5 e 3.3.6 respectivamente). Deste modo, durante a execução do robô, o controle neural **ART1-R-MLPs-RR-Marcos** detecta bifurcações e becos sem saídas para associá-los a marcos. Como tarefa de navegação, o robô deve, além de seguir paredes evitando obstáculos, aprender uma trajetória livre de becos sem saída. Desta forma, o robô depara-se com o problema de aprender a tomar decisões mediante duas opções corretas possíveis, sempre que se localizar em um marco do tipo bifurcação.

Para as simulações realizadas com a arquitetura **ART1-R-MLPs-RR-Marcos** tem-se a seguinte configuração, ilustrada na Figura 7.5. A rede ART1 recorrente (ART1-R) apresenta 11 neurônios na camada F1, para receber o

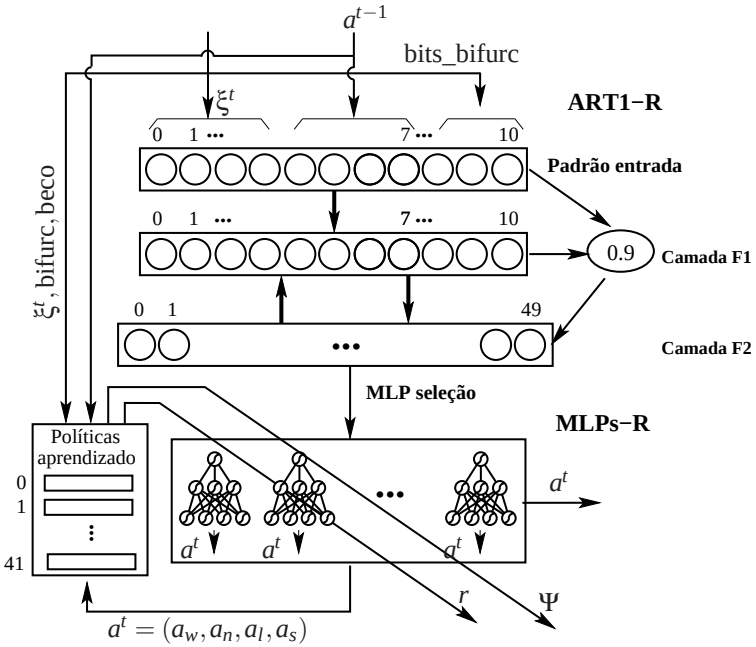


Figura 7.5: Configuração da arquitetura neural reativa **ART1-R-MLPs-RR-Marcos**.

padrão de entrada equivalente ao estado do ambiente, ξ^t, a recorrência da ação anterior, a^{t-1} e mais três bits, denominados "**bits_bifurc**", que indica a ocorrência ou não de uma bifurcação de acordo com a sinalização (bifurc). A camada F2 é disposta com no máximo 50 neurônios. Este acréscimo de neurônios se deve ao aumento do número de neurônios do padrão de entrada, e se mostra um limite aceitável para categorizar as redes MLPs no nível abaixo. O parâmetro de vigilância ρ é estabelecido como 0.9. E assim como na abordagem anterior, as redes MLPs possuem 1 neurônio na camada de entrada, 3 neurônios na camada intermediária e 4 neurônios na camada de saída, para determinar a ação corrente a^t. A base de regras para as políticas de aprendizado online das redes MLPs somam 42 regras (Seção A.2). Este aumento no número de regras se deve ao módulo **Políticas de aprendizado** implementar um processo de tomada de decisão na ocorrência de marcos, através da aplicação de dois tipos de punições r e Ψ respectivamente.

Os resultados de simulação comparam o desempenho da arquitetura

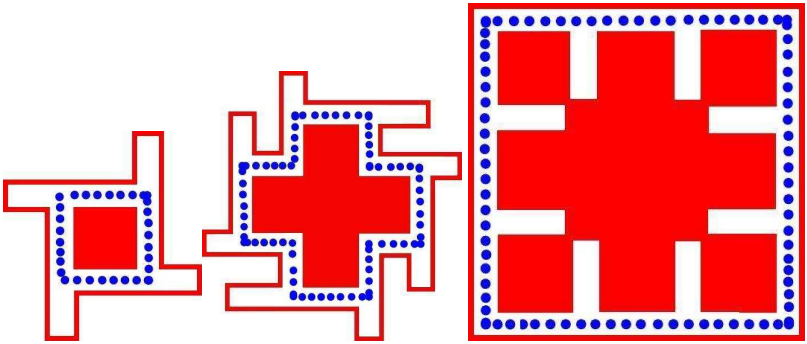


Figura 7.6: Aprendizado de caminho do controle **ART1-R-MLPs-RR-Marcos**.

neural **ART1-R-MLPs-RR-Marcos** proposta para o controle do robô simulado em três labirintos do tipo T, mostrados na Figura 7.6. O robô foi introduzido nos labirintos (a), (b) e (c) em posições arbitrárias. Na fase de aprendizado, ele vaga pelo labirinto, eventualmente se dirigindo a becos sem saída e recebendo punição por esta ação. Ao fim da fase de aprendizado, o robô aprende a percorrer todo o labirinto, memorizando a trajetória ótima, livre de obstáculos e becos sem saída, através do mapeamento neural de percepção-ação aprendido. A Tabela 7.2 mostra o momento em que ocorre a finalização de ambos o aprendizado auto-supervisionado da rede ART1-R, e o aprendizado por reforço das redes MLPs, para os três labirintos analisados.

Labirinto (Fig.7.6)	Complexidade (sub-trajetórias + becos)	Classes ART1-R	% média de acerto	Passos
(a)	8	17	43.55%	537
(b)	20	32	70.46%	4053
(c)	20	32	54.49%	2371

Tabela 7.2: Número de passos onde ocorre a finalização da fase de aprendizado auto-supervisionado (quando uma última classe nova é criada) e por reforço (quando uma última punição é dada a uma rede MLP.

A partir destes resultados se constata que tanto o número de classes obtidas quanto a quantidade de passos de controle necessários para a finali-

zação do aprendizado ART1 são proporcionais à complexidade do labirinto. Esta complexidade está relacionada à quantidade de becos sem saída, bifurcações e cantos. Nos cantos o robô muda a direção de sua trajetória tomando uma direção perpendicular e em cada bifurcação ele deve aprender a não tomar a direção que leva a um beco sem saída. Em ambos os casos ocorre um aumento na variação dos padrões de entrada apresentados à rede ART1, ocasionando um aumento do número de classes criadas e como consequência o aumento no número de punições executadas.

Como a complexidade do labirinto (a) da Figura 7.6 é a menor, onde a trajetória final apresenta apenas 4 sub-trajetórias, o robô completa sua fase de aprendizado mais rapidamente, com 537 passos de controle (Tabela 7.2). Por outro lado, como no labirinto (b) o robô deve aprender uma trajetória ótima composta por 12 sub-trajetórias, são necessários então mais passos de controle (4053) para que ocorra a finalização do aprendizado do sistema neural. Porém com a porcentagem média de ações corretas mais alta neste ponto (70.46%), com relação aos outros 2 casos onde o aprendizado ocorre anteriormente.

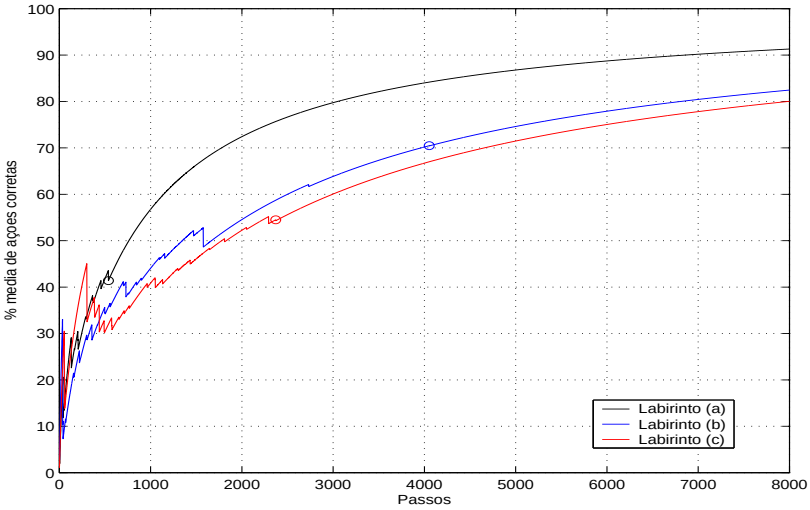
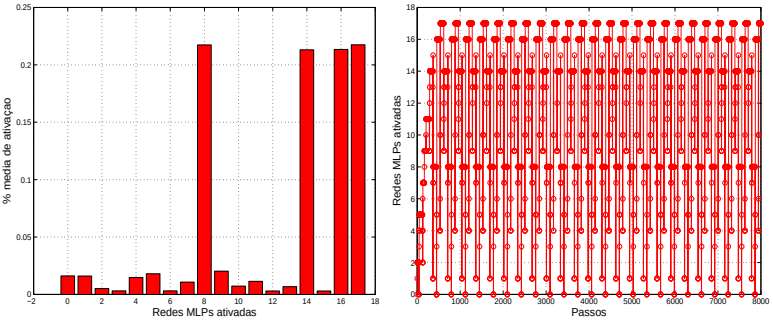
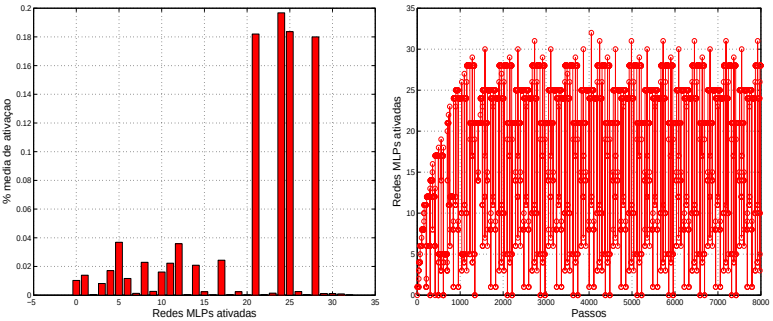


Figura 7.7: Desempenhos do robô em cada um dos labirintos da Figura 7.6.

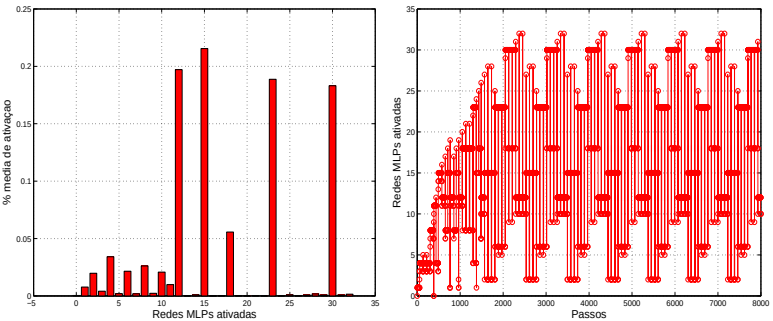
No gráfico de desempenho da Figura 7.7 é observada a curva das porcentagens médias de ações corretas para os três casos analisados. Os círculos



(a) Simulação referente ao labirinto da Figura 7.6 (a).



(b) Simulação referente ao labirinto da Figura 7.6 (b).



(c) Simulação referente ao labirinto da Figura 7.6 (c).

Figura 7.8: À esquerda gráficos com a porcentagem de ativação das redes MLPs e à direita o momento em que as mesmas são ativadas pelo neurônio vencedor da camada F2 da rede ART1-R.

no gráfico indicam o momento em que a arquitetura reativa **ART1-R-MLPs-RR-Marcos** completou o aprendizado de seu mapeamento neural de percepção-ação, durante a simulação nos três labirintos (Figura 7.6).

Durante os 8000 passos de controle, em que o robô explora cada labirinto, foram computadas também a porcentagem de vezes em cada rede direta foi ativada, resultando os gráficos à esquerda da Figura 7.8. Neles vê-se que quatro redes possuem as maiores porcentagens de ativação, devido ao comprimento dos corredores. Já os gráficos à direta representam o momento de ativação das redes diretas durante os 8000 passos, ou seja, a cada passo, a rede ativada é marcada por um círculo. Neles também percebe-se o maior uso das quatro redes destacadas nos respectivos gráficos à esquerda e, depois de determinado ponto, o padrão de ativação das redes se repete, devido à estabilidade das fases de aprendizado do sistema neural de navegação.

Portanto, o sistema de controle neural implementado pela arquitetura reativa **ART1-R-MLPs-RR-Marcos** mostrou-se eficiente no aprendizado de trajetórias livres de obstáculos e becos sem saída nos labirintos analisados. Todavia, como discutido na Seção 5.5.3, esta arquitetura apenas apresenta um aprendizado efetivo do sistema neural e da tarefa de navegação em tipos específicos de labirintos T. Sendo assim, a abordagem **ART1-R-MLPs-RR-Marcos** não proveu um sistema flexível e adaptativo com a estratégia de tratar de forma reativa a detecção de marcos e o processo de tomada de decisão. Por esta razão a arquitetura **ART1-R-MLPs-RR** manteve-se como camada reativa da arquitetura **NeuroCog**.

7.4 Camadas Deliberativas Prévias

As arquiteturas expostas nesta seção apresentam camadas deliberativas que foram desenvolvidas anteriormente à camada deliberativa da arquitetura **NeuroCog**. A principal diferença entre as camadas deliberativas prévias e a camada deliberativa da arquitetura **NeuroCog** reside no fato de que as primeiras executam na sequência as fases de exploração e planejamento de caminho durante a operação do robô. Enquanto que a deliberação na arquitetura **NeuroCog** trata exploração e planejamento como comportamentos a serem executados de forma dinâmica, para equilibrar o dilema exploração versus aproveitamento. Enquanto a primeira camada deliberativa prévia **Map-Tree** trata somente com labirintos T sem caminhos cíclicos e estáticos, a segunda camada deliberativa prévia **Map-Dijkstra** provê planejamento de caminho em labirintos T com ciclos e que podem ser modificados durante a operação do robô.

7.4.1 Camada Deliberativa Map-Tree

A camada deliberativa **Map-Tree** é esquematizada na Figura 7.9 e realiza a tarefa de navegação executando duas fases (exploração e planejamento) distintas e em sequência. Ou seja, com ela o robô primeiramente explora todo o labirinto e em seguida planeja caminho de um lugar origem a um lugar destino pré estabelecidos. A camada deliberativa **Map-Tree** trata somente com labirintos T estáticos e que não apresentam caminhos cíclicos no grafo topológico resultante.

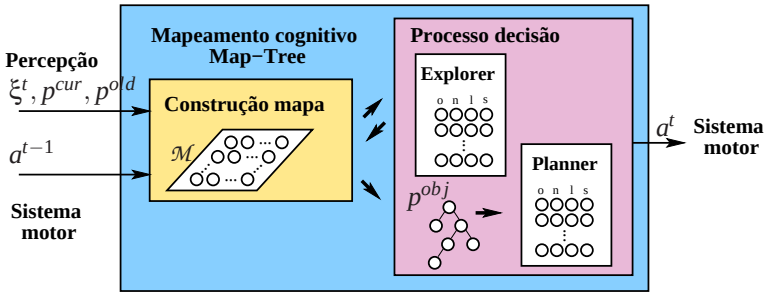


Figura 7.9: Primeira proposta: mapeamento cognitivo **Map-Tree**.

Toda vez que uma bifurcação ou beco sem saída é detectado, a camada deliberativa **Map-Tree** recebe as informações ξ^t, p^{cur} e p^{old} através do módulo **Percepção** (Seção 6.3) e a^{t-1} a partir do módulo **Sistema motor**. A camada de mapeamento cognitivo **Map-Tree** é assim denominada devido à fase de exploração do robô ser responsável por construir o mapa cognitivo enquanto a fase de planejamento de caminho transpõe este mapa para uma árvore binária a fim de determinar o planejamento de ações em cada bifurcação que levará ao lugar objetivo.

O mapa cognitivo, $\mathcal{M}$, é representado por um grafo, onde os nós representam lugares (becos sem saída e bifurcações no labirinto) e as arestas armazenam informações referentes a ações de saída e chegada entre os nós. Como descrito no Algoritmo 7.1, $\mathcal{M}$ é construído à medida que novos lugares são encontrados, e enquanto a fase de exploração é mantida (linhas 4 a 10). Assim como definido na Seção 6.5.1, $out_{p^{old}}$ e $in_{p^{cur}}$, correspondem as ações de saída em p^{old} e chegada em p^{cur}, respectivamente. A ação de saída em p^{old} é obtida a partir de $\mathcal{A}$ (linha 2), que contém os valores de pesos para as ações alternativas no lugar anteriormente alcançado, p^{old}. Já a ação de chegada em p^{cur} é obtida diretamente através de a^{t-1} (linha 3).

Algoritmo 7.1 Mapeamento cognitivo: **Map-Tree**

procedure MAP-TREE(ξ^t, p^{old}, p^{cur}, a^{t-1})
  $out_{p^{old}} \leftarrow \nabla(\mathcal{A}[p^{old}])$
  $in_{p^{cur}} \leftarrow i$ onde $a_i^{t-1} = 1$, para $i = o, n, l, s$
  if $\neg(p^{cur} \in [0, L-1])$ **then**
    $\mathcal{M}[p^{old}][p^{cur}] \leftarrow out_{p^{old}}$
    $\mathcal{M}[p^{cur}][p^{old}] \leftarrow in_{p^{cur}}$
    $A^{explorer}[p^{cur}] \leftarrow \neg\xi^t$
    $L \leftarrow L+1$
    $explorando \leftarrow$ **true**
  end if
  if $explorando$ **then** ▷ Robô explorando
    $A^{explorer}[p^{cur}][in_{p^{cur}}] \leftarrow \gamma \cdot A^{explorer}[p^{cur}][in_{p^{cur}}]$
    $A^{explorer}[p^{old}][out_{p^{old}}] \leftarrow \gamma \cdot A^{explorer}[p^{old}][out_{p^{old}}]$
    $completou\_exploracao \leftarrow \triangle(A^{explorer})$
    if $completou\_exploracao =$ **true then**
      $explorando \leftarrow$ **false** ▷ Exploração completada
      $(filho\_esq, filho\_dir) \leftarrow$ DESCOBRE_FILHOS $(\mathcal{M}, p^{origem}, \emptyset, L)$
      MAPA_PARA_ARVORE $(\mathcal{M}, L, Tree_{p^{origem}}, p^{origem}, \emptyset, filho\_esq, filho\_dir, 0)$
      $(filho\_esq, filho\_dir) \leftarrow$ DESCOBRE_FILHOS $(\mathcal{M}, p^{destino}, \emptyset, L)$
      MAPA_PARA_ARVORE $(\mathcal{M}, L, Tree_{p^{destino}}, p^{destino}, \emptyset, filho\_esq, filho\_dir, 0)$
      $p^{obj} \leftarrow p^{origem}$
      $Tree\_\text{PARA}\_\mathcal{A}(Tree_{p^{origem}}, p^{origem}, A^{planner}, \mathcal{M})$
    end if
    $\mathcal{A}[p^{cur}] \leftarrow A^{explorer}[p^{cur}]$
  else ▷ Robô navegando
    if $p^{obj} = p^{origem}$ **then**
      if $p^{cur} = p^{origem}$ **then**
        $p^{obj} \leftarrow p^{destino}$
        $Tree\_\text{PARA}\_\mathcal{A}(Tree_{p^{destino}}, p^{destino}, A^{planner}, \mathcal{M})$
      end if
    else if $p^{obj} = p^{destino}$ **then**
      if $p^{cur} = p^{destino}$ **then**
        $p^{obj} \leftarrow p^{origem}$

Algoritmo 7.1 Mapeamento cognitivo: **Map-Tree** (cont.)

```
                    Tree_PARA_A(Tree_{p^origem}, p^origem, A^planner, M)
                end if
            end if
            A[p^cur] ← A^planner[p^cur]
        end if
        a^t ← ∂(A[p^cur])
        Sistema motor(a^t)
    end procedure

    procedure DESCOBRE_FILHOS(M, no_id, pai, L)
        filhos[0] ← −1
        filhos[1] ← −1
        x ← 0
        for i ← 0, L do
            if M[no_id][i] ≠ ∅ and i ≠ pai then
                filhos[x] ← i
                x ← x + 1
            end if
        end for
        return filhos[0], filhos[1]
    end procedure

    procedure MAPA_PARA_ARVORE(M, L, Tree, no_id, pai, filho_esq,
    filho_dir, nivel)
        Tree[no_id][0] ← pai
        Tree[no_id][1] ← filho_esq
        Tree[no_id][2] ← filho_dir
        Tree[no_id][3] ← nivel
        if filho_esq ≠ −1 then
            child_left, child_right ← DESCOBRE_FILHOS (M, filho_esq,
    no_id, L)
            MAPA_PARA_ARVORE (M, L, Tree, filho_esq, no_id,
    child_esq, child_dir, nivel + 1)
        end if
        if filho_dir ≠ −1 then
            child_left, child_right ← DESCOBRE_FILHOS (M, filho_dir,
    no_id, L)
```

O subsistema **Explorer** (linhas 11 - 24) através da variável $A^{explorer}$ guia o comportamento de exploração fazendo com que todos os lugares do labirinto sejam alcançados e todas as ações alternativas nas bifurcações sejam executadas. Ao verificar que todo o labirinto foi explorado (linha 15), **Explorer** inicializa a fase de planejamento transpondo o mapa cognitivo para árvores de tomada de decisão, através do procedimento MAPA_PARA_ARVORE. Estas árvores são do tipo binária, onde o nó raiz corresponde ao lugar objetivo e os nós restantes aos nós em $\mathcal{M}$. No caso desta implementação o lugar objetivo foi pré estabelecido respectivamente como p^{origem} e $p^{destino}$(lugares origem e destino), onde $p^{obj} \leftarrow p^{origem}$ (linha 21). Ou seja, primeiramente, o robô deve planejar um caminho até o lugar origem e em seguida planejar um caminho até o lugar destino e vice versa.

Algoritmo 7.1 Mapeamento cognitivo: **Map-Tree** (cont.)

```
65:        MAPA_PARA_ARVORE (M, L, Tree, filho_dir, no_id,
           child_esq, child_dir, nivel + 1)
66:    end if
67: end procedure

68: procedure Tree_PARA_A(Tree, no_id, A, M)
69:    if Tree[no_id][1] ≠ −1 then
70:        for i ← o,s onde i = o,n,l,s do
71:            A[Tree[no_id][1]][i] ← 0
72:        end for
73:        A[Tree[no_id][1]][M[Tree[no_id][1]][no_id]] ← 1
74:        Tree_PARA_A(Tree, Tree[no_id][1], A, M)
75:    end if
76:    if Tree[no_id][2] ≠ −1 then
77:        for i ← o,s onde i = o,n,l,s do
78:            A[Tree[no_id][2]][i] ← 0
79:        end for
80:        A[Tree[no_id][2]][M[Tree[no_id][2]][no_id]] ← 1
81:        Tree_PARA_A(Tree, Tree[no_id][2], A, M)
82:    end if
83: end procedure
```

O subsistema **Planner** (linhas 25 - 38) implementa a fase de planejamento de caminho, inicializada ao término da fase de exploração. A cada lugar corrente alcançado, p^{cur}, **Planner** verifica se este corresponde ao lugar objetivo, que neste caso pode ser ambos origem ou destino. Caso o lugar

objetivo tenha sido alcançado, o procedimento *Tree*_PARA_$\mathcal{A}$ transpõe a respectiva árvore de tomada de decisão, cuja raiz é o lugar objetivo corrente para pesos de ações em $A^{Planner}$ para todos os lugares no labirinto. Este procedimento consiste de um algoritmo que percorre a árvore binária em pré-ordem, obtendo através do mapa cognitivo a informação das direções referentes a cada par de lugares origem-destino (nós pai - filho). Desta forma, o planejamento de caminho a partir de qualquer lugar até o lugar objetivo é armazenado em $A^{Planner}$ e consiste da composição de sub-caminhos que seguem a hierarquia da árvore no sentido *"bottom-up"*, para guiar o robô nas escolhas de ações que levam ao lugar objetivo (nó raiz). Portanto, o robô torna-se hábil para executar caminhos entre quaisquer dois lugares pré estabelecidos no labirinto.

Finalmente, quando $\mathcal{A}$ (linha 39) contém valores de $A^{explorer}$, caso ocorra empate dentre os valores dos pesos das ações alternativas em p^{cur}, o critério de desempate priorizará uma ação conforme a seqüência oeste, norte, leste e sul. Porém, isto não ocorre quando o robô está na fase de planejamento de caminho, onde $\mathcal{A}$ contém valores de $A^{planner}$. Neste caso, apenas um peso terá o valor 1 dentre os pesos das ações alternativas em p^{cur}.

1^o Experimento: Camada Deliberativa Map-Tree e a Evolução das Abordagens Reativas

Este primeiro experimento tem por objetivo analisar o aprendizado efetivo de camadas reativas ora implementadas pelos arranjos neurais (i)**MLP-RR**, (ii)**ART1-MLPs-RR** e (iii)**ART1-R-MLPs-RR** (Seção 5.4.2) respectivamente. Como ilustrado na Figura 7.10, as arquiteturas simuladas apresentam como camada deliberativa o processo de mapeamento cognitivo **Map-Tree** descrito anteriormente. O esquema de funcionamento destas arquiteturas é semelhante ao da arquitetura **NeuroCog**, ou seja, nos corredores do labirinto, a camada reativa é responsável pela resposta da ação, enquanto nas bifurcações e becos sem saída, a camada deliberativa atua construindo o mapa cognitivo durante a fase de exploração e planejando caminhos durante a fase de planejamento. No nível abaixo, o módulo **Percepção** atua como descrito na Seção 6.3 e o módulo **Sistema motor** como descrito na Seção 6.4, com exceção de fornecer apenas a informação da ação anterior às camadas reativa e deliberativa.

Na simulação das três arquiteturas, como medida de desempenho do aprendizado efetivo das camadas reativas (i), (ii) e (iii), foram computadas a quantidade e a ocorrência de punições aplicadas às redes MLPs, através do módulo de **Políticas de aprendizado**, bem como a quantidade de classes

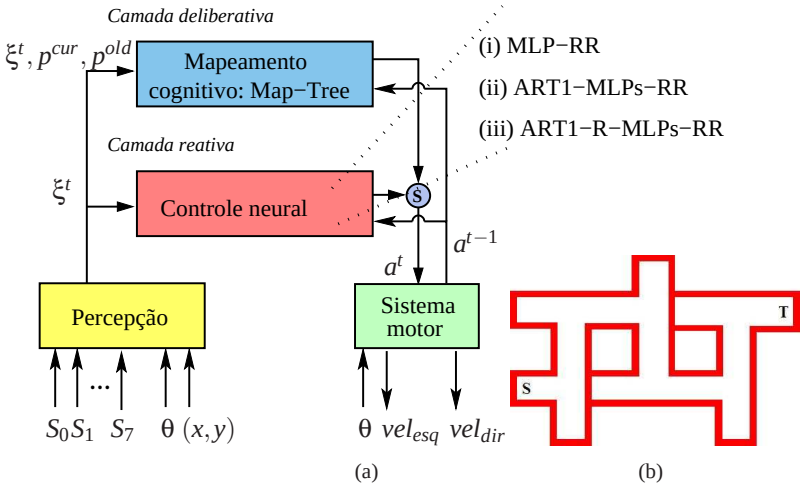


Figura 7.10: Arquitetura híbrida Map-Tree e a evolução das abordagens reativas. (a) Três arquiteturas: camada deliberativa **Map-Tree** versus controles reativos **MLP-RR**, **ART1-MLPs-RR** e **ART1-R-MLPs-RR** (b) Labirinto simulado com lugar origem em S e lugar destino em T.

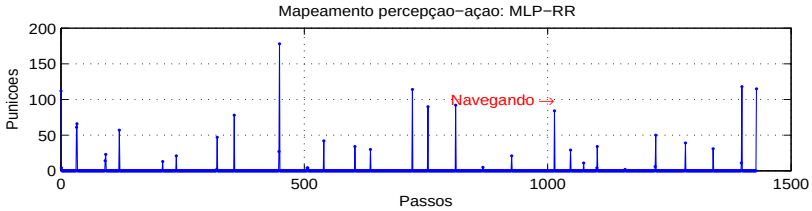
criadas pelas redes ART1. O módulo de **Políticas de aprendizado**, descrito no Algoritmo 5.2, realiza o treinamento das redes do tipo MLP, e se mantém igual para as três camadas reativas (i), (ii) e (iii). Este módulo recebe como entrada as informações ξ^t e a^{t-1} e produz como saída o processo de punição r a ser aplicada à rede MLP corrente. As informações de quantidade, bem como ocorrência de punições executadas às redes MLPs, mais a quantidade de classes criadas pela rede ART1, foram computadas durante a fase de exploração de todo labirinto, seguida da execução de um caminho planejado a partir do lugar origem S para alcançar o lugar destino T.

A Tabela 7.3 mostra os resultados obtidos para as três arquiteturas híbridas com as diferentes camadas reativas. Durante as simulações, o robô primeiramente explora todo o labirinto, realizando a construção de seu mapa cognitivo e então navega apenas uma vez entre o lugar S até o lugar destino T. Para esta tarefa são necessários em média 1400 passos (Tabela 7.3).

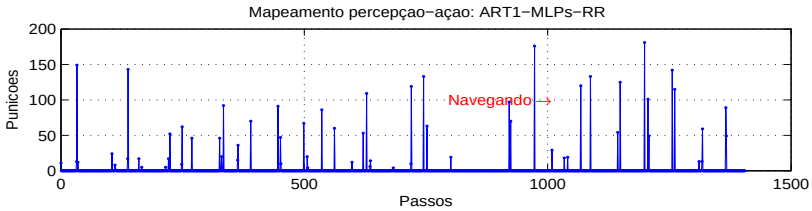
A camada reativa (i) MLP-RR é composta apenas por uma rede MLP que apresenta quatro neurônios em sua camada de entrada, para receber o padrão ξ^t, três neurônios na camada intermediária e quatro neurônios na camada de saída, para produzir a ação a^t.

Arquitetura Map-Tree / (Camada reativa)	Punições	n^o MLPs	Passos (Explor. + Naveg.)
(i) MLP-RR	1674	1	1429
(ii) ART1-MLPs-RR	3454	14	1405
(iii) ART1-R-MLPs-RR	901	29	1386

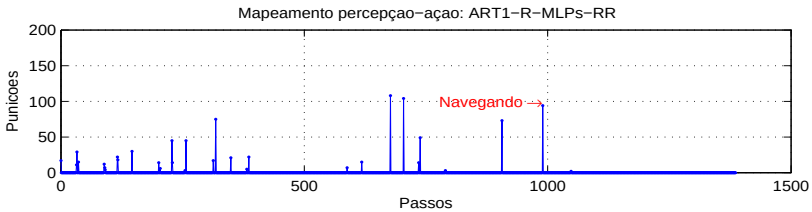
Tabela 7.3: Desempenhos da arquitetura **Map-Tree**. Camada deliberativa **Map-Tree** versus diferentes camadas reativas: medidas de desempenho dos aprendizados on-line das redes MLPs, rede ART1 e da tarefa de navegação.



(a)



(b)



(c)

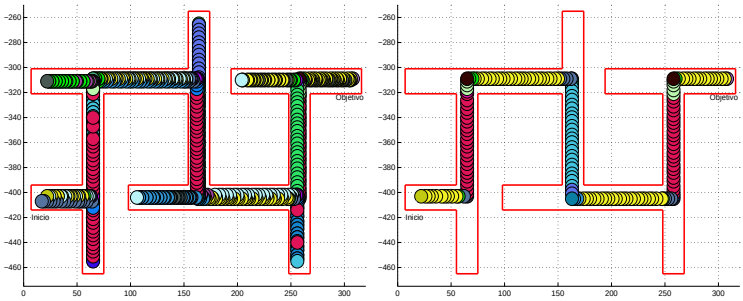
Figura 7.11: Desempenho do aprendizado online das diferentes camadas reativas das arquiteturas híbridas **Map-Tree**/[(a), (b) e (c)].

Devido à camada deliberativa Map-Tree, o robô executa corretamente o comportamento de exploração e planejamento de caminho, porém como pode ser observado no gráfico (a) da Figura 7.11, o processo de punição atua durante todo o período de execução do robô, não ocorrendo um aprendizado efetivo desta rede MLP. Isto se deve ao modo online e aleatório do algoritmo de aprendizado, associado ao estilo arquitetural da rede, conforme discutido na Seção 5.3.3.

A camada reativa (ii) ART1-MLPs-RR se caracteriza por apresentar uma rede ART1 comutadora de redes MLPs, que recebe como entrada apenas o padrão do estado do ambiente ξ^t, e por esta razão apresenta apenas 4 neurônios em sua camada F1 e 20 neurônios em sua camada F2. Como visto na Tabela 7.3, a rede ART1 deste tipo de camada reativa criou 14 redes MLPs através da ativação de seus neurônios na camada F2. Porém o aprendizado efetivo deste arranjo neural não ocorre devido à ausência da realimentação da ação executada no passo anterior, o que faz com que o robô não tenha uma memória de curto prazo que indique a direção em que estava executando. Desta forma a rede ART1 sempre seleciona uma mesma rede MLP para aprender ações corretas mas ao mesmo tempo contraditórias, o que impossibilita um aprendizado efetivo do conjunto de MLPs, conforme também pode ser visto no gráfico (b) da Figura 7.11.

Por outro lado, constata-se a ocorrência de um aprendizado efetivo da camada reativa da arquitetura (iii) Map-Tree/ART1-R-MLPs-RR. Neste caso, a rede ART1 é recorrente e possui 8 neurônios em sua camada F1 para receber o padrão de entrada (ξ^t, a^{t-1}). Além disto, a quantia de 40 neurônios para a camada F2 se mostra suficiente para realizar a categorização do espaço de padrões de entrada. Através da Tabela 7.3, nota-se a diminuição da quantidade de punições (901) aplicadas às redes MLPs, porém ocorre o aumento do número de classes (29) criadas pela rede ART1. O aprendizado efetivo deste arranjo neural é melhor visualizado no gráfico (c) da Figura 7.11, ao término da fase de exploração e início da fase de planejamento de caminho, nota-se que as redes MLPs não mais recebem punições.

Os gráficos (a) e (b) da Fig. 7.12 ilustram a ativação dos neurônios da camada F2 da rede ART1 durante a simulação da arquitetura (iii) Map-Tree/ART1-R-MLPs-RR. Em outras palavras, enquanto o robô executa as trajetórias durante a fase de exploração (a) e em seguida durante a fase de planejamento (b), são plotados cada neurônio ativado, que corresponde ao acionamento de uma respectiva rede MLP, responsável pela resposta da ação a^t, enquanto o robô percorre os corredores do labirinto.



(a) Exploração do labirinto. (b) Planejamento de caminho.

Figura 7.12: Neurônios ativados na camada F2 (rede ART1), durante a simulação da arquitetura (iii) **Map-Tree/ART1-R-MLPs-RR**.

2^o **Experimento: Parâmetros de Vigilância** ρ

O objetivo deste segundo experimento foi verificar através da arquitetura Map-Tree/ART1-R-MLPs-RR, o quanto o valor do parâmetro de vigilância, ρ, repercute no processo de aprendizado efetivo do arranjo neural que implementa a camada reativa da arquitetura. Foram testados os valores de ρ iguais a $0,6$, $0,7$, $0,8$, $0,9$ e $1,0$ respectivamente. Em cada simulação da arquitetura foram computados a quantidade de classes criadas e a porcentagem média de ações corretas executadas pelas redes MLPs, durante 8000 passos. Via de regra, para um dado conjunto de padrões a serem classificados por uma rede ART1, com um valor elevado de ρ, resulta em uma discriminação mais refinada entre classes do que com um valor mais baixo.

Portanto, nota-se através do gráfico da Figura 7.13, que o menor número de classes (20) é obtido com $\rho = 0,6$ e o maior (31) com $\rho = 0,9$ e $\rho = 1,0$. Como consequência, quanto menor o parâmetro de vigilância da rede ART1, menos eficiente se torna a aprendizagem por reforço das redes MLPs. Dado que baixos ρ'*s* ocasionam maior ocorrência de punições, verificadas através das curvas da porcentagem de acertos mais acidentadas, conforme pode ser visualizado no gráfico da Figura 7.14. Por esta razão, o valor de ρ igual a $0,9$ é o valor utilizado para a maioria das simulações apresentadas nesta tese.

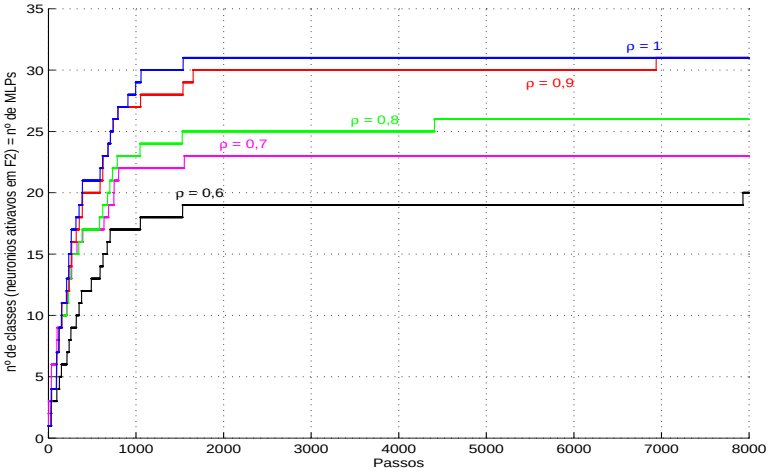


Figura 7.13: Arquitetura **Map-Tree/ART1-R-MLPs-RR** simulada a partir de diferentes valores de ρ. Número de classes criadas pelas redes ART1-R.

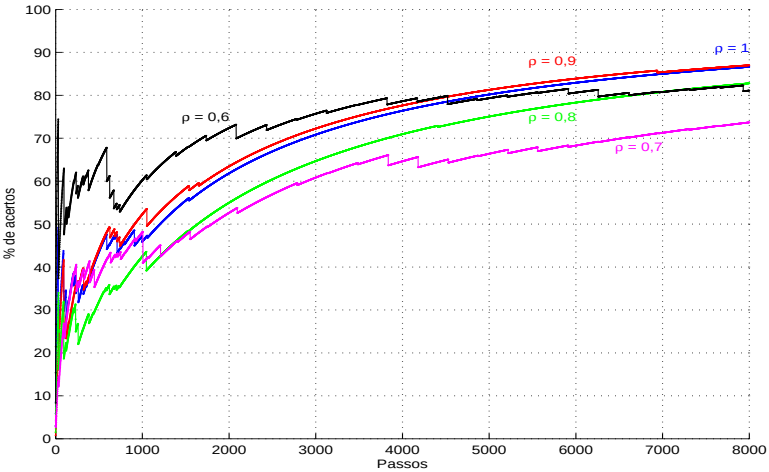


Figura 7.14: Arquitetura Map-Tree/ART1-R-MLPs-RR simulada a partir de diferentes valores de ρ. A porcentagem média de ações corretas executadas pelas redes MLPs.

3^o Experimento: Arquitetura Map-Tree/ART1-R-MLPs-RR e Navegação Planejada

Este experimento visa mostrar o funcionamento da camada deliberativa **Map-Tree** ao construir uma representação de conhecimento do ambiente, durante a fase de exploração do robô. Considera-se que o labirinto é estático e que não contém caminhos cíclicos. Nas simulações que serão apresentadas a seguir, o robô tem por tarefa de navegação explorar todo o labirinto e em seguida planejar um caminho entre os lugares origem e destino, que são pré-estabelecidos.

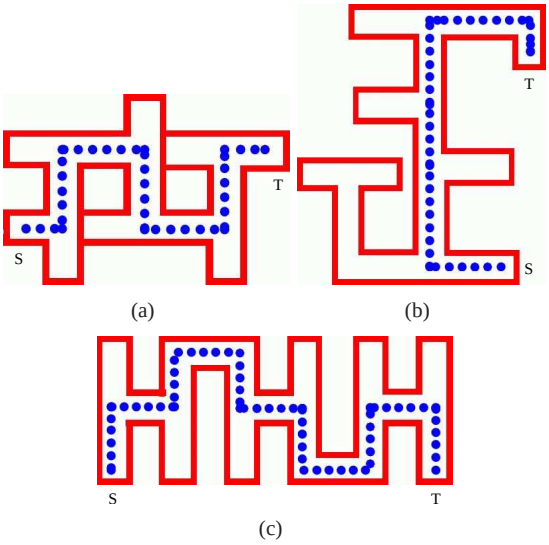


Figura 7.15: Labirintos de simulação para a arquitetura **Map-Tree/ART1-R-MLPs-RR** e os planejamentos de caminho executados.

Como ilustrado nas Figuras 7.15 (a), (b) e (c), foram utilizados três labirintos do tipo T, onde o robô iniciou sua operação a partir dos lugares nos labirintos identificados como S (lugar origem). O lugar objetivo é identificado com T. Foram computados o número de passos para completar a fase de exploração, bem como a quantidade de classes criadas (mapeamentos percepção-ação) pela rede ART1 na camada reativa da arquitetura analisada.

A Tabela 7.4 mostra os dados de simulação obtidos durante a execução do robô nos três labirintos. A fase de exploração no labirinto (a) termina com a menor quantidade de passos, pelo fato deste labirinto ser o menor em com-

Labirinto	Passos exploração	Mapeamento percepção-ação	Nº de lugares	Nº de cantos
(a)	987	27	14	0
(b)	1256	33	12	2
(c)	1425	30	14	4

Tabela 7.4: Arquitetura **Map-Tree/ART1-R-MLPs-RR**. Passos de controle para completar a fase de exploração e quantidade de mapeamentos percepção ação, nos labirintos (a), (b) e (c) da Figura 7.15.

primento de corredores. Já para o labirinto (c) são necessários mais passos para completar a operação, devido à sua maior extensão em corredores, porém ele necessita de menos classes (MLPs) em seu nível reativo, para o aprendizado do mapeamento percepção-ação, devido à frequência de apresentação de padrões à rede ART1 ser razoavelmente simétrica. A Figura 7.16 ilus-

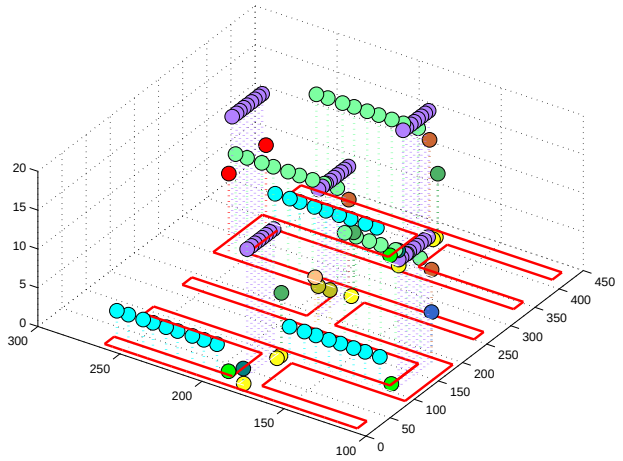


Figura 7.16: Camada reativa. Mapeamento percepção-ação para o labirinto (c) da Figura 7.15 durante a execução do caminho planejado entre os lugares S e T.

tra o mapeamento percepção-ação do labirinto (c) da Figura 7.15, realizado pela camada reativa da arquitetura **Map-Tree/ART1-R-MLPs-RR**, durante a execução do caminho planejado entre S e T . Os círculos em cores distintas ilustram a atuação de diferentes redes MLPs que determinam ações a serem

executadas de acordo com o padrão de entrada apresentado à rede ART1, enquanto o robô percorre os corredores do labirinto. Em corredores onde paredes em paralelo são detectadas, por exemplo, a rede MLP que executa a ação é a mesma durante todo o percurso em uma mesma direção (círculos com cores iguais). Assim que os sensores detectam mudanças nos corredores, quanto à configuração de paredes no labirinto, um outro neurônio vencedor na camada F2 é ativado, ativando assim uma outra rede MLP para determinar a ação a ser executada.

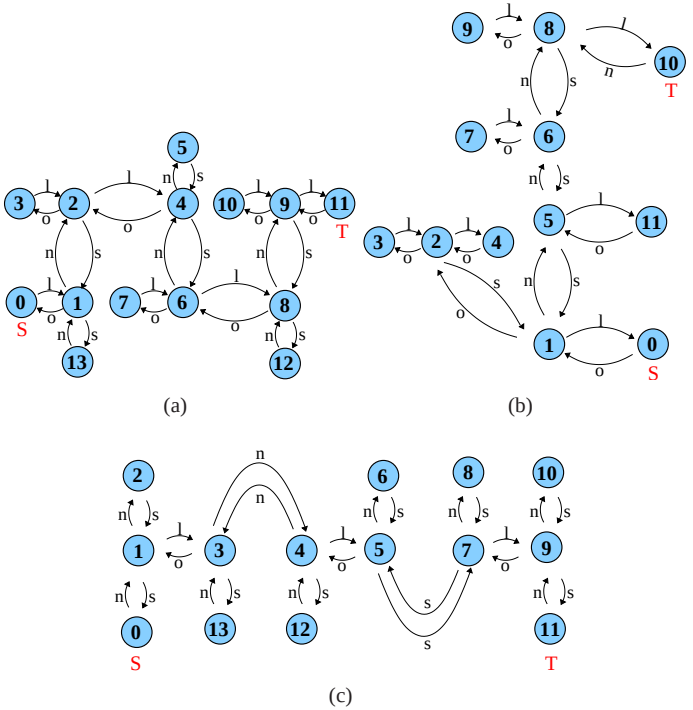


Figura 7.17: Os respectivos mapas topológicos aprendidos após a fase de exploração dos labirintos (a), (b) e (c) da Figura 7.15.

As Figuras 7.17 (a), (b) e (c) correspondem aos mapas topológicos aprendidos durante a fase de mapeamento cognitivo (exploração) dos labirintos (a), (b) e (c) da Figura 7.15. Os mapas cognitivos correspondem a grafos onde os nós equivalem aos lugares (becos sem saída e bifurcações) nos labirintos e as arestas indicam as ações (oeste, norte, leste ou sul) que levam a

um lugar vizinho. Os números que identificam os nós (lugares) do mapa são estabelecidos conforme à ordem em que são alcançados pela primeira vez. Durante a exploração, quando o robô se encontra em um lugar do tipo bifurcação e ocorre empate entre os pesos das ações alternativas (Algoritmo 7.1, linha 39), a ação corrente é selecionada, considerando-se a prioridade oeste, norte, leste e sul.

Após a fase de exploração, o mapa topológico aprendido é transpostos em árvores de decisão do tipo binária, como as que são mostradas na Figura 7.18. Estas árvores de decisão apresentam como nó raiz o lugar correspondente ao objetivo. Todo os nós restantes na árvore armazenam ações que representam sub-planos para se alcançar o nó raiz (objetivo). Visto que o robô deve, após completar a exploração de cada labirinto, primeiramente se dirigir até o lugar origem S, para em seguida alcançar o lugar destino T, antes de iniciar a fase de planejamento de caminho, são criadas as árvores de decisão, cujos nós raízes sejam estabelecidos não somente como lugares origem S, mas também como lugares destinos T, para cada labirinto (Algoritmo 7.1, linhas 18 e 20).

Deste modo, as árvores de decisão (a) I, (b) I e (c) I da Figura 7.18 correspondem aos conjuntos de planos construídos a partir dos mapas cognitivos da Figura 7.17, considerando-se os lugares origem S, nos labirintos da Figura 7.15, como lugares objetivos. Já as árvores (a) II, (b) II e (c) II correspondem igualmente aos conjuntos de planos construídos a partir dos respectivos mapas cognitivos da Figura 7.17, considerando-se os lugares destino T como objetivos.

Por exemplo, considere que no labirinto (c) da Figura 7.15, o robô detecta o fim da fase de exploração ao alcançar o último lugar do labirinto, p^{13}. Neste momento as árvores (c) I e II da Figura 7.18 são criadas, e o robô assume primeiramente a árvore (c) I, porque ele deve primeiramente executar o caminho até o lugar origem 0. Através da árvore (c) I, o robô tomará as seguintes ações (a cada lugar alcançado) para executar a trajetória entre o lugar corrente 13 até o lugar origem 0: (n, o, s). Ao alcançar o lugar objetivo 0, o lugar destino, 11, passa então a ser o novo lugar objetivo, e então o robô agora assume a árvore de decisão (c) II, executando as seguintes ações em cada lugar alcançado até chegar ao lugar objetivo 11: (n, l, n, l, s, l, s).

As trajetórias executadas, partindo-se de S com destino a T, durante a fase de planejamento de caminho, nos três labirintos analisados (a), (b) e (c), são descritas através de círculos azuis plotados na Figura 7.15.

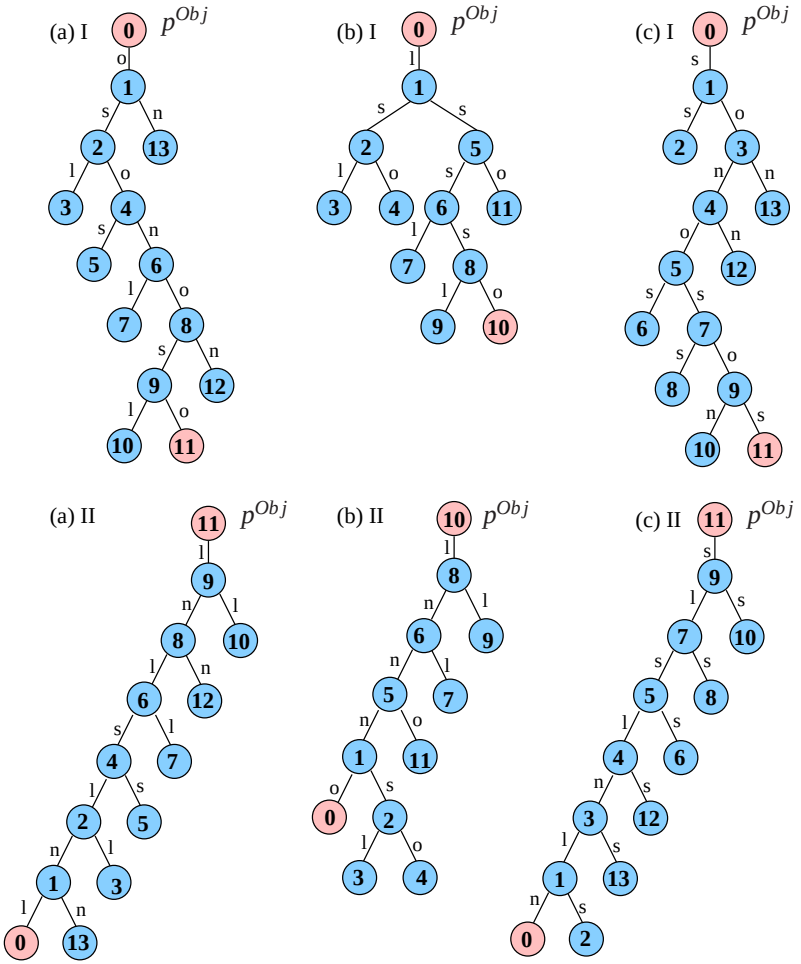


Figura 7.18: Árvores de decisão criadas a partir dos mapas cognitivos (a), (b) e (c) da Figura 7.17.

7.4.2 Camada Deliberativa Map-Dijkstra

A segunda camada deliberativa prévia, **Map-Dijkstra**, é assim denominada por obter navegação planejada e desvios através do algoritmo de Dijkstra, que utiliza o mapa cognitivo, aprendido após uma exploração completa do labirinto. Desta forma, a arquitetura híbrida, formada pela camada deliberativa **Map-Dijkstra** mais a camada reativa **ART1-R-MLPs-RR**, provê um sistema de planejamento de caminho em labirintos T com ciclos e que podem ser modificados durante a operação do robô. O desenvolvimento da camada deliberativa **Map-Dijkstra** precedeu ao desenvolvimento da camada deliberativa da arquitetura **NeuroCog**, proposta neste trabalho de tese.

A Figura 7.19 ilustra a estrutura da camada deliberativa **Map-Dijkstra** que é semelhante à camada **Map-Tree** vista na seção anterior, ou seja ambas as camadas executam em sequência, primeiro a fase de exploração do labirinto, para em seguida iniciar a fase de planejamento de caminho. A principal diferença entre as camadas **Map-Tree** e **Map-Dijkstra** é a forma como o planejamento é realizado. Na primeira, o mapa topológico do labirinto é transposto para árvores binárias de tomada de decisão, enquanto no segundo, algoritmo de Dijstra é aplicado diretamente ao mapa topológico $\mathcal{M}$.

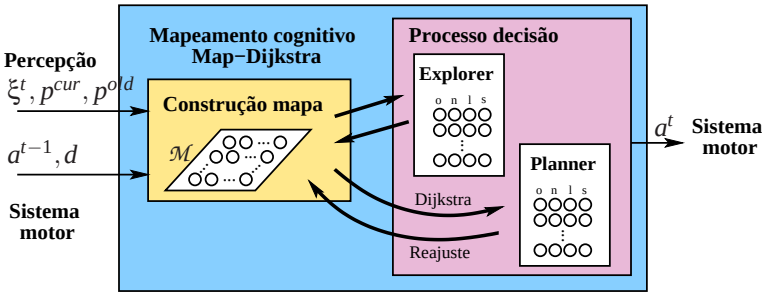


Figura 7.19: Segunda proposta: mapeamento cognitivo **Map-Djkstra**.

Assim que um marco (bifurcação ou beco sem saída) é detectado, a camada deliberativa **Map-Dijkstra** recebe as informações (ξ^t, p^{cur} e p^{old}) através do módulo **Percepção** (Seção 6.3) e (a^{t-1} e d) a partir do módulo **Sistema motor**. A informação de distância d é incorporada ao mapa cognitivo para ser utilizada pelo algoritmo de Dijkstra. Desta forma, os nós do mapa cognitivo, $\mathcal{M}$, representam lugares enquanto as arestas armazenam informações referentes a ações de saída e chegada entre os nós, mais a informação de distância d.

O Algoritmo 7.2 descreve a camada deliberativa **Map-Dijkstra**. A

Algoritmo 7.2 Camada deliberativa: **Map-Dijkstra**

procedure MAP-DIJKSTRA(ξ^t, p^{old}, p^{cur}, a^{t-1}, d,*status*)
  $out_{p^{old}} \leftarrow \nabla(\mathcal{A}[p^{old}])$
  $in_{p^{cur}} \leftarrow a^{t-1}$
  if $status = explorando$ **then** ▷ Robô explorando
    if $\neg(p^{cur} \in [0, L-1])$ **then**
      $\mathcal{M}[p^{old}][p^{cur}] \leftarrow out_{p^{old}}$
      $\mathcal{M}[p^{cur}][p^{old}] \leftarrow in_{p^{cur}}$
      $\mathcal{M}[p^{old}][p^{cur}] \leftarrow \mathcal{M}[p^{cur}][p^{old}] \leftarrow d$
      $A^{explorer}[p^{cur}] \leftarrow \neg\xi^t$
      $L \leftarrow L+1$
    end if
    $A^{explorer}[p^{cur}][in_{p^{cur}}] \leftarrow \gamma \cdot A^{explorer}[p^{cur}][in_{p^{cur}}]$
    $A^{explorer}[p^{old}][out_{p^{old}}] \leftarrow \gamma \cdot A^{explorer}[p^{old}][out_{p^{old}}]$
    $\mathcal{A} \leftarrow A^{explorer}[p^{cur}]$
    $completou\_exploracao \leftarrow \triangle(A^{explorer})$
    if $completou\_exploracao =$ **true then**
      $status \leftarrow navegando$
      PLANEJA_CAMINHO(p^{origem}, $p^{destino}$, $A^{planner\_origem}$, $A^{planner\_destino}$)
      $p^{obj} \leftarrow p^{origem}$
      $\mathcal{A} \leftarrow A^{planner\_origem}[p^{cur}]$
    end if
  else if $status = navegando$ **then** ▷ Robô navegando
    if $p^{obj} = p^{origem}$ **then**
      if $p^{cur} = p^{origem}$ **then**
        $p^{obj} \leftarrow p^{destino}$
        $\mathcal{A} \leftarrow A^{planner\_destino}$
      end if
    else if $p^{obj} = p^{destino}$ **then**
      if $p^{cur} = p^{destino}$ **then**
        $p^{obj} \leftarrow p^{origem}$
        $\mathcal{A} \leftarrow A^{planner\_origem}$
      end if
    end if
    if $\neg(p^{cur} \in [0, L-1])$ **then** ▷ Mudança no labirinto: desvio
      $A^{explorer}(p^{cur}) \leftarrow \neg\xi^t$
      $L \leftarrow L+1$
      $p^{exp} \leftarrow \cdot(\mathcal{M}, p^{old}, out_{p^{old}})$

Algoritmo 7.2 Camada deliberativa: **Map-Dijkstra** (cont.)

38: $\mathcal{M}[p^{old}][,p^{exp}] \leftarrow \mathcal{M}[p^{exp}][p^{old}] \leftarrow \emptyset$
39: $\mathcal{M}[p^{old}][,p^{cur}] \leftarrow out_{p^{old}}$
40: $\mathcal{M}[p^{cur}][,p^{old}] \leftarrow in_{p^{cur}}$
41: $\mathcal{M}[p^{old}][,p^{cur}] \leftarrow \mathcal{M}[p^{cur}][p^{old}] \leftarrow d$
42: PLANEJA_CAMINHO(p^{origem}, $p^{destino}$, $\mathcal{M}$, $A^{planner\_origem}$, $A^{planner\_destino}$)
43: **end if**
44: **end if**
45: $a^t \leftarrow \partial(\mathcal{A}[p^{cur}])$)
46: **Sistema motor**(a^t)
47: **end procedure**

fase de exploração é executada através das linhas 4 a 21, que corresponde à implementação do subsistema **Explorer** (Figura 7.19). O mapa topológico $\mathcal{M}$ é construído, à medida que novos lugares são encontrados (linhas 5 a 11). O mapa recebe as informações $out_{p^{old}}$ e $in_{p^{cur}}$, correspondentes às ações de saída em p^{old} e chegada em p^{cur}, respectivamente (Seção 6.5.1). A ação de saída em p^{old} é obtida a partir de $\mathcal{A}$ (linha 2), que contém os valores de pesos para as ações alternativas no lugar anteriormente alcançado, p^{old}. Já a ação de chegada em p^{cur} é obtida diretamente através de a^{t-1} (linha 3). Além disso, o comportamento de exploração é implementado através da variável $A^{explorer}$, que permite que todos os lugares do labirinto sejam alcançados e todas as ações alternativas nas bifurcações sejam executadas.

Ao verificar que todo o labirinto foi explorado (linha 16), o subsistema **Explorer** inicializa a fase de planejamento, mudando a variável *status* para "*navegando*" e invocando o procedimento PLANEJA_CAMINHO (Algoritmo 7.3). Este procedimento transpõe diretamente o conjunto de planos, calculados a partir do algoritmo de Dijkstra, para as variáveis $A^{planner\_origem}$ e $A^{planner\_destino}$. O algoritmo de Dijkstra recebe como parâmetro o mapa cognitivo e os lugares referentes à origem e destino para cálculo do menor caminho. Devido à tarefa de navegação, os lugares p^{origem} e $p^{destino}$ (origem e destino), foram respectivamente estabelecidos como lugar objetivo, porque o robô deve, ao completar a exploração, primeiramente planejar um caminho até o lugar origem (linha 19) e em seguida, planejar um caminho até o lugar destino e vice versa.

A fase de planejamento de caminho, compreendida pelo subsistema **Planner** (linhas 22 - 44), verifica se o lugar corrente alcançado, p^{cur}, corresponde ao lugar objetivo. Neste caso, o lugar objetivo corrente é trocado

Algoritmo 7.3 Planejamento de caminho **Map-Dijkstra**

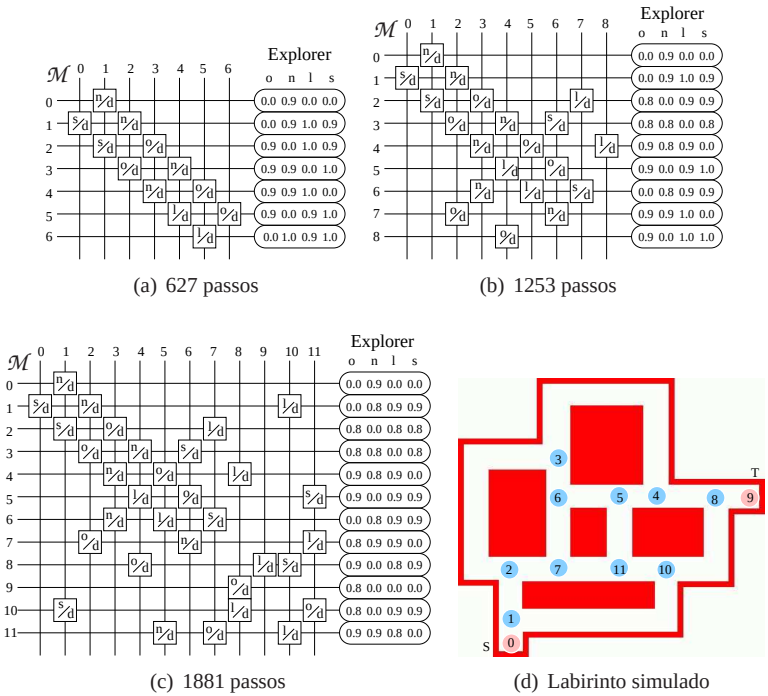
1: **procedure** PLANEJA_CAMINHO(p^{origem}, $p^{destino}$, $\mathcal{M}$, $A^{planner\_origem}$, $A^{planner\_destino}$)
2: **for** $i \leftarrow 0, L$ **do**
3: **if** $i \neq p^{origem}$ **and** $i \neq p^{destino}$ **then**
4: $A^{planner\_origem} \leftarrow dijkstra(i, p^{origem}, \mathcal{M})$
5: $A^{planner\_destino} \leftarrow dijkstra(i, p^{destino}, \mathcal{M})$
6: **end if**
7: **end for**
8: $A^{planner\_origem} \leftarrow dijkstra(p^{destino}, p^{origem}, \mathcal{M})$
9: $A^{planner\_destino} \leftarrow dijkstra(p^{origem}, p^{destino}, \mathcal{M})$
10: **end procedure**

(linhas 25 e 30), e um novo conjunto de planos de ações para se chegar ao novo lugar objetivo é estabelecido (linhas 26 e 31). E ainda, devido à possibilidade de ocorrer mudanças no labirinto durante a operação do robô, o procedimento PLANEJA_CAMINHO calcula através do algoritmo de Dijkstra os menores caminhos entre todos os outros lugares do labirinto até o lugar objetivo corrente (Algoritmo 7.3, linhas 2 a 7). Desta forma, o planejamento de caminho a partir de qualquer lugar no labirinto até o lugar objetivo é armazenado em $\mathcal{A}$. A variável $\mathcal{A}$ guiará o robô nas escolhas de ações que levam ao lugar objetivo e permitirá ainda que, durante a fase de planejamento, o robô realize desvios quando for detectados bloqueios em caminhos planejados. Em outras palavras, quando o robô detecta um novo beco sem saída (bloqueio) (Algoritmo 7.2, linha 34), o subsistema **Planner** insere o novo lugar no mapa cognitivo, efetuando os devidos reajustes, através da informação do lugar esperado p^{exp}, obtido a partir de $\mathcal{M}$ (linha 37).

Assim como na camada **Map-Tree**, na camada **Map-Dijkstra** quando $\mathcal{A}$ (linha 45) contém valores de $A^{explorer}$, caso ocorra empate dentre os valores dos pesos das ações alternativas em p^{cur}, o critério de desempate priorizará uma ação conforme a seqüência oeste, norte, leste e sul. Porém, isto não ocorre quando o robô está "navegando", onde $\mathcal{A}$ contém valores $A^{planner\_origem}$ ou $A^{planner\_destino}$. Nestes casos, apenas um peso terá o valor 1 dentre os pesos das ações alternativas em p^{cur}.

1^o Experimento: Mapeamentos Cognitivo Map-Dijkstra e Percepção-Ação

Este experimento tem por objetivo exemplificar o funcionamento da arquitetura de controle híbrida, formada pela camada deliberativa **Map-Dijkstra** e pela camada reativa **ART1-R-MLPs-RR**. A simulação executada no labirinto (d) da Figura 7.20, será descrita em duas partes. A primeira mostra um exemplo de como é realizado o mapeamento cognitivo (camada deliberativa), e a segunda ilustra o funcionamento do mapeamento percepção-ação (camada reativa), durante a operação do robô. O labirinto apresenta caminhos cíclicos e pode ser modificado quando o robô está na fase de planejamento. No labirinto simulado, os lugares origem e destino são pré definidos como S e T respectivamente. A tarefa do sistema de navegação implementado pela arqui-



(a) 627 passos (b) 1253 passos

(c) 1881 passos (d) Labirinto simulado

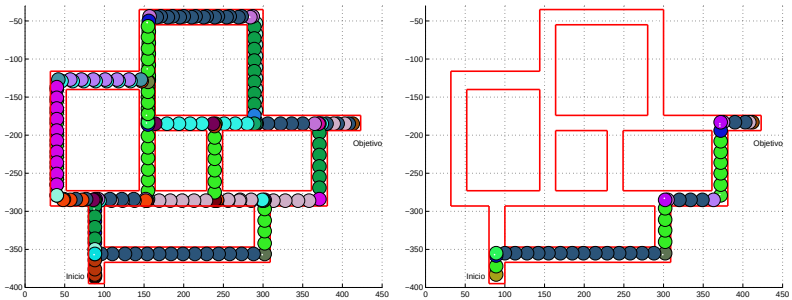
Figura 7.20: Exemplo de construção do mapa cognitivo.

tetura híbrida **Map-Dijkstra**/**ART1-R-MLPs-RR** é primeiramente explorar todo o labirinto para em seguida planejar o menor caminho entre os lugares S

a T.

Os gráficos (a), (b) e (c) da Figura 7.20 descrevem em três etapas subsequentes, a exploração completa e a construção do mapa cognitivo do labirinto (d). Cada gráfico apresenta a configuração do mapa cognitivo no momento em que o robô executou os respectivos passos de controle. À direita de cada mapa são apresentados os valores da variável $A^{explorer}$ gerenciadas pelo subsistema Explorer, que segue a prioridade oeste, norte, leste e sul, quando ocorre empate entre os valores dos pesos das ações alternativas em $A^{explorer}$, para um dado lugar corrente.

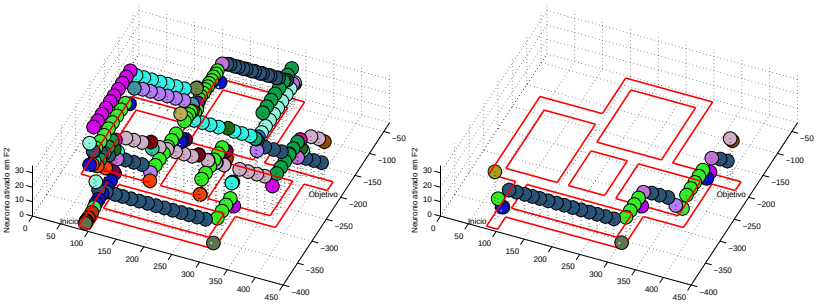
Por exemplo, o mapa topológico $\mathcal{M}$ em (a), Figura 7.20, é obtido com 627 passos de controle. Ou seja, até este momento, o robô tem alcançado 7 lugares no labirinto, onde são incorporados 7 nós em seu grafo topológico. O grafo armazena informações referentes as ações de entrada e saída entre nós vizinhos (o, n, l, ou s) mais a distância percorrida em número de passos de controle (d). Neste primeiro intervalo de exploração, entre os passos 1 até 627, o robô percorreu a seguinte trajetória: $0\ \overrightarrow{n}\ 1\ \overrightarrow{n}\ 2\ \overrightarrow{o}\ 3\ \overrightarrow{n}\ 4\ \overrightarrow{o}\ 5\ \overrightarrow{o}\ 6$. No segundo intervalo, entre os passos 627 até 1253, o robô obtém o mapeamento cognitivo correspondente ao gráfico (b), onde mais dois novos lugares são adicionados a $\mathcal{M}$, para tal o robô executa a seguinte trajetória: $6\ \overrightarrow{n}\ 3\ \overrightarrow{o}\ 2\ \overrightarrow{l}\ 7\ \overrightarrow{n}\ 6\ \overrightarrow{n}\ 3\ \overrightarrow{n}\ 4\ \overrightarrow{l}\ 8$. E finalmente, no terceiro intervalo, entre os passos 1254 a 1881 a fase de exploração é completada, com a inclusão de mais três lugares restantes a $\mathcal{M}$, ao passo que o robô executa a seguinte trajetória: $8\ \overrightarrow{l}\ 9\ \overrightarrow{o}\ 8\ \overrightarrow{s}\ 10\ \overrightarrow{o}\ 11\ \overrightarrow{o}\ 7\ \overrightarrow{o}\ 2\ \overrightarrow{s}\ 1\ \overrightarrow{l}\ 10\ \overrightarrow{l}\ 11\ \overrightarrow{n}\ 5$. E assim o mapeamento o cognitivo do labirinto é apresentado no gráfico (c) da Figura 7.20.



(a) Exploração (b) Planejamento

Figura 7.21: Visualização 2D: mapeamento percepção-ação durante as fases de exploração e planejamento de caminho.

A segunda etapa deste experimento ilustra o funcionamento do mapeamento percepção-ação aprendido pela camada reativa **ART1-R-MLPs-RR**, durante a operação da arquitetura híbrida **Map-Dijkstra/ART1-R-MLPs-RR**. Através dos gráficos (a) e (b) da Figura 7.21 é descrita, através da visualização no plano, a atuação das redes MLPs ativadas pela rede ART1, durante as fases de exploração do labirinto e planejamento de caminho respectivamente. Os gráficos (a) e (b) da Figura 7.22 correspondem aos mesmos gráficos anteriores, apenas sendo visualizados no espaço. Em ambas as figuras, os círculos em cores plotados ao longo das trajetórias de exploração e planejamento de caminho entre os lugares S e T, correspondem a redes MLPs ativadas pelo neurônio vencedor na camada F2 da rede ART1. Em outras palavras, cada cor corresponde a uma ação respondida por uma rede MLP específica enquanto o robô interage com o ambiente nos corredores. Em especial nos gráficos (a) e (b) da Figura 7.22, os índices das redes MLPs são plotados ao longo do eixo *z*.

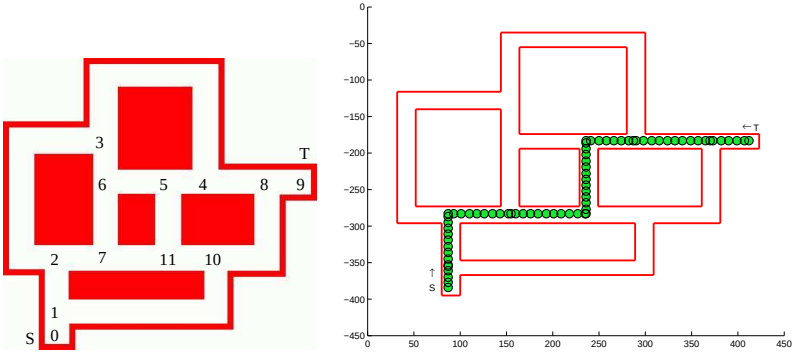


(a) Exploração (b) Planejamento

Figura 7.22: Visualização 3D: mapeamento percepção-ação durante as fases de exploração e planejamento de caminho.

2^{o} Experimento: Comportamentos de Desvio

Este experimento tem por objetivo demonstrar comportamentos de desvio implementados pela arquitetura **Map-Dijkstra/ART1-R-MLPs-RR**. O robô foi novamente introduzido no mesmo labirinto do experimento anterior, [Figura 7.23 (a)], onde foram estabelecidos os lugares S e T como origem e destino respectivamente. Em seguida, o robô executa o menor caminho entre S e T descrito pela trajetória da Figura 7.23 (b). O mapa cognitivo, $\mathcal{M}$



(a) (b)

Figura 7.23: Arquitetura **Map-Dijkstra/ART1-MLPs-RR**. (a) Mapeamento do labirinto após exploração completa. (b) Um dos menores caminhos planejados entre S e T.

aprendido neste experimento é similar ao obtido na Figura 7.20 (c), apenas com relação às informações de ações de entrada e saída entre nós vizinhos, sendo assim ambos os lugares S e T são identificados em $\mathcal{M}$ como 0 e 9 respectivamente. Entretanto, a diferença existente entre o mapa topológico aprendido neste experimento e o do anterior reside no fato de que as distâncias d entre os mesmos pares de lugares vizinhos não coincidem às vezes, devido à atuação da camada reativa **ART1-R-MLPs-RR** nos corredores. Como a distância d corresponde à quantia de passos de controle executados entre um lugar e outro, esta quantia de passos pode variar entre simulações de mesmo protocolo, devido principalmente ao aprendizado aleatório das redes MLPs. Logo, o menor caminho será determinado, pelo algoritmo de Dijkstra, como aquele que possui a menor soma de d's entre os lugares início e objetivo.

Por esta razão, enquanto no experimento anterior [Figura 7.21 (b)] o robô executou o menor caminho navegando entre os lugares ($0 \sim 1 \sim 10 \sim 8 \sim 9$), neste 2^o experimento, a trajetória executada compreende os lugares ($0 \sim 1 \sim 2 \sim 7 \sim 11 \sim 5 \sim 4 \sim 8 \sim 9$) [Figura 7.23 (b)]. Contudo, existem ainda dois outros menores caminhos que também seriam válidos e que poderiam ser executados, tais como o ($0 \sim 1 \sim 2 \sim 7 \sim 11 \sim 10 \sim 8 \sim 9$) e o ($0 \sim 1 \sim 2 \sim 7 \sim 6 \sim 5 \sim 4 \sim 8 \sim 9$).

Em seguida, comportamentos de desvio são observados, enquanto o robô executa o menor caminho saindo de S com destino a T e durante o percurso, detecta a presença de bloqueios, como ilustrados no labirinto (a) da

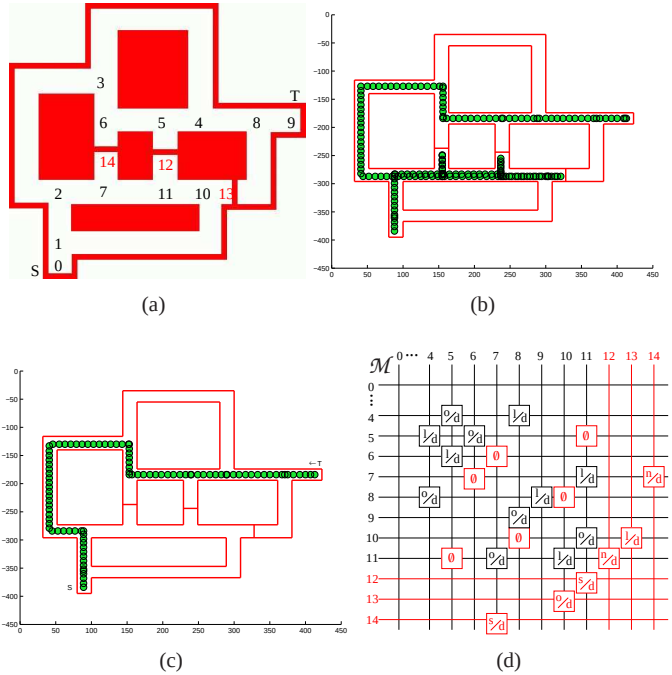


Figura 7.24: Arquitetura **Map-Dijkstra/ART1-MLPs-RR**. (a) Modificações no labirinto: bloqueios em caminhos conhecidos. (b) Comportamentos de desvio executados. (c) Comportamento de desvio final. (d) Mapa cognitivo final.

Figura 7.24. Conforme pode ser visualizado na Figura 7.24 (b) o robô parte do lugar S com o plano de executar a trajetória ($0 \sim 1 \sim 2 \sim 7 \sim 11 \sim 5 \sim 4 \sim 8 \sim 9$), mas ao sair do lugar 11 para alcançar o lugar 5, por exemplo, o robô detecta um bloqueio, o qual reconhece como um beco sem saída, atribuindo a este novo lugar a identificação 12 [Figura 7.24 (a)]. Neste momento, o mapa cognitivo é reajustado [Figura 7.24 (d)], onde $\mathcal{M}[11][5] \leftarrow \mathcal{M}[5][11] \leftarrow \emptyset$, $\mathcal{M}[11][12] \leftarrow n$ e $\mathcal{M}[12][11] \leftarrow s$, e a partir de lugar 12, um novo caminho de desvio é planejado: ($12 \sim 11 \sim 10 \sim 8 \sim 9$). Porém, quando o robô saí do lugar 10 para alcançar o lugar 8, um novo bloqueio é detectado no percurso, deste modo, o novo beco sem saída é identificado como 13 [Figura 7.24 (a)]. E neste lugar, o mapa cognitivo é novamente reajustado: $\mathcal{M}[10][8] \leftarrow \mathcal{M}[8][10] \leftarrow \emptyset$, $\mathcal{M}[10][13] \leftarrow l$ e $\mathcal{M}[13][10] \leftarrow o$ [Figura 7.24

(d)]. Em seguida, a partir do lugar 13, o novo caminho de desvio planejado é então (13 $\sim$ 10 $\sim$ 11 $\sim$ 7 $\sim$ 6 $\sim$ 5 $\sim$ 4 $\sim$ 8 $\sim$ 9). Contudo a presença do terceiro bloqueio entre os lugares 7 e 6, faz com que o robô detecte o lugar 14, reajuste o mapa cognitivo para $\mathcal{M}$ [7][6] $\leftarrow \mathcal{M}$ [6][7] $\leftarrow \emptyset$, $\mathcal{M}$ [7][14] $\leftarrow n$ e $\mathcal{M}$ [14][7] $\leftarrow s$, e em seguida recalcule um novo caminho de desvio que compreende: (14 $\sim$ 7 $\sim$ 2 $\sim$ 3 $\sim$ 6 $\sim$ 5 $\sim$ 4 $\sim$ 8 $\sim$ 9). Finalmente, após o robô não mais detectar bloqueios durante o percurso entre S e T, o menor caminho final encontrado é descrito através da trajetória ilustrada na Figura 7.24 (c): (0 $\sim$ 1 $\sim$ 2 $\sim$ 3 $\sim$ 6 $\sim$ 5 $\sim$ 4 $\sim$ 8 $\sim$ 9).

Portanto, a arquitetura **Map-Dijkstra/ART1-R-MLPs-RR** provê ao robô comportamentos de desvios mediante modificações no labirinto durante sua operação. Porém estas mudanças incluem somente o bloqueio de caminhos conhecidos e ainda, elas podem ocorrer somente quando o robô estiver na fase de planejamento de caminho. Isto caracteriza pouca flexibilidade quanto ao tratamento das dinâmicas de um ambiente. Porém a arquitetura **Map-Dijkstra/ART1-R-MLPs-RR** serviu como ponto de partida para o desenvolvimento da arquitetura **NeuroCog**, que trata de maneira mais flexível modificações no ambiente, intercalando por exemplo as fases de exploração e planejamento de caminho.

3^{o} Experimento: Comparando Prioridades de Exploração

Este experimento, tem por objetivo comparar diferentes aspectos do desempenho da arquitetura **Map-Dijkstra/ART1-R-MLP-RR** mediante diferentes opções de prioridades a ser seguida pela camada deliberativa, nas bifurcações, durante a fase de exploração do labirinto. Uma prioridade estabelecida é utilizada apenas quando há empate entre pesos de maior valor dentre as ações alternativas, caso contrário, a ação determinada é aquela que apresenta o maior peso. Em todas as simulações apresentadas até o momento, as camadas deliberativas prévias **Map-Tree** (Algoritmo 7.1, linha 39) e **Map-Dijkstra** (Algoritmo 7.2, linha 45) utilizaram a prioridade oeste, norte, leste e sul, na ocorrência de empate entre pesos de ações na variável $\mathcal{A}$.

Neste experimento a prioridade de escolha estabelecida como (oeste, norte, leste e sul) é denominada como "horária". Além desta, duas outras prioridades de escolha são definidas, tais como a "anti-horária" (sul, leste, norte e oeste) e a "aleatória", a qual se determina uma ação, dentre ações de pesos iguais, de forma aleatória. O objetivo é investigar se estas diferentes prioridades de escolha interferem, por exemplo, no tempo para se completar a fase de exploração, ou na quantidade de classes e consequentemente na quantidade de punições aplicadas às redes MLPs na camada reativa da arquitetura

de controle em questão.

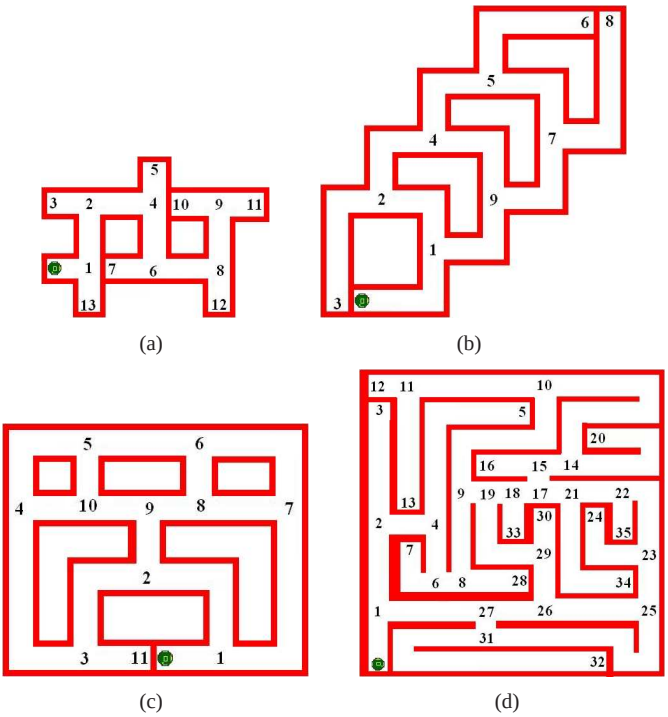


Figura 7.25: Labirintos simulados para a avaliação do desempenho da arquitetura **Map-Dijkstra/ART1-R-MLPs-RR**, segundo as prioridades de exploração “horária”, “anti-horária” e “aleatória”.

Desta forma, foram executadas três simulações para cada um dos labirintos (a), (b), (c) e (d) ilustrados na Figura 7.25. As simulações realizadas se diferenciam pelas prioridades de escolha (horária, anti-horária e aleatória) estabelecidas na camada deliberativa. Em cada labirinto, o robô simulado se encontra sobreposto ao lugar identificado como 0, que corresponde ao lugar onde ele inicia sua tarefa de exploração. A cada simulação, o robô executa uma exploração completa, onde ao final são anotadas medidas de desempenhos, relacionadas à utilização de recursos de tempo e memória. As numerações indicadas nos labirintos da Figura 7.25 exemplificam a seqüência de identificações dos lugares encontrados e inseridos no mapa cognitivo, à medida que o robô explora os labirintos, seguindo a prioridade “horária” na

camada deliberativa **Map/Dijkstra**.

Os desempenhos da arquitetura **Map-Dijkstra/ART1-R-MLPs-RR**, a partir das simulações realizadas, são descritos através dos gráficos (a), (b), (c) e (d) da Figura 7.26. Eles medem respectivamente a (a) quantidade de passos de controle, o (b) tempo em segundos, a (c) quantidade de classes criadas pela rede ART1 (que determina a quantidade de redes MLPs alocadas) e a (d) quantidade de punições aplicadas às redes MLPs, para completar uma exploração completa dos labirintos apresentados na Figura 7.25.

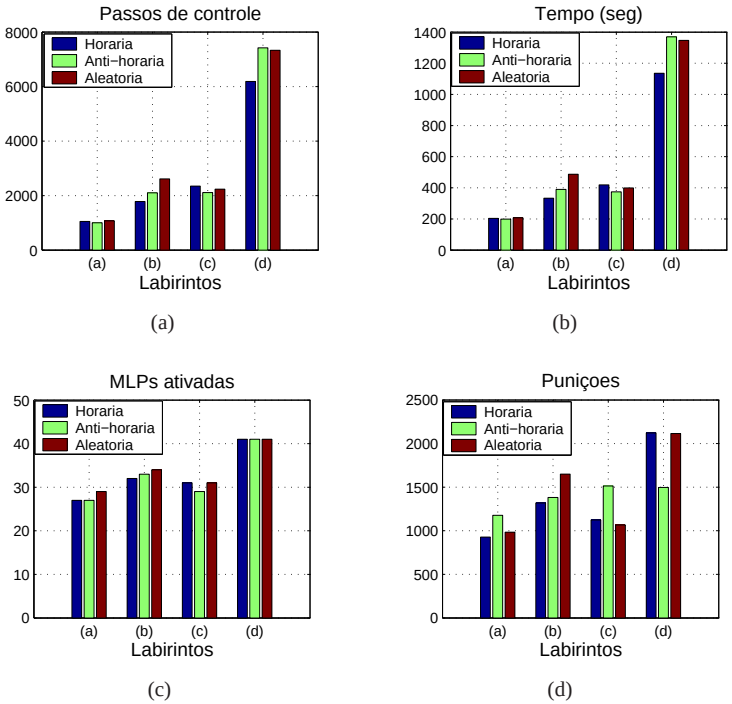


Figura 7.26: Medidas de desempenho relacionadas às prioridades “horária”, “anti-horária” e “aleatório”, estabelecidas pela camada deliberativa, para a tarefa de exploração dos labirintos (a), (b), (c) e (d) da Figura 7.25.

Os gráficos relacionados a passos de controle e tempo em segundos [Figura 7.26 (a) e (b)] necessários à execução de uma exploração completa dos labirintos são equivalentes e constituem desempenhos aceitáveis com relação ao consumo de tempo. Nota-se também que quanto maior o labirinto,

maior o tempo despendido para completar a tarefa. Por outro lado, as diferentes medidas relacionadas às prioridades "horária", "anti-horária" e "aleatória", em cada labirinto, não apresentam diferenças significativas e demonstram um desempenho médio.

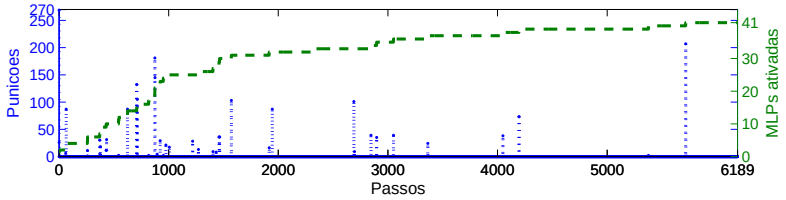
Já os gráficos relacionados à quantidade de redes MLPs ativadas e punições aplicadas [Figura 7.26 (c) e (d)] correspondem aos custos da atuação da camada reativa **ART1-R-MLPs-RR**, durante o processo de exploração dos labirintos. A quantidade de redes MLPs corresponde diretamente à classificação de padrões executada pela rede ART1, e ainda, esta classificação sofre influência quanto à sequência e frequência com que os padrões de entrada são apresentados à rede. Já com relação à quantidade de punições aplicadas, estas medidas estão relacionadas em parte à quantidade de classes ou MLPs criadas. Entretanto, estas são as medidas mais desiguais entre as prioridades de exploração estabelecidas, devido à característica aleatória do aprendizado online das rede MLPs.

Portanto, com base nestes quatro tipos de informações coletadas, pode-se concluir que as diferentes prioridades de exploração consomem em média a mesma quantia de recursos de memória e tempo e que as pequenas diferenças observadas quanto ao desempenho da arquitetura **Map-Dijkstra/ART1-R-MLPs-RR** dependerá do tamanho e da configuração do formato dos labirintos.

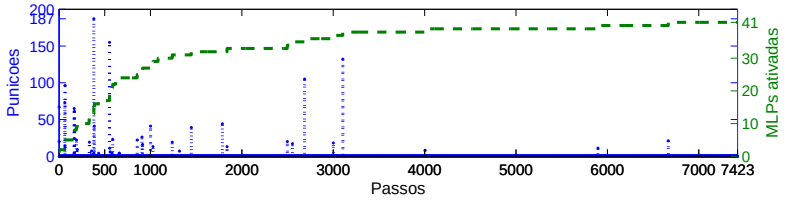
Já os gráficos (a), (b) e (c) da Figura 7.27 mostram respectivamente as informações sobre punições e quantidade de MLPs ativadas mediante as prioridades de exploração "horária", "anti-horária" e "aleatória" estabelecidas pela camada deliberativa, durante as simulações no labirinto (d).

4^{o} Experimento: Comparando Medidas de Exploração Média

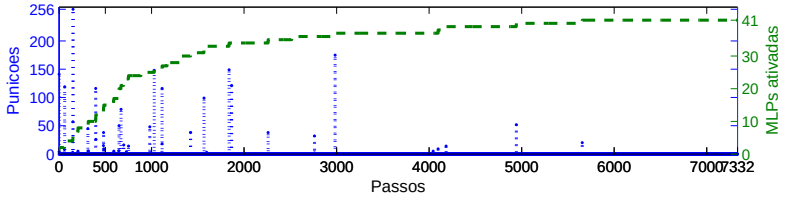
Este experimento tem por objetivo avaliar o desempenho da arquitetura **Map-Dijkstra/ART1-R-MLPs-RR**, comparando medidas de exploração média, para as simulações realizadas no experimento anterior, relacionadas a diferentes prioridades de exploração. Em adição, para efeito de comparação, outra modalidade de exploração, denominada randômica, foi utilizada nas simulações da arquitetura para os labirintos (a), (b), (c) e (d) da Figura 7.25. Com o intuito de medir a exploração média em cada labirinto, são reunidos a cada lugar visitado, a quantidade total de lugares descobertos até o momento, L, e a quantidade de ações alternativas ainda não executadas, com relação a todos os lugares já visitados. Desta forma, a exploração média pode



(a) Labirinto (d) da figura 7.25: prioridade de exploração “horária”.



(b) Labirinto (d) da figura 7.25: prioridade de exploração “anti-horária”.



(c) Labirinto (d) da figura 7.25: prioridade de exploração “aleatória”.

Figura 7.27: Desempenhos da camada reativa **ART1-R-MLPs-RR** versus prioridades de exploração estabelecidas pela camada deliberativa, para o labirinto (d) da Figura 7.25.

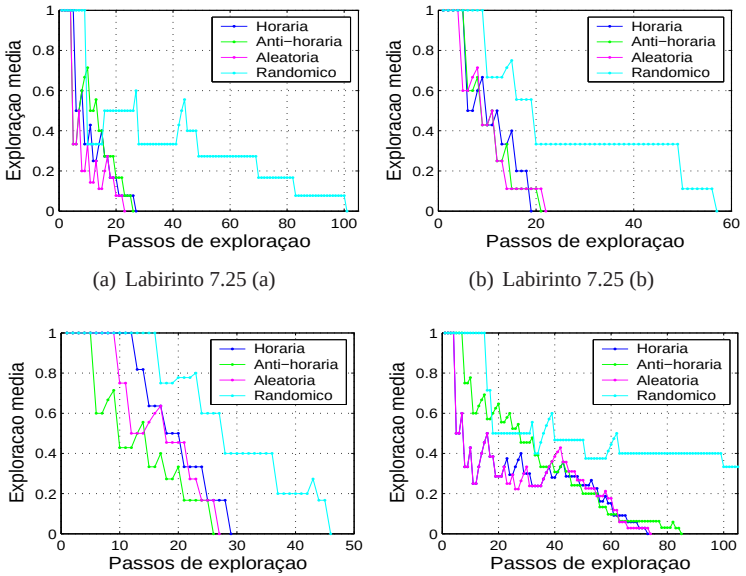
ser expressa através da seguinte função:

$$\overline{\mathcal{A}(t)} = \frac{\sum_{j=0}^{L-1} A_t^{explorer}[j][i]}{L} \quad (\forall A^{explorer}[p^j][i] = 1), \quad (i = o,n,l,s) \text{ e } (0 \leq j < L) \tag{7.2}$$

Uma ação que ainda não foi executada corresponde àquela que apresenta valor de peso igual 1 na variável $A^{explorer}$ (Algoritmo 7.2, linhas 9, 12 e 13). Sendo assim, a exploração média, $\overline{\mathcal{A}(t)}$, equivale ao somatório de todas as ações ainda não executadas, dividido pela quantidade total de lugares no

mapa cognitivo, L, a cada passo de exploração t. A variável t representa um macro passo de tempo que é incrementado de 1 a cada percurso transcorrido entre um lugar anteriormente alcançado e um lugar corrente. Quando a exploração média se encontra no intervalo $(0 < \overline{\mathcal{A}(t)} \leq 1)$, denota que ainda há possíveis regiões no labirinto a serem exploradas. Caso contrário, quando $\overline{\mathcal{A}(t)}$ é igual a zero, significa que uma exploração completa do labirinto foi finalizada.

Como dito anteriormente, na modalidade de exploração randômica (THRUN, 1992), as experiências do robô são desconsideradas. Em outras palavras, a cada bifurcação alcançada, a escolha da próxima ação é sempre determinada de forma aleatória, com distribuição de probabilidade uniforme, desconsiderando qualquer informação da ação de exploração anterior, expressa, por exemplo, nas linhas 12 e 13 do Algoritmo 7.2, as quais foram utilizadas nas simulações do experimento anterior.



(a) Labirinto 7.25 (a) (b) Labirinto 7.25 (b)

(c) Labirinto 7.25 (c) (d) Labirinto 7.25 (d)

Figura 7.28: Exploração média: comparando o desempenho da arquitetura **Map-Dijkstra/ART1-R-MLPs-RR**, mediante as prioridades de exploração "horária", "anti-horária" e "aleatória" com a modalidade de exploração randômica, nos labirintos (a), (b), (c) e (d) da Figura 7.25.

Os gráficos (a), (b), (c) e (d) da Figura 7.28 comparam os dados obtidos a partir das simulações executadas no experimento anterior, mais as simulações executadas com a modalidade de exploração randômica, para os labirintos (a), (b), (c) e (d) da Figura 7.25. A cada passo de exploração, a exploração média, $\overline{\mathcal{A}(t)}$ é computada para as três prioridades de exploração "horária", "anti-horária" e "aleatória" e a modalidade randômica estabelecidas na camada deliberativa da arquitetura. Como pode ser observado nos gráficos, $\overline{\mathcal{A}(t)}$ é mantida em 1 enquanto o número de ações alternativas ainda não executadas for igual ao número de lugares encontrados até o momento. E ainda, se no início da tarefa de exploração, o robô for alcançando mais novas bifurcações do que becos sem saída, o valor da exploração média é mantida próxima ou igual a 1.

Portanto, a partir dos gráficos da Figura 7.28, nota-se a importância de considerar a experiência realizada pelo robô na decisão da ação corrente. Caso contrário, o robô consome muito tempo para completar a exploração, como no caso do labirinto (d) da Figura 7.25. Sendo assim, o desempenho da arquitetura **Map-Dijkstra/ART1-R-MLPs-RR**, ao estabelecer a modalidade de exploração randômica na camada deliberativa, torna-se bastante inferior ao desempenho da mesma arquitetura, ao estabelecer as prioridades de exploração "horária", "anti-horária" ou "aleatória".

7.5 Arquitetura NeuroCog

Esta seção apresenta as simulações e resultados da arquitetura híbrida final **NeuroCog**, proposta neste trabalho de tese e descrita no Capítulo 6. A arquitetura **NeuroCog** aplica-se a labirintos que podem ser modificados durante a operação do robô, pois sua camada deliberativa é formada por um processo de mapeamento cognitivo adaptativo, que trata de maneira dinâmica, as fases de exploração e planejamento de caminho.

As simulações computacionais realizadas para medir o desempenho da arquitetura **NeuroCog** considerou uma tarefa de navegação que possui dois objetivos. O primeiro consiste do robô, partindo de um lugar origem **S**, descobrir a localização de um lugar destino **T**. O outro objetivo consiste do robô tentar obter o conhecimento tanto do menor caminho, quanto de caminhos alternativos entre ambos **S** e **T**, dado que o labirinto pode sofrer modificações.

As simulações foram realizadas no labirinto da Figura 7.29, onde primeiramente o ambiente é mantido estático (Seção 7.5.1) e em seguida dinâmico (Seção 7.5.2), durante a execução do robô. O labirinto apresenta 18 lugares, que correspondem a 4 becos sem saída e 14 bifurcações. Os lugares definidos como **S** e **T** equivalem aos lugares pré-definidos como origem e

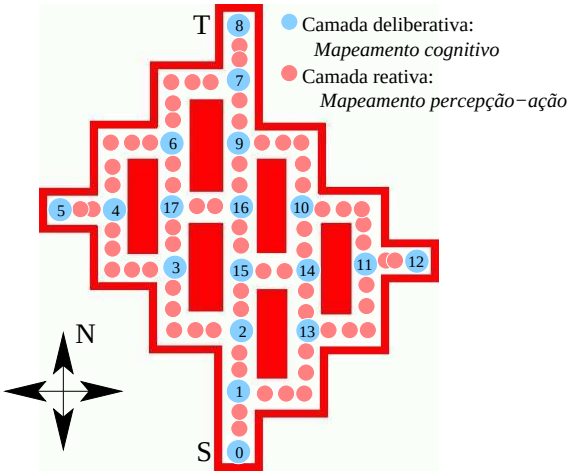


Figura 7.29: O labirinto simulado: 4 becos sem saída e 14 bifurcações em T.

destino respectivamente. Portanto, quando o labirinto se mantém sem modificações, o mapa cognitivo produzido apresentará no máximo 18 nós, correspondentes aos lugares encontrados. E ainda, quando o labirinto for dinâmico, o mapa cognitivo poderá obter mais de 18 nós, devido a bloqueios (becos-sem-saída) e/ou novas aberturas (bifurcações) que podem ser encontrados.

Nas simulações apresentadas nesta seção, a camada reativa da arquitetura **NeuroCog**, implementada pelo arranjo neural **ART1-R-MLPs-RR**, possui a mesma configuração conforme descrita na Seção 7.3.1, com exceção da camada F2 que agora apresenta 45 neurônios para prever um possível acréscimo na quantidade de classes criadas pela rede ART1-R, devido ao aumento da complexidade do labirinto em questão. Já na camada deliberativa, o parâmetro Γ, estabelecido pelo subsistema **Tomada de decisão**, visa especificar o ajuste do dilema entre os comportamentos de explorar o ambiente e planejar caminho, durante a execução da tarefa de navegação. Para efeito de comparação, o parâmetro Γ será configurado com diferentes valores, cujos desempenhos correspondentes para a arquitetura **NeuroCog** serão apresentados nas Seções 7.5.1 e 7.5.2. O fator de desconto fixo utilizado para caracterizar o comportamento de exploração foi estabelecido como $\gamma = 0.995$ (Seção 6.5.2, Eq. 6.2) e finalmente, durante a decisão sobre a ação corrente a^t, ocorrendo empate nos valores dos pesos das ações alternativas em $\mathcal{A}(p^{cur})$ (Eq. 6.5), a prioridade a ser seguida é oeste, norte, leste e sul.

7.5.1 1^o Experimento: *"Exploration* $\times$ *Exploitation"* em Labirinto Estático

Neste experimento, o labirinto utilizado nas simulações foi mantido estático durante a execução da tarefa de navegação e corresponde ao labirinto da Figura 7.29. A tarefa consiste do robô, ao partir do lugar origem S, ter por objetivo alcançar o lugar destino T. Em seguida, assim que o robô visita o destino T, o objetivo alterna-se para o lugar origem S, e assim sucessivamente. O valor do parâmetro Γ estabelecido na camada deliberativa, determinará o grau de "curiosidade" do robô, durante a execução dos percursos entre ambos os lugares S e T. Ou seja, o valor de Γ promove o ajuste entre os comportamentos de explorar o labirinto e seguir o caminho planejado entre S e T ou T e S.

Sendo assim, estas simulações objetivam investigar primeiro, o valor para o parâmetro Γ na Eq. 6.4 (Seção 6.5.2) que melhor ajuste os comportamentos de explorar ambiente e planejar caminho, como medida de desempenho da camada deliberativa, implementada pelo módulo de **Mapeamento cognitivo**. Segundo, qual a implicação deste ajuste no custo estrutural despendido pela camada reativa, implementada pelo controle neural **ART1-R-MLPs-RR**. O melhor ajuste entre os comportamentos de explorar e planejar, no caso em que o labirinto é estático, corresponde ao robô, após uma exploração completa do labirinto, priorizar a execução do menor caminho encontrado entre ambos lugares origem e destino.

Os desempenhos dos controles reativo e deliberativo são descritos da seguinte maneira. A cada percurso, de **S** a **T** ou de **T** a **S**, completado pelo robô, apontou-se a quantia de ciclos de controle executados (passos). Foram computadas as médias de passos sobre 25 execuções, em cada simulação onde o valor do parâmetro Γ é fixo, durante todo o período de execução da tarefa. As simulações diferenciam-se por seus valores de Γ, estabelecidos no intervalo de $(0 \leq \Gamma < 1)$, com variação de 0.1.

Desta forma, na Figura 7.30 são mostradas as quantias médias de ciclos de controle (**Desempenho médio (S - T)**), executados pelo robô, para percorrer caminhos entre **S** e **T**. Este gráfico representa a curva de desempenho da camada deliberativa, implementada pelo módulo **Mapeamento cognitivo**. Ao mesmo tempo, ainda na Figura 7.30, são mostrados os custos (**Custos controle reativo**) despendidos pelo aprendizado online da camada reativa, implementada pelo arranjo neural **ART1-R-MLPs-RR**. O custo estrutural computado compreende a soma total da quantia de neurônios (classes) ativados pela rede ART1 recorrente, a qual corresponde à mesma quantia de redes MLPs alocadas, o que também interfere de maneira indireta na quantia

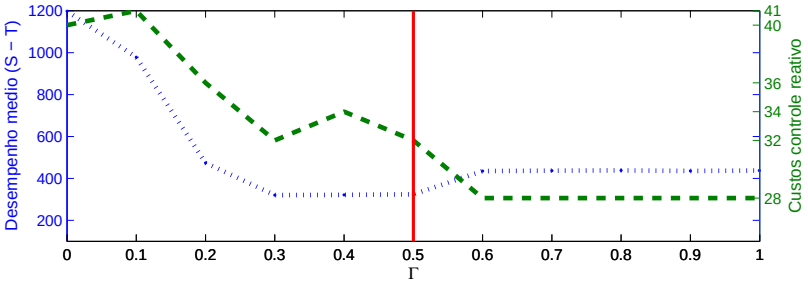


Figura 7.30: Arquitetura **NeuroCog**: dilema da exploração versus aproveitamento. Desempenho da camada deliberativa mediante o problema de equilibrar os comportamentos de exploração e planejamento versus os custos da camada reativa, em labirinto estático.

de punições aplicadas a estas redes durante o aprendizado online.

Como pode ser observado na curva **Desempenho médio (S - T)** do gráfico na Figura 7.30, os valores de Γ no intervalo de $(0.3 \leq \Gamma \leq 0.5)$ levaram o robô a executar caminhos mais eficientes (menor número de passos) entre os lugares **S** e **T**. O ajuste realizado por estes valores levou a um equilíbrio entre os comportamentos de explorar o ambiente e planejar caminho durante a realização da tarefa de navegação. Ao passo que, na curva do **Custos controle reativo**, os custos para o módulo de **Controle neural** se mostram maiores quando o comportamento de explorar predomina sobre o comportamento de planejar $(0 \leq \Gamma < 0.3)$. Em contrapartida, os custos para o controle reativo reduziram-se, quando o comportamento de planejar foi predominante $(0.6 \leq \Gamma \leq 0.9995)$, fazendo com que o robô explorasse pouco o ambiente (apenas até encontrar o lugar objetivo) e em seguida planejasse apenas o primeiro caminho descoberto. O fato de ocorrer um maior custo estrutural para o módulo de **Controle neural** quando o robô explora mais e, ao contrário, um menor custo quando o robô planeja mais, se justifica devido à característica do reconhecimento temporal de padrões feito pela rede ART1 recorrente. Em outras palavras, quando o robô explora mais, a seqüência dos padrões de entrada para a rede ART1 sofre maior variação, interferindo na criação de novas classes. Quando esta variação é menor isto implica em uma menor quantidade de classes criadas.

Quando o peso do comportamento de exploração é nulo $(\Gamma = 1)$, o robô sequer sai do lugar origem **S**, devido à Eq. 6.4. Isto implica em custo próximo a zero para o controle reativo, porém impossibilita o robô de plane-

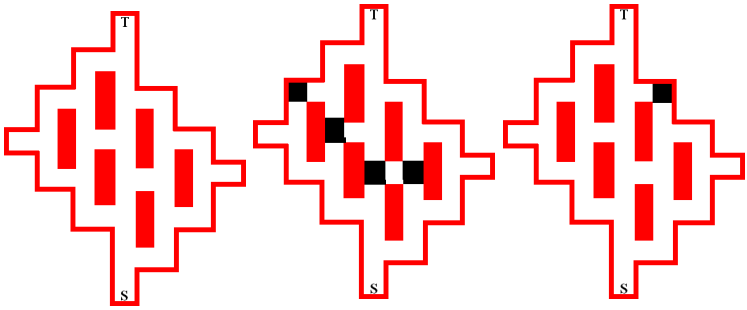
jar qualquer caminho, visto que o comportamento de exploração propicia a coleta de informações junto ao mapa cognitivo. Portanto, nestas simulações, os valores de Γ que melhor ajustam o dilema *exploração versus planejamento versus custos* estão no intervalo de $(0.3 \leq \Gamma \leq 0.5)$. Ou seja, eles permitem uma boa combinação entre ambos os comportamentos de explorar e planejar no nível deliberativo, permitindo ainda um dispêndio razoável no nível reativo. Com estes valores de Γ, a arquitetura **NeuroCog** proporciona ao robô um desempenho satisfatório na execução da tarefa de navegação, mediante um labirinto estático. No experimento mostrado a seguir, são avaliados os desempenhos do robô em um labirinto dinâmico, para os diferentes valores de Γ vistos acima.

7.5.2 2^o Experimento: *"Exploration* $\times$ *Exploitation"* em Labirinto Dinâmico

As simulações apresentadas nesta seção, também foram realizadas no labirinto da Figura 7.29, entretanto, durante a realização da tarefa de navegação, foram introduzidas modificações no ambiente, incluindo bloqueios de caminhos anteriormente livres e novas aberturas anteriormente bloqueadas, em diferentes intervalos de tempo. Nestas simulações, a tarefa de navegação foi aplicada a um contexto de resgate de vítimas. O robô deve, primeiramente, partir da localização inicial **S**, com a missão de encontrar o lugar **T**, onde vítimas encontram-se reunidas. Dado que o robô consegue apenas resgatar uma vítima de cada vez, a cada percurso que corresponde sair de **S**, alcançar **T** e retornar a **S**, considera-se o salvamento de uma vítima. O robô deve buscar melhorar seu desempenho, de forma a maximizar o número de vítimas resgatadas e minimizar o intervalo de tempo em que esta tarefa é executada.

Assim como nas simulações da seção anterior, estas simulações visam medir os desempenhos de ambos os módulos de **Mapeamento cognitivo** e **Controle neural**, mediante um labirinto dinâmico. Portanto, são observados, em primeiro os valores para o parâmetro Γ que melhor ajustam os comportamentos de explorar o ambiente e planejar caminho e, segundo, qual a implicação deste ajuste no custo estrutural despendido pelo controle reativo, considerando-se modificações ocorridas no ambiente, durante a simulação da tarefa de navegação.

A Figura 7.31 em (a), (b) e (c) ilustra em quais períodos de ciclos de controle (passos), as modificações foram introduzidas no labirinto. Primeiramente, como ilustrado na Figura 7.31(a), entre os passos $0 - 0.4(\times 10^4)$, o robô realiza a tarefa de resgate onde vários caminhos alternativos entre os lugares **S** e **T** no labirinto podem ser descobertos. Porém, após este intervalo,



(a) $0-0.4\ (\times 10^4)$ passos (b) $0.4001-1.2\ (\times 10^4)$ passos (c) $1.2001-2\ (\times 10^4)$ passos

Figura 7.31: Modificando o labirinto em tempo de operação: (a) labirinto sem modificação, (b) primeira modificação e (c) segunda modificação.

o labirinto sofre uma primeira modificação onde todos dos caminhos entre **S** e **T**, com exceção de um, são interceptados. Eles são mantidos assim durante o intervalo de $0.4001-1.2\ (\times 10^4)$ passos, como pode ser visto na Figura 7.31(b). Em seguida, durante o intervalo de $1.2001-2\ (\times 10^4)$ passos, como mostrado na Figura 7.31(c), o labirinto sofre uma segunda modificação onde os caminhos anteriormente bloqueados são liberados, ao passo que o único caminho livre entre **S** e **T** até então torna-se interceptado.

O objetivo da primeira modificação no labirinto - Figura 7.31(b) - é observar o quanto uma exploração completa é necessária quando o ambiente sofre modificações bruscas. Uma exploração completa proporciona um comportamento mais adaptativo onde o robô torna-se apto para tentar outros caminhos alternativos até o objetivo, quando caminhos conhecidos são bloqueados. Neste caso, o valor mais apropriado ao parâmetro Γ na Eq. 6.4 é aquele que detecta o momento em que o comportamento de explorar já foi executado o suficiente, para então ser substituído pelo comportamento de planejar. Ainda com o mesmo valor para o parâmetro Γ , o objetivo da segunda modificação - Figura 7.31(c) - é avaliar se o robô será hábil para substituir o comportamento de planejar pelo comportamento de explorar, a fim de encontrar novamente o caminho mais curto então desbloqueado, e como conseqüência salvar mais vítimas.

Os gráficos das Figuras 7.32 e 7.33 mostram as medidas de desempenhos das camadas deliberativa e reativa, implementadas pelos módulos de **Mapeamento cognitivo** e de **Controle neural** respectivamente. O eixo **Performance S-T (T-S)** da Figura 7.32 mede a quantidade de ciclos de controle

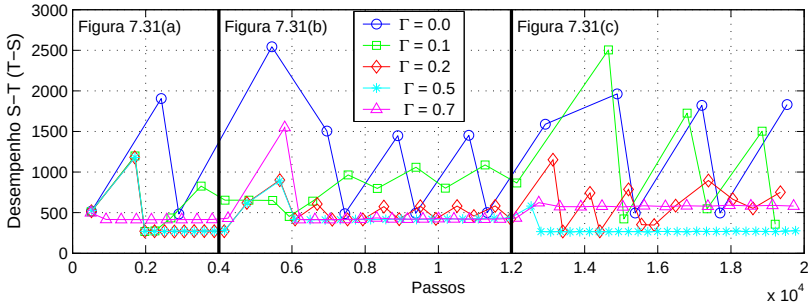


Figura 7.32: Desempenho do módulo **Mapeamento cognitivo NeuroCog**.

(passos) que o robô executa para sair de **S** e chegar a **T** e vice-versa. Da mesma maneira, o eixo **Custos controle reativo** da Figura 7.33 mede o respectivo custo estrutural para a camada reativa, que como visto anteriormente, compreende o somatório da quantia de classes criadas pela rede ART1 recorrente.

Uma análise dos desempenhos do módulo de **Mapeamento cognitivo** (Figura 7.32) é descrita a seguir, para os diferentes valores de Γ, no intervalo de 0 a 0.4 ($\times 10^4$) passos, durante a execução da tarefa de resgate. Com $\Gamma = 0.0$ e $\Gamma = 0.1$ obtém-se baixos desempenhos onde os caminhos executados entre **S** e **T** possuem as maiores quantidades de passos. Isto se deve ao fato destes valores de Γ configurarem o comportamento de explorar como predominante sobre o comportamento de planejar. De maneira diferente, os valores de $\Gamma = 0.2$ e $\Gamma = 0.5$ proporcionam desempenhos melhores (menores quantidades de passos para executar caminhos entre **S** e **T**) devido ao ajuste equilibrado entre ambos os comportamentos. Já com $\Gamma = 0.7$, o desempenho é mediano pois observa-se que o robô explora o ambiente até encontrar o primeiro caminho entre **S** e **T** e em seguida apenas planeja e executa este caminho específico. Ao mesmo tempo, como pode ser visto na Figura 7.33, o custo para o módulo **Controle neural**, no primeiro intervalo, com $\Gamma = 0.0$ foi o maior enquanto com $\Gamma = 0.7$ foi o menor. Pois estes valores configuram diferentes comportamentos globais para o sistema **NeuroCog**, e isto reflete na característica do reconhecimento temporal de padrões feito pela rede ART1 recorrente, visto anteriormente na Seção 7.5.1.

No segundo intervalo de 4001 a 12000 passos, onde o robô se depara com a primeira mudança no ambiente durante a execução da tarefa de resgate, encontrando todos os caminhos entre **S** e **T** bloqueados, com exceção de um,

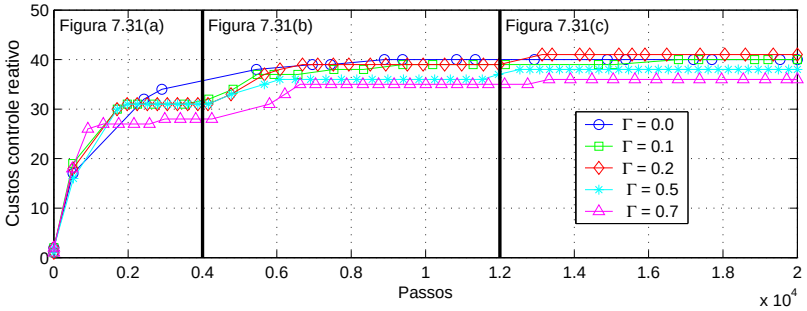


Figura 7.33: Desempenho - **Controle neural NeuroCog**: custos do mapeamento percepção-ação.

Figura 7.31(b), os desempenhos são mostrados nos gráficos das Figuras 7.32 e 7.33. Neste intervalo, por esta mudança ser de grande escala, para todos os valores de Γ na Figura 7.33 dá-se um aumento nos custos do controle reativo. Como a maioria dos caminhos entre **S** e **T** são bloqueados, o padrão de seqüência das percepções de entrada para rede ART1 sofrem modificações, aumentando assim o número de novas classes e conseqüentemente punições. Ainda durante o segundo intervalo, o desempenho do módulo **Mapeamento cognitivo**, Figura 7.32, para valores de Γ iguais a 0.0 e 0.1 são baixos, pois somente o comportamento de exploração é predominante nestes casos. Para $\Gamma = 0.7$, o robô consome algum tempo explorando, pois o único caminho conhecido entre **S** e **T** até o momento fora bloqueado, e quando ele encontra um novo caminho de acesso a **S** e **T**, o comportamento de planejar torna-se predominante. Para os valores de Γ iguais a 0.2 e 0.5 vê-se que o robô também encontra o único caminho disponível, porém, antes disso, o mesmo consome algum tempo tentando o acesso ao lugar objetivo por outros caminhos alternativos conhecidos.

Durante o terceiro intervalo da execução da tarefa de resgate, entre os passos 12001 a 20000 o robô deve tratar com a segunda mudança no labirinto. Esta mudança envolve o bloqueio do único caminho então disponível entre **S** e **T**, Figura 7.31(c), levando o robô a novamente explorar o ambiente a fim de buscar por outros caminhos alternativos. Como visto na Figura 7.33, os desempenhos do controle reativo, para todos os valores de Γ, obtiveram uma pequena elevação dos custos, devido à mudança na seqüência temporal dos padrões de entrada para a rede ART1 no módulo **Controle neural**. Considerando-se o desempenho do módulo **Mapeamento cognitivo**, para os valores de Γ que fazem predominar o comportamento de exploração, 0.0, 0.1

e 0.2, os desempenhos continuaram mais baixos. Com $\Gamma = 0.7$, assim que o robô descobre novamente um caminho entre **S** e **T**, ele passa a executar somente este caminho, devido a predominância do comportamento de planejar.

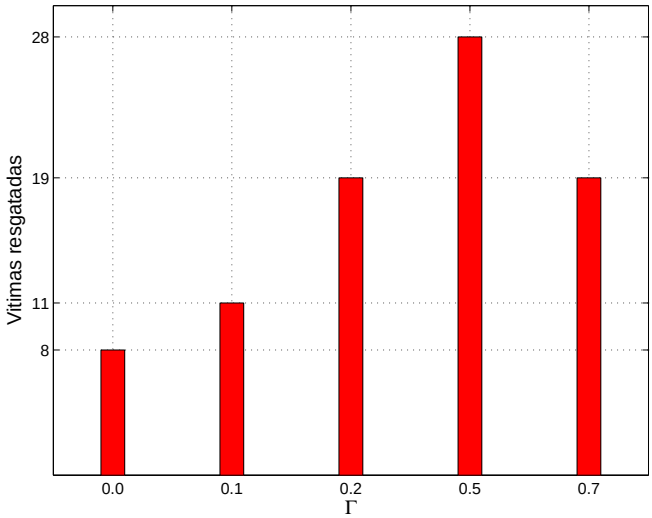


Figura 7.34: Número de vítimas resgatadas com os diferentes valores de Γ.

A Figura 7.34 aponta o número de vítimas resgatadas pelo robô, durante a tarefa de resgate, configurando-se a arquitetura **NeuroCog** a partir dos diferentes valores de Γ. Como pode ser observado, o melhor desempenho é obtido fazendo $\Gamma = 0.5$, pois desta forma, o sistema **NeuroCog** equilibra os comportamentos de explorar ambiente e planejar caminho. Em outras palavras, isto faz com que o robô volte a encontrar o menor caminho no labirinto, após a segunda modificação de seu ambiente. Como resultado, ele obtém um maior número de vítimas resgatadas ao final da tarefa de navegação.

7.6 Comparação Funcional das Abordagens Propostas

A Tabela 7.5 é um resumo das principais características das arquiteturas de controle inteligente, propostas nesta tese, a fim de descrever a evolução funcional até se chegar à arquitetura final **NeuroCog**. A tabela compara as abordagens sob dois aspectos. Primeiro, de acordo com os tipos de labirintos utilizados durante as simulações apresentadas neste capítulo, e segundo, de acordo com os respectivos comportamentos implementados por cada uma de-

		Arquiteturas propostas:				
		Reativas		Híbridas		
		ART1-R-MLPs-RR*	***ART1-R-MLPs-RR-Marcos***	***Map-Tree/****	***Map-Dijkstra/****	***NeuroCog***
Ambientes	LESBB	×	×	×	×	×
	LEE		×	×	×	×
	LESC			×	×	×
	LCM1				×	×
	LCM2					×
Comportamentos	RF	×	×	×	×	×
	RE	×	×	×	×	×
	EPE			×	×	
	D				×	×
	EPD					×

Legenda:

Ambientes	LESBB = labirintos estáticos sem bifurcações e becos sem saída. LEE = labirintos T estáticos e específicos LESC = labirintos T estáticos e sem ciclos. LCM1 = labirintos T com ciclos e modificáveis por inclusão de bloqueios LCM2 = labirintos T com ciclos e modificáveis por inclusão de bloqueios ou novas aberturas.
Comportamentos	RF = reflexivo. RE = reativo. EPE = exploração e planejamento estáticos. D = desvio. EPD = exploração e planejamento dinâmicos.

Tabela 7.5: Diferenças entre as arquiteturas de controle inteligente propostas.

las. Os diferentes tipos de labirintos são descritos na sequência. LESBB equivale ao acrônimo de labirintos estáticos sem bifurcações e becos sem saída, cujo exemplo corresponde aos labirintos utilizados na Seção 7.3.1. Ambientes do tipo LEE correspondem a ambientes estáticos, com bifurcações T e becos sem saída os quais possuem uma configuração específica conforme discutido na Seção 5.5.3. Exemplos de simulações que utilizaram este tipo de ambiente foram apresentados na Seção 7.3.2. Ambientes LESC que correspondem a labirintos T estáticos e sem ciclos são aqueles cujas representações produzidas pelo mapeamento cognitivo equivalem a árvores binárias, exemplos deste tipo de labirinto foram apresentados na Figura 7.15 da Seção 7.4.1. Já a sigla LCM1 está relacionada a labirintos T com ciclos e que podem ser modificados durante a operação do robô, através da inclusão de bloqueios em caminhos conhecidos. Labirintos deste tipo foram utilizados na Seção 7.4.2. Finalmente, a sigla LCM2 se refere a labirintos T com ciclos e modificáveis durante a operação do robô através da inclusão de bloqueios ou novas aberturas em caminhos conhecidos, como foi descrito no experimento da Seção 7.5.2. Como pode ser visto na Tabela 7.5, a arquitetura final **NeuroCog** é hábil para atuar em qualquer um dos tipos de labirinto listados.

Com relação aos tipos de comportamentos implementados pelas abordagens propostas tem-se o RF, reflexivo, que corresponde a não colidir com obstáculos, o reativo, RE, que equivale a andar para frente sem colidir e sem executar movimentos em vai e vem. A sigla EPE corresponde aos comportamentos de explorar ambiente e planejar caminhos executados em sequência, ou seja, o planejamento só é executado quando todo o labirinto foi explorado. A sigla D corresponde ao comportamento de desvio, executado após a detecção de bloqueios em caminhos conhecidos no labirinto. Finalmente a sigla EPD corresponde ao comportamento descrito através do equilíbrio entre ambos explorar ambiente e planejar caminho, considerando-se que estes comportamentos sejam competitivos. Ou seja, o comportamento global EPD é uma forma de tratar o dilema conhecido como *exploration* versus *exploitation* durante a execução da tarefa de navegação do robô.

7.7 Conclusão

Os resultados das simulações computacionais apresentados neste capítulo mostraram como foram validadas as consecutivas arquiteturas de controle desenvolvidas, até se chegar à arquitetura final **NeuroCog**, proposta neste trabalho de tese. E ainda, questões de desempenho e aprendizado efetivo em ambos os níveis reativo e deliberativo foram avaliados, à medida que estas arquiteturas eram aplicadas à tarefas de navegação em labirintos, cuja

complexidade aumentou gradativamente.

Desta maneira, os resultados das simulações referentes às arquiteturas neurais reativas **ART1-R-MLPs-RR** e **ART1-R-MLPs-RR-Marcos**, demonstraram um aprendizado efetivo de mapeamentos percepção-ação e desempenhos aceitáveis, quando aplicadas, a tarefas de navegação reativa, tais como seguir corredores evitando obstáculos e executar trajetórias livres de becos sem saída em labirintos mais simples.

Já as simulações realizadas a partir das arquiteturas híbridas prévias **Map-Tree** e **Map-Dijkstra**, apresentaram desempenhos satisfatórios para tarefas de navegação planejada, primeiramente em labirintos estáticos e sem caminhos cíclicos e posteriormente em labirintos com ciclos e que apresentam mudanças, após a fase de exploração, produzindo assim o comportamento de desvio.

E finalmente os resultados experimentais, obtidos a partir das simulações com a arquitetura híbrida final **NeuroCog**, demonstraram características de aprendizado adaptativo e desempenho eficiente, durante a resolução de uma tarefa de navegação complexa, do ponto de visto computacional. A maneira dinâmica de tratar os comportamentos de exploração e planejamento de caminho no nível deliberativo, junto à análise de como esta dinâmica repercute no nível reativo, fornecem uma nova medida de desempenho, que descreve o dilema *"exploration"* versus *"exploitation"*, no contexto de tarefas de navegação planejada.

CAPÍTULO 8

Conclusão

8.1 Principais Considerações

Este trabalho de tese propôs uma arquitetura baseada em comportamentos, que foi resultado da evolução de uma série de arquiteturas de controle prévias. A abordagem proposta denominada **NeuroCog** apresentou soluções aos seguintes problemas relacionados à navegação planejada ou baseada em mapas.

Primeiro, foi proposta uma maneira de integrar níveis de controle deliberativo e reativo que resultou em uma arquitetura de controle híbrida. A arquitetura proposta provê um sistema de navegação flexível e adaptativo, que combina de maneira simplificada e eficiente, um método de aprendizado de mapa topológico, com um processo de aprendizado reativo, implementado através de um arranjo de RNAs. Como resultado, este sistema permite a um robô, contando apenas com seus sensores de proximidade, ângulo de orientação e posição, resolver tarefas de navegação complexas, do ponto de vista computacional.

Segundo, a abordagem final foi validada em labirintos estáticos e dinâmicos. Nesta tese, considerou-se como característica dinâmica o fato do ambiente sofrer modificações quanto a bloqueios (novos becos sem saída) ou aberturas (novas bifurcações), após o aprendizado de ambos a representação do labirinto (pelo módulo deliberativo) e do mapeamento de percepção-ação (pelo módulo reativo).

Em terceiro, para tratar a dinâmica do ambiente, a arquitetura de controle proposta provê um método de ajuste que trata o dilema conhecido como *exploration* versus *exploitation*, o qual foi descrito e estendido neste trabalho de tese sob dois aspectos:

- O primeiro relacionado ao módulo deliberativo da arquitetura, que implementa o método de ajuste entre ambos os comportamentos de explorar ambiente e planejar caminho. Estes comportamentos apresentam uma competição inerente, a qual interfere no desempenho da estratégia de navegação.

- O segundo aspecto está relacionado ao custo do controle neural reativo, que é influenciado pelo método de ajuste da camada deliberativa.

Sendo assim, o desenvolvimento da abordagem **NeuroCog**, como uma arquitetura de controle inteligente para navegação de robôs móveis em labirintos, tanto estáticos quanto dinâmicos, se desdobrou em quatro etapas principais e consecutivas:

- A primeira onde foram produzidas sucessivas arquiteturas reativas, a partir de diferentes acoplamentos de RNAs e algoritmos de aprendizagem. Mais especificamente, a utilização de dois tipos de RNAs, com aprendizado em tempo de operação, para a implementação da camada reativa da arquitetura proposta, propiciou a criação de um processo de aprendizado de mapeamento percepção-ação flexível e adaptativo. Nesse sentido, as arquiteturas reativas resultantes tiraram vantagens das propriedades plástica e estável do aprendizado auto-organizável de uma rede ART, juntamente com o aprendizado online (por reforço) de redes diretas MLPs.

- Na segunda etapa, se identificou uma limitação do arranjo neural, quanto ao aprendizado de comportamentos mais complexos, tais como planejamento de caminho em labirintos com bifurcações. Sendo assim, se buscou, através da teoria de mapas cognitivos, inspiração para a criação de uma representação de conhecimento explícita do ambiente. Desta forma, a abordagem proposta passou a utilizar o conceito de marcos, estabelecendo locais do tipo bifurcações e becos sem saída como marcos identificáveis, a serem incorporados ao mapa topológico de labirintos. Portanto, foi estabelecido que nestes locais atuaria o controle deliberativo, enquanto que em corredores atuaria o controle reativo, sendo esta a forma encontrada para integrar os níveis reativo e deliberativo nas arquiteturas híbridas subsequentes.

- Na terceira etapa, foram desenvolvidas camadas deliberativas prévias, cujo suporte de controle reativo foi fornecido pela camada reativa resultante da primeira etapa. Na sequência, as arquiteturas híbridas prévias resultaram no desenvolvimento do módulo deliberativo **Neuro-**

Cog. Mais especificamente, todas as camadas deliberativas foram implementadas através do aprendizado e utilização de memórias de longo termo (mapa topológico ou cognitivo) e de curto termo (experiências de exploração e planejamento).Por serem aplicadas a ambientes do tipo labirinto, a atuação do controle reativo nas arquiteturas híbridas predomina sobre a atuação do controle deliberativo.

- Finalmente na quarta etapa, no desenvolvimento da arquitetura híbrida final **NeuroCog**, foi desenvolvido um processo para equilibrar o dilema conhecido como "*exploration* versus *exploitation*" entre os comportamentos deliberativos. E como medida de desempenho, se propôs a análise da repercussão produzida por este processo, nos custos do aprendizado do arranjo neural, que implementa o nível reativo da arquitetura.

Durante o desenvolvimento desta tese, foram revisadas as arquiteturas mais representativas segundo os paradigmas de controle hierárquico, reativo e híbrido. A arquitetura **NeuroCog** serve como um exemplo de arquitetura de subsunção e híbrida, sendo que a camada deliberativa subsume a camada reativa nos locais que se referem a marcos no ambiente.

Foram estudados também uma hierarquia com diversos modelos de navegação autônoma, inspirados na teoria de mapas cognitivos. Estes modelos vão desde sistemas puramente reativos até sistemas baseado em mapas. Este estudo também conduziu a criação das estratégias de navegação desenvolvidas ao longo deste trabalho de tese. Onde as arquiteturas reativas propostas podem ser classificadas como uma estratégia de navegação do tipo *disparo de reconhecimento de lugar*. Enquanto as híbridas podem ser classificadas como estratégias de navegação do tipo *topológica*.

Igualmente foram revisados alguns tipos de RNAs e modalidades de aprendizado, que podem ser aplicados na resolução dos diferentes sub-problemas envolvidos na problemática da navegação de robôs móveis. Esta revisão foi útil no desenvolvimento das arquiteturas reativas resultantes, que mostraram eficiência no aprendizado de mapeamentos percepção-ação, incorporando capacidades de aprendizado autônomo e adaptativo, na navegação de um robô móvel em labirintos simples, sem bifurcações.

Com o aumento da complexidade dos labirintos, através do acréscimo de bifurcações, por exemplo, pode-se aumentar o grau de dificuldade da tarefa de navegação. Desta forma, surgiu a necessidade de implementação de comportamentos mais complexos, tais como exploração do ambiente e planejamento de caminho. Sendo assim, a estratégia seguinte foi construir sucessivas camadas deliberativas, buscando inspiração na teoria dos mapas cognitivos,

até se chegar à arquitetura proposta final **NeuroCog**, que provê o controle inteligente de um robô móvel para navegação tanto em labirintos estáticos como dinâmicos.

8.2 Principais Contribuições

As principais contribuições obtidas a partir deste trabalho de tese são:

- O processo de investigação sobre a inserção de RNAs, como blocos de construção básicos no desenvolvimento de uma arquitetura de controle para robô móvel.
- O desenvolvimento de um método de acoplamento entre diferentes arquiteturas de RNAs (Redes ART e MLPs), para aprendizado adaptativo de comportamentos reativos.
- Criação de um processo de aprendizado baseado em AR, como treinamento em tempo de operação de redes MLPs, para aprendizado de comportamentos reflexivos.
- Desenvolvimento de um processo de aprendizado de mapeamento cognitivo ou topológico, que provê um método eficiente de permutação entre comportamentos competitivos, produzindo desempenho satisfatório, mediante execução de tarefas de navegação em labirintos dinâmicos.
- Novo método de integração funcional e eficiente entre ambas as camadas reativa e deliberativa das arquiteturas híbridas propostas, realizado através do conceito de marcos, relacionados a lugares específicos no ambiente.
- Desenvolvimento de uma nova descrição para o dilema conhecido como "*exploration* versus *exploitation*", como medida de desempenho tanto no contexto de comportamentos deliberativos, quanto no contexto da integração entre os níveis reativos e deliberativos das arquiteturas híbridas propostas.
- Desenvolvimento de uma arquitetura de controle híbrida e baseada em comportamentos, **NeuroCog**, que provê um sistema de navegação computacional facilmente aplicável a um robô real, com limitados recursos sensórios e computacionais, e ao mesmo tempo direcionado à resolução de tarefas de navegação complexas.

Os resultados obtidos nesta tese geraram publicações em congresso internacional (MARIN; ROISENBERG; PIERI, 2006b) e três artigos em congressos nacionais (MARIN; ROISENBERG; PIERI, 2006a), (MARIN; ROISENBERG; PIERI, 2007b) e (MARIN; ROISENBERG; PIERI, 2007a). Um artigo foi submetido a uma revista internacional da área de robótica e sistemas autônomos e está sob revisão.

8.3 Trabalhos Futuros

Na sequência são apresentados alguns assuntos importantes a serem considerados como trabalhos futuros, visando tanto uma maior validação das principais contribuições dadas, bem como o acréscimo de maior portabilidade e robustez à abordagem **NeuroCog**.

8.3.1 Questões Preliminares

Apesar do sistema **NeuroCog** se aplicar a labirintos de qualquer tamanho e que possam ser modificados durante a operação do robô, a ortogonalidade das paredes, bem como o formato em "T" das bifurcações são condições básicas para o seu funcionamento. Entretanto, é importante salientar que, embora labirintos com bifurcações em cruz "+" não tenham sido utilizados para validar a arquitetura **NeuroCog**, apenas um pequeno acréscimo no processo de detecção para este tipo de bifurcação seria necessário, a fim de que o sistema proposto possa atuar diretamente.

Além disto, para tornar a abordagem proposta mais adaptativa, visando estender sua aplicação a labirintos com formatos mais complexos, uma série de implementações seriam necessárias. Um dos principais desafios é prever a leitura dos dados dos sensores de proximidade em pontos de bifurcações. Por exemplo, como no caso do labirinto hexagonal utilizado em (MALLOT et al., 1995), cujas bifurcações são em formato "Y", além de labirintos com outros tipos de junções, tais como "$\curlyvee$" ou "$\curlywedge$". Afora isso, uma outra dificuldade, ocasionada por estes tipos de bifurcações, está no fato de que os ângulos entre possíveis movimentos, além de não serem perpendiculares entre si, não poderem ser considerados como constantes, com relação à frente do robô.

A ortogonalidade dos labirintos também permite que o robô, utilizando a arquitetura **NeuroCog**, execute apenas trajetórias em linha reta. Além disso, o robô pára, toda vez que necessita mudar de direção, girando ao redor de seu próprio eixo. Redes MLPs com treinamento off-line poderiam ser aplicadas, a fim de que o robô execute trajetórias com curvaturas nestes

locais.

Outro aspecto importante é estender a aplicação e utilização de RNAs como blocos de construção básicos também para o desenvolvimento da camada deliberativa da arquitetura **NeuroCog**. Onde por exemplo, uma RNA com aprendizado auto-supervisionado pudesse gerar um comportamento exploratório e ao mesmo tempo representar de maneira incremental e explícita o mapa topológico do ambiente.

Durante todo o processo de desenvolvimento da abordagem **NeuroCog**, não foram consideradas questões de plausibilidade biológica, em ambos os níveis comportamental e neurofisiológico. Entretanto seria importante dar maior atenção a estes aspectos, dado o avanço das pesquisas em robótica bioinspirada.

Todos os experimentos realizados nesta tese foram testados em um ambiente de simulação. Para uma validação mais concreta do sistema **NeuroCog**, se faz necessário testar a abordagem proposta em um robô real, que está sendo desenvolvido no LCA/DAS.

Além disto, questões de portabilidade devem ser desenvolvidas a fim de adaptar o sistema **NeuroCog** a ambientes abertos, incorporando os aspectos da abordagem topológica implementados no contexto da navegação em labirinto.

8.3.2 Simplificação da Camada Reativa

Uma metáfora de uma importante característica presente em seres vivos referente a reflexo inato poderia ser implementado na abordagem **NeuroCog**, com o intuito de reduzir a complexidade da camada reativa, que é baseada no arranjo neural composto pela rede ART1 recorrente "chaveadora" de redes MLPs, com aprendizado em tempo de operação, visto na Seção 5.4.2. A implementação do reflexo inato para a camada reativa simplificaria tanto a estrutura quanto a forma do aprendizado do sistema de controle reativo, que poderia ser reduzido, por exemplo, a uma única rede MLP com aprendizado off-line por retro-propagação, como visto ao final da Seção 5.3.3, na Figura 5.9(a).

Este "reflexo neural inato" (andar para frente evitando obstáculo), presente na camada reativa da arquitetura **NeuroCog**, se aplicaria tanto a labirintos estáticos como dinâmicos, pois somente a camada deliberativa é responsável por tratar as mudanças que podem ocorrer no ambiente, durante o tempo de operação do robô. A aplicação desta implementação de reflexo neural inato torna-se simples e direta, devido à redução de complexidade do problema de navegação, proporcionada por ambientes do tipo labirinto T, o que

leva à simplicidade na construção dos conjuntos de treinamento e validação, necessários ao aprendizado por retro-propagação, os quais são compostos por simples mapeamentos percepção-ação do robô.

8.3.3 Limitação do Aprendizado Aleatório

O aprendizado aleatório, invocado pelo módulo **Políticas de Aprendizado** mediante o reforço *punição* a ser aplicado às redes MLPs (Algoritmo 5.2), em geral não apresentou problemas de falta de convergência, durante os experimentos realizados ao longo deste trabalho de tese. De fato, esta propriedade é fundamental para viabilizar o processo de aprendizado em tempo de operação destas redes, ao comporem os arranjos neurais reativos propostos no Capítulo 5. Além disto, esta convergência pode ser garantida para atuação da abordagem **NeuroCog** em labirintos T de qualquer tamanho.

A convergência do aprendizado aleatório é garantida neste contexto, devido a duas características principais relacionadas às redes MLPs utilizadas. Primeira, à pequena quantidade de neurônios presentes em especial na camada de saída e segunda, à discretização dos valores de resposta destes neurônios para unidades binárias. É importante notar que estas características das redes MLPs, as quais viabilizam a aplicação do aprendizado aleatório, são proporcionadas por razões da simplicidade, quanto à representação dos padrões de estado do ambiente e da ação a ser executada pelo robô.

Entretanto, em um contexto de maior complexidade, originado a partir do ambiente e do aumento do grau de liberdade do robô por exemplo, o que poderia ocasionar um aumento de dados a serem tratados diretamente como valores contínuos pelo sistema de controle, a garantia da convergência do aprendizado aleatório aplicado as redes MLPs não poderia ser mantida.

Sendo assim, para se garantir um aprendizado em tempo de operação, como trabalhos futuros, a camada reativa da abordagem **NeuroCog** poderia ser implementada com técnicas de aprendizado por reforço. Um exemplo em especial são as abordagens propostas nos trabalhos de Lin (1991) e Lin (1992), para aprendizado autônomo no domínio de robôs móveis, onde as abordagens combinam tanto métodos de diferença temporal (SUTTON, 1988), quanto aprendizagem Q (WATKINS, 1989), juntamente com redes neurais treinadas pelo algoritmo de retro-propagação. Isto permitiria também uma nova maneira de implementar a camada deliberativa, o que proporcionaria uma maior conexão com as métricas da aprendizagem por reforço como medida de desempenho.

8.3.4 Tratando a Problemática de Ruídos

Durante a execução do software *WSU Java Khepera Simulator* utilizado para desenvolver, em especial, a arquitetura **NeuroCog** proposta nesta tese, pôde-se verificar que em condições nominais, os valores dos dados dos sensores de proximidade IR apresentavam constantes variações, mesmo quando o robô permanecia em uma mesma posição relativa. Esta característica é uma tentativa do software supracitado de simular a presença de ruídos nos dados dos sensores, a fim de imitar as condições reais de um robô Khepera físico em seu ambiente. Entretanto, este método de introdução de ruídos nos sensores do robô Kherera simulado em nenhum momento repercutiu negativamente na execução e desempenho dos controladores implementados ao longo do trabalho, os quais puderam considerar, com cem por cento de certeza, as configurações de parede do labirinto fornecidas pelo módulo **Percepção**.

Todavia, fora de um ambiente de simulação sabe-se que um robô físico em um ambiente real torna-se, consequentemente, mais vulnerável às condições de imprecisão originada tanto dos sensores quanto dos atuadores do robô. É fato que a execução e desempenho da arquitetura **NeuroCog** seriam altamente comprometidos caso uma porcentagem maior de ruídos nos sensores de proximidade pudessem ocasionar efeitos como falsos positivos e falsos negativos com relação à presença de obstáculos nas quatro direções estabelecidas (oeste, norte, leste e sul).

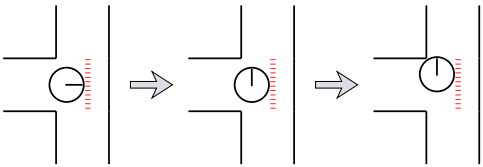


Figura 8.1: Ruído nos dados sensórios: exemplo de colisão devido a uma parede “imaginária”.

Por exemplo, um ou mais falsos positivos corresponderiam a paredes “imaginárias” que produziriam, desde a execução de comportamentos inadequadados, comprometendo a atuação tanto da camada reativa quanto da camada deliberativa da arquitetura **NeuroCog**, bem como a quebra do robô, devido à colisão. No primeiro caso, um comportamento incorreto poderia ser causado por um beco sem saída imaginário ao longo de um corredor, o que faria com que o robô retrocedesse de maneira indevida em sua trajetória. Já o segundo caso em que o robô quebraria, devido à colisão com parede, pode

ser descrito na Figura 8.1. Considerando-se uma parede imaginária a leste, que faria com que o robô parasse em uma posição onde as paredes ao norte e ao sul já não pudessem mais ser percebidas pelos sensores das laterais. A decisão da ação imediata, fornecida neste caso pela camada reativa, seria fazer com que o robô tomasse a direção ou a norte ou a sul, o que causaria a colisão do robô em ambos os casos, devido ao seu posicionamento inadequado no corredor.

Já a ocorrência de um ou mais falsos negativos com relação à configuração de paredes correspondem a falsas direções livres as quais trazem consequências mais graves principalmente se a direção em que o robô se encontra em movimento estiver envolvida, o que levaria diretamente à colisão e quebra.

Além da imprecisão dos sensores, a abordagem **NeuroCog** também se tornaria vulnerável à imprecisão originada dos atuadores que é resultante do deslizamento das rodas do robô. Neste caso, tanto a execução quanto o desempenho da camada deliberativa seriam potencialmente afetados devido à utilização, pelo processo de construção e manutenção do mapa cognitivo, da contagem de passos executados pelo robô, entre lugares específicos no labirinto. Em outras palavras, o acúmulo de erros métricos no mapa comprometeria o cálculo do caminho mais curto até o objetivo corrente.

Mudanças quanto à codificação dos dados dos sensores de proximidade, bem como a utilização de técnicas alternativas referente a lógica fuzzy (nebulosa) (ZADEH, 1965), redes Fuzzy ART (CARPENTER; GROSSBERG; ROSEN, 1991a) e rede ART2 (CARPENTER; GROSSBERG; ROSEN, 1991b) por exemplo, poderiam proporcionar uma maior robustez à abordagem **NeuroCog** mediante à ocorrência de ruídos, responsáveis pela imprecisão e ambiguidade dos padrões de entrada, fornecidos às camadas reativa e deliberativa.

Como dito anteriormente, uma primeira modificação estaria relacionada à codificação dos dados dos sensores de proximidade, que definem a variável de estado do ambiente ξ^t. Ou seja, ao invés desta variável apresentar quatro componentes com valores discretos (binários), as quatro componentes poderiam apresentar valores contínuos ou analógicos entre 0 e 1, os quais representariam a probabilidade ou o grau de certeza (pertinência) para a existência de uma parede em cada direção determinada (oeste, norte, leste e sul).

Deste modo, dentro de um Controle reativo neural-fuzzy mostrado na Figura 8.2, um sistema especialista nebuloso (fuzzy), poderia ser implementado para prover tanto a variável de estado do ambiente nebulosa ξ^t_{fuzzy} para uma rede Fuzzy ART ou ART2, quanto a variável de estado do ambiente crisp (não nebulosa), ξ^t_{crisp}, para o módulo de **Políticas de aprendizado** e também para a camada deliberativa. Ambas as redes Fuzzy ART e ART2 tratam

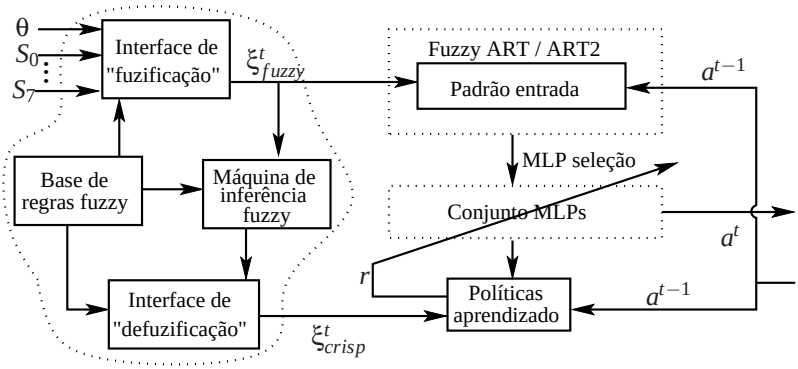


Figura 8.2: Controle neural-fuzzy.

ξ^{t-1}_{fuzzy}	ξ^t_{fuzzy}			
	MB	B	A	MA
MB	ZE	ZE	ZE	UM
B	ZE	ZE	UM	UM
A	ZE	UM	UM	UM
MA	UM	UM	UM	UM

Tabela 8.1: Exemplo de tabela de regras de decisão para o controle nebuloso de uma componente da variável ξ^t_{crisp}

tanto padrões de entradas com unidades discretas, quanto analógicas. Uma maior vantagem é apresentada pela rede Fuzzy ART, que incorpora computações da teoria de conjuntos nebulosos combinando um rápido aprendizado de sequências arbitrárias de padrões analógicos ou binários, com uma regra de esquecimento que protege a memória do sistema contra ruídos nos padrões de entrada (CARPENTER; GROSSBERG; ROSEN, 1991a).

Dentro do sistema especialista nebuloso na Figura 8.2, o módulo Interface de "fuzificação" executa um mapeamento de escala que converte dados de entrada não nebulosos, como por exemplo os valores dos dados dos sensores IR, em valores linguísticos que poderiam ser descritos através da função de pertinência visualizada na Figura 8.3.

Deste modo, pode-se estabelecer a variável de estado do ambiente nebulosa ξ^t_{fuzzy}, através de uma fusão de dados dos sensores IR, mais o ângulo de orientação θ, para determinar a presença de paredes nas direções oeste norte, leste e sul.

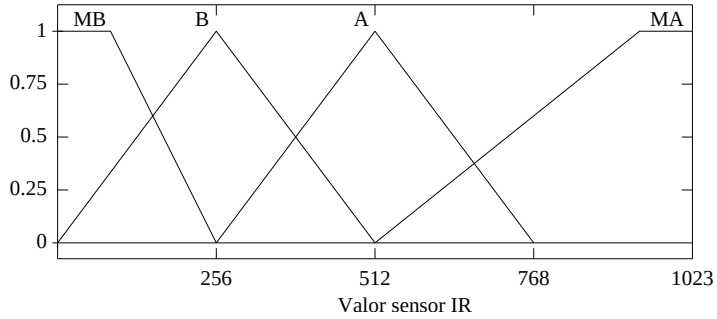


Figura 8.3: Função de pertinência para valores de entrada de sensores de proximidade. Notações - MB: Nenhum obstáculo no sinal. B: Obstáculo está longe. A: Obstáculo está próximo. MA: Quase tocando no obstáculo.

A Base de regras fuzzy consiste de um conjunto de regras de controle linguístico, escritas da forma: "SE um conjunto de condições são satisfeitas, ENTÃO um conjunto de consequências são deduzidos. Para o caso do controle reativo da Figura 8.2, um exemplo de tabela de regras de decisão, para o controle nebuloso de uma componente da variável ξ^t_{crisp}, poderia ser descrito através de Tabela 8.1, utilizando-se, por exemplo, das respectivas componentes das variáveis ξ^t_{fuzzy} e ξ^{t-1}_{fuzzy}. Já a Máquina de inferência fuzzy consiste de uma lógica de tomada de decisão, que emprega as regras da Base de regras fuzzy, para deduzir o controle nebuloso de ações, em resposta às entradas "fuzificadas". Já a interface de "defuzificação" executa um mapeamento que produz uma ação de controle não nebulosa (crisp) a partir de uma ação de controle nebuloso deduzido. Um regra comumente utilizada é o método centróide (ZADEH, 1965; ZADEH, 1973; LEE, 1990).

Figura 8.4: Controle reativo fuzzy.

Uma outra alternativa ao Controle neural-fuzzy da Figura 8.2, seria substituir todo o arranjo neural **ART1-R-MLPs-RR** que compõe a camada reativa da abordagem **NeuroCog**, por um controlador reativo fuzzy, implementado através de um sistema especialista fuzzy, com base em heurísticas

de controle para construção das bases de regras (Figura 8.4).

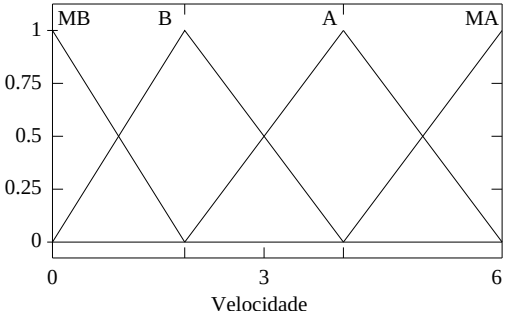


Figura 8.5: Função de pertinência para a variável de saída a^t_{fuzzy}. Notações - MB: Muito baixa. B: Baixa. A: Alta. MA: Muito alta.

Deste modo, as quatro componentes da ação da ação do robô também poderiam apresentar valores contínuos entre 0 e 1, indicando a velocidade com que o robô deve se mover em determinada direção: $a^t_{fuzzy} = \{a^o_{fuzzy}, a^n_{fuzzy}, a^l_{fuzzy}, a^s_{fuzzy}\}$. Ou seja, a cada passo de controle, uma das quatro componentes da ação corrente a ser executada, poderia ser inferida através da função de pertinência mostrada na Figura 8.5. Logo, como trabalhos futuros, também sugere-se a construção das bases de regras para este controlador fuzzy reativo, semelhante à maneira como foi apresentada no trabalho de (ZHANG; WILLE; KNOLL, 1996), por exemplo.

APÊNDICE A

Bases de Regras para as Políticas de Aprendizado Online das redes MLPs

Este apêndice descreve as duas bases de regras que compõem os módulos de **Políticas de Aprendizado** das respectivas arquiteturas de controle neural reativas: **ART1-R-MLPs-RR** e **ART1-R-MLPs-RR-Marcos**. Estas bases de regras são apresentadas em linguagem Java e cada caso de teste onde a rede MLP corrente pode receber punição é considerado uma regra ou política de aprendizado. A base de regras do controle neural **ART1-R-MLPs-RR**, descrita na Seção A.1, totaliza 33 regras enquanto a base de regras do controle neural **ART1-R-MLPs-RR-Marcos**, descrita na Seção A.2, totaliza 41 regras. Ao total ambas as bases ainda acrescentam mais 1 regra correspondente ao controle da resposta da rede MLP corrente quanto à codificação do movimento do robô.

A.1 Base de Regras do Controle Neural ART1-R-MLPs-RR

```
/*   OESTE             NORTE           LESTE           SUL    caso = 0000*/
if(estado[0]==0 && estado[1]==0 && estado[2]==0 && estado[3]==0){
    if(movimento[mov_ant] != 1) //movimento atual diferente movimento anterior
         r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE           SUL    caso = 0001*/
else if(estado[0]==0 && estado[1]==0 && estado[2]==0 && estado[3]==1){
    if (movimento[3]==1) // ir para o Sul, punicao
        r = punicao;
    // direcao do mov. anterior esta livre &&
    // nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else if (estado[mov_ant] == 1 && movimento[1]==1) // quer ir para Norte
        r = punicao;
    else r = recompensa;
```

```
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0010*/
else if(estado[0]==0 && estado[1]==0 && estado[2]==1 && estado[3]==0){
    if (movimento[2]==1)
        r = punicao;
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else if (estado[mov_ant] == 1 && movimento[0]==1) // quer ir para Oeste
        r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0011*/
else if(estado[0]==0 && estado[1]==0 && estado[2]==1 && estado[3]==1){
    if (movimento[2]==1 || movimento[3]==1)
        r = punicao;
    else if (estado[mov_ant] == 1){ //direcao do mov. anterior NAO esta livre
        //se mov. anterior == Leste e movimento atual != Norte OU
        //se mov. anterior == Sul e movimento atual != Oeste
        if ( (mov_ant==2 && movimento[1]!= 1) || (mov_ant==3 && movimento[0]!= 1))
            r = punicao;
        else r = recompensa;
    }
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0100*/
else if(estado[0]==0 && estado[1]==1 && estado[2]==0 && estado[3]==0){
    if (movimento[1]==1)
        r = punicao;
    // direcao do mov. anterior esta livre
    // nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else if (estado[mov_ant] == 1 && movimento[3]==1) // quer ir para SUL
        r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0101*/
else if(estado[0]==0 && estado[1]==1 && estado[2]==0 && estado[3]==1){
    if (movimento[1]==1 || movimento[3]==1 )
        r = punicao;
    //direcao do mov. anterior esta livre
    // nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0110*/
else if(estado[0]==0 && estado[1]==1 && estado[2]==1 && estado[3]==0){
    if (movimento[1]==1 || movimento[2]==1)
        r = punicao;
    else if (estado[mov_ant] == 1){ //direcao do mov. anterior NAO esta livre
        //se mov. anterior == Leste e movimento atual != SUL OU
        //se mov. anterior == NORTE e movimento atual != Oeste
        if ( (mov_ant==2 && movimento[3]!= 1) || (mov_ant==1 && movimento[0]!= 1))
```

```
            r = punicao;
        else r = recompensa;
    }
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0111*/
else if(estado[0]==0 && estado[1]==1 && estado[2]==1 && estado[3]==1){
    if (movimento[1]==1 || movimento[2]==1 || movimento[3]==1 )
        r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 1000*/
else if(estado[0]==1 && estado[1]==0 && estado[2]==0 && estado[3]==0){
    if (movimento[0]==1)
        r = punicao;
    // direcao do mov. anterior esta livre
    // nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else if (estado[mov_ant] == 1 && movimento[2]==1) // quer ir para LESTE
        r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 1001*/
else if(estado[0]==1 && estado[1]==0 && estado[2]==0 && estado[3]==1){
    if (movimento[0]==1 || movimento[3]==1 )
        r = punicao;
    else if (estado[mov_ant] == 1){ //direcao do mov. anterior NAO esta livre
        //se mov. anterior == OESTE e movimento atual != NORTE OU
        //se mov. anterior == SUL e movimento atual != LESTE
        if ( (mov_ant==0 && movimento[1]!= 1) || (mov_ant==3 && movimento[2]!= 1))
            r = punicao;
        }
        else r = recompensa;
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE             SUL   caso = 1010*/
else if(estado[0]==1 && estado[1]==0 && estado[2]==1 && estado[3]==0){
    if (movimento[0]==1 || movimento[2]==1 )
        r = punicao;
    //direcao do mov. anterior esta livre
    // nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 1011*/
else if(estado[0]==1 && estado[1]==0 && estado[2]==1 && estado[3]==1){
    if (movimento[0]==1 || movimento[2]==1 || movimento[3]==1 )
        r = punicao;
    else r = recompensa;
}
```

```
/*      OESTE             NORTE           LESTE            SUL    caso = 1100*/
else if(estado[0]==1 && estado[1]==1 && estado[2]==0 && estado[3]==0){
    if (movimento[0]==1 || movimento[1]==1)
        r = punicao;
    else if (estado[mov_ant] == 1){ //direcao do mov. anterior NAO esta livre
        //se mov. anterior == OESTE e movimento atual != SUL OU
        //se mov. anterior == NORTE e movimento atual != LESTE
        if ( (mov_ant==0 && movimento[3]!= 1) || (mov_ant==1 && movimento[2]!= 1))
            r = punicao;
        else r = recompensa;
    }
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 1101*/
else if(estado[0]==1 && estado[1]==1 && estado[2]==0 && estado[3]==1){
    if (movimento[0]==1 || movimento[1]==1 || movimento[3]==1 )
        r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 1110*/
else if(estado[0]==1 && estado[1]==1 && estado[2]==1 && estado[3]==0){
    if (movimento[0]==1 || movimento[1]==1 || movimento[2]==1 )
        r = punicao;
    else r = recompensa;
}
```

A.2 Base de Regras do Controle Neural ART1-R-MLPs-RR-Marcos

```
/*   OESTE             NORTE           LESTE           SUL    caso = 0000*/
if(estado[0]==0 && estado[1]==0 && estado[2]==0 && estado[3]==0){
    if(movimento[mov_ant] != 1) //movimento atual diferente movimento anterior
       r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0001*/
else if(estado[0]==0 && estado[1]==0 && estado[2]==0 && estado[3]==1){
    if (movimento[3]==1) // ir para o Sul, punicao
        r = punicao;
    // direcao do mov. anterior esta livre &&
    // nao repetiu mov. anterior
    else if (!ocorreu_bifurcacao && estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
        r = punicao;
    else if (!ocorreu_bifurcacao && estado[mov_ant] == 1 && movimento[1]==1)
            // quer ir para Norte
        r = punicao;
    else if (ocorreu_bifurcacao){
        if((mov_ant + 2) > 3 ){
            if(movimento[mov_ant - 2] == 1)
                r = punicao;
            else r = recompensa;
        }
        else if (movimento[mov_ant + 2] == 1)
            r = punicao;
        else r = recompensa;
```

```
    }
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0010*/
else if(estado[0]==0 && estado[1]==0 && estado[2]==1 && estado[3]==0){
    if (movimento[2]==1)
        r = punicao;
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (!ocorreu_bifurcacao && estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else if (!ocorreu_bifurcacao && estado[mov_ant] == 1 && movimento[0]==1)
            // quer ir para Oeste
        r = punicao;
    else if (ocorreu_bifurcacao){
        if((mov_ant + 2) > 3 ){
            if(movimento[mov_ant - 2] == 1)
                r = punicao;
            else r = recompensa;
        }
        else if (movimento[mov_ant + 2] == 1)
            r = punicao;
        else r = recompensa;
    }
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0011*/
else if(estado[0]==0 && estado[1]==0 && estado[2]==1 && estado[3]==1){
    if (movimento[2]==1 || movimento[3]==1)
        r = punicao;
    else if (estado[mov_ant] == 1){ //direcao do mov. anterior NAO esta livre
        //se mov. anterior == Leste e movimento atual != Norte OU
        //se mov. anterior == Sul e movimento atual != Oeste
        if ( (mov_ant==2 && movimento[1]!= 1) || (mov_ant==3 && movimento[0]!= 1))
            r = punicao;
        else r = recompensa;
    }
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0100*/
else if(estado[0]==0 && estado[1]==1 && estado[2]==0 && estado[3]==0){
    if (movimento[1]==1)
        r = punicao;
    // direcao do mov. anterior esta livre
    // nao repetiu mov. anterior
    else if (!ocorreu_bifurcacao && estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else if (!ocorreu_bifurcacao && estado[mov_ant] == 1 && movimento[3]==1)
            // quer ir para SUL
        r = punicao;
    else if (ocorreu_bifurcacao){
        if((mov_ant + 2) > 3 ){
            if(movimento[mov_ant - 2] == 1)
                r = punicao;
            else r = recompensa;
        }
```

```
        else if (movimento[mov_ant + 2] == 1)
            r = punicao;
        else r = recompensa;
    }
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0101*/
else if(estado[0]==0 && estado[1]==1 && estado[2]==0 && estado[3]==1){
    if (movimento[1]==1 || movimento[3]==1 )
        r = punicao;
    //direcao do mov. anterior esta livre
    // nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0110*/
else if(estado[0]==0 && estado[1]==1 && estado[2]==1 && estado[3]==0){
    if (movimento[1]==1 || movimento[2]==1)
        r = punicao;
    else if (estado[mov_ant] == 1){ //direcao do mov. anterior NAO esta livre
        //se mov. anterior == Leste e movimento atual != SUL OU
        //se mov. anterior == NORTE e movimento atual != Oeste
        if ( (mov_ant==2 && movimento[3]!= 1) || (mov_ant==1 && movimento[0]!= 1))
            r = punicao;
        else r = recompensa;
    }
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 0111*/
else if(estado[0]==0 && estado[1]==1 && estado[2]==1 && estado[3]==1){
    ocorreu_beco = true;
    if (movimento[1]==1 || movimento[2]==1 || movimento[3]==1 )
        r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE            SUL    caso = 1000*/
else if(estado[0]==1 && estado[1]==0 && estado[2]==0 && estado[3]==0){
    if (movimento[0]==1)
        r = punicao;
    // direcao do mov. anterior esta livre
    // nao repetiu mov. anterior
    else if (!ocorreu_bifurcacao && estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else if (!ocorreu_bifurcacao && estado[mov_ant] == 1 && movimento[2]==1)
            // quer ir para LESTE
        r = punicao;
    else if (ocorreu_bifurcacao){
        if((mov_ant + 2) > 3 ){
            if(movimento[mov_ant - 2] == 1)
                r = punicao;
            else r = recompensa;
        }
        else if (movimento[mov_ant + 2] == 1)
            r = punicao;
        else r = recompensa;
```

```
    }
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE             SUL    caso = 1001*/
else if(estado[0]==1 && estado[1]==0 && estado[2]==0 && estado[3]==1){
    if (movimento[0]==1 || movimento[3]==1 )
        r = punicao;
    else if (estado[mov_ant] == 1){ //direcao do mov. anterior NAO esta livre
        //se mov. anterior == OESTE e movimento atual != NORTE OU
        //se mov. anterior == SUL e movimento atual != LESTE
        if ( (mov_ant==0 && movimento[1]!= 1) || (mov_ant==3 && movimento[2]!= 1))
            r = punicao;
        else r = recompensa;
    }
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE              SUL   caso = 1010*/
else if(estado[0]==1 && estado[1]==0 && estado[2]==1 && estado[3]==0){
    if (movimento[0]==1 || movimento[2]==1 )
        r = punicao;
    //direcao do mov. anterior esta livre
    // nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE             SUL    caso = 1011*/
else if(estado[0]==1 && estado[1]==0 && estado[2]==1 && estado[3]==1){
    ocorreu_beco = true;
    if (movimento[0]==1 || movimento[2]==1 || movimento[3]==1 )
        r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE             SUL    caso = 1100*/
else if(estado[0]==1 && estado[1]==1 && estado[2]==0 && estado[3]==0){
    if (movimento[0]==1 || movimento[1]==1)
        r = punicao;
    else if (estado[mov_ant] == 1){ //direcao do mov. anterior NAO esta livre
        //se mov. anterior == OESTE e movimento atual != SUL OU
        //se mov. anterior == NORTE e movimento atual != LESTE
        if ( (mov_ant==0 && movimento[3]!= 1) || (mov_ant==1 && movimento[2]!= 1))
            r = punicao;
        else r = recompensa;
    }
    //direcao do mov. anterior esta livre   &&
    //nao repetiu mov. anterior
    else if (estado[mov_ant] == 0 && movimento[mov_ant]!= 1)
            r = punicao;
    else r = recompensa;
}
/*      OESTE             NORTE           LESTE             SUL    caso = 1101*/
else if(estado[0]==1 && estado[1]==1 && estado[2]==0 && estado[3]==1){
    ocorreu_beco = true;
    if (movimento[0]==1 || movimento[1]==1 || movimento[3]==1 )
        r = punicao;
    else r = recompensa;
```

```
}
/*      OESTE              NORTE           LESTE            SUL    caso = 1110*/
else if(estado[0]==1 && estado[1]==1 && estado[2]==1 && estado[3]==0){
    ocorreu_beco = true;
    if (movimento[0]==1 || movimento[1]==1 || movimento[2]==1 )
        r = punicao;
    else r = recompensa;
}
```

Referências Bibliográficas

ALBUS, J. S.; MCCAIN, H. G.; LUMIA, R. NASA/NBS *standard reference model for tele-robot control system architecture (*NASREM*)*. [S.l.], 1989.

ARBIB, M. A. The handbook of brain theory and neural networks. In: ____. [S.l.]: Michael A. Arbib, MIT Press, 2003. cap. "Schema Theory", p. 993–998.

ARKIN, R. C.; RISEMAN, E. M.; HANSEN, A. AuRa: An architecture for vision-based robot navigation. In: *DARPA Image Understanding Workshop*. Los Angeles, CA: [s.n.], 1987. p. 417–413.

ARLEO, A.; GERSTNER, W. Spatial cognition and neuro-mimetic navigation: A model of hippocampal place cell activity. *Biol. Cyb.*, v. 83, p. 287–299, 2000.

ARLEO, A.; MILLáN, J. del R.; FLOREANO, D. Efficient learning of variable-resolution cognitive maps for autonomous indoor navigation. *IEEE Transactions on Robotics & Automation*, v. 15, n. 6, p. 990–1000, December 1999.

ARLEO, A.; SMERALDI, F.; GERSTNER, W. Cognitive navigation based on nonuniform gabor space sampling, unsupervised growing networks, and reinforcement learning. *IEEE Transactions On Neural Networks*, v. 15, n. 3, p. 639–652, May 2004.

BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 13, p. 834–846, 1983.

BROOKS, R. A. A robust layered control system for a mobile robot. *IEEE Jornal on Robotics and Automation*, RA-2, n. 1, p. 14–23, March 1986.

BROOKS, R. A. “elephants don’t play chess”. *Robotics and Autonomous Systems*, v. 6, p. 3–15, 1990.

BROWNING, B. *Biologically Plausible Spatial Navigation for a Mobile Robot*. Tese (PhD thesys) — Department of Computer Science and Electrical Engineering, The University of Queesland, August 2000.

BRäUNL, T. *Embedded Robotics - Mobile Robot Design and Applications with Embedded Systems*. [S.l.]: Springer-Verlag, 2003. ISBN 3-540-03436-6.

BURES, J. et al. Rodent navigation after dissociation of the allocentric and idiothetic representations of space. *Neuropharmacology, Elsevier Science Ltd.*, v. 37, p. 689–699, 1998.

CAPI, G.; DOYA, K. Evolution of neural architecture fitting environmental dynamics. *Adaptive Behavior*, v. 13, n. 1, p. 53–66, 2005.

CARPENTER, G. A.; GROSSBER, S.; REYNOLDS, J. ARTMAP: A self-organizing neural network architecture for fast supervised learning and pattern recognition. *IEEE Conference - Neural Networks for Ocean Engineering,*, p. 863–868, 1991.

CARPENTER, G. A.; GROSSBERG, S. The ART of adaptive pattern recognition self-organizing by a neural network. *IEEE Computer*, v. 21, n. 3, p. 77–88, March 1988.

CARPENTER, G. A.; GROSSBERG, S. The handbook of brain theory and neural networks. In: ____. [S.l.]: Michael A. Arbib, MIT Press, 2003. cap. “Adaptive Resonance Theory”, p. 87–90.

CARPENTER, G. A.; GROSSBERG, S.; ROSEN, D. B. Fuzzy ART: An adaptive resonance algorithm for rapid, stable classification of analog patterns. *D.B. Neural Networks - IJCNN-91-Seattle International Joint Conference IEEE*, v. 2, p. 411–416, July 1991.

CARPENTER, G. A.; GROSSBERG, S.; ROSEN, D. B. ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition. *D.B. Neural Networks - IJCNN-91-Seattle International Joint Conference IEEE*, v. 2, p. 151–156, 1991.

CASTELLANO, G. et al. Reactive navigation by fuzzy control. In: *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*. [S.l.: s.n.], 1996. v. 3, p. 2143–2149.

CROWLEY, J. L. Navigation for an intelligent mobile robot. *IEEE Journal of Robotics and Automation*, RA-1, n. 1, p. 31–41, March 1985.

DIJKSTRA, E. W. A note on two problems in connection with graphs. *Numeriske Mathematik*, v. 1, p. 269–271, 1959.

ELFES, A. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, p. 46–57, 1989.

FILLIAT, D.; MEYER, J.-A. Map-based navigation in mobilie robots: I. A review of localization strategies. *Cognitive Systems*, v. 4, p. 243–282, 2003.

FILLIAT, D.; MEYER, J.-A. Map-based navigation in mobilie robots: II. A review of map-learning and path-planning strategies. *Cognitive Systems*, v. 4, p. 283–317, 2003.

FIRBY, R. J.; PROKOPWICZ, R. E.; SWAIN, M. J. An architecture for vision and action. In: *proceedings of 1995 International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann Pub., 1995. v. 1, p. 72–79.

FRANZ, M. O.; MALLOT, H. A. Biomimetic robot navigation. *Robotics and Autonomous Systems*, v. 30, p. 133–153, 2000.

FREEMAN, J. A.; SKAPURA, D. M. *Neural Networks: Algoritms, Aplications and Programming Techniques*. [S.l.]: Addison-Wesley Publishing Company, 1992.

GIRALT, G.; CHATILA, R.; VAISSET, M. An integrated navigation and motion control system for multisensory robots. In: BRADY; PAUL (Ed.). *Robotics Research*. Cambridge, MA: MIT Press, 1984. v. 1, p. 191–214.

GLASIUS, R.; KOMODA, A.; GIELEN, S. Neural networks dynamics for path planning and obstacle avoidance. *Neural Networks*, v. 8, n. 1, p. 125–133, 1995.

GOODRIDGE, S. G.; KAY, M. G.; LUO, R. C. Multilayered fuzzy behavior fusion for real-time reactive control of systems with multiple sensors. *IEEE Trans. Ind. Electron.*, v. 43, June 1996.

GROSSBERG, S. Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biol. Cybern.*, v. 23, p. 121–134, 1976.

GROSSBERG, S. Adaptive pattern classification and universal recoding: II. feedback, expectation, olfaction, and illusions. *Biol. Cybern.*, v. 23, p. 187–202, 1976.

GUIVANT, J. et al. Navigation and mapping in large unstructured environments. *The International Journal of Robotics Research*, v. 23, n. 4-5, p. 449–472, April-May 2004.

HAFNER, V. V. Cognitive maps in rats and robots. *Adaptive Behavior*, v. 13, n. 2, p. 87–96, 2005.

HAYKIN, S. *Redes neurais: princípios e práticas*. 2. ed. [S.l.]: Bookman®, 2001. ISBN 85-7307-718-2.

HEIKKONEN, J.; KOIKKALAINEN, P. Neural systems for robotics. In: ____. [S.l.]: Omid Omidvar and Patrick Van Der Smagt, 1997. cap. Self-Organization and Autonomous Robots, p. 297–337.

IVANJKO, E.; VASAK, M.; PETROVIC, I. Kalman filter theory based mobile robot pose tracking using occupancy grid maps. In: *Control and Automation, 2005. ICCA '05. International Conference on*. [S.l.: s.n.], 2005. v. 2, p. 869–874.

JAN, G. E.; CHANG, K.-Y.; PARBERRY, I. A new maze routing approach for path planning of a mobile robot. *Proceedings of the 2003 IEEE/ASME. International Conference on Advanced lntelligency Mechatronics (AIM 2003)*, p. 552–557, 2003.

JARADAT, M.; LANGARI, R. Line map construction using a mobile robot with a sonar sensor. In: *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*. [S.l.: s.n.], 2005. p. 1251–1256.

JEFFERIES, M. E.; YEAP, W.-K. Neural network approaches to cognitive mapping. In: *IEEE*. [S.l.: s.n.], 1995. p. 75–78.

JORGENSEN, C. C. Neural network representation of sensor graphs in autonomous robot path planning. In: *IEEE First International Conference on Neural Networks IV*. San Diego: [s.n.], 1987. p. 507–515.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Jounal of Artificial Intelligence Research*, p. 237–285, 1996.

KOHONEN, T. *Self-Organization and Associative Memory*. [S.l.]: New York: Springer-Verlag, 1988.

KONIDARIS, G. D.; HAYES, G. M. An architecture for behavior-based reinforcement learning. *Adaptive Behavior*, v. 13, n. 1, p. 5–32, 2005.

KONOLIGE, K.; MYERS, K. Artificial intelligence and mobile robots. In: ____. [S.l.]: D. Kortenkamp, R. Bonasson, R. Murphy, editors, MIT Press, 1998. cap. The Saphira Architecture for Autonomous Mobile Robots.

KRöSE, B.; van Dam, J. Adaptive state space quantization for reinforcement learning of collision-free navigation. In: *Proceedings of the IEEE International Conference on Intelligent Robots and System*. [S.l.: s.n.], 1992. p. 1327–1332.

KRöSE, B.; van Dam, J. Learning to avoid colision: A reinforcement learning paradigm for mobile robot navigation. In: *IFAC/IFIP/IMACS Internacional Symposium on Artificial Intelligence in Real-Time Control*. Delft: [s.n.], 1992. p. 295–300.

KRöSE, B.; van Dam, J. Neural systems for robotics. In: ____. [S.l.]: Omid Omidvar and Patrick Van Der Smagt, 1997. cap. Neural Vehicles, p. 297–337.

KRöSE, B. J. A.; EECEN, M. A self-organizing representation of sensor space for mobile robot navigation. In: *Proceedings of the 1994 IEEE/RSJ Internacional Conference on intelligent Robots and System*. Munich: [s.n.], 1994. p. 9–14.

KURZ, A. Constructing maps for mobile robot navigation based on ultrasonic range data. *IEEE Trans. Syst. Man Cybern.*, v. 26, n. Part B, p. 233–242, 1996.

LEBOUTHILLIER, A. E. W. Grey Walter and his turtle robots. *The Robot Builder*, v. 11, n. 5, p. 1–3, May 1999.

LEE, C. C. Fuzzy logic in control systems: Fuzzy logic controller - Part I. *IEEE Transactions in Systems, Man and Cibernetics*, v. 20, n. 2, p. 404–418, March-April 1990.

LEVITT, T. S.; LAWTON, D. T. Qualitative navigation for mobile robots. *Artificial Intelligence*, v. 44, p. 305–360, 1990.

LIN, L. J. Programming robots using reinforcement learning and teaching. In: *AAAI*. [S.l.: s.n.], 1991. p. 781–786.

LIN, L. J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, v. 8, p. 293–321, 1992.

LUMELSKY, V. J. A comparative study on the path length performance of maze-searching and robot motion planning algorithms. *IEEE Transactions on Robotics And Automation*, v. 7, n. 1, p. 57–66, February 1991.

MALLOT, H. A. et al. View-based cognitive map learning by an autonomous robot. In: *Proceedings of ICANN'95 - International Conference on Artificial Neural Networks*. Nanterre, France: [s.n.], 1995. (EC2, II), p. 381–386.

MARIN, L. de O.; ROISENBERG, M.; PIERI, E. R. D. Aprendizado de trajetória mímico da natureza na navegação de robôs móveis. In: *Anais da Sociedade Brasileira de Computação SBC 2006*. Campo Grande, Brasil: [s.n.], 2006.

MARIN, L. de O.; ROISENBERG, M.; PIERI, E. R. D. A neural architecture for online path learning in maze navigation. In: *8th International IFAC Symposium on Robot Control SYROCO 2006*. Bologna, Italy: [s.n.], 2006.

MARIN, L. de O.; ROISENBERG, M.; PIERI, E. R. D. Arquitetura cognitiva para navegação labirintos. In: *VIII Simpósio Brasileiro de Automação Inteligente*. Florianópolis, Brasil: [s.n.], 2007.

MARIN, L. de O.; ROISENBERG, M.; PIERI, E. R. D. Evolução de redes neurais artificiais em uma arquitetura cognitiva biologicamente inspirada para navegação autônoma em labirintos. In: *VIII Congresso Brasileiro de Redes Neurais*. Florianópolis, Brasil: [s.n.], 2007.

MARQUES, C.; LIMA, P. Multisensor navigation for nonholonomic robots in cluttered environments. *IEEE Robotics & Automation Magazine*, p. 70–82, September 2004.

MATARIC, M. J. Navigating with a rat brain: a neurobiologically-inspired model of robot spatial representation. *From Animals to Animats*, v. 1, p. 169–175, 1991. MIT Press, Cambridge, MA.

MATARIC, M. J. Behavior-based control: Main properties and implications. In: *Workshop on Intelligent Control System, IEEE International Conference on Robotics an Automation*. Nice, France: [s.n.], 1992.

MATARIC, M. J. Integration of representation into goal-driven behavior-based robots. *IEEE Trans. on Robotics and Automation*, v. 8, n. 3, p. 304–312, June 1992.

MEDEIROS, A. A. D. A survey of control architectures for autonomous mobile robots. *Journal of the Brazilian Computer Society*, v. 4, n. 3, 1998.

MESBAHI, L. et al. Reactive navigation of mobile robot by temporal radial basis function approach. In: *IEEE Region 10 Conference TENCON*. [S.l.: s.n.], 2004. v. 4, p. 483–486.

MILLAN, J. del R.; ARLEO, A. Neural network learning of variable grid-based maps for the autonomous navigation of robots. In: *Computational Intelligence in Robotics and Automation, CIRA'97. IEEE International Symposium*. [S.l.: s.n.], 1997. p. 40–45.

MILLáN, J. del R. Incremental acquisition of local networks for the control of autonomous robots. In: SPRINGER-VERLAG (Ed.). *Proceedings of the Seventh International Conference on Artificial Neural Networks*. Heidelberg, Germany: [s.n.], 1997. p. 739–744.

MILLáN, J. del R. The handbook of brain theory and neural networks. In: ____. [S.l.]: Michael A. Arbib, MIT Press, 2003. cap. "Robot Navigation", p. 987–990.

MONDADA, F.; E.FRANZI; LENNE, P. Mobile robot miniaturization: a tool for investigation in control algorithms. *Int. Symposium on Experimental Robotics. Kyoto, Japan*, 1993.

MORASSO, P.; VERCELLI, G.; ZACCARIA, R. Hybrid systems for robot planning. In: ALEKSANDER, I.; TAYLOR, J. (Ed.). *Artificial Neural Networks*. North-Holland/Elsevier, Amsterdam: [s.n.], 1992. v. 2, p. 691–697.

MORAVEC, H. P. The stanford cart and the cmu rover. *Proceedings of the IEEE*, v. 71, n. 7, p. 872–884, July 1993.

MURPHY, R. R. *Introduction to AI Robotics*. [S.l.]: Massachusetts Institute of Technology, 2000. ISBN 0-262-133383-0.

NAJAND, S.; LO, Z.-P.; BAVARIAN, B. Application of self-organizing neural networks for mobile robot environment learning. In: BEKEY, G.; GOLDBERG, K. (Ed.). *Neural Netwoks in Robotics*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1992. p. 85–96.

NEHMZOW, U. *Mobile Robotics: A Practical Introduction*. [S.l.]: Springer-Verlag London Limited, 2000. ISBN 1852331739.

NEHMZOW, U.; MCGONIGLE, B. Achieving rapid adaptations in robots by means of external tuition. *in: D.T. Cliff, P. Husbands, J.-A. Meyer, S. W. Wilson (Eds.), From Animals to Animats 3: Proceedings of the Adaptive Behavior, MIT Press, Cambridge, MA*, p. 301–308, 1994.

NILSSON, N. J. *Shakey The Robot*. 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1984.

NILSSON, N. J. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence*, v. 1, p. 139–158, Jan 1994.

O'KEEFE, J.; NADEL, L. *The hippocampus as a cognitive map*. [S.l.]: Oxford University Press, 1978.

OLIVEIRA, L. O. L. *Mapas auto-organizáveis de Kohonen aplicados ao mapeamento de ambientes de robótica móvel*. Dissertação (Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação) — INE/UFSC/Brasil, Florianópolis/SC, Outubro 2001.

OMIDVAR, O.; SMAGT, P. van der (Ed.). *Neural Systems for Robotics*. [S.l.]: Academic Press, 1997. ISBN 0-12-526280-9.

OVERHOLT, J. L.; HUDAS, G. R.; CHEOK, K. C. A modular neural-fuzzy controller for autonomous reactive navigation. In: *Fuzzy Information Processing Society. NAFIPS 2005. Annual Meeting of the North American*. [S.l.: s.n.], 2005. p. 121–126.

PEARL, J. *Causality*. New York: Cambridge University Press, 2000.

POMERLEAU, D. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, v. 3, n. 1, p. 88–97, 1991.

PRESCOTT, A. J. *Explorations in reinforcement and model-based learning*. Tese (Ph.D. thesis) — Dept of Psychology, Universit of Sheffield, U.K., 1994.

ROISENBERG, M. *PiramidNet - Redes Neurais Modulares e Hierárquicas: Fundamentos e Aplicações em Robótica*. Florianópolis, SC, 2000.

ROISENBERG, M. et al. Pyramidnet: A modular and hierarchical neural network architecture for behavior based robotics. In: *ISRA - International Symposium on Robotics and Automation*. Querétaro, México: [s.n.], 2004. p. 32–37.

ROSEN, B. E.; GOODWIN, J. M.; VIDAL, J. J. Adaptive range coding. *In: D. Touretzky (ed.) Neural Information Processing Systems*, v. 3, 1990.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey: Alan Apt, 1995. ISBN 0-13-103805-2.

RYLATT, R. M.; CZARNECKI, C. A.; ROUTEN, T. W. A partially recurrent gating network approach to learning action selection by reinforcement. In: *International Conference on Neural Networks*. [S.l.: s.n.], 1997. v. 3, p. 1689 – 1692.

SALICHS, M. A.; MORENO, L. Navigation of mobile robots: Open questions. *Robotica*, v. 18, n. 3, p. 227–234, 2000. Cited By (since 1996): 35. Disponível em: <www.scopus.com>.

SCHöLKOPF, B.; MALLOT, H. A. View-based cognitive mapping and path planning. *Adaptive Behavior*, v. 3, n. 3, p. 311–348, 1995.

SCHMAJUK, N. A.; THIEME, A. D. Purposive behavior and cognitive mapping: An adaptive neural network. *Biol. Cybern.*, v. 67, p. 165–174, 1992.

SEHAD, S.; TOUZET, C. Neural reinforcement path planning for the miniature robot khepera. *Proceedings of the 1995 World Congress on Neural Networks*, v. 2, 1995.

SELIG, J. M. *Introductory Robotics*. UK: Prentice Hall, 1992. ISBN 0-13-488875-8.

SILVA, F. de A. *Redes neurais hierárquicas para a implementação de comportamentos em agentes autônomos*. Dissertação (Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação) — INE/UFSC/Brasil, Florianópolis/SC, Outubro 2001.

SIMMONS, R. et al. A layered architecture for office delivery robots. In: *Autonomous Agents 97*. [S.l.: s.n.], 1997. p. 245–252.

SITTI, M. Miniature micro/nano-robotics. Carnegie Mellon University `http://nanoscience.bu.edu/seminars.asp`, acessed on June 8, 2006. April 2005.

SUTTON, R. S. *Temporal credit assignment in reinforcement learning*. Tese (Doutorado) — University of Massachusets, 1984.

SUTTON, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, v. 3, p. 9–44, 1988.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

THRUN, S. B. Handbook of intelligent control: Neural, fuzzy and adaptive approaches. In: ____. [S.l.]: Van Nostrand Reinhold, Florence, Kentucky 41022, 1992. cap. "The role of exploration in learning control", p. 1–27.

TOLEDO, F. J. et al. Map building with ultrasonic sensors of indoor environments using neural networks. *IEEE International Conference on Systems, Man, and Cybernetics*, v. 2, p. 920–925, 8-11 Oct 2000.

TOLMAN, E. C. Cognitive maps in rats and men. *Psychol. Rev.*, v. 55, p. 189–208, 1932.

TRULLIER, O. et al. Biologically based artificial navigation systems: review and prospects. *Progress in Neurobiology. Elsevier Science Ltd*, v. 51, p. 483–544, 1997.

VIEIRA, R. C. *Descrição de comportamentos robóticos utilizando uma abordagem gramatical e sua implementação através de redes neurais artificiais*. Dissertação (Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação) — INE/UFSC/Brasil, Florianópolis/SC, Abril 2004.

VOICU, H. Hierarchical cognitive maps. *Neural Networks*, v. 16, p. 569–576, 2003.

VOICU, H.; SCHMAJUK, N. A. Exploration, navigation and cognitive mapping. *Adapt. Behav.*, v. 8, p. 207–223, 2000.

VOICU, H.; SCHMAJUK, N. A. Three-dimensional cognitive mapping with a neural network. *Robot. Auton. Syst.*, v. 35, p. 21–35, 2001.

VOICU, H.; SCHMAJUK, N. A. Latent learning, shortcuts and detours: a computational model. *Behavioural Processes*, v. 1, n. 59, p. 67–86, 2002.

WATKINS, C. J. C. H. *Learning from delayed rewards*. Tese (Doutorado) — University of Cambridge, England, 1989.

WYETH, G.; BROWNING, B. Cognitive models of spatial navigation from a robot builder's perspective. *Adaptive Behaviour*, v. 6, n. 3/4, p. 509–534, 1998.

XU, X.; WANG, X.-N.; HE, H.-G. A self-learning reactive navigation. *Proceedings of the Second International Conference on Machine Learning and Cybernetics*, p. 2384–2388, 2-5 November 2003.

ZADEH, L. A. *Fuzzy sets*. [S.l.]: Inform. and Control, 1965. 705-740 p.

ZADEH, L. A. Out line of a new approach ot the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Mans, and Cybernetics SMC-3*, p. 28–44, 1973.

ZHANG, J.; WILLE, F.; KNOLL, A. Fuzzy logic rules for mapping sensor data to robot control. *Advanced Mobile Robots, Euromicro Workshop on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 29, 1996.

ZHU, A.; YANG, S. X. Self-organizing behavior of a multi-robot system by a neural network approach. *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, p. 1204–1209, October 2003.

ZIMMER, U. W. Robust world-modelling and navigation in a real world. *Neurocomputing*, v. 13, p. 247–260, 1996.