# APRENDIZADO DA COORDENAÇÃO DE COMPORTAMENTOS PRIMITIVOS PARA ROBÔS MÓVEIS

**Antonio Henrique Pinto Selvatici***
antonio.selvatici@poli.usp.br

**Anna Helena Reali Costa***
anna.reali@poli.usp.br

*Laboratório de Técnicas Inteligentes – LTI
Escola Politécnica da Universidade de São Paulo – EPUSP
Av. Prof. Luciano Gualberto, trav.3, n.158
Cidade Universitária – 05508-900
São Paulo-SP, Brasil

## ABSTRACT

In most real world applications, mobile robots should perform their tasks in previously unknown environments. Thus, a mobile robot architecture capable of adaptation is very suitable. This work presents an adaptive architecture for mobile robots, AAREACT, which has the ability of learning how to coordinate primitive behaviors codified by the Potential Fields method by using Reinforcement Learning. The proposed architecture's performance is compared to that showed by an architecture that performs a fixed coordination of its behaviors, and shows a better performance for different environments. Experiments were performed in the robot Pioneer's simulator, from ActivMedia Robotics®. The obtained results also suggest that AAREACT has good adaptation skills for specific environment and task.

**KEYWORDS**: Mobile robotics, control architecture, reactive behaviors, reinforcement learning.

ARTIGO CONVIDADO:
Versão completa e revisada de artigo apresentado no SBAI-2005
Artigo submetido em 01/06/2006
1a. Revisão em 22/08/2006
Aceito sob recomendação do Editor Convidado
Prof. Osvaldo Ronald Saavedra Mendez

## RESUMO

Para ter uma aplicação real, um robô móvel deve poder desempenhar sua tarefa em ambientes desconhecidos. Uma arquitetura para robôs móveis que se adapte ao meio em que o robô se encontra é então desejável. Este trabalho apresenta uma arquitetura adaptativa para robôs móveis, de nome AAREACT, que aprende como coordenar comportamentos primitivos codificados por Campos Potenciais através de Aprendizado por Reforço. A atuação da arquitetura proposta, após uma fase de aprendizado inicial, é comparada com a apresentada por uma arquitetura com coordenação fixa dos comportamentos, demonstrando melhor desempenho para diversos ambientes. Os experimentos foram realizados no simulador do robô Pioneer, da ActivMedia Robotics®. Os resultados experimentais obtidos neste trabalho apontam também a alta capacidade de adaptação da arquitetura AAREACT para um ambiente e tarefa especificos.

**PALAVRAS-CHAVE**: Robótica móvel, arquitetura de controle, comportamentos reativos, aprendizado por reforço.

## 1 INTRODUÇÃO

Define-se um robô móvel inteligente como um agente inteligente, artificial, autônomo, com capacidade de locomoção, imerso no mundo físico real. *Agente inteligente* por decidir de forma racional; *artificial* por ser máquina e não uma entidade criada pela natureza; *autônomo* por ser capaz de decidir

por si só, de se auto-governar, de atuar no ambiente de forma propositada, não passiva, de se adaptar a mudanças ocorridas, no ambiente ou em si próprio, e continuar a atingir suas metas; *com capacidade de locomoção* por poder se mover no ambiente. (Ribeiro et al., 2001; Murphy, 2000).

A execução da tarefa atribuída a um robô móvel inteligente, ou simplesmente robô móvel, inclui necessariamente a sua locomoção de forma autônoma até localidades no espaço diferentes de sua posição inicial. É de interesse que essa locomoção possa ser feita em ambientes não estruturados, onde há, possivelmente, a presença de objetos obstruindo e dificultando a navegação. Por isso um robô móvel deve estar bem adaptado ao seu ambiente para que obtenha sucesso no cumprimento de sua tarefa. Em muitas situações, é necessário que ele possa se adaptar a novas condições, o que requer que seu sistema de controle tenha a capacidade de aprendizado. Ele pode, assim, observar e criticar sua própria atuação, julgando-a segundo alguma medida de utilidade, e ajustar seus parâmetros no intuito de melhorá-la. Russel and Norvig (2003) apresentam a idéia do agente aprendiz, cuja arquitetura incorpora um agente não aprendiz, que tem seus parâmetros modificados por um elemento de aprendizado.

Devido à natureza autônoma dos robôs móveis, é mais interessante que o aprendizado a ser executado por eles não seja supervisionado. Dessa forma, o robô aprende sem a necessidade de um tutor, podendo adaptar-se automaticamente a mudanças no ambiente. Aprendizado por Reforço (AR) (Kaelbling et al., 1996) mostra-se assim bastante promissor para a utilização em arquiteturas de robôs móveis. De fato, AR é um dos tipos de aprendizado mais largamente utilizados em sistemas robóticos para a adaptação do sistema de controle (Arkin, 1998).

Este trabalho propõe uma arquitetura para robôs móveis, de nome AAREACT, inspirada na idéia do agente aprendiz. Ela utiliza como base uma arquitetura baseada em comportamentos reativos codificados por campos potenciais, de nome REACT. Uma camada de aprendizado por reforço baseado em módulos (Kalmár et al., 1998), que constitui o elemento de aprendizado, supervisiona o desempenho do robô e ajusta a contribuição de cada comportamento na sua atuação final.

Neste trabalho, a seção 2 contextualiza a arquitetura proposta, relacionando-a com outros trabalhos. A seção 3 apresenta os conceitos fundamentais do aprendizado que foi empregado na arquitetura. A seção 4 descreve como é a plataforma robótica adotada neste trabalho. Então, a seção 5 descreve a arquitetura REACT, que serve de base para a arquitetura proposta, e a seção 6 descreve os incrementos realizados na arquitetura original de modo a dotá-la da capacidade de aprendizado. Os resultados obtidos com a arquitetura AAREACT são discutidos na seção 7. Finalmente, a seção 8 traz as conclusões deste trabalho, bem como sugestões para trabalhos futuros.

## 2 TRABALHOS CORRELATOS

REACT é uma arquitetura reativa aplicada à navegação de robôs móveis, projetada com base em comportamentos primitivos coordenados de forma cooperativa (Selvatici and Costa, 2004; Pacheco and Costa, 2002). Dessa forma, as ações comandadas por cada comportamento contribuem na geração da ação final. Os comportamentos são codificados com o método dos Campos Potenciais, gerando forças, que incindem sobre o robô, baseadas em alguma função potencial que associa cargas repulsivas aos obstáculos detectados e cargas atrativas à posição objetivo (Arkin, 1998). Assim, a REACT é de simples implementação e demanda pouco esforço computacional, sendo muito adequada para o controle do robô em tempo real. Uma vez que a codificação dos comportamentos é contínua, o movimento do robô é mais suave, apresentando poucas frenagens ou guinadas bruscas. Além disso, devido à alta modularidade da abordagem comportamental, é relativamente fácil dotar o robô de outras capacidades, bastando, para isso, acrescentar novos comportamentos.

Apesar da REACT funcionar bem para diversos ambientes, ela possui os problemas intrínsecos das arquiteturas que utilizam o método de Campos Potenciais. A literatura tradicionalmente aponta duas principais falhas nessas arquiteturas (Koren and Borenstein, 1991). A primeira é a possibilidade da existência de mínimos locais do campo potencial gerado pelos comportamentos, configurando regiões de atração indevidas para onde o robô pode se dirigir e permanecer. A outra falha é a forte oscilação na trajetória apresentada pelo robô quando passa dentro de corredores estreitos, devido à rápida alternância de direção do campo força resultante experimentada pelo robô. Esse efeito é em parte atenuado pelo comportamento *moveAhead* na arquitetura REACT. Esses problemas ocorrem porque o campo gerado pelos comportamentos nessas arquiteturas é fixo. Desse modo, caso o campo resultante possua regiões de atração diferentes do objetivo ou ainda regiões de oscilação, nada impede o robô de entrar nessas regiões.

Borenstein and Koren (1990) propuseram uma arquitetura para navegação de robôs móveis em ambientes com obstáculos baseada na análise do mapa do ambiente, representado por uma grade de ocupação (Elfes, 1989). O vetor de movimento é determinado a partir de um histograma polar de valores de ocupação, de forma que sempre aponta para a direção com o melhor compromisso entre baixa probabilidade de obstáculos e alto alinhamento com a direção do alvo. No entanto, devido à falta de modularidade dessa arquitetura, ela foge da abordagem comportamental, o que torna difícil adaptar capacidades adicionais ao robô, além do desvio de obstá-

culos e do movimento em direção ao objetivo.

Para manter a abordagem comportamental e aproveitar as vantagens do método de campos potenciais, Ranganathan and Koenig (2003) propuseram o acréscimo de uma camada de planejamento e seqüenciamento de comportamentos de forma a coordená-los de forma mais eficiente. Ela funciona como supervisora da arquitetura: quando é detectada uma situação em que o robô não consegue sair de uma região de atração indevida, a camada de planejamento e seqüenciamento determina uma posição alvo intermediária, que se encontre no caminho até o objetivo final, e a informa ao comportamento de ir até o alvo, *move-to-goal*. Assim fica caracterizada uma mudança no valor de um parâmetro desse comportamento através de planejamento na tentativa de adaptar a arquitetura às condições ambientais.

Kalmár et al. (1998) também utilizaram uma arquitetura comportamental que executa planejamento e seqüenciamento sobre os comportamentos. Uma função de seleção utiliza aprendizado por reforço baseado em módulos na escolha do comportamento mais adequado para cada possível situação encontrada pelo robô, configurando, assim, uma coordenação competitiva dos comportamentos, ou seja, apenas um deles está ativo cada vez. Os comportamentos são definidos através de uma decomposição da tarefa que o robô deve executar em sub-tarefas mais simples, de forma que cada comportamento é projetado para executar uma determinada subtarefa. A vantagem dessa arquitetura é que o aprendizado baseia-se na própria experiência do robô, o que caracteriza um maior grau de autonomia.

A arquitetura aprendiz proposta neste trabalho também emprega AR para coordenar comportamentos primitivos. De forma semelhante a Kalmár et al. (1998), a AAREACT procura fazer com que a coordenação dos comportamentos se adapte às situações encontradas pelo robô. Uma camada de aprendizado atua sobre parâmetros que influenciam a coordenação dos comportamentos com base no conhecimento adquirido pela experiência do robô. Vale ressaltar que, neste trabalho, utiliza-se uma coordenação cooperativa de comportamentos, na qual mais de um comportamento contribui para a atuação final do robô, de forma a guiá-lo entre os obstáculos até chegar a uma posição alvo. Uma vez que a participação de cada comportamento muda conforme a situação do ambiente, o campo potencial gerado não é mais fixo, evitando situações de mínimos locais permanentes ou que levariam a trajetórias oscilantes.

# 3 APRENDIZADO POR REFORÇO

AR permite que um agente aprenda uma política de atuação na base de interações do tipo tentativa e erro com um ambiente dinâmico (Kaelbling et al., 1996). Para ambientes estacionários, a teoria de processos markovianos de decisão — em inglês, *Markov Decision Processes*, ou MDPs — garante um tratamento matemático adequado. De fato, a maior parte dos trabalhos em AR baseiam-se em MDPs, embora os conceitos envolvidos possam ser aplicados de forma mais genérica (Monteiro, 2002).

## 3.1 Processos markovianos de decisão (MDPs)

Processos markovianos de decisão são processos a tempo discreto onde um agente deve decidir que ação tomar dado que o ambiente se encontra no estado $s$ e satisfaz a condição de Markov. A condição de Markov diz que o estado corrente do ambiente resume o passado de forma compacta, de modo que estados futuros não dependem de estados anteriores caso se conheça o estado corrente. Isto é, pode-se predizer qual será o próximo estado dado o estado corrente e a ação a ser tomada (Sutton and Barto, 1998).

Formalmente, um MDP consiste de:

- um conjunto discreto de $N_s$ estados do ambiente $S = \{s_1, s_2, \ldots, s_{N_s}\}$;
- um conjunto discreto de $N_a$ possíveis ações $A = \{a_1, a_2, \ldots, a_{N_a}\}$;
- uma função de probabilidade de transição $P(s'|s, a)$, que determina a probabilidade de se ir para o estado $s'$ dado que se está no estado $s$ e é tomada a ação $a$;
- funções de reforço associadas à ação $a$ tomada no estado $s$, $r(s, a) \in R$, onde $R$ é um conjunto de funções $r : S \times A \rightarrow \mathbb{R}$ que podem não ser deterministas.

Uma política $\pi$, em um MDP, é aquela que relaciona um estado a uma ação a ser tomada. Diz-se que $\pi^*$ é a política ótima quando ela garante a maior soma dos reforços a longo prazo, representada por um modelo de otimalidade para a atuação do agente. Uma possibilidade é o modelo de horizonte infinito com desconto, que leva em consideração a recompensa a longo prazo, mas as recompensas que são recebidas no futuro são descontadas de acordo com um fator de desconto $\gamma$, com $0 < \gamma < 1$:

$$R_T = E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| \pi\right]. \quad (1)$$

Este é o modelo utilizado na maior parte dos trabalhos em MDP, e que embasa os algoritmos mais conhecidos de AR. Uma vez escolhido esse modelo de otimalidade, pode-se de-

finir um valor ótimo para a atuação do agente como sendo:

$$V^* = \max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| \pi\right]. \quad (2)$$

Esse valor ótimo só depende do estado corrente e, para cada estado, pode ser definido como a solução do sistema de equações recursivas

$$\begin{aligned} V^*(s_i) &= \max_a \left( r(s_i, a) + \gamma \sum_{s' \in S} P(s'|s_i, a) V^*(s') \right) \\ &= \max_a Q^*(s_i, a), \quad i = 1, 2, \ldots, N_s, \end{aligned} \quad (3)$$

onde $Q^*(s, a)$ é o reforço total esperado caso se tome a ação $a$ no estado $s$ e depois se siga agindo otimamente. Isso significa que o valor de um estado $s$ é dado pela recompensa instantânea esperada mais o valor esperado para o próximo estado, descontado de $\gamma$, quando se usa a melhor ação disponível. Dada a função valor $V^*(s)$, a política ótima é aquela que garante (3), ou seja,

$$\pi^*(s) = \arg\max_a Q^*(s, a). \quad (4)$$

A função $Q^*(s, a)$ também pode ser definida de forma recursiva como

$$Q^*(s_i, a) = r(s_i, a) + \gamma \sum_{s' \in S} P(s'|s_i, a) \max_{a'} Q^*(s', a'). \quad (5)$$

## 3.2 Algoritmo SARSA

De modo geral, o papel dos algoritmos de AR é construir, de modo iterativo e através de interações com o ambiente, uma função $Q(s, a)$, inspirada na definição de $Q^*(s, a)$ dada em (5). $Q(s, a)$ é uma medida de utilidade de se escolher a ação $a$ dado que o ambiente se encontra no estado $s$. Como o espaço de estados e ações é finito e enumerável, essa função é guardada em uma tabela, geralmente denominada tabela Q. Assumindo-se certas condições, e após um grande número de interações, esses algoritmos aproximam $Q^*(s, a)$ por $Q(s, a)$, encontrando a política ótima através de (4).

Neste trabalho, optou-se por usar o algoritmo *SARSA* (Sutton and Barto, 1998). Quando o agente se encontra no estado $s$ e executa a ação $a$, obtém como resposta do ambiente um reforço $r$ e um novo estado $s'$. Uma vez nesse novo estado $s'$, o agente deve escolher uma nova ação $a'$. Assim, a partir dessa quíntupla $<s, a, r, s', a'>$, tem-se a regra de aprendizado do algoritmo:

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)), \quad (6)$$

onde $r \in \mathbb{R}$ é o sinal de reforço, $\alpha \in ]0, 1[$ é a taxa de aprendizado e $\gamma$ é a taxa de desconto, sendo $\alpha$ e $\gamma$ parâmetros de projeto. O algoritmo SARSA utiliza (6) para construir a tabela Q ao longo das iterações do agente com seu ambiente.

No entanto, (6) não fornece meios para determinar a ação seguinte a ser executada, o que depende da estratégia de atuação escolhida. Caso se confie no resultado do aprendizado obtido até então, pode-se adotar uma estratégia *gulosa — greedy*, em inglês — que escolhe sempre a ação com maior valor de utilidade $Q$ para o estado em que o agente se encontra. Caso contrário, pode-se adotar alguma estratégia que explore mais o espaço de ações, na tentativa de se acelerar o aprendizado. Note-se, porém, que no caso do algoritmo SARSA, uma condição para garantir a convergência para a política ótima é que a estratégia de atuação convirja para a estratégia gulosa, pois, assim, (6) torna-se

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a' \in A}\{Q(s', a')\} - Q(s, a)), \quad (7)$$

permitindo a aplicação da prova de convergência dada em (Mitchell, 1997).

## 3.3 AR baseado em módulos

Para a resolução de um problema através dos algoritmos de AR mais conhecidos, e em especial, do SARSA, é necessário que ele seja antes modelado nos termos de um MDP. Assim, o desempenho do algoritmo de AR depende de como são definidos os espaços de estados e ações do agente. Em muitos casos, definir esses conjuntos do modo mais direto implica na inviabilidade do uso dos algoritmos de AR, pois, além do número muito elevado de estados e ações, os estados poderiam não ser precisamente observados pelo agente.

Uma alternativa para a definição dos conjuntos de estados e ações é utilizar o conhecimento qualitativo *a priori* sobre a tarefa do agente de modo a decompô-la em sub-tarefas a serem executadas por controladores previamente projetados, também chamados de macro-ações. Nessa abordagem, o conhecimento *a priori* também é usado para determinar que atributos do ambiente correspondem a condições de operação para as macro-ações. Desse modo, AR baseado em módulos consiste no aprendizado de uma função de seleção das macro-ações a partir da situação de presença ou ausência dos atributos previamente definidos do ambiente (Kalmár et al., 1998).

### Atributos

Pode acontecer que as variáveis de estado envolvidas no problema sejam contínuas e ilimitadas, como em muitos problemas do mundo real. Além do mais, nesses problemas é muito comum que o estado do sistema não possa ser determi-

nado precisamente. No caso de robôs móveis, em especial, as variáveis de estado envolvidas diretamente no problema da navegação até um determinado objetivo correspondem à postura (posição e orientação do robô no ambiente) e as leituras dos sensores. Modelar os estados desse problema através de uma discretização simples de suas variáveis de estado nem sempre e possível, pois essas variáveis podem não ser todas mensuráveis com a precisão necessária. E quando isso é possível, há o inconveniente da necessidade de se definir a granularidade do modelo. Para isso, deve-se levar em conta o compromisso entre desempenho do algoritmo e a qualidade do resultado. Uma discretização muito fina das variáveis de estado resultaria em um grande número de estados, o que acarretaria em um tempo muito grande de processamento, ou mesmo à não convergência do algoritmo de AR. Em contrapartida, uma discretização grosseira pode resultar em uma política pouco útil para o problema em questão. Além do mais, caso o ambiente em que está inserido o robô for suficientemente grande e com muitos obstáculos, torna-se improvável obter uma granularidade do modelo capaz de levar a um resultado satisfatório.

Uma alternativa para a modelagem feita diretamente a partir dos dados dos sensores é a utilização de atributos, que são funções de observação do estado do sistema. Quando bem projetados, definir o estado do sistema através de atributos pode resolver o problema de dimensionalidade alta e dificuldade na observação (Kalmár et al., 1998).

### Módulos

Da mesma forma, as possíveis ações de um robô móvel formam um espaço contínuo, apresentando as mesmas dificuldades para sua modelagem. Uma alternativa bem conhecida é o emprego das macro-ações, que podem ser entendidas como controladores projetados para solucionar um problema específico, geralmente de baixa complexidade. Tendo-se em conta um modelo qualitativo do problema, é possível dividi-lo em sub-problemas suficientemente simples de tal forma que seja possível projetar uma macro-ação que o solucione. A ativação de uma macro-ação depende de uma determinada condição de operação. A cada par formado por uma macro-ação e sua respectiva condição de operação denomina-se módulo (Kalmár et al., 1998).

O emprego de atributos para modelar os estados do ambiente e de macro-ações para definir as possíveis ações do agente torna conveniente atrelar as condições de operação das macro-ações aos atributos. Definidos assim, os atributos passam a informar sobre a presença das várias condições de operação no ambiente através de um valor booleano, o que os caracteriza como atributos binários. Diz-se, então, que um atributo binário está ativo caso a respectiva condição de operação esteja presente no ambiente, e inativo caso contrário. Então, o estado do ambiente é dado por um vetor de atributos, que indica a ativação ou inatividade de cada atributo binário predefinido.

Esse modo de definição dos atributos fornece um meio muito apropriado de se marcar o tempo. Em vez de utilizar um relógio independente do processo, convém coincidir os instantes de transição de estados com os eventos de ativação ou desativação dos atributos. Dessa forma, os instantes de transição de estados, que medem a duração do MDP, ficam desacoplados do tempo real de duração dos estados, o que é mais interessante quando as sub-tarefas possuem naturalmente diferentes tempos de execução.

Dependendo de como foram projetados os módulos, pode ocorrer que mais de uma condição de operação ocorra ao mesmo tempo. Nesse caso, o papel do algoritmo de AR é determinar uma função de seleção que escolhe a macro-ação mais adequada para cada estado do ambiente, definido pelo vetor de atributos. Embora seja possível projetar condições de operação auto-excludentes, resultando em uma função de seleção predefinida, resultados obtidos por Kalmár et al. (1998), estendidos por Colombini and Ribeiro (2005), mostram que o uso de funções de seleção aprendidas apresentam melhor desempenho, uma vez que podem captar características do domínio de aplicação que passariam despercebidas pelo projetista humano. Kalmár et al. (1998) apontam ainda duas precauções que devem ser tomadas na aplicação de AR para a seleção das macro-ações. A primeira delas é a necessidade de que o tempo de duração de um estado seja sempre finito. Isso pode ser facilmente conseguido impondo-se restrições na política do agente, de forma a garantir que, qualquer que seja o estado, sempre será escolhida uma macro-ação que leve a uma mudança de estado em tempo finito. Outra precaução é que sempre haja pelo menos um caminho possível no espaço de estados até algum estado objetivo, independentemente do estado corrente.

## 4 ROBÔ UTILIZADO

A arquitetura proposta neste trabalho considera que o robô móvel utilizado é terrestre e move-se sobre um solo plano. Além disso, seus sensores devem permitir que ele possa se localizar através de odometria e ainda detectar os obstáculos à sua frente por meio de um anel frontal de sonares.

O modelo de robô utilizado neste trabalho é o Pioneer 2-DX, da ActivMedia Robotics®, ilustrado na figura 1. Ele possui um corpo de alumínio com 44cm de comprimento, 38cm de largura e 22cm altura, pesando 9kg e suportando 23kg de carga adicional. A direção é diferencial, sendo que as duas rodas principais possuem 19cm de diâmetro e 5cm de largura. Uma roda castor, localizada na parte traseira do robô, dá estabilidade à plataforma. O robô pode chegar a uma ve-

Figura 1: Foto frontal do robô Pioneer 2-DX.

locidade de translação máxima de 1,6m/s, e sua velocidade de rotação pode alcançar 230º/s.

## 4.1 Sensores do Pioneer 2-DX

Os sensores do Pioneer a serem considerados neste trabalho são os sonares e os *encoders*, utilizados para medir os deslocamentos das rodas. Dessa forma, esses sensores fornecem a localização do robô por odometria. Esse tipo de localização foi adotado neste trabalho, apesar de apresentar erro acumulativo. Os *encoders* possuem 500 marcas, podendo, dessa forma, detectar deslocamentos nas rodas de até 0,72º.

Os sonares do robô trabalham a uma freqüência de 25Hz, e estão distribuídos em forma de anel na parte frontal do robô. Os seis sonares mais à frente estão uniformemente distribuídos em um ângulo de 90º, de forma que as direções de emissão do ultrassom variam 15ºentre cada sonar. Além desses, há um sonar lateral em cada lado, e suas direções de atuação formam ângulos retos com o eixo longitudinal do robô.

## 4.2 Simulador do Pioneer-2DX

Os experimentos deste trabalho foram realizados no simulador do robô Pioneer 2-DX, distribuído conjuntamente com o *software* de comunicação com o robô. A simulação das leituras dos sensores, incluindo os erros, é bem realista, de forma que o simulador pôde ser utilizado sem prejuízo para a validade dos resultados.

# 5 ARQUITETURA REACT

A arquitetura REACT é baseada em comportamentos modelados por *Motor-Schemas* (Arkin, 1998). Um *Motor-Schema* permite, essencialmente, a definição e implementação de um comportamento em dois módulos principais: o módulo de percepção, que é responsável por extrair dos estímulos sensoriais as informações relevantes para o comportamento em questão, e o módulo de codificação do comportamento que, alimentado pelo módulo de percepção, executa o mapeamento dos estímulos sensoriais nas respostas motoras. Os parâmetros que definem uma ação — que corresponde à saída de cada *Motor-Schema* — são a magnitude e a direção do movimento. Estes parâmetros são representados por um vetor, e correspondem, respectivamente, à velocidade e à rotação que o robô deve executar, segundo o comportamento em questão. A coordenação dos comportamentos é feita de modo cooperativo: as contribuições de cada comportamento são somadas, de maneira que o problema da coordenação se resume a dimensionar corretamente as saídas dos diferentes comportamentos de forma que a sua soma resulte no comportamento global desejado para o robô.

## 5.1 Comportamentos da REACT

Os comportamentos utilizados na arquitetura REACT são:

**Comportamento *avoidCollision:*** este comportamento tem por objetivo evitar colisões em relação a múltiplos obstáculos. O módulo de percepção interpreta as leituras dos sonares de modo a identificar os obstáculos presentes no ambiente. O módulo de codificação da ação constrói o campo potencial do comportamento associando cargas repulsivas aos obstáculos. A direção da força é definida como radial aos obstáculos. A magnitude é determinada por uma função de decaimento que depende apenas da distância do robô ao obstáculo. Deste modo, o módulo de codificação deste comportamento é composto pelas equações

$$V(d) = \begin{cases} V_{AC}\, e^{\frac{-d+S}{T}} & \text{para } d > S \\ V_{AC} & \text{para } d \leq S \end{cases} \qquad (8)$$
$$\phi = \pi - \phi_{\text{robô-obstáculo}}$$

onde $V$ é a magnitude do comportamento (velocidade), $d$ é a distância do centro de massa do robô ao obstáculo, $V_{AC}$ é a velocidade máxima do comportamento, $S$ é o *standoff* do robô (distância abaixo da qual a magnitude do comportamento satura em seu valor máximo), $T$ é a constante de escala da curva de decaimento, $\phi$ é a direção do vetor resultante do comportamento e $\phi_{\text{robô-obstáculo}}$ é a direção definida pela reta que une o obstáculo ao centro de massa do robô.

**Comportamento *moveToGoal:*** este comportamento tem o objetivo de atrair o robô para um determinado ponto no ambiente. A posição inicial do alvo no sistema global de coordenadas é informada ao sistema por meio de um agente externo. A posição atual do robô, estimada pelo módulo de percepção, baseia-se na informação dos odômetros. O módulo de codificação determina a direção do movimento como sendo igual à direção dada pela reta que une os pontos que definem as posições do robô e do alvo ($\phi_{\text{robô-alvo}}$). A magnitude apre-

senta valor constante ($V_{MTG}$), de modo que a codificação do comportamento é dada por

$$\begin{aligned} V &= V_{MTG} \\ \phi &= \phi_{\text{robô-alvo}}. \end{aligned} \tag{9}$$

**Comportamento *moveAhead:*** este faz com que o robô sempre tenha uma forte tendência de continuar na direção em que se encontra no momento. O comportamento *moveAhead* não possui módulo de percepção e o seu módulo de codificação é elementar, retornando um campo de magnitude constante ($V_{MA}$) e direção sempre igual à do próprio robô naquele instante ($\phi_{\text{robô}}$):

$$\begin{aligned} V &= V_{MA} \\ \phi &= \phi_{\text{robô}}. \end{aligned} \tag{10}$$

## 5.2 Ponderação dos comportamentos na REACT

Na arquitetura REACT original, todos comportamentos são igualmente considerados para o cálculo da ação do robô. As magnitudes dos comportamentos ($V_{AC}$, $V_{MA}$ e $V_{MTG}$) são fixas e foram determinadas experimentalmente para um modelo específico de robô, de forma que seu desempenho fosse satisfatório para diversos ambientes (Pacheco and Costa, 2002). No entanto, um melhor desempenho poderia ser alcançado caso essas magnitudes pudessem variar dependendo da situação do ambiente encontrada pelo robô. Isso pode ser feito multiplicando a saída de cada comportamento por um determinado peso que dependa do estado do ambiente, ponderando assim a participação dos comportamentos de acordo com a conveniência.

Dessa forma, um conjunto de três pesos, $w_{AC}$, $w_{MTG}$ e $w_{MA}$, que ponderam, respectivamente, as saídas de *avoidCollision*, *moveToGoal* e *moveAhead*, define também um controlador para o robô. Pode-se, então, dispor de vários conjuntos de pesos, projetados *a priori*, obtendo-se, assim, diversos controladores diferentes. Conseqüentemente, torna-se possível inserir na arquitetura uma camada de aprendizado com o objetivo de escolher o melhor controlador para a situação em que o ambiente se encontra. Caso seja definido um conjunto discreto de possíveis situações, a camada de aprendizado deve então aprender uma política associando cada possível situação a um determinado controlador, cenário muito conveniente para a utilização de AR.

# 6 ARQUITETURA AAREACT

A arquitetura AAREACT, acrônimo para *Adaptação Automática* da REACT, consiste de uma camada de aprendizado por reforço baseado em módulos que supervisiona a atuação do robô e modifica os pesos que multiplicam as saídas dos comportamentos (Selvatici, 2005), conforme mostra a figura 2.

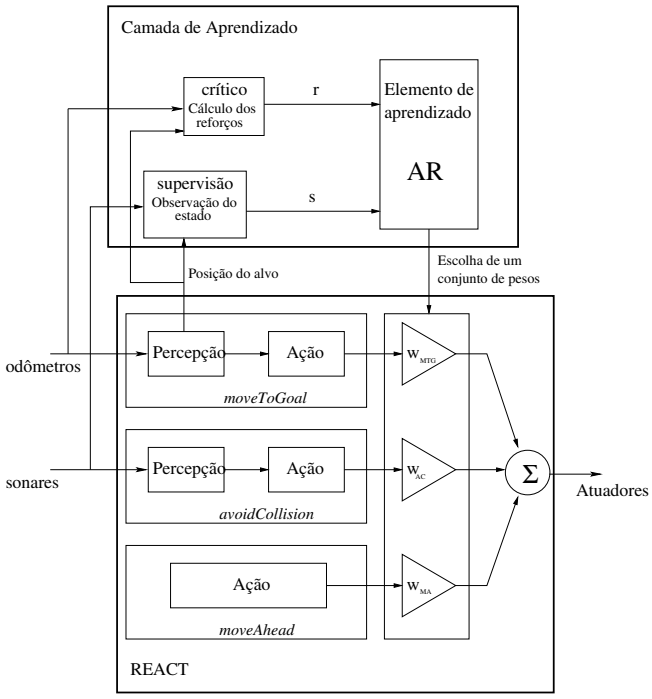


Figura 2: Diagrama de blocos da arquitetura AAREACT

A camada de aprendizado da AAREACT, através do componente de **supervisão**, percebe o estado $s$ do ambiente dado por atributos binários calculados a partir dos dados dos sensores do robô (sonares e odômetros). Além disso, através do componente **crítico**, define reforços $r$ que dependem da atuação do robô. Então o **elemento de aprendizado**, que implementa o algoritmo SARSA, determina a política de atuação adequada àquele estado, $\pi(s)$, correspondendo a um conjunto de pesos para os comportamentos da arquitetura.

## 6.1 A modelagem do problema

A tarefa principal que deve ser executada pelo robô móvel é a navegação até um determinado ponto objetivo no ambiente, com o desvio de obstáculos. Possíveis sub-tarefas relacionadas com a tarefa principal são a ultrapassagem de cada obstáculo que se encontra no caminho até o alvo. Cada ultrapassagem pode, ainda, se subdividir em etapas, nas quais a percepção do obstáculo pelo robô vai se modificando. Considerando os comportamentos da REACT, descritos na seção 5, supõe-se que há uma melhor combinação desses comportamentos para cada etapa do desvio de um obstáculo, representada por um conjunto de pesos que multiplicam as saídas dos comportamentos. Dessa forma, um conjunto de pesos pode ser definido como uma macro-ação, cuja condição de operação é dada pela percepção do robô com relação ao alvo e aos

obstáculos.

Note-se que, no trabalho de Kalmár et al. (1998), cada macro-ação corresponde a um determinado comportamento. De fato, no campo da robótica móvel, essa correspondência é bastante natural (Colombini and Ribeiro, 2005), uma vez que é comum definir cada comportamento em vista da execução de uma tarefa específica dentre aquelas que o robô deve realizar. No entanto, neste trabalho, cada macro-ação corresponde a uma combinação de comportamentos, e não a um comportamento isolado, caracterizando uma abordagem essencialmente diferente.

## 6.2 A definição dos atributos

Os atributos definidos procuram fornecer informação suficiente para refletir a percepção do robô com relação aos obstáculos próximos e ao alvo. Cada atributo, quando ativo de forma isolada, leva naturalmente a uma combinação propositada dos comportamentos, representada por um conjunto de pesos. A relação entre um atributo e seu respectivo conjunto de pesos correlato estão apresentados na seção 6.3.

Os atributos definidos neste trabalho são:

**FreeTarget:** este atributo está ativo quando o robô detecta que não há obstáculos entre ele e o alvo, ou ainda que os obstáculos na direção do alvo estão muito distantes. Um obstáculo distante é caracterizado por uma leitura de sonar superior a um limiar $L_{far}$, definido arbitrariamente.

**BackTarget:** a ativação deste atributo se dá quando o alvo se encontra atrás do robô, de forma que os sonares, dispostos em um anel na parte frontal do robô, não conseguem detectar se há algum impedimento entre ele e o alvo.

**SideObstacle:** este atributo é ativado quando um dos sonares laterais do robô detecta a presença de um obstáculo próximo, o que equivale a uma leitura de sonar inferior a um limiar $L_{near}$, definido arbitrariamente, com $L_{near} < L_{far}$.

**DiagonalObstacle:** este atributo é ativado quando um dos sonares das diagonais anteriores do robô detecta a presença de um obstáculo próximo, com leitura de distância inferior a $L_{near}$.

**MiddleObstacle:** o que determina a ativação deste atributo é a detecção de algum obstáculo a uma distância média do robô, caracterizada por uma leitura de qualquer sonar entre os limiares $L_{near}$ e $L_{far}$. A presença isolada deste atributo configura uma situação confortável, onde não há perigo iminente de obstáculo.

**NarrowPath:** este atributo está ativo quando ambos os sonares laterais do robô detectam a presença de um obstáculo próximo, caracterizando a passagem do robô por um corredor estreito. Quando este atributo está ativo, os demais descritos anteriormente (*FreeTarget, BackTarget, SideObstacle, DiagonalObstacle* e *MiddleObstacle*) são ignorados, uma vez que a navegação em um corredor estreito é uma situação especial na qual não se deve preocupar com a posição do alvo ou a presença dos obstáculos fora do corredor.

**FrontalObstacle:** este atributo está ativo quando um dos sonares frontais do robô detecta a presença de um obstáculo próximo, a uma distância menor do que $L_{near}$, caracterizando um perigo iminente de colisão. Quando este atributo é ativado, todos os demais são ignorados, já que a reação mais prudente do robô é se afastar desse obstáculo e não se preocupar com o alvo ou com obstáculos mais distantes.

A figura 3 ilustra a definição de cada atributo em termos da distribuição espacial dos obstáculos e do alvo em torno do robô. O estado do ambiente é caracterizado pelo vetor de atributos que indica quais estão ativos e quais estão inativos. Uma mudança de estado ocorre quando há a ativação de um ou mais atributos antes inativos, ou ainda quando um ou mais atributos deixam de estar ativos. A princípio, dado o número de atributos definidos, pode-se pensar que o número de estados possíveis é $2^7 = 128$. No entanto, o modo como estão definidos os atributos resulta em que vários sejam mutuamente exclusivos, reduzindo drasticamente o tamanho do espaço de estados. Uma análise dos atributos definidos mostra que há apenas 24 estados possíveis para o ambiente.

## 6.3 Definição dos conjuntos de pesos

A elaboração dos pesos é baseada na idéia de se atrelar um controlador a cada atributo definido. Quando o estado do ambiente for definido pela ativação *isolada* de um único atributo, o conjunto de pesos escolhido é aquele atrelado ao atributo em questão. Nos demais estados, nos quais há *mais de um atributo* ativo, o módulo de AR deve *decidir* o melhor conjunto de pesos dentre aqueles relacionados na tabela 1. O modo como foram definidos os atributos garante que não há a possibilidade de todos os atributos estarem desativados simultaneamente.

Ao atributo *FreeTarget* foi atrelado o conjunto de pesos 1, que ativa apenas o comportamento *moveToGoal*. Já ao atributo *BackTarget* foi atrelado o conjunto de pesos 2, que, além do *moveToGoal*, ativa também o comportamento *avoidCollision* como uma forma de prevenção contra colisões com objetos não detectados. Possíveis obstáculos posicionados atrás do robô não podem ser percebidos previamente pela falta de

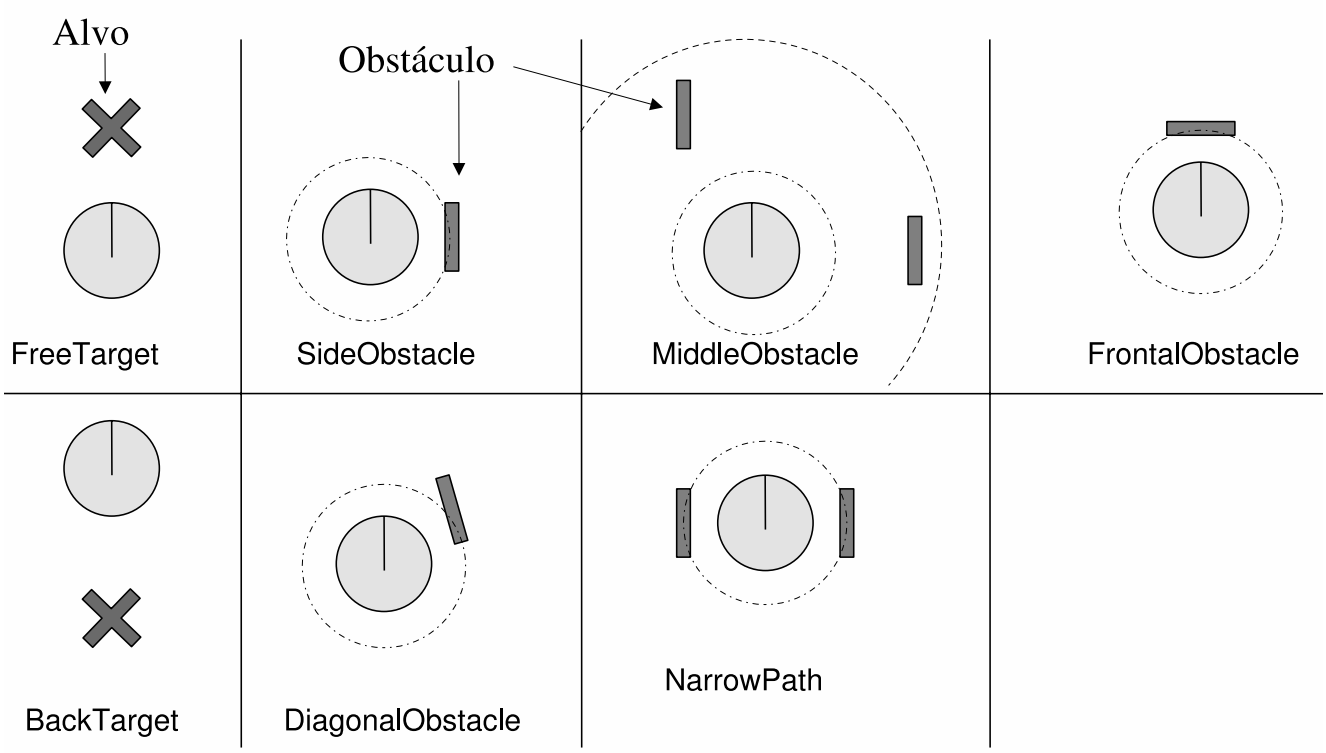


Figura 3: Ilustração de como foram definidos os atributos. O robô está representado pelo círculo central, sendo que o segmento de reta em seu interior define a direção do movimento. A cruz representa o alvo até onde o robô deve se locomover, e os retângulos representam os obstáculos detectados pelos sonares. As circunferências de linha pontilhada representam as distâncias $L_{near}$ e $L_{far}$, sendo que $L_{near}$ é a circunferência mais próxima do robô.

sonares traseiros, de forma que se o giro do robô em direção ao alvo for muito rápido, sem haver tempo para a mudança de estados, e, com isso, mudança no conjunto de pesos escolhidos, o comportamento *avoidCollision* evitará a colisão. No caso de *SideObstacle*, além do comportamento de andar para frente, é interessante haver uma participação de *avoidCollision* para que haja um afastamento gradual do obstáculo lateral. Já o atributo *DiagonalObstacle* requer uma participação maior do comportamento *avoidCollision*, pois o risco de colisão é maior. Como a ativação isolada do atributo *MiddleObstacle* indica que não há perigo de colisão iminente, o conjunto de pesos atrelado determina o grau de participação dos comportamentos igual à utilizada, de forma fixa, na arquitetura REACT. O atributo *FrontalObstacle*, que representa perigo de colisão iminente, requer a ativação máxima do comportamento *avoidCollision*, mais uma pequena participação de *moveAhead* para evitar movimentos muitos bruscos do robô. A ativação do atributo *NarrowPath* indica uma situação semelhante à ativação isolada de *SideObstacle*.

Nota-se que o conjunto de pesos 8 não está atrelado a nenhum atributo do ambiente. Ele foi elaborado de modo a ser bastante diferente dos demais definidos até então: além dele, apenas o conjunto 5 conjuga os três comportamentos simultaneamente, mas cada um desses conjuntos de pesos enfatiza comportamentos diferentes. Dessa forma, o conjunto de pesos 8 está presente entre as possíveis ações do módulo de AR como uma opção *a mais* a ser explorada no caso de *mais de um atributo* estar presente ao mesmo tempo.

Tabela 1: Relação entre os atributos definidos e o conjunto de pesos dos comportamentos associados a cada um.

| Índice | Atributo | Conjunto de pesos associado $\{w_{MA}; w_{MTG}; w_{AC}\}$ |
|---|---|---|
| 1 | *FreeTarget* | {0,0 ; 1,0 ; 0,0} |
| 2 | *BackTarget* | {0,0 ; 1,0 ; 1,0} |
| 3 | *SideObstacle* | {1,0 ; 0,0 ; 0,3} |
| 4 | *DiagonalObstacle* | {1,0 ; 0,0 ; 1,0} |
| 5 | *MiddleObstacle* | {0,6 ; 0,4 ; 1,0} |
| 6 | *NarrowPath* | {1,0 ; 0,0 ; 0,3} |
| 7 | *FrontalObstacle* | {0,3 ; 0,0 ; 1,0} |
| 8 | — | {0,5 ; 1,0 ; 0,7} |

## 6.4 Modelo da função de reforço

Em AR, reforços positivos são utilizados para premiar situações desejáveis para o agente, podendo-se aplicar também penalidades para situações indesejáveis. O objetivo primário do robô é chegar à posição alvo. Dessa forma, a camada de aprendizado recebe um grande reforço positivo quando isso ocorre, definido por $r_{goal}$.

No entanto, é desejável também que o robô apresente um bom desempenho na sua atuação, o que é difícil de expressar quantitativamente. Neste trabalho, tentou-se modelar esse desempenho através de dois parâmetros: a velocidade média desenvolvida e a velocidade média de aproximação do alvo. Assim, recompensas proporcionais a essas medidas de desempenho são fornecidas à camada de aprendizado sempre que é percebida uma mudança na situação do ambiente, caracterizando um *reforço intermediário*, recebido antes do cumprimento da tarefa especificada. Reforços intermediários são importantes para acelerar o aprendizado, no entanto, esses reforços devem ter valores inferiores àquele recebido quando o robô atinge o alvo. O reforço intermediário recebido é dado por

$$r_{int} = K_1 v_m + K_2 v_a, \tag{11}$$

onde:

- $K_1$ e $K_2$ são ganhos arbitrariamente definidos;
- $v_m$ é a velocidade média desenvolvida pelo robô durante o tempo em que o ambiente se encontra num determinado estado ou situação;
- $v_a$ é a velocidade média de aproximação do alvo pelo robô durante o tempo de duração de uma situação, dada pela diferença entre as distâncias inicial e final do alvo, dividida pelo tempo de permanência na situação.

A formulação acima permite valores positivos e negativos para $r_{int}$. O valor desse reforço intermediário é saturado em $\pm\bar{r}$, para garantir que será sempre menor do que $r_{goal}$.

# 7 EXPERIMENTOS COM A AAREACT

No início do aprendizado, não se conhece uma política adequada para a escolha dos conjuntos de pesos, de modo que é esperada uma atuação ruim do robô em qualquer ambiente. A única política predefinida é a que relaciona a ativação de um único atributo do ambiente com o respectivo conjunto de pesos; esta porém é fixa e não é influenciada pelo resultado do aprendizado. Por isso, decidiu-se dividir o aprendizado da arquitetura em duas fases. Na primeira fase, denominada de aprendizado inicial, o módulo de AR ainda não tem uma boa estimativa das utilidades $Q(s, a)$, e adota uma estratégia $\epsilon$-gulosa (Sutton and Barto, 1998), que, ou escolhe uma ação aleatória, com probabilidade $\epsilon$, ou escolhe a ação com maior utilidade, com probabilidade $1 - \epsilon$. Nessa fase é esperada uma atuação ruim para o robô, que, inicialmente, deve apresentar até mesmo um comportamento errante. Com isso, espera-se que ele experimente as várias situações diferentes e aprenda a lidar com elas, resultando assim em uma política inicial satisfatória, válida para qualquer ambiente, definida a partir da tabela Q obtida.

Na segunda fase, utiliza-se o conhecimento adquirido no aprendizado inicial, de forma que já é esperado um desempenho razoável do robô em qualquer ambiente que venha a atuar. Nessa fase, o módulo de AR não mais explora os possíveis conjuntos de pesos, e passa a adotar a estratégia gulosa, visando apenas a obter um bom desempenho. No entanto, vale ressaltar que o aprendizado por reforço continua ativo. Os experimentos que avaliam o desempenho da arquitetura AAREACT são realizados nessa fase.

## 7.1 Aprendizado inicial

Como o conhecimento adquirido pela AAREACT se estende a diversos ambientes, é interessante aproveitar o aprendizado inicial para treinar o robô em um ambiente que apresente diversas situações possíveis a serem exploradas. Foi elaborado um cenário onde o robô é obrigado a passar por vários tipos de configurações de obstáculos para atingir seu objetivo. Esse cenário é constituído de uma sala retangular, onde foram definidas quatro posições alvo de modo a ficarem próximas das quinas da sala, conforme mostra a figura 4. O robô foi então comandado a procurar um alvo por vez, de modo a garantir que, ao percorrer os quatro alvos consecutivos, o robô circunde a sala.

Na implementação da arquitetura AAREACT, os seguintes parâmetros foram utilizados:

- Parâmetros dos comportamentos
  - *Stand off*: $S = 40$cm
  - Constante de decaimento: $T = 15$cm
  - Velocidade *moveAhead*: $V_{MA} = 15,0$cm/s
  - Velocidade *avoidCollision*: $V_{AC} = 15,0$cm/s
  - Velocidade *moveToGoal*: $V_{MTG} = 15,0$cm/s
- Parâmetros dos atributos
  - Limiar de detecção de obstáculo próximo: $L_{near} = 50$cm
  - Limiar de detecção de obstáculo distante: $L_{far} = 200$cm

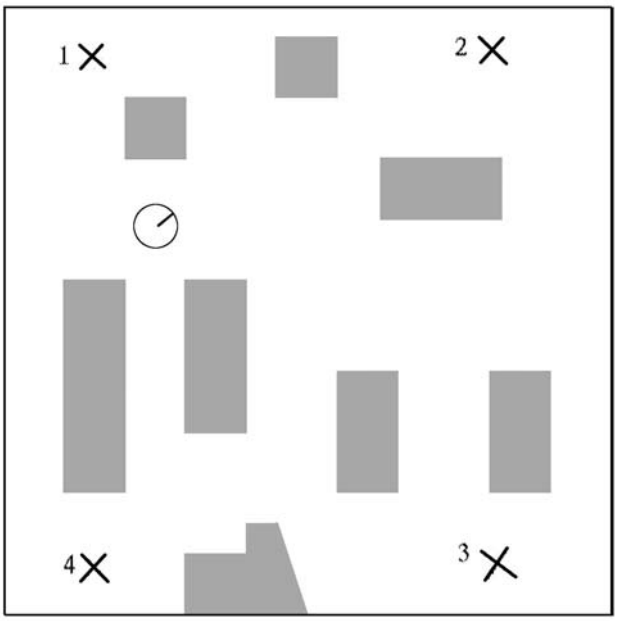


Figura 4: Ambiente onde o robô simulado iniciou seu aprendizado. Os objetos em cinza na figura definem os obstáculos. As posições indicadas com um $\times$ são os alvos. O círculo indica a posição do robô, sendo que o segmento de reta em seu interior indica sua orientação. Os números de 1 a 4 indicam a ordem na qual os alvos são percorridos.

- Parâmetros do aprendizado
  - Taxa de aprendizado: $\alpha = 0,3$
  - Taxa de desconto: $\gamma = 0,99$
  - Probabilidade de exploração: $\epsilon = 20\%$
- Parâmetros do reforço
  - Reforço no objetivo: $r_{goal} = 100$
  - Reforço máximo intermediário: $\bar{r} = 3$
  - Ganho sobre a velocidade média desenvolvida: $K_1 = 0,02$s/mm
  - Ganho sobre a velocidade média de aproximação do alvo: $K_2 = 0,002$s/mm

Uma vez que o aprendizado por reforço acompanha toda a vida do agente, ou seja, sempre está ativo, adotou-se um critério de parada flexível para o aprendizado inicial. A permanência prolongada nessa fase não acarreta nenhum prejuízo para o conhecimento adquirido, de modo que o experimento foi executado pelo tempo necessário para que o robô passe a apresentar pouca variação no desempenho. O desempenho do robô foi medido através do tempo gasto para completar

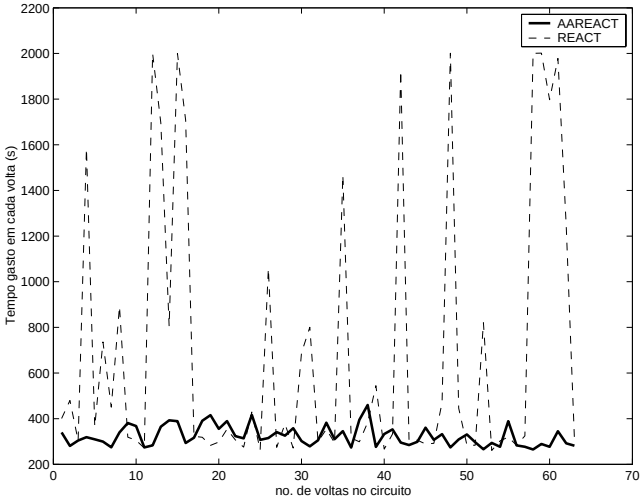


Figura 5: Comparação gráfica dos resultados desempenhados pelas arquiteturas AAREACT e REACT no simulador para o cenário utilizado no aprendizado inicial, representado na figura 4.

cada volta no circuito, o que corresponde a passar pelos quatro alvos estipulados. Notou-se que, após completar o circuito pela primeira vez, os tempos das voltas posteriores ficaram limitados em uma faixa de menor valor. Dessa forma, considerou-se que o período de aprendizado inicial foi executado por um tempo suficiente para garantir um bom desempenho ao robô. A tabela Q resultante foi então adotada para outros experimentos, realizados numa fase posterior. Nesses experimentos a probabilidade de exploração $\epsilon$ é fixada em zero.

## 7.2 Experimentos comparativos com o simulador

Foram realizados alguns testes para comparar o desempenho da arquitetura AAREACT com relação ao da arquitetura REACT. Ambas as arquiteturas foram implementadas no simulador do robô Pioneer 2-DX (ver seção 4).

O primeiro cenário utilizado é aquele usado no aprendizado inicial da AAREACT. Da mesma forma que no aprendizado inicial, o robô deve percorrer o circuito formado pelos quatro alvos, de forma seqüencial. Caso o robô demore muito a cumprir seu objetivo (mais do que 2000s), considera-se que possivelmente ele ficou "perdido", de modo que o simulador recoloca o robô na sua posição inicial, iniciando um novo ciclo. A arquitetura com aprendizado AAREACT apresentou melhor resultado, enquanto que a arquitetura com coordenação fixa REACT freqüentemente não conseguiu cumprir os objetivos durante o tempo máximo estipulado. Os resultados de desempenho obtidos estão apresentados na figura 5, en-

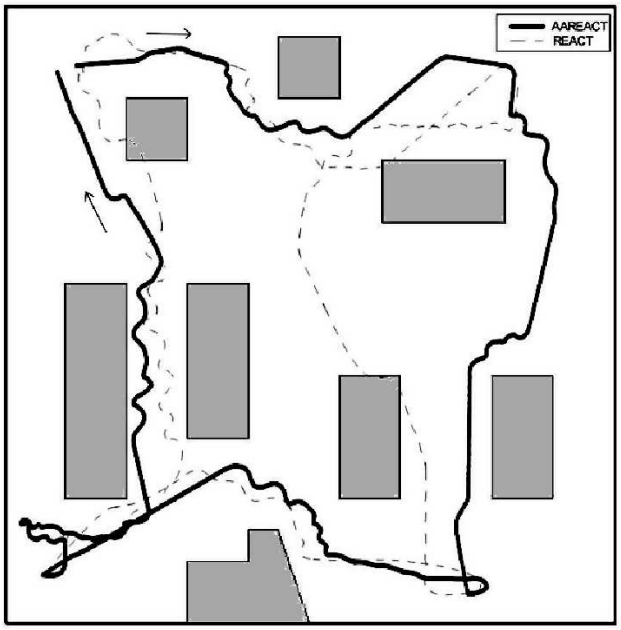


Figura 6: Comparação entre trajetórias típicas apresentadas pelas arquiteturas REACT e AAREACT nos experimentos realizados no simulador, com o cenário do aprendizado inicial. As setas indicam o sentido do movimento do robô.

quanto que a figura 6 compara trajetórias típicas obtidas com as arquiteturas REACT e AAREACT. As trajetórias foram obtidas através do modelo de odometria com erro apresentado pelo simulador do robô Pioneer. Dessa forma, as trajetórias marcadas indicam apenas um caminho aproximado. As regiões onde as trajetórias se sobrepõem aos obstáculos são conseqüência desse erro de odometria. As oscilações nas curvas de desempenho da figura 5 devem-se ao fato de que, mesmo para uma determinada arquitetura, os trajetos percorridos são diferentes a cada volta no cenário, uma vez que os sensores e atuadores são ruidosos. Observa-se uma grande variação no desempenho da arquitetura REACT neste cenário, de forma que ela apresenta bons resultados (tempo gasto menor do que 400s) apenas esporadicamente. Enquanto isso, a AAREACT se mostra mais constante, sendo que, na grande maioria dos casos, ela apresenta um resultado muito melhor do que a REACT.

Em outro experimento, foi utilizado um segundo cenário, mostrado na figura 7. Nesse cenário há dois alvos que se alternam, de modo que o robô deve percorrer um circuito de ida e volta entre eles.

No entanto, observou-se que nesse cenário, a arquitetura aprendiz apresentou desempenho muito melhor, já que, enquanto a arquitetura REACT se mostrou incapaz de, muitas vezes, cumprir o objetivo, a arquitetura AAREACT iniciou

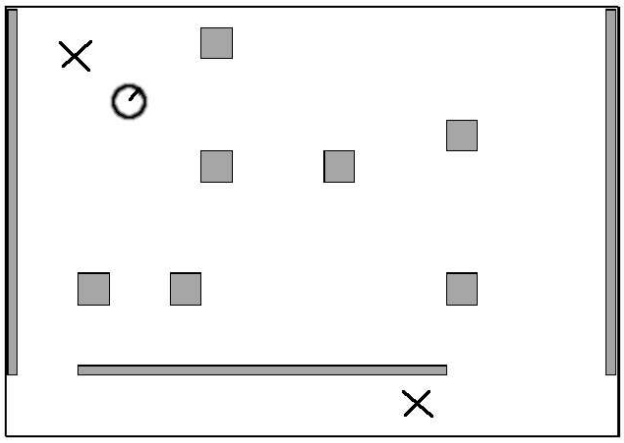

Figura 7: Ambiente da segunda experiência no simulador. Os alvos estão indicados com um $\times$.

sua atuação com um desempenho satisfatório, e, à medida em que se adaptava ao ambiente específico, melhorava seu desempenho de modo visível, como mostra a figura 8. No entanto, a fase de adaptação é caracterizada por uma grande oscilação na curva de desempenho, resultando nos "picos" observados no gráfico referente à AAREACT até a 24ª volta.

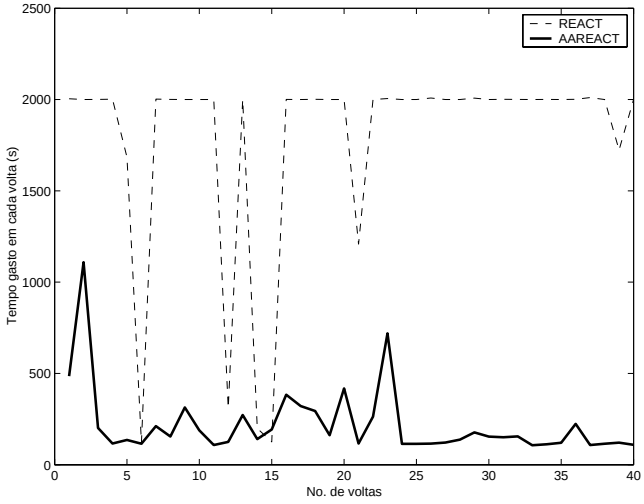


Figura 8: Comparação gráfica dos resultados entre os desempenhos das arquiteturas AAREACT e REACT no segundo cenário do simulador.

## 8 CONCLUSÃO

Foi apresentada uma arquitetura adaptativa para robôs móveis baseada em comportamentos reativos e com capacidade de aprendizado. Os resultados obtidos com a arquitetura AAREACT demonstram que a adaptação da influência de cada

comportamento na atuação do robô de acordo com a situação encontrada de fato melhora o seu desempenho, quando comparado com o obtido pela ponderação fixa dos comportamentos na coordenação cooperativa dos mesmos, conforme implementada na REACT. Alguns experimentos mostraram, inclusive, que a arquitetura adaptativa é capaz de fazer o robô atingir seu objetivo em alguns ambientes nos quais a arquitetura REACT dificilmente tem sucesso. O aprendizado da AAREACT procura ajustar a atuação do robô a uma situação percebida localmente. Assim, ela não foi projetada para se adaptar a um ambiente específico, mas seu aprendizado é orientado de forma a ajustar a atuação do robô de modo a se comportar bem nas várias configurações possíveis do ambiente. Por isso, o resultado do aprendizado inicial, realizado em um ambiente específico, é utilizado pela arquitetura em todos os outros ambientes, onde o aprendizado continua a ser realizado, de forma incremental. Mesmo que o robô opere sempre em um ambiente específico e se especialize nele, a transição para outro ambiente ocorre facilmente, uma vez que o aprendizado evolui naturalmente.

No entanto, não é possível concluir que o conjunto de atributos e de pesos, e a função reforço utilizados são realmente os que proporcionam melhor resultado para o aprendizado. É importante ressaltar, porém, que a tarefa para qual foi projetada a arquitetura é bastante genérica no âmbito de robótica móvel, de forma que a investigação a respeito do refinamento dos atributos, pesos e reforços definidos podem consistir em uma contribuição importante. Por isso, como trabalho futuro, sugere-se a realização de estudos comparativos entre possíveis atributos e reforços, além da determinação de atributos eficientes baseados em outros tipos de sensores utilizados no desvio de obstáculos de robôs móveis.

## AGRADECIMENTO

Antonio Henrique P. Selvatici agradece o apoio dado pela Fundação de Amparo à Pesquisa do Estado de São Paulo — FAPESP, através do processo no. 02/11792-0, e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico — CNPq, através do processo número 1408682005-4. Este trabalho teve apoio parcial do projeto MultiBot CAPES/GRICES, proc. n. 099/03.

## REFERÊNCIAS

Arkin, R. C. (1998). *Behavior-Based Robotics*, The MIT Press, Cambridge, MA.

Borenstein, J. and Koren, Y. (1990). Real-time obstacle avoidance for fast mobile robots in cluttered environments, *Proceedings of the 1990 IEEE Conference on Robotics and Automation*, Vol. 1, pp. 572–577.

Colombini, E. L. and Ribeiro, C. H. (2005). An analysis of feature-based and state-based representations for module-based learning in mobile robots, *Proceedings of the 5th International Conference on Hibrid Intelligent Systems*, IEEE Computer Society, Rio de Janeiro, Brasil, pp. 163–168.

Elfes, A. (1989). *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*, PhD thesis, Department of Electical and Computer Engineering, Carnegie Mellon University.

Kaelbling, L. P., Littman, M. L. and Moore, A. (1996). Reinforcement learning: A survey, *Journal of Artificial Intelligence Research* **4**: 237–285.

Kalmár, Z., Szepesvári, C. and Lörincz, A. (1998). Module-based reinforcement learning: Experiments with a real robot, *Machine Learning* **31**: 55–85.

Koren, Y. and Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation, *Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento, California, pp. 1398–1404.

Mitchell, T. (1997). *Machine Learning*, McGraw-Hill, Boston, MA.

Monteiro, S. T. (2002). *Estudo de desempenho de agoritmos de aprendizagem sob condições de ambigüidade sensorial*, Master's thesis, Instituto Tecnológico de Aeronáutica, São José dos Campos, Brasil.

Murphy, R. (2000). *Introduction to AI Robotics*, The MIT Press, Cambridge, MA.

Pacheco, R. N. and Costa, A. H. R. (2002). Navegação de robôs móveis utilizando o método de campos potenciais, *in* M. T. S. Sakude and C. de A. Castro Cesar (eds), *Workshop de Computação – WORKCOMP'2002*, Instituto Tecnológico de Aeronáutica, São José dos Campos, SP, pp. 125–130.

Ranganathan, A. and Koenig, S. (2003). A reactive robot architecture with planning on demand, *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Vol. 2, Las Vegas, Califórnia, pp. 1462–1468.

Ribeiro, C. H. C., Costa, A. H. R. and Romero, R. A. F. (2001). Robôs móveis inteligentes: princípios e técnicas, *in* A. T. Martins and D. L. Borges (eds), *I Jornada de Atualização em Inteligência Artificial - JAIA. Anais do XXI Congresso da Sociedade Brasileira de Computação*, Vol. 3, SBC, Fortaleza, CE, pp. 257–306.

Russel, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, $2^{\mathrm{nd}}$ edn, Prentice Hall, Upper Saddle River, New Jersey.

Selvatici, A. H. P. (2005). *AAREACT: Uma arquitetura comportamental adaptativa para robôs móveis que integra visão, sonares e odometria*, Master's thesis, Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil.

Selvatici, A. H. P. and Costa, A. H. R. (2004). Combinação de sensores através da cooperação de comportamentos primitivos, *Anais do XV Congresso Brasileiro de Automática CBA'2004*, Gramado, RS.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*, MIT Press, Massachussets, MA.