EXTRAÇÃO AUTOMÁTICA DE MAPAS DE ATRIBUTOS BASEADA EM TÉCNICA BAYESIANA PARA LOCALIZAÇÃO DE ROBÔS MÓVEIS

Fábio L. N. de Miranda[*]
fabio.ita04@gmail.com

Carlos H. C. Ribeiro[*]
carlos@comp.ita.br

[*]Divisão de Ciência da Computação
Insituto Tecnológico de Aeronáutica
Praça Mal. Eduardo Gomes, 50
12228-900 - São José dos Campos, SP - Brasil

RESUMO

A solução do problema de determinação da postura (ou localização) é de fundamental importância para a incorporação de autonomia em robôs móveis. Entretanto, devido à natureza inexata do movimento, uma localização precisa não é possível usando unicamente odometria. Faz-se necessário extrair das leituras sensoriais do robô informações que permitam corrigir os desvios intrínsecos a cada ação executada. Neste contexto, os algoritmos de Monte Carlo estimam e atualizam a postura (com base em modelos *a priori* de sensores e atuadores) através de um conjunto de partículas que simbolizam possíveis posturas do robô no ambiente, associadas a uma crença que indica quão bem estas se aproximam de sua localização real. Complexa, no entanto, é a tarefa de obtenção do modelo sensorial, principalmente quando realizada por meio de técnicas não-automáticas. A idéia central aqui utilizada consiste no treinamento de redes neurais artificiais para extração automática de um mapa de atributos que simplifica o modelo sensorial, usando um método bayesiano (BaLL – Bayesian Landmark Learning) a partir de leituras sensoriais. Este trabalho descreve a implementação utilizando-se como base a plataforma ARIA para simulação de um robô móvel Magellan Pro cujos sensores externos são sonares. Os resultados mostram o funcionamento da técnica e a sua aplicabilidade para a obtenção automática de mapas de atributos.

PALAVRAS-CHAVE: Modelos de sensores, Localização de robôs móveis, Aprendizado de máquina.

ABSTRACT

Solving the problem of pose determination is a fundamental issue for incorporating the autonomy concept in mobile robots. However, due to the inaccurate nature of movement, localization is not possible if based solely on odometry. Information from other sensor readings must be extracted in such a way that intrinsic errors from action execution can be compensated for. In this context, Monte Carlo Localization estimates and updates pose (based on previously designed sensor and actuator models) by using a set of particles that define possible states the robot can occupy in its working environment, associated to a belief that indicates how close these particles are from the real robot localization. However, sensor model design is a complex task, and hard to be done by non-automatic techniques. The key idea in this paper is to train artificial neural networks for automatically extracting feature maps that simplifies the sensorial model, using a Bayesian method (BaLL – Bayesian Landmark Learning) from sensor data. This work describes an implementation, using as a base the Aria platform to simulate a mobile robot whose external sensors are sonars. Results show the suitability of the method and its applicability for automatic feature

Artigo submetido em 08/10/2005
1a. Revisão em 23/05/2006
2a. Revisão em 28/11/2006
Aceito sob recomendação do Editor Associado
Prof. Ivan Nunes Da Silva

map extraction from sonar readings.

KEYWORDS: Sensor models, Mobile robot localization, Machine learning.

1 INTRODUÇÃO

A incorporação do conceito de autonomia em máquinas envolve a compreensão de suas formas de interação com o mundo. No caso de robôs, isto ocorre por meio de sensores e atuadores. A análise destes dispositivos revela que eles funcionam de maneira não-determinística: uma série de fatores faz com que sensores não possam ser considerados totalmente confiáveis, e que as ações executadas pelos atuadores nem sempre produzam o mesmo efeito.

Tais questões demandam estudo especial para possibilitar soluções flexíveis e robustas, que consigam contornar as dificuldades impostas pelo não-determinismo intrínseco do funcionamento dos dispositivos robóticos. Neste contexto, as técnicas probabilísticas vêm se revelando como promissoras candidatas a prover soluções compreensivas e de tempo real. Um caso particular onde estas técnicas vêm sendo empregadas com êxito consiste no Problema da Localização, definido em linhas gerais como a capacidade autônoma de o robô conseguir estimar sua própria localização, tendo como base somente uma representação do ambiente onde atua e as leituras de seus sensores.

Muitos problemas do mundo real envolvem a estimativa de grandezas que não podem ser determinadas com exatidão: apesar de apresentarem um estado real único e bem definido, consegue-se somente representá-lo segundo uma densidade de probabilidades sobre o espaço de todos os estados possíveis. Costuma-se denotar esta densidade como *crença* – quanto maior a crença em um estado específico, então melhor ele deve se aproximar do real estado da grandeza que se deseja medir.

O Problema da Localização de Robôs Móveis constitui um problema desta espécie, uma vez que a natureza não-determinística dos dispositivos mecânicos e perceptivos de um robô não permite a determinação exata de sua localização, informação indispensável para ser capaz de operar de maneira autônoma. Assim, deve-se dotá-lo de subsistemas que consigam estimar a localização a partir dos dados disponíveis em tempo real. Em geral, estes dados consistem de modelos *a priori* sobre o ambiente, e das leituras percebidas por meio de sensores.

A disponibilidade de conhecimento *a priori* e de dados observáveis que permitam inferir a crença possibilitam uma modelagem segundo princípios bayesianos. O mecanismo de inferência se baseia no Teorema de Bayes, e a técnica fundamental – que de fato corresponde a um modelo bayesiano geral para a localização baseada em modelos sensoriais e de atuação do robô – é conhecida como Localização Markov (Fox et al., 1999). Por meio da modelagem dos dados observáveis em funções de probabilidades, pode-se atualizar a crença à medida que novas observações do problema se tornam disponíveis. Entretanto, a Localização Markov – por razões mostradas na Seção 2.1 – é uma técnica computacionalmente intratável na grande maioria dos problemas de localização enfrentados por robôs reais.

Neste contexto, técnicas Monte Carlo (Doucet et al., 2001) consistem num conjunto de métodos que provêem aproximações atrativas e convenientes. Muito estudados nos últimos anos – sob diversos nomes, por exemplo: *filtros de partículas*, *filtros Monte Carlo* e *algoritmos de condensação* – métodos Monte Carlo modelam as distribuições segundo um conjunto de amostras – freqüentemente denominadas *partículas* – para aproximar as funções de probabilidade envolvidas.

Comumente utiliza-se o termo MCL – *Monte Carlo Localization* – para se referir à aplicação de técnicas Monte Carlo ao Problema da Localização. Neste caso, as únicas funções probabilísticas necessárias para a inferência da localização referem-se a um modelo dinâmico e um modelo de sensores, que possam representar funções de probabilidades que modelem as imperfeições do movimento e da percepção sensorial de determinado agente móvel.

Embora a técnica MCL tenha já se mostrado bastante efetiva para a localização de robôs móveis (Thrun et al., 2001), a obtenção dos modelos constitui um problema não trivial, principalmente no tocante ao modelo de sensores, devido ao número de dimensões do espaço das leituras possíveis. Apesar de ser possível realizar simplificações sobre este espaço, estas podem acabar gerando modelos pobres, por desprezarem muitas das informações sensoriais disponíveis. A própria obtenção do modelo se faz de modo bastante trabalhoso, por envolver a análise de uma grande massa de dados de sensores colhida no ambiente onde o agente opera.

A técnica BaLL – *Bayesian Landmark Learning* (Thrun, 1997) – aponta uma solução interessante para a questão da modelagem sensorial. Baseada em uma medida *bayesiana* do erro de localização utilizada para o treinamento de redes neurais artificiais, ela permite a construção de modelos de sensores sem desprezar dados, e de forma automática. Neste sentido, a obtenção do modelo sensorial é feita considerando a necessidade real de localização, e não com base em uma caracterização *a priori*. Esta obtenção automática é um problema pouco estudado em Robótica Móvel, embora seja reconhecido como uma questão relevante em Inteligência Artificial (Greiner e Isukapalli, 1994) e em Ciências Cognitivas (Chown et al., 1995).

Este trabalho tem como objetivo a implementação do algoritmo BaLL para obtenção automática de um mapa de atributos equivalente a um modelo de sensores do tipo sonar em um robô móvel ActivMedia Pioneer simulado (ActivMedia Robots, 2001), bem como a análise de seu funcionamento, de modo a disponibilizá-lo para utilização integrada com um módulo MCL existente.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Formulação Bayesiana e Técnicas Monte Carlo

Seja Ξ o espaço de estados de uma grandeza ξ não-observável diretamente. Seja S o espaço das observações s a partir das quais se pode inferir aquela grandeza. Sejam ainda as seguintes distribuições, com o índice t denotando o instante em que elas foram avaliadas:

$$P(\xi_0) \tag{1}$$

$$P(\xi_{t+1}|\xi_t), \text{com} t \geq 0 \tag{2}$$

$$P(s_t|\xi_t), \text{com} t \geq 0 \tag{3}$$

A Equação 1 representa o conhecimento *a priori* que se tem do problema. A Equação 2 modela a função de transição entre estados em Ξ, podendo ser interpretada como "a probabilidade de ocorrência do estado ξ_{t+1}, supondo que o estado anterior era ξ_t", representando a dinâmica de estados. Finalmente, a Equação 3 modela a distribuição marginal "da observação s_t ser observada no estado ξ_t".

A crença da grandeza de interesse pode ser avaliada em dois momentos: antes e depois da incorporação das observações. Neste caso, definem-se as crenças *a priori* e *a posteriori*:

$$Bel_{prior}(\xi_t) = P(\xi_t|s_1, \xi_1, \ldots, s_{t-1}, \xi_{t-1}) \tag{4}$$

$$Bel_{posterior}(\xi_t) = P(\xi_t, |s_1, \xi_1, \ldots, s_{t-1}, \xi_{t-1}, s_t) \tag{5}$$

A expressão (4) permite relacionar o conhecimento *a priori* com a crença inicial do estado:

$$Bel_{prior}(\xi_0) = P(\xi_0) \tag{6}$$

Por meio do Teorema de Bayes e do Teorema da Probabilidade Total, respectivamente, podem ser deduzidas as seguintes expressões (Doucet et al., 2001):

$$Bel_{\text{posterior}}(\xi_t) = \frac{P(s_t|\xi_t) \cdot Bel_{\text{prior}}(\xi_t)}{\int\limits_{\Xi} P(s_t|\xi_t) \cdot Bel_{\text{prior}}(\xi_t)\, d\xi_t} \tag{7}$$

$$Bel_{\text{prior}}(\xi_{t+1}) = \int\limits_{\Xi} P(\xi_{t+1}|\xi_t) \cdot Bel_{\text{posterior}}(\xi_t)\, d\xi_t \tag{8}$$

As equações (7) e (8) sugerem um algoritmo recursivo para atualização da crença: $Bel_{posterior}$ pode ser obtida a partir de Bel_{prior} por meio da função de observações (7), enquanto Bel_{prior} pode ser obtida a partir de $Bel_{posterior}$ por meio da função de transição de estados (8). Denomina-se por *passo de atualização* a incorporação dos dados observáveis, representada por (7), e por *passo de predição* a estimativa da crença *a priori* quando ocorrem transições de estados, representada por (8).

A complexidade em avaliar as integrais destas duas expressões levou à busca de aproximações baseadas em aproximações gaussianas das distribuições, como os filtros de Kalman (Kalman, 1960; Smith et al., 1990) e suas variantes menos restritivas, como os filtros de Kalman multi-hipótese (Jensfelt e Kristensen, 1999). Técnicas Monte Carlo para estimação surgiram também em meados dos anos 60, no entanto foram deixadas de lado devido às limitações computacionais da época. Retomadas com o crescimento do poder computacional experimentado no final dos anos 80, possibilitaram vários avanços no estudo de filtros bayesianos (Doucet, Freitas e Gordon, 2001).

Em Weisstein (2004) e Wittwer (2004), as técnicas Monte Carlo são definidas, em linhas gerais, como técnicas que empregam distribuições probabilísticas aleatórias de tamanho finito para obtenção de soluções numéricas de problemas cujas soluções analíticas apresentam alto grau de complexidade. Esta idéia pode ser utilizada para aproximar as crenças contínuas (7) e (8) por um conjunto finito de amostras (*partículas*), associadas a fatores de importância (pesos):

$$Bel(\xi_t) = \{\xi_{i,t}, w_{i,t}\}_{i=1,\ldots,m} \tag{9}$$

Cada partícula $\xi_{i,t}$consiste numa amostra de um estado escolhido aleatoriamente no espaço de estados, segundo as funções de observação e de transição. O algoritmo de evolução da crença nos instantes t consiste nos passos mostrados no quadro da Figura 1.

Entradas:

s_t

$B_{el}(\xi_{t-1}) = \{\xi_{i,t-1}, w_{i,t-1}\}_{i=1,\ldots,m}$

Saída:

$B_{el}(\xi_t)$

Inicialização:

$B_{el}(\xi_t) = \varnothing$

Ciclo: Obter as 'm' amostras de $B_{el}(\xi_t)$ *através dos seguintes passos:*

1) Gerar partículas $\xi_{i,t-1}$ *aleatórias segundo seus respectivos pesos* $w_{i,t-1}$.

2) Gerar partículas $\xi_{i,t}$ *segundo o modelo de transição de estados* $P(\xi_t|\xi_{t-1})$

3) Atribuí-las um peso $w_{i,t}$, *avaliado pelo modelo de observações* $P(s_t|\xi_t)$.

4) Adicionar o par obtido $\{\xi_{i,t}, w_{i,t}\}$ *ao conjunto* $B_{el}(\xi_t)$.

Finalização:

Normalizar os pesos $w_{i,t}$ *em* $B_{el}(\xi_t)$.

Figura 1: Algoritmo Monte Carlo

O passo 1 do algoritmo consiste em uma etapa de preparação, onde se substitui as partículas com baixo fator de importância por outras, selecionadas aleatoriamente no espaço de estados Ξ. O passo 2 utiliza o modelo de transição de estados para obter as novas partículas para a próxima iteração. O passo 3 utiliza a observação obtida nesta iteração para avaliar as partículas obtidas no passo 2, atribuindo-lhes um fator de importância. Extraídas as $m$partículas, obtém-se a crença *Bel* (ξ_t) pela normalização dos respectivos pesos.

2.2 O Problema da Localização

O problema da localização consiste em determinar a posição – ou *postura* – de agentes móveis, condicionada às leituras de sensores dos quais eles sejam dotados. São exemplos de instâncias deste problema: a determinação da postura de um robô no ambiente em que atua; a guiagem de aeronaves por radar; o reconhecimento de ambientes marítmos por submarinos dotados de sonares; os subsistemas de localização autônoma de veículos não tripulados; entre outros. Como não se pode determinar exatamente a localização de tais agentes, resta representá-la segundo uma crença sobre possíveis estados.

Para a modelagem dos comportamentos dinâmico e perceptivo de um robô, deve-se considerar que:

***As ações são não-determinísticas*:** As ações do robô nem sempre terão efeitos similares, devido a uma série de fatores: ocorrência de derrapagens intrísecas dos próprios atuadores, variações no atrito com a superfície do ambiente, ocorrências de falhas na execução de uma ação, inacurácia odométrica, desbalanceamento dos atuadores de movimento, etc.

***O sensoriamento é incompleto ou imperfeito*:** exceto no caso de robôs cujos comportamentos podem ser totalmente programados *a priori*, toda informação disponível para o robô será proveniente dos seus sensores. No entanto, tais informações não podem ser tomadas como totalmente confiáveis. Pelo contrário, a maioria dos sensores de baixo custo, como sonares, costuma falhar na detecção de objetos ou ao avaliar sua distância relativa a eles. Isso acontece mesmo em dispositivos mais sofisticados. Por exemplo, a utilização de câmeras (visão computacional) é bastante prejudicada por variações de intensidade luminosa, o que torna complexo o tratamento de dados tomados em diferentes horas do dia, em diferentes meses do ano, em ambientes onde ocorrem efeitos de reflexão e refração, etc. Algoritmos de processamento sensorial devem, portanto, ser capazes de lidar com as limitações dos dispositivos perceptivos.

Seja a postura ξ_t – localização real do robô no instante t – definida pela terna:

$$\xi_t = (x_t, y_t, o_t) \tag{10}$$

As variáveis de estado x_t e y_t indicam coordenadas de ξ_t num sistema de coordenadas cartesianas do ambiente, enquanto o_t indica a orientação de ξ_t. O espaço de todos os possíveis estados que o robô pode ocupar representa o seu ambiente de atuação. A Figura 2, extraída de Thrun et al.. (2001), ilustra aspectos relevantes da representação da localização por densidades de probabilidades, bem como o comportamento das amostras Monte Carlo que aproximam estas densidades, aplicada ao Problema da Localização de Robôs Móveis.

As distribuições na coluna A da Figura 2 simbolizam uma situação onde as variáveis x_{t-1}e y_{t-1} encontram-se bem determinadas, porém nada se pode afirmar sobre o_{t-1}. Sob tal condição, a execução de uma ação 'mover a distância d para frente' (coluna B) provoca uma dispersão da nuvem de probabilidades que aproxima a postura do robô, num raio d em torno de (x_{t-1}, y_{t-1}). Após a ação, o robô passa a ocupar o estado ξ_t, onde realiza a leitura s_t de seus sensores. Na coluna C, a função de observações avalia a probabilidade de a leitura ter sido tomada nos diversos estados do ambiente. A interseção das regiões (coluna B) e (coluna C) permitem estimar a distribuição posterior de ξ_t, mostrada na coluna D. Em respeito à orientação o_{t-1}, anteriormente desconhecida,

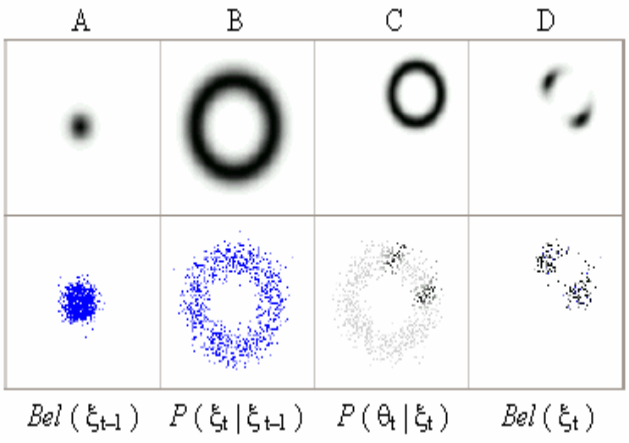

Figura 2: Representação da localização por densidades de probabilidade (extraído de Thrun et al., 2001).

já se pode afirmar (observe a coluna D) que se tratava de um ângulo do primeiro quadrante.

2.2.1 Modelo Dinâmico

O estado ξ_{t+1} alcançado após a execução de uma ação de movimento (translação ou rotação) depende do estado ξ_t anterior, e da ação $a$executada. Sendo não-determinística a natureza do movimento, a modelagem da função de transição de estados deve buscar reproduzir a dinâmica real do robô. Assim, denota-se por Modelo de Movimento do robô, ou simplesmente Modelo Dinâmico, a função de transição de estados reescrita como:

$$P(\xi_{t+1}|\xi_t, a_t), \text{com } t \geq 0 \tag{11}$$

2.2.2 Modelo de Sensores

A modelagem da função de observações é um problema complexo. A dificuldade consiste em determinar, para cada estado e possível leitura de um ambiente, uma boa estimativa de P ($s_t|\xi_t$), que também deve ser computacionalmente viável para poder ser empregada no robô em aplicações em tempo real. Uma opção para contornar tamanha dificuldade consiste em analisar atributos que possam ser extraídos das leituras, ao invés de tratar as leituras sensoriais em si. Assim, denota-se a função de mapeamento σ como o modelo que transforma leituras sensoriais em vetores de atributos:

$$\sigma \quad : S \rightarrow \quad F \tag{12}$$

$$f = \sigma(s) \tag{13}$$

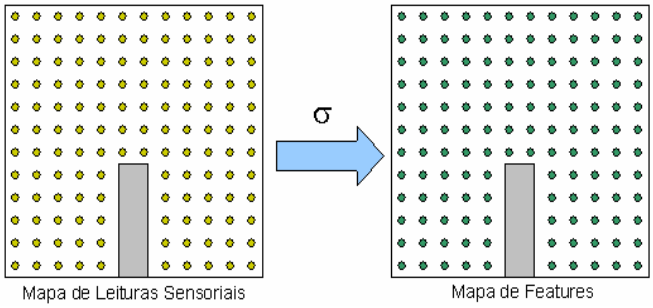

Figura 3: Modelagem de leituras sensoriais por atributos.

Ao invés de trabalhar com P ($s|\xi$), basta modelar

$$P(f|\xi) \tag{14}$$

de tratamento mais simples. Por relacionar vetores de atributos f aos diversos estados ξ do ambiente, (14) é frequentemente denominada de mapa de atributos do ambiente.

A Figura 3 ilustra um exemplo esquemático de mapeamento. No mapa físico de ambiente representado à esquerda, cada círculo representa um estado onde dados sensoriais foram previamente coletados e armazenados. Por meio de uma função de extração de atributos, estes dados são mapeados em um mapa de atributos, que permite estimar probabilidades condicionais $P(f|\xi)$ adequadas para utilização em substituição ao modelo de sensores $P(s|\xi)$. Na seção 3.3 mostrar-se-ão duas soluções distintas para este modelo, sendo uma delas obtida de forma automática pelo algoritmo *BaLL*.

2.2.3 Localização Monte Carlo de Robôs Móveis

Para incorporar o conceito de localização a partir de atributos, faz-se necessário reescrever as expressões (4) e (5):

$$Bel_{prior}(\xi_t) = P(\xi_t|f_1, a_1, \ldots, f_{t-1}, a_{t-1}) \tag{15}$$

$$Bel_{posterior}(\xi_t) = P(\xi_t|f_1, a_1, \ldots, a_{t-1}, f_t) \tag{16}$$

Bem como as expressões (7) e (8):

$$Bel_{\text{posterior}}(\xi_t) = \frac{P(f_t|\xi_t) \cdot Bel_{\text{prior}}(\xi_t)}{\int_{\Xi} P(f_t|\xi_t) \cdot Bel_{\text{prior}}(\xi_t)\, d\xi_t} \tag{17}$$

$$Bel_{\text{prior}}(\xi_{t+1}) = \int_{\Xi} P(\xi_{t+1}|\xi_t, a_t) \cdot Bel_{\text{posterior}}(\xi_t)\, d\xi_t \tag{18}$$

Pode-se então aplicar o algoritmo Monte Carlo mostrado no quadro da Figura 1 ao problema da Localização de Robôs Móveis, fazendo o Modelo Dinâmico (11) atuar como a função transição de estados, e o Modelo de Sensores (14) atuar como a função de observações.

2.3 O Algoritmo BaLL

2.3.1 Erro Bayesiano de Localização

Mostrou-se até aqui a determinação da postura de um agente móvel como um caso específico de estimativa de estados. Pode-se avaliar eficácia deste estimador avaliando-se uma medida de erro da estimativa. Numa determinada distribuição, supondo que ξ^* corresponda ao verdadeiro estado, calcula-se o erro em relação a este pela expressão:

$$E(\xi^*) = \int_{\Xi} e(\xi^*, \xi) \cdot Bel(\xi)\, d\xi \tag{19}$$

A função e (x, y) representa uma medida de distância entre os estados x e y, podendo ser aplicada, por exemplo, a métrica euclidiana. Para calcular o erro sobre toda a distribuição, pondera-se a contribuição (19) de cada um dos estados por ela representados. Assim, obtém-se:

$$E = \iint_{\Xi} e(\xi^*, \xi) Bel(\xi)\, P(\xi^*)\, d\xi\, d\xi^* \tag{20}$$

A avaliação de $P(\xi^*)$ depende da forma da distribuição *Bel* (ξ) utilizada. Em Thrun (1997), utiliza-se (20) para avaliar o erro posterior de uma distribuição:

$$E_{\text{posterior}} = \iint_{\Xi} e(\xi^*, \xi) Bel_{\text{posterior}}(\xi)\, P(\xi^*)\, d\xi\, d\xi^* \tag{21}$$

Pode-se então desenvolver o termo da crença posterior na equação (21) utilizando a equação (17), de modo a identificar dependências da medida de erro com o mapa de atributos. Observando-se que

$$P(f) = \int_{\Xi} P(f|\xi)\, Bel_{\text{prior}}(\xi)\, d\xi \tag{22}$$

e computado-se $E_{posterior}$ médio sobre todos os possíveis vetores de atributos, obtém-se:

$$\bar{E}_{\text{posterior}} = \iint_{\Xi} e(\xi^*, \xi) Bel_{\text{prior}}(\xi)\, P(\xi^*) \int_{F} \frac{P(f|\xi)P(f|\xi^*)}{P(f)} df\, d\xi\, d\xi^* \tag{23}$$

Pode-se notar que a Equação (23) guarda dependência com a crença *a priori* que modela a determinação de estados, bem como com a escolha do mapa de atributos. Assim, pode-se questionar se, dentre todos os filtros σ que transformam leituras sensoriais em atributos, existe **algum** capaz de minimizar o erro $\bar{E}_{\text{posterior}}$. O algoritmo *BaLL* emprega redes neurais artificiais para analisar esta questão.

2.3.2 Implementação do Algoritmo BaLL

Seja um filtro σ definido por:

$$\sigma = (g_1, g_2, \ldots g_n) \tag{24}$$

cujas componentes representem uma coleção de saídas de redes neurais *feedforward* (Figura 4) que recebem como entradas informações provenientes de sensores, ou seja:

$$g_i : S \quad \rightarrow [0, 1] \tag{25}$$

Cada combinação (g_1 , g_2 ,... g_n) poderia ser considerada um atributo no espaço Sconsiderado. Sendo a saída um número real no intervalo [0,1], tal interpretação não possui muito significado prático, por tornar infinito o espaço F – o qual se busca modelar para realizar uma simplificação do espaço S.

No entanto, Thrun (1997) sugere uma interpretação conveniente para a expressão (24), que consiste em admitir Fcomo sendo o conjunto {0,1}^{n}, ou seja, um vetor de n componentes que podem assumir somente valores 0 e 1. Neste caso, o número de atributos mapeados será finito e igual a 2^n. Assume-se, então, que a saída da i-ésima rede neural corresponde à probabilidade da i-ésima componente do vetor de atributos ser igual à unidade:

$$P(f_i = 1|s) = g_i(s) \tag{26}$$

$$P(f_i = 0|s) = 1 - g_i(s) \tag{27}$$

Pode-se então afirmar que cada uma das redes empregadas funciona como uma extratora estocástica de componentes de

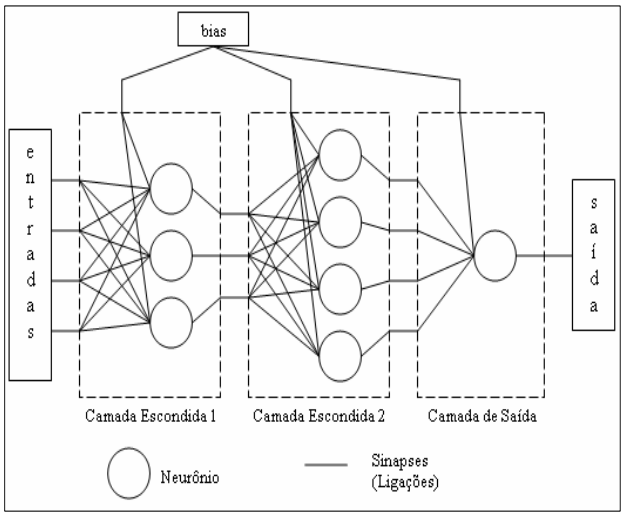

Figura 4: Uma rede neural artificial multicamadas

atributos. Considerando-se o atributo $f = (f_1, f_2, \ldots, f_n)$, calcula-se $P(f|s)$ pela probabilidade conjunta:

$$P(f|s) = \prod_{i=1}^{n} P(f_i|s) \tag{28}$$

calculada a partir das saídas das redes neurais. A expressão (28) é um termo que relaciona a medida de erro (23) com o modelo de sensores $P(s|\xi)$, através do teorema da probabilidade total:

$$P(f|\xi) = \int_S P(f|s,\xi)\, P(s|\xi)\, ds \tag{29}$$

Considerando a independência condicional de f e ξ dado s (Pearl, 1999), obtém-se:

$$P(f|\xi) = \int_S P(f|s)\, P(s|\xi)\, ds \tag{30}$$

Consolidadas as definições que relacionam as saídas das redes neurais com o modelo de sensores, pode-se derivar o algoritmo BaLL buscando-se a minimização dessa medida de erro.

Inicialmente, destaca-se que as integrais da Equação (23) são, de modo geral, intratáveis computacionalmente. Devem então ser utilizadas aproximações baseadas em somatórios e obtidas da aplicação da Equação (30) sobre as Equações (22) e (23), resultando respectivamente (Thrun, 1997):

$$P(f) = \sum_{\langle s,\xi\rangle \in X} P(f|s)\, Bel_{\text{prior}}(\xi) \tag{31}$$

$$\bar{E}_{\text{posterior}} = \sum_{\langle s^*,\xi^*\rangle \in X} \sum_{\langle s,\xi\rangle \in X} e(\xi^*,\xi)\, Bel_{\text{prior}}(\xi)\, P(\xi^*) \sum_{f=(0,0,0,\ldots 0)}^{f=(1,1,1,\ldots 1)} \frac{P(f|s)\, P(f|s^*)}{P(f)} \tag{32}$$

Neste caso, ao invés de calcular a medida de erro por todo o espaço de estados Ξ, emprega-se um conjunto de estados-referência para determinar um conjunto X de pares de treinamento das redes neurais:

$$X = \{\langle s_k, \xi_k\rangle \,|k = 1, \ldots K\} \tag{33}$$

com K denotando a cardinalidade ou tamanho deste conjunto. É importante lembrar que a obtenção e o armazenamento do conjunto de treinamento devem ser realizados previamente à sua utilização na extração do modelos de sensores.

O treinamento das redes neurais extratoras de atributos e correspondente ao filtro σ da expressão (24) baseia-se na descida do gradiente e no algoritmo *backpropagation* para realizar a busca de um conjunto de redes que minimizem a medida do erro posterior de localização (32). Para cada rede g_i calculam-se os gradientes das respectivas camadas de saída pela equação (34). Nesta expressão, o operador $\delta_{A,B}$ representa o *delta de Kronecker*, cujo resultado é 1, se A = B, e 0 caso contrário. O termo $\partial g_i(\bar{s})/\partial w_{i\mu\nu}$ refere-se à derivada da função de ativação utilizada no cálculo das saídas das redes (Haykin, 1999). Maiores detalhes podem ser encontrados em Thrun (1997).

Por meio do algoritmo *backpropagation*, pode-se obter os gradientes das camadas internas das redes segundo as medidas obtidas em (34).

$$\frac{\partial E_{\text{posterior}}}{\partial w_{i\mu\nu}} = \sum_{\langle \bar{s},\bar{\xi}\rangle \in X} \frac{\partial g_i(\bar{s})}{\partial w_{i\mu\nu}} \sum_{\langle s^*,\xi^*\rangle \in X} \sum_{\langle s,\xi\rangle \in X} e(\xi^*,\xi) P(\xi^*) Bel_{\text{prior}}(\xi) \cdot \sum_{f_1=0}^{1} \sum_{f_2=0}^{1} \cdots \sum_{f_n=0}^{1} \prod_{j\neq i} P(f_j|s^*) P(f_j|s) (2\delta_{f_i,1} - 1) \cdot \left[\frac{\delta_{\xi_i^*,\bar{\xi}} P(f_i|s) + \delta_{\xi_i,\bar{\xi}} P(f_i|s^*)}{\sum_{\langle \tilde{s},\tilde{\xi}\rangle \in X} \prod_{j=i} P(f_j|\tilde{s})} - \frac{P(f_i|s^*) P(f_i|s) \prod_{j\neq i} P(f_j|\bar{s}) Bel_{\text{prior}}(\bar{\xi})}{\left(\sum_{\langle \tilde{s},\tilde{\xi}\rangle \in X} \prod_{j=i} P(f_j|\tilde{s}) \right)^2} \right] \quad (34)$$

Entradas
Conjunto de treinamento $X = \{\langle s_k, \xi_K\rangle | k = 1, \ldots, K\}$
Crença de Treinamento $Bel_{\text{prior}}(\xi)$
Saídas
Parâmetros (pesos) otimizados das redes neurais que compõem σ.

Inicialização:
Os parâmetros das redes devem ser iniciados com pequenos valores aleatórios.

Ciclo:
Para cada par $\langle s_k, \xi_k\rangle$, *calcular as probabilidades condicionais* (26) e (27).
Calcular o erro posteirior (32)
Calcular o gradiente (34) *do erro posterior, para cada uma das redes.*
Atualizar os pesos das redes de acordo com (35).

Figura 5: Algoritmo BaLL.

Calculados os gradientes de todos os neurônios da rede, a atualização de pesos obedece à equação de atualização (35):

$$w_{i\mu\nu} \leftarrow w_{i\mu\nu} + \eta \frac{\partial E_{\text{posterior}}}{\partial w_{i\mu\nu}} \quad (35)$$

onde $w_{i\mu\upsilon}$ é o peso da conexão entre os neurônios ie $\mu\upsilon$ e η é a taxa de aprendizagem. O quadro da Figura 5 sumariza os passos do algoritmo BaLL.

A Seção 3.3.2 apresenta detalhes da implementação deste algoritmo.

2.4 Integração MCL – BaLL

A técnica BaLL permite a obtenção de mapas de atributos de um ambiente – $P(\ f|s\ )$. No entanto, a localização Monte Carlo necessita de um modelo sensorial relacionando os atributos f aos diversos estados do ambiente, ou seja, $P\ (\ f|\xi\ )$. Caso fosse possível uma descrição completa de todos os possíveis estados ξ do ambiente, então haveria um relacionamento direto entre estes dois modelos, pelo estabelecimento de uma relação biunívoca entre se ξ.

Entretanto, deve-se lembrar que, num mesmo estado, podem-se observar diferentes leituras, devido ao não-determinismo intrínseco do funcionamento dos sensores. Além disso, um mapeamento completo do ambiente não seria interessante, devido à complexidade de realização e aos custos computacionais envolvidos. Por estes motivos, opta-se por realizar apenas um mapeamento parcial do ambiente em estados de referência, a partir dos quais o modelo $P\ (\ f|\xi\ )$ deve ser capaz de aproximar todos os demais.

Seja um modelo sensorial que se utiliza de três redes neurais, e um fragmento de mapa de um ambiente representado na Figura 6. Na parte inferior são mostrados alguns vetores simbolizando supostos atributos em estados de referência, considerando-se que toda a tarefa de obtenção do mapa de atributos tenha sido realizada, ou seja:

- Coleta prévia de dados dos sensores em estados de referência.
- Treinamento das redes neurais utilizando este repositório de dados sensoriais.
- Aplicação das redes neurais obtidas às leituras de todos os estados de referência do mapa, obtendo-se o mapa de atributos.

Sob estas condições, espera-se que o atributo observada na posição real do robô seja um vetor cujas componentes pertençam a intervalos próximos das componentes dos atributos dos estados de referência vizinhos. Ou seja, para qualquer estado do ambiente, pode-se calcular um 'atributo médio', baseada no mapa de atributos. Assim, define-se $\sigma_{i(esperado)}(\ s\ )$ o vetor obtido compondo-se as médias das componentes dos *m atributos* vizinhos:

$$\sigma_{(\text{esperado})}(s) = (g_{1(\text{esperado})}(s), \ldots, g_{n(\text{esperado})}(s)) = (\sum_{j=1}^{m} \frac{g_j(s)}{m}, \ldots, \sum_{j=1}^{m} \frac{g_j(s)}{m}) \quad (36)$$

A avaliação de amostras da crença se dá por meio da equação

$$P(f|\xi) = \Pi[1 - |g_{i(esperado)}(s) - g_{i(real)}(s)|] \quad (37)$$

que compara $\sigma_{i(esperado)}(s)$ com $\sigma_{(real)}(s)$, apresentando resultados no intervalo [0,1]. Quão mais próxima uma amostra estiver da localização real do robô, mais o resultado de (36) se aproxima da unidade. Desta forma, este modelo consegue representar uma boa medida de probabilidade de se tomar uma determinada leitura s em diversos estados do espaço de estados, traduzindo-se na informação desejada para o modelo de sensores.

Na Figura 6, **A** e **B** representam duas amostras, mostrando-se em destaque os atributos dos estados de referência vizinhos. Nas condições indicadas, obtém-se $\sigma_{esperado}$ (A) = (0.8, 0.5, 0.3) e $\sigma_{esperado}$ (B) = (0.8, 0.6, 0.5). Supondo que as saídas das redes segundo a leitura s coletada na real localização do robô sejam dadas por σ_{real} (s) = (0.8, 0.7, 0.6), obtém-se P (f| A) = 0.56 e P (f| B) = 0.81, ocorrendo portanto o reforço da crença de amostras em regiões que melhor se aproximam da verdadeira localização do robô.

3 ARRANJO EXPERIMENTAL

3.1 Plataforma de Simulação

Optou-se pela utilização do *software ARIA – ActiveMedia Robot Interface for Application*, uma interface de programação de aplicações para controle de robôs distribuída pela empresa *ActiveMedia Robotics* (ActivMedia Robots, 2001) sob termos de licença GPL (GPL, 2004).

A Figura 7 ilustra a arquitetura básica deste software. Nela, ARIA desempenha o papel de ferramenta para a criação – via linguagens de programação C / C++ – de programas-cliente que controlam um robô-servidor por meio de conexões *TTY* ou *TCP* / *IP*. O robô-servidor pode ser real ou simulado.

Apesar de prover uma série de funcionalidades de interesse para a simulação de robôs móveis, este software apresenta mal-funcionamento em alguns métodos de acesso, acarretando dificuldades para obtenção da localização real do robô. Tanto o levantamento do modelo dinâmico quanto o de sensores necessitam desta informação: o primeiro para combinar os pares < ξ_t , ξ_{t+1}> relacionados pela ação a, e o segundo para a construção de tabelas < s_t, ξ_t >, uma vez que sobre estas coleções de dados é que se faz a análise para construção dos modelos.

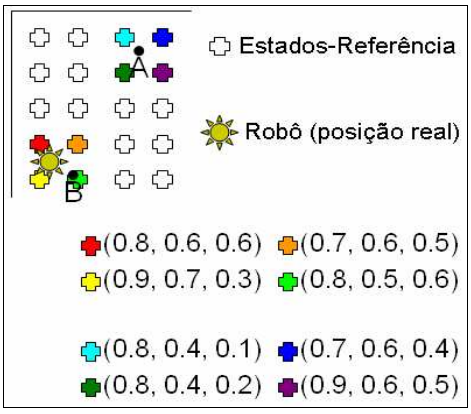

Figura 6: Fragmento do mapa de atributos de um ambiente

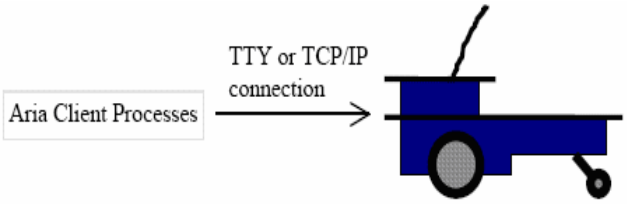

Figura 7: Arquitetura cliente-servidor de controle do robô.

A obtenção do modelo dinâmico pôde ser contornada pela reutilização de trabalho anterior (Miranda e Ribeiro, 2004). No entanto, a coleta de leituras sensoriais no ambiente de operação do robô exigiu a engenharia de um artifício para possibilitar a determinação exata dos estados onde as leituras seriam tomadas. Tal artifício será descrito em detalhes na seção 3.2.

3.2 Projeto da Simulação

3.2.1 Artifícios para Utilização do Software ARIA

Nas seções anteriores buscou-se deixar em evidência o fato de que a obtenção de modelos de sensores ocorre de forma *offline*, ou seja, a partir de repositórios de dados previamente coletados pelo robô em seu ambiente de operação. A partir destes dados, os modelos são obtidos por meio de alguma técnica de análise, automática ou não.

No entanto, como citado na seção 3.1, a informação do estado ξ não pode ser obtida por meio da interface ARIA de programação de robôs. Para possibilitar a utilização desta

plataforma de simulação, foi desenvolvido um artifício, baseado no funcionamento do simulador e em processos do Sistema Operacional, de modo a se conseguir associar os dados sensoriais às informações do estado onde foram coletados.

O funcionamento normal do simulador consiste nos seguintes passos:

1. Acionar o robô-servidor, configurado com um arquivo de parâmetros e um arquivo de ambiente.
2. Acionar o programa-cliente que controla o robô.
3. Comandar o robô (executando, por exemplo, ações de movimentação e aquisição de dados de sensores).
4. Desconectar o programa-cliente do robô.
5. Desligar o robô-servidor.

No arquivo de parâmetros constam informações de configuração do robô, por exemplo: suas dimensões, dados sobre o posicionamento de sensores, dos atuadores disponíveis, etc. No arquivo de ambiente devem estar presentes informações sobre todos os elementos que compõem o ambiente de operação do robô, sobre o sistema de coordenadas que suporta tais informações, e por fim sobre a posição inicial do robô no instante do acionamento do simulador (Figura 8).

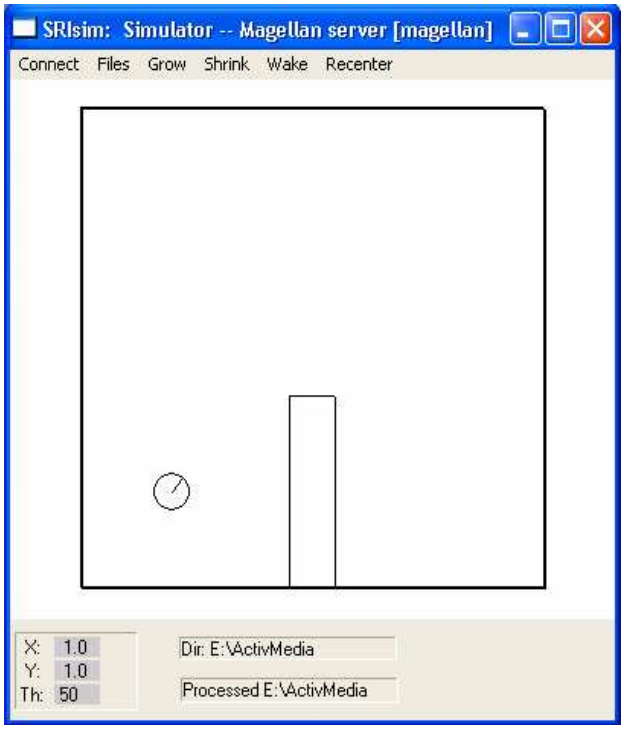

Figura 8: Simulador do Robô-Servidor. O estado inicial do robô, configurado no arquivo de ambiente, corresponde ao estado = (1000mm, 1000mm, 50^o), com origem do sistema de coordenadas no canto inferior esquerdo do ambiente.

Este último papel atribuído ao arquivo de ambiente – relativo ao posicionamento inicial do robô – permite que, pelo menos no instante do acionamento, a posição do robô esteja bem definida para a coleta do par $\langle \ s, \ \xi \ \rangle$. Esta característica abriu caminho para uma alternativa de coleta de dados de sensores: configurar a posição inicial do robô no arquivo de ambiente; iniciar a simulação; coletar as leituras neste estado e desligar o robô; repetir estes passos para todos os estados de interesse.

Apesar de aparentemente trivial, esta tarefa é impraticável de se realizar senão através de programas que iniciem automaticamente os processos relacionados aos programas envolvidos na simulação e na aquisição das leituras, devido ao número de vezes que estas etapas devem ser repetidas – mesmo num ambiente pequeno, o número de pares $\langle \ s, \ \xi \ \rangle$ necessários para análise do modelo de sensores facilmente chega à ordem de dezenas de milhar. Caso o sistema operacional utilizado seja da família Microsoft Windows, as informações necessárias para lidar com criação e tratamento de processos e *threads* podem ser encontrados em (MSDN, 2004). Utilizando-se destes recursos de sistema operacional, foi criado um programa-cliente capaz de iniciar automaticamente o processo da simulação do robô-servidor. Após ter realizado a conexão e coletado as leituras sensoriais na posição inicial, o programa termina o processo do robô, modifica o arquivo de ambiente de modo a configurar uma nova posição inicial, e repete o processo automaticamente, quantas vezes forem necessárias para a coleta de dados de sensores por todo o ambiente.

Este engenho é uma contribuição para a comunidade que utiliza o ARIA para estudo de simulações de robôs móveis, que se depara freqüentemente com o problema de levantar modelos baseados na análise de dados sensoriais. Mas apesar de contornar a dificuldade inicial, esta solução é paliativa. A análise de alternativas de simuladores que apresentem maior flexibilidade consiste numa importante tarefa para o estudo de Robótica Móvel. Por este motivo, buscou-se neste trabalho utilizar os recursos do software da *ActivMedia* por meio de um componente intermediário – denominado de IPC, ou Interface Padrão de Comunicação.

A idéia consiste em determinar os mecanismos de controle do robô por meio da IPC. Esta abordagem provê maior flexibilidade de utilização de simuladores: caso haja mudança na tecnologia de robô empregada – por exemplo, ao considerar o uso de outros tipos de simulador – basta considerar uma implementação da interface que suporte a nova tecnologia. Desta forma, os programas-cliente vêm a sofrer pouco ou nenhum impacto no caso da citada mudança.

A solução apresentada nesta seção tornou possível a aquisição de dados a partir do simulador, e de forma automática.

3.2.2 Repositório de Dados de Sensores

Solucionada a questão da disponibilidade de dados de sensores, surge a questão do armazenamento, recuperação e manipulação destes. Este requisito funcional para persistência de dados foi incorporado à arquitetura de solução na forma de uma Interface de Acesso a Dados. O estudo da melhor maneira de realizar este requisito levou à análise de alternativas de armazenamento via arquivos XML e tecnologias de bancos de dados.

Arquivos XML mostraram-se de início como atraente alternativa, por prover facilidade para serialização e desserialização de complexas estruturas de dados em arquivos de texto. No entanto, além de armazenadas, estas estruturas devem ser facilmente recuperadas e manipuladas para posterior utilização. Por este motivo, preferiu-se utilizar ferramentas de bancos de dados relacionais, as quais provêm a linguagem SQL para recuperação eficiente de dados armazenados em disco.

Apesar de não empregados no armazenamento dos dados sensoriais, os conhecimentos de XML adquiridos neste estudo foram úteis em duas outras situações: para o armazenamento das redes neurais artificiais obtidas e para manipulação do arquivo do ambiente de modo a modificar-lhe somente o parâmetro de posicionamento inicial do robô.

3.2.3 Arquitetura da Solução

Foi estruturado um desenvolvimento da solução na forma de componentes independentes e interoperáveis.

O caso da utilização do software de simulação pôde ser contornado pela criação de uma camada intermediária de software, responsável exclusivamente pelo controle do robô, visando isolar os programas-cliente – no caso deste trabalho, o Módulo de Localização Monte Carlo – da tecnologia de robô empregada. Por outro lado, buscou-se uniformizar as funcionalidades de persistência de informações por meio de uma Interface de Acesso a Dados. Tais idéias encontram-se sumarizadas no Diagrama de Componentes da Figura 9.

3.2.4 Modelagem da Solução de Software

O diagrama de classes da Figura 10 mostra um conjunto mínimo de classes que provê uma estrutura de dados de suporte à IPC, ou seja, tipos abstratos de dados necessários para a programação de um robô. A classe *AriaRobot* – onde ocorre a programação dos controles ARIA – implementa a interface *AbstractRobot*, e o polimorfismo entre elas permite que os programas-clientes possuam apenas uma referência para a

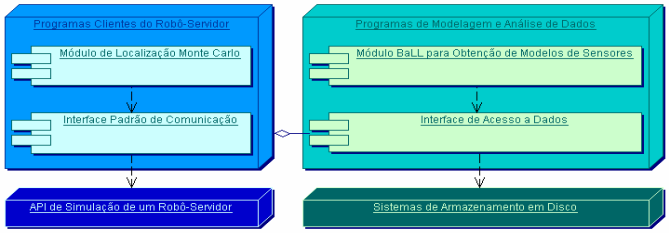

Figura 9: Arquitetura de solução para a implementação do Algoritmo BaLL e sua integração com o módulo Monte Carlo existente.

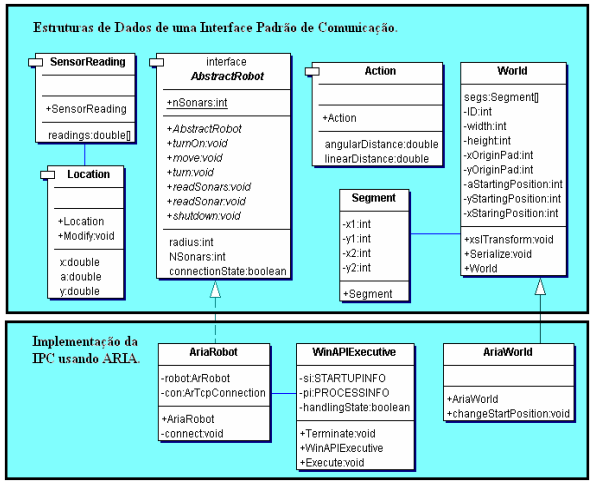

Figura 10: Projeto da IPC e do controle ARIA.

interface, não estabelecendo direta dependência em relação aos controles ARIA. A flexibilidade desta modelagem permite que outros tipos de simulador sejam facilmente agregados, bastando codificar os controles específicos em uma classe que implemente a interface *AbstractRobot*.

O diagrama de classes da Figura 11 sugere uma maneira de realizar a integração com implementações Monte Carlo. Na superclasse *MCL* são declarados métodos abstratos *MotionModel* e *PerceptualModel*, os quais devem ser implementados em subclasses que representem soluções particulares para os modelos dinâmicos e de sensores do problema da localização. A interface *Map* atua como estrutura de dados para representação dos atributos de um ambiente. *BallMCL* e *DistanceMCL* constituem exemplos de utilização da integração sugerida, diferindo entre si apenas quanto ao modelo sensorial empregado.

3.3 Soluções de Modelo Sensorial

Na Seção 2 ficou mostrado que um modelo de sensores consiste de uma representação do ambiente e de uma análise das

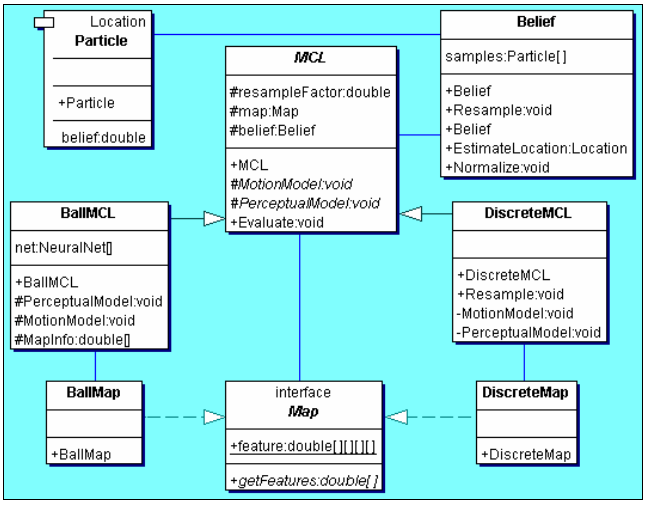

Figura 11: Módulo de localização Monte Carlo.

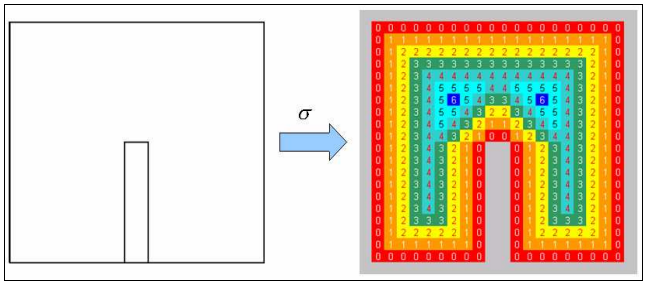

Figura 12: Escolha não-automática de *atributos* do ambiente para o modelo $P(f|\xi)$.

leituras sensoriais para avaliação da densidade $P(\ f|\xi\ )$.

Esta seção apresenta os principais aspectos de um modelo simples, obtido de forma não-automática. Em seguida, apresenta a obtenção de um modelo pelo algoritmo BaLL, bem como artifícios para possibilitar o treinamento das redes neurais em um tempo considerado aceitável.

3.3.1 Modelo Baseado em Proximidade

Uma solução para representação do mapa de atributos, consiste em descrever os estados do ambiente segundo a proximidade do robô a obstáculos de referência (Almeida e Ribeiro, 2004). Em (Miranda e Ribeiro, 2004), esta abordagem foi implementada considerando-se como referência as paredes do ambiente de operação do robô, como ilustrado na Figura 12. Para modelar o funcionamento dos sensores, foram coletadas leituras posicionando o robô a variadas distâncias de uma parede.

Sendo d a indicação do atributo no estado de realização da leitura, pôde-se observar que, a cada 100 leituras realizadas,

Tabela 1: Levantamento de dados para modelo de sensores baseado em proximidade de obstáculos do tipo parede. Os números da primeira linha (0 a 9) correspondem à observação, e os da primeira coluna são os valores reais.

	0	1	2	3	4	5	6	7	8
0	93.4	6.6	0	0	0	0	0	0	0
1	16.7	76.3	7	0	0	0	0	0	0
2	0	17.5	78.1	4.4	0	0	0	0	0
3	0	0	20.2	68.4	11.4	0	0	0	0
4	0	0	0	21.9	69.3	8.8	0	0	0
5	0	0	0	0.9	19.3	73.7	6.1	0	0
6	0	0	0	0	1.8	20.2	71.9	6.1	0
7	0	0	0	0	0	3.5	24.6	61.4	10.5
8	0	0	0	0	0	0.9	3.5	21.9	64.9

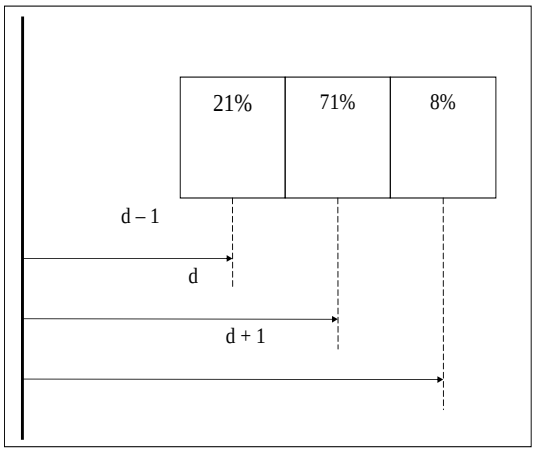

Figura 13: Modelo 21:71:8

em média 71 indicavam que o robô estaria de fato ocupando um estado cujo atributo seria d, porém 21 indicaram $d-1$e 8 indicaram $d+1$. Os dados relativos a este levantamento foram arranjados na Tabela 1, a partir da qual se pôde propor o Modelo "21:71:8", ilustrado na Figura 13.

Dispondo deste modelo, resta analisar como ocorre a inferência de $P\ (\ f\ |\xi\ )$. Através das variáveis de estado (x, y) da amostra, pode-se consultar no mapa da Figura 12 qual o atributo a ela associada. Supondo ser Δ o atributo da amostra A, e δ o atributo observada de uma leitura dos sensores, então se avalia $P\ (\ \delta\ |A\ )$ por meio da função da Figura 14.

Maiores detalhes sobre o modelo baseado em proximidade podem ser encontrados em (Almeida e Ribeiro, 2004).

$$P(\delta|A) = \begin{cases} 0.21 & , \text{se} \Delta = \delta - 1 \\ 0.71 & , \text{se} \Delta = \delta \\ 0.08 & , \text{se} \Delta = \delta + 1 \\ 0 & , \text{nos demais casos} \end{cases}$$

Figura 14: $P(f|\xi)$, segundo o modelo 21:71:8.

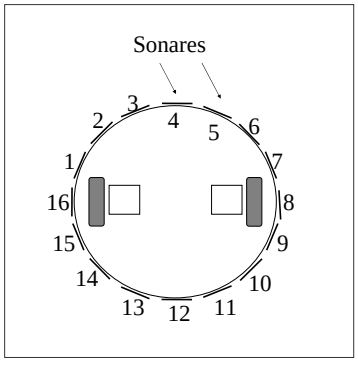

Figura 15: Distribuição dos sonares do Robô Magellan.

3.3.2 Modelo Obtido pelo Algoritmo BaLL

Diferentemente do modelo baseado em proximidade de obstáculo, cuja obtenção requer grande esforço de análise, o algoritmo *BaLL*, se propõe a obter automaticamente um mapa de atributos a partir do treinamento de redes neurais. A obtenção de dados para treinamento das redes e a execução dos algoritmos para cálculo das equações (32) e (34) são o objeto de estudo a seguir.

Como preparação para o algoritmo, criou-se um banco de dados (Tabela 2) para armazenar registros de leituras sensoriais coletadas pelo robô no ambiente representado na Figura 8. Coletou-se um total de 300.000 leituras para variadas posturas, em estados distantes entre si por 5 cm, com 36 leituras realizadas para cada posição (x,y) – uma a cada 10^o de giro do robô em relação a um sistema global de coordenadas. O robô utilizado foi o Magellan Pro^{TM}(IS Robotics, 2000), que possui 16 sonares, distribuídos de maneira simétrica, 8 na parte frontal e 8 na parte traseira, conforme mostra a Figura 15.

O algoritmo *BaLL* acessa este repositório para obtenção de distribuições *Bel*(ξ) de treinamento. Considerou-se uma distribuição uniforme, a exemplo dos testes que foram propostos em Thrun (1997). O tamanho da distribuição depende da incerteza de treinamento, ou seja, do tamanho da região considerada aceitável para concentrar as amostras das crenças de modo a propiciar a localização do robô.

Numa determinada iteração, constrói-se o conjunto de treinamento de acordo com esta incerteza, pois serão atribuídas crenças não nulas apenas dentro da região da distribuição *Bel*(ξ). Em seguida, calcula-se, a partir do conjunto obtido, cada um dos termos que compõem as equações (32) e (34).

Computados estes termos parciais, são executados os fluxogramas das Figuras 16 e 17, a partir dos quais se obtém as medidas de erro posterior e os gradientes de aprendizado.

O termo ContribF_ABC presente na Figura 17 diz respeito ao termo da equação de cálculo de gradientes transcrito na expressão:

$$\begin{aligned} &\text{ContribF\_ABC} = \\ &\sum_{f_1=0}^{1} \sum_{f_2=0}^{1} \cdots \sum_{f_n=0}^{1} \prod_{j \neq i} P\left(f_j|s^*\right) P\left(f_j|s\right) \left(2\delta_{f_i,1} - 1\right) \cdot \\ &\left[\frac{\delta_{\xi_i^*,\bar{\xi}} P\left(f_i|s\right) + \delta_{\xi_i,\bar{\xi}} P\left(f_i|s^*\right)}{\sum\limits_{\langle \tilde{s},\tilde{\xi} \rangle \in X} \prod\limits_{j=i} P\left(f_j|\tilde{s}\right)} - \right. \\ &\left. \frac{P\left(f_i|s^*\right) P\left(f_i|s\right) \prod\limits_{j \neq i} P\left(f_j|\bar{s}\right) Bel_{\text{prior}}\left(\bar{\xi}\right)}{\left(\sum\limits_{\langle \tilde{s},\tilde{\xi} \rangle \in X} \prod\limits_{j=i} P\left(f_j|\tilde{s}\right) \right)^2} \right] \end{aligned} \quad (38)$$

O índice i destaca os termos cuja avaliação ocorre na iteração de cálculo relacionada à rede i.

Treinamento das Redes

A Figura 18 apresenta o fluxograma do algoritmo BaLL (ver Figura 5). Na prática, o cálculo do erro posterior mostrou-se necessário apenas para análise e monitoramento da evolução das redes neurais. Por outro lado, o tempo de cálculo dos gradientes cresce numa potência cúbica com o aumento do tamanho (número de amostras) da distribuição, inviabilizando treinamento para maiores distribuições. Estes fatos levaram a duas decisões:

1. Monitorar o erro a cada 250 iterações e utilizar um único cálculo de gradientes para realizar 25 atualizações das redes.
2. Seccionar o ambiente em porções menores, treinando vários conjuntos de redes neurais de acordo com referências observáveis nestas seções (presença de paredes, cantos, corredores ou quinas).

Apesar de a última decisão inserir uma etapa não-automática no processo, sua análise e execução é relativamente simples

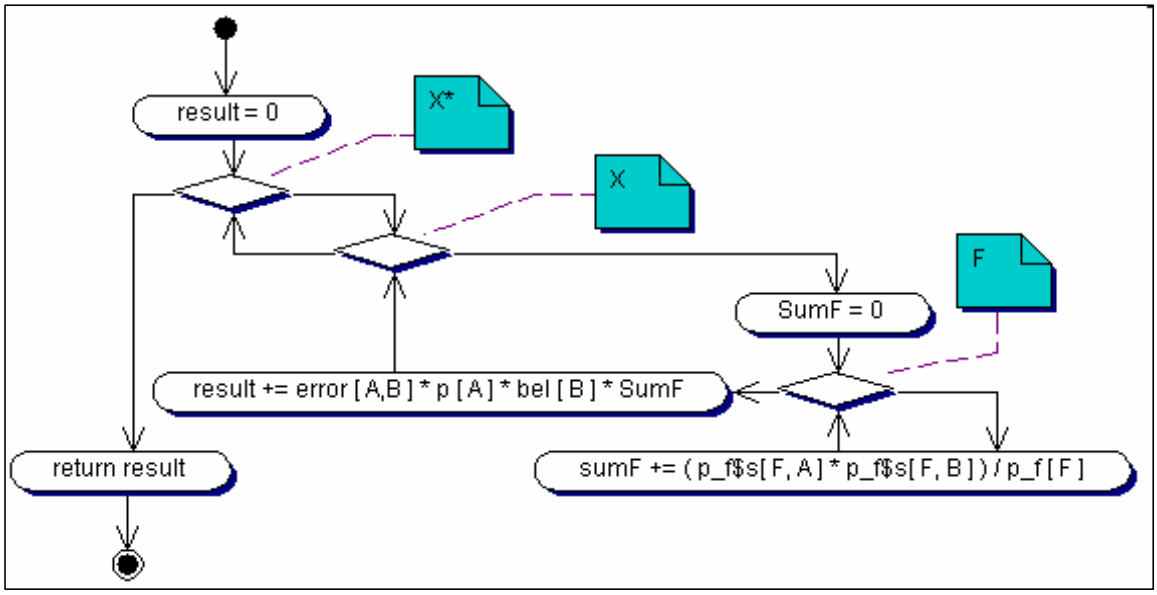

Figura 16: Fluxograma para cálculo do erro posterior.

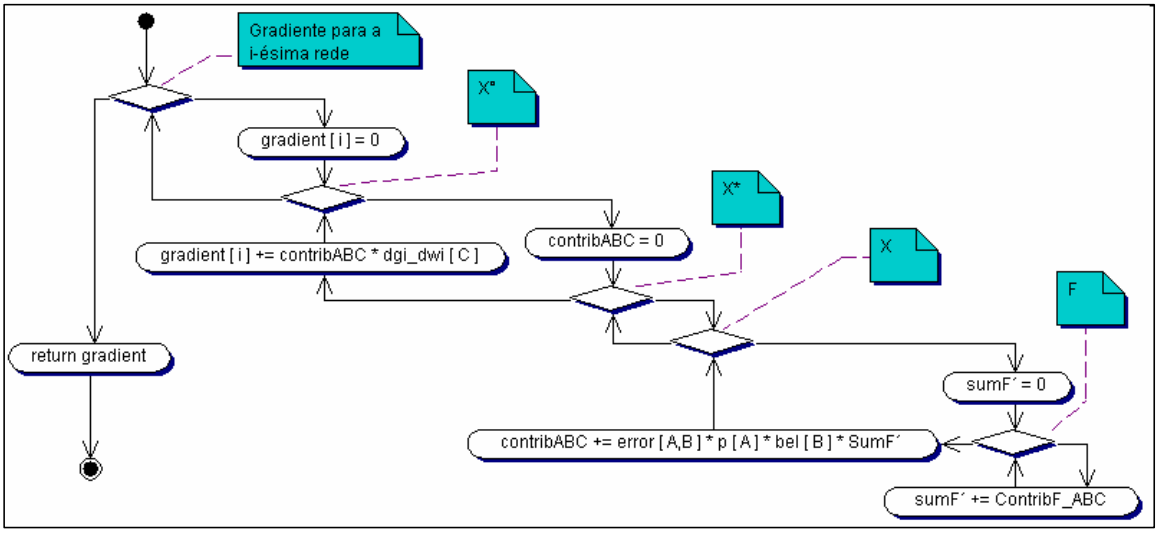

Figura 17: Fluxograma para cálculo dos gradientes das redes neurais.

– a Figura 19 mostra um exemplo aplicado ao ambiente de testes utilizado neste trabalho – e viabiliza o algoritmo em termos de tempo de treinamento.

Resta destacar que o treinamento das redes baseou-se na técnica de validação cruzada (Haykin, 1999) para garantir uma avaliação adequada das redes. Para realizar esta análise, os dados do semi-espaço direito do ambiente foram utilizados para treinamento, e os dados do semi-espaço esquerdo para a validação (monitoramento da medida de erro posterior).

4 TESTES E ANÁLISE DOS RESULTADOS

Nesta seção são apresentados os resultados obtidos para execução do treinamento das redes do mapa de atributos obtido pelo algoritmo BaLL.

4.1 Aspectos Preliminares

Normalização das Entradas: As 16 leituras dos sensores foram mapeadas no intervalo [0,1] antes de aplicá-las como

Tabela 2: Estruturas de dados utilizadas para o cálculo das equações (32) e (34) do algoritmo BaLL.

Estrutura	Informação Armazenada na Estrutura
Error[A,B]	Métrica de distância entre 2 estados A e B do ambiente.
p[A]	Probabilidade do estado A representar a verdadeira localização do robô, dada a distribuição bel[A].
Bel[A]	Crença ou Densidade de Probabilidade de Treinamento do estado A.
p_fi$s[N,A,{0,1}]	Probabilidades Condicionais da Característica Mapeada pela N-ésima rede, condicionada às leituras do estado A.
dgi_dwi[N,A]	Derivada sigmoidal da saída da N-ésima rede, tendo como entradas as leituras sensoriais do estado A.
p_f$s[F,A]	Probabilidade Conjunta da Atributo F, condicionada aoestado A.
sum_pf[F]	Soma das probabilidades conjuntas da Atributo F.
qsum_pf[F]	Quadrado de sum_pf[F].

entradas nas redes por meio da transformação

$$\text{Input[i]} = (\text{máximo} - \text{leitura[i]})/\text{máximo}$$

onde "máximo" indica o valor da maior leitura que pode ser lida pelos sensores do robô. Desta forma, leituras pouco significativas (longe de obstáculos) são transformadas em entradas de pequena magnitude, enquanto as leituras que indicam proximidade de obstáculos são transformadas em entradas próximas da unidade. A vantagem da normalização consiste no fato de ela possibilitar que os pesos sejam atualizados proporcionalmente à relevância de uma leitura.

Distribuição de Treinamento: Todas as distribuições de treinamento utilizadas nos testes foram uniformes e de mesmo tamanho. Nestas condições, o termo

$$\sum_{\langle s^*,\xi^*\rangle \in X} \sum_{\langle s,\xi\rangle \in X} e\left(\xi^*,\xi\right) Bel_{\text{prior}}\left(\xi\right) P\left(\xi^*\right) \qquad (39)$$

contribui para a magnitude do erro em um viés multiplicador fixo (ver Equação 32), dependente da métrica utilizada no cálculo de e(A,B) e da distribuição de treinamento *Bel*(ξ). Nos gráficos de análise dos testes do algoritmo BaLL, tal viés foi descontado para melhor visualização dos resultados.

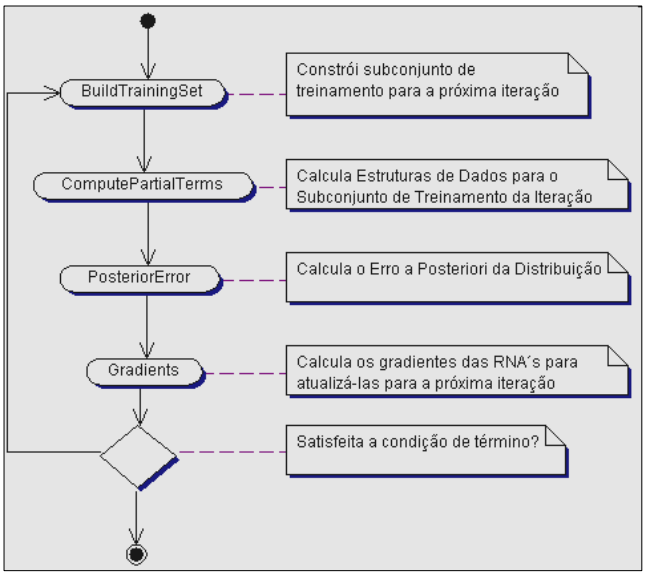

Figura 18: Seqüência de execução da rotina de aprendizado das redes.

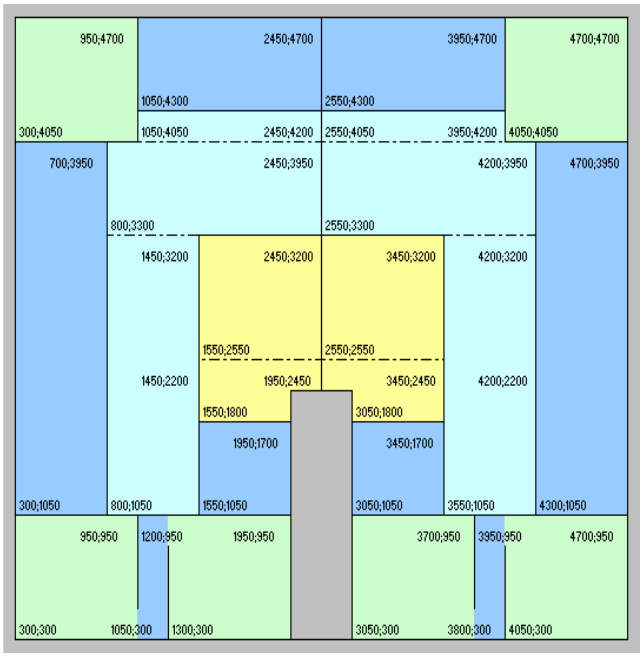

Figura 19: Pré-classificação de estados do ambiente baseada na presença de tipos de obstáculos pré-determinados (cantos, paredes, corredores e quinas). Os números indicam os limites de cada região no sistema de coordenadas do ambiente, medidos em mm.

Arquitetura das Redes Neurais: a investigação do comportamento do algoritmo foi realizada empregando-se redes neurais artificiais de três camadas, de acordo com as arquiteturas indicadas na Tabela 3.

Tabela 3: Arquiteturas das redes utilizadas.

Nome da Rede	Número de Neurônios		
	1^a Camada	2^a Camada	3^a Camada
32_16	32	16	1
48_24	48	24	1
64_32	64	32	1
96_48	96	48	1

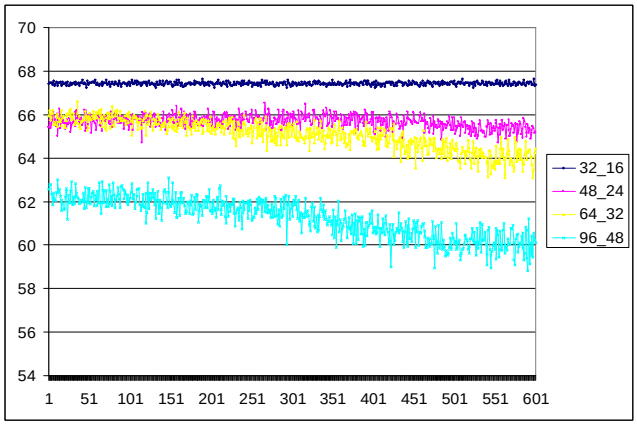

Figura 20: Treinamento para seções do tipo parede. Neste treinamento, as redes “96:48:1"apresentaram acentuada minimização do erro bayesiano.

4.2 Análise do Treinamento das redes

Os gráficos das Figuras 20, 21 e 22 ilustram a evolução da medida do erro bayesiano de localização ao longo do treinamento das redes neurais. Foram realizados 150.000 passos de treinamento, com uma medida de gradiente servindo para 25 passos de atualizações sucessivas dos pesos das. O erro foi calculado a cada 250 iterações no conjunto de validação (região direita do ambiente). O eixo vertical indica a medida de erro, e o eixo horizontal o número de passos de treinamento. Cada unidade neste eixo corresponde a 250 passos de atualização de pesos. Todos os valores da medida de erro foram descontados do viés da distribuição, constante e igual a 100,400748.

4.3 Conclusões sobre o Algoritmo BaLL

A investigação realizada neste trabalho considerou apenas quatro tipos de arquitetura de redes neurais devido ao elevado custo computacional de treinamento. Cada um dos testes mostrados levou pelo menos 5 horas (alguns chegando a demorar o dobro) para serem realizados em computadores *Athlon 2.0 GHz*, com 1 *GB* de *RAM*. De fato, a fase de treinamento do algoritmo BaLL tem complexidade O ($N2^nK^3$ + *NKnW*), onde N é o número de iterações, n é o número de redes neurais, K é o tamanho do conjunto de treinamento e

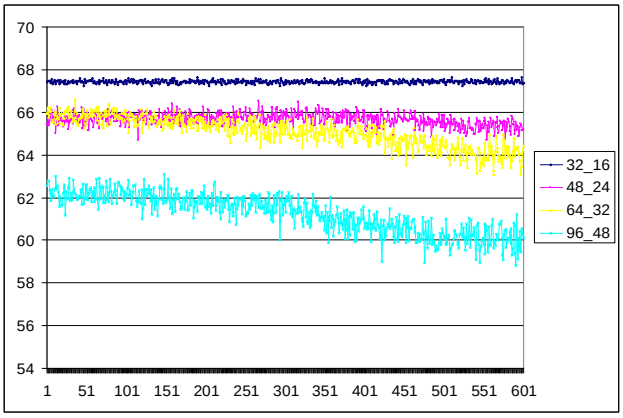

Figura 21: Treinamento para seções do tipo canto. Neste treinamento, as redes apresentaram a minimização do erro bayseano com a arquitetura “96:48:1"proporcionando os melhores resultados.

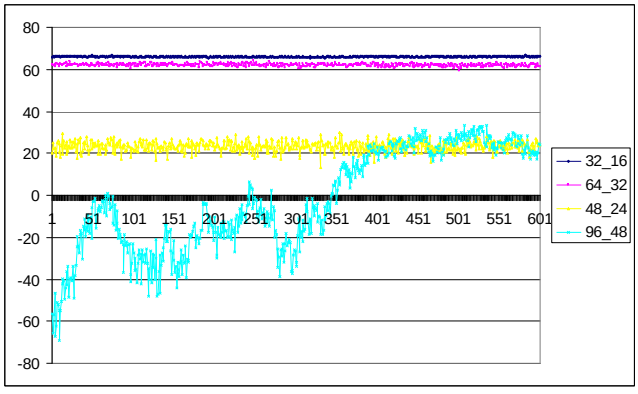

Figura 22: Treinamento para seções do tipo quina. A evolução do comportamento da medida de erro apresentou um padrão bastante irregular.

W é o número de pesos na rede (Thrun, 1997). Uma possibilidade para redução da complexidade é o treinamento sequencial (e não paralelo) das redes, de modo similar ao que ocorre de modo incremental em algoritmos do tipo CasCor (Fahlman e Lebiere, 1990).

Ainda assim, com apenas alguns testes preliminares, pôde-se observar situações em que o treinamento das redes neurais proporciona acentuada redução do erro bayesiano de localização, característica desejada para a obtenção do modelo de sensores. Isto em si aponta para a necessidade de mais investigações sobre o algoritmo, inclusive para a verificação da existência de redes neurais adequadas para os casos onde a minimização do erro não pôde ser observada nas arquiteturas testadas.

No que se refere à extração de atributos para diferentes tipos

de seções do ambiente, observou-se um desempenho pior em regiões do tipo quina. Tal situação é típica para sensores do tipo sonar, devido aos problemas de *crosstalk* e reflexão especular produzido pelas quinas (Nehmzow, 2000).

5 CONCLUSÕES

Este trabalho tratou de uma série de aspectos relacionados ao desenvolvimento de uma solução para algoritmos de localização Monte Carlo, focando na análise de uma solução para o algoritmo *BaLL* para obtenção automática de um modelo de sensores de um robô simulado no software *ARIA*.

Desenvolvida em linguagem *C++.NET*, compatível com o simulador utilizado, buscou-se empregar o paradigma de Orientação a Objetos e de ferramentas *CASE* para modelagem da solução proposta. A aplicação destes conceitos e ferramentas propiciou o desenvolvimento de um modelo relativamente simples para a integração das soluções *MCL* e *BaLL*, produtos principais deste trabalho.

Foi proposto o projeto e implementada uma camada intermediária de *software* isolando as funcionalidades do módulo de localização das características de programação do simulador utilizado. Este tratamento pôde proporcionar maior independência, usabilidade e tratabilidade sobre o *software* de simulação. Estas características são desejáveis devido a algumas limitações apresentadas pelo simulador durante o desenvolvimento do trabalho.

A principal das limitações residiu na dificuldade que ele impõe para coleta de dados dependentes da localização real do robô simulado, necessários para o levantamento dos modelos dinâmico e sensorial empregados no módulo *MCL*. No caso da aquisição de dados para construção de um banco de dados de leituras de sensores, requisito funcional do algoritmo *BaLL*, tais dificuldades puderam ser contornadas por uma solução paliativa, a qual se aproveita de algumas características intrínsecas da plataforma *ARIA* e da utilização de processos e *threads* do sistema operacional *Microsoft Windows*.

Por meio desta solução, pôde-se compor um banco de dados contendo cerca de 300.000 registros de leituras de sensores coletadas em um ambiente de testes, uma tarefa que demorou 19 dias para ser completada e cujo produto poderá ser empregado como base para trabalhos futuros de análise de modelos de sensores.

A técnica *BaLL* revela-se uma interessante alternativa para obtenção de modelos de sensores a serem utilizados em implementações do tipo Monte Carlo. Os resultados dos testes confirmam o sucesso da técnica e da solução em testes preliminares, nos quais pôde-se observar situações em que o treinamento das redes neurais proporciona acentuada redução do erro bayesiano de localização, característica desejada para a obtenção do modelo de sensores. Isto em si aponta para a necessidade de mais investigações sobre o algoritmo, inclusive para a verificação da existência de redes neurais adequadas para os casos onde a minimização do erro não pôde ser observada.

No que se refere à extração de atributos para diferentes tipos de seções do ambiente, observou-se um desempenho pior em regiões do tipo quina, devido aos problemas de crosstalk e reflexão especular produzido por estas em sonares. Soluções para estes problemas implicam modificações de hardware através de configurações alternativas (Stanley e McKerrow, 1997) ou alterações de firmware exigindo acesso ao sistema operacional que controla a emissão dos pulsos de som (Kleeman e Kuc, 1994).

O algoritmo BaLL foi originalmente proposto em (Thrun, 1997), que relata testes em um robô do tipo B21 equipado com anel de sonares e câmera colorida. Os resultados aqui relatados envolvem uma classe diferente de robôs (Magellan), equipados apenas com anel de sonares. É interessante observar que, mesmo neste último caso, o algoritmo fez a identificação correta dos tipos de atributos relevantes para navegação, à exceção das quinas. Na proposta original, os resultados do algoritmo BaLL evidenciaram uma superioridade dos atributos extraídos *vis-a-vis* o problema de localização, comparativamente a técnicas de localização baseadas em atributos pré-definidos. Neste sentido, possíveis desenvolvimentos futuros podem incluir uma comparação entre a localização baseada no algoritmo BaLL e a localização baseada em atributos pré-definidos, utilizando-se exclusivamente sonares ou outro tipo de sensor. Como o algoritmo – especialmente em sua fase de treinamento – é muito custoso computacionalmente, tal estudo pode ajudar a identificar que atributos podem ser pré-definidos e quais devem ser extraídos automaticamente, considerando as especificidades dos sensores considerados.

Do ponto de vista de implementação, a integração entre o módulo MCL e o modelo de sensores baseado nas redes neurais obtido pelo algoritmo BaLL é uma sugestão para trabalhos futuros. O presente estudo também abre caminho para uma série de outras propostas e investigações:

- descobrir as configurações das melhores redes neurais;
- considerar ambientes mais complexos;
- testar utilizando outros simuladores;
- aplicar em robôs reais;
- utilizar os módulos de localização desenvolvidos como base para a navegação de robôs em tarefas de maior complexidade;

- integrar o método BaLL ao algoritmo MCL de localização e estudar outros problemas qu podem ser baseados em métodos Monte Carlo, por exemplo, o problema da localização aplicado a times de robôs;
- aproveitar a modelagem realizada para portar o código para projetos *open-source*.

REFERÊNCIAS

ActivMedia Robots (2001). Saphira 6.1, Saphira 8.0 e Aria 1.1. Manuais de Referência.

Almeida, L. F. e Ribeiro, C.H.C. (2004). Localização Monte Carlo para robôs móveis baseada em ponderação de fatores de importância ao longo do tempo. In: Anais do XV Congresso da Sociedade Brasileira de Automática, Gramado, RS.

Chown, E., Kaplan, S. e Kortenkamp, D. (1995). Prototypes, location and associative networks (PLAN): Towards a unified theory of of cognitive mapping. *Cognitive Science*, v. 19, p. 1-51.

Doucet, A., Freitas, N. e Gordon, N. (2001). Sequential Monte Carlo Methods in Practice. Springer-Verlag.

Fahlman, S. e Lebiere, C. (1990). Cascade-correlation learning architecture. In: D. Touretzky (Ed.), Advances in Neural Information Processing Systems. Morgan Kaufmann, San Mateo, v. 2, p. 524-532.

Fox, D., Burgard, W. E Thrun, S. (1999). Markov Localization for Mobile Robots in Dynamic Environments. *Journal of Artificial Intelligence Research*, v. 11, p. 391-427.

GPL. (2004). GNU General Public License. GNU Project, Free Software Foundation. http://www.gnu.org.

Greiner, R. e Isukapalli, R. (1994). Learning to select useful landmarks. Procs. Of National Conference on Artificial Intelligence, pp. 1251-1256. AAAI Press / The MIT Press.

Haykin, S. (1999). Neural Networks: A Comprehensive Foundation. Prentice-Hall.

IS Robotics (2000). IS ROBOTICS, Inc. *Magellan Pro Compact Mobile Robot User's Guide*. USA: Real World Interface Division.

Jensfelt, P. e Kristensen, S. (1999). Active global localisation for a mobile robot using multiple hypothesis tracking. Procs. of the IJCAI-99 Workshop on Reasoning with Uncertainty in Robot Navigation.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, v. 32, p. 35-45.

Kleeman, L. e Kuc, R. (1994). An optimal sonar array for target localization and classification. Procs. of the IEEE International Conference on Robotics and Automation, p. 3130-3135.

Miranda, F.L.N. e Ribeiro, C.H.C. (2004). Análise do Desempenho de um Algoritmo de Localização Monte Carlo para Robôs Móveis. Relatório de Iniciação Científica. FAPESP, proc. no. 02/10825-2.

MSDN. (2004). Platform SDK: DLLs,Processes,and Threads.http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/startupinfo_str.asp.

Murphy, R. R. (2000). Introduction to AI Robotics. MIT Press.

Nehmzow, U. (2000). Mobile Robotics: A Practical Introduction. Springer.

Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kauffman Publishers.

Ribeiro, C.H.C., Costa, A.H.R., Romero, R.A.F. (2001). Robôs Móveis Inteligentes: Princípios e Técnicas. In: Martins, Ana Teresa; Borges, Díbio Leandro. (Org.). Anais do XXI Congresso da Sociedade Brasileira de Computação. 2001, v. 3, p. 257-306.

Smith, R., Self, M, Cheeseman, P. (1990). Estimating uncertain spatial relationship in robotics. In: Cox, I.J., Wilfong, G.T. (Eds), Autonomous robot vehicles, New York: Springer Verlag, p. 167-193.

Stanley, B. e McKerrow, L. (1997). Measuring range and bearing with a binaural ultrasonic sensor. Proc. IEEE/RSJ IROS, vol. 2, pp. 565-571.

Thrun, S., Fox, D., Burgard, W. e Dellaert, F. (2001). Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, n.128, pp.99-141.

Thrun, S. (1997). Bayesian Landmark Learning for Mobile Robot Localization. *Machine Learning*, p. 1-32. abr. 1997.

Weisstein, E.W. (2004) Monte Carlo Method. MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/MonteCarloMethod.html.

Wittwer, J.W. (2004). Monte Carlo Simulation Basics. Vertex42.com.http://vertex42.com/ExcelArticles/mc/MonteCarloSimulation.html.