DANIEL TEIXEIRA SILVA
RODRIGO TAMIAZZO DO NASCIMENTO
THIAGO L. BARDELLA DOS SANTOS

APRENDIZADO POR REFORÇO NA NAVEGAÇÃO ROBÓTICA

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a conclusão do curso de graduação em Engenharia de Computação, junto ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

São Paulo
2013

DANIEL TEIXEIRA SILVA
RODRIGO TAMIAZZO DO NASCIMENTO
THIAGO L. BARDELLA DOS SANTOS

APRENDIZADO POR REFORÇO NA NAVEGAÇÃO ROBÓTICA

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a conclusão do curso de graduação em Engenharia de Computação, junto ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

Área de Concentração:

Sistemas Digitais

Orientador:

Prof.ª Dr.ª Anna Helena Reali Costa

São Paulo
2013

FICHA CATALOGRÁFICA

Teixeira Silva, Daniel; Tamiazzo do Nascimento, Rodrigo; Bardella dos Santos; Thiago L.

Aprendizado por Reforço na Navegação Robótica / D. Teixeira Silva, R. Tamiazzo do Nascimento, T. L. Bardella dos Santos. – São Paulo, 2013.

48 p.

Monografia (Graduação em Engenharia de Computação) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

1. Inteligência artificial. 2. Navegação robótica. 3. Aprendizado por reforço. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais (PCS). II. t.

AGRADECIMENTOS

Agradecemos

À nossa orientadora, Anna Helena Reali Costa, pela direção e paciência.

A nossas famílias, pelo apoio e compreensão.

RESUMO

Aprendizado por reforço permite que um agente aprenda a realizar tarefas de forma otimizada através de experiências obtidas de interações com o ambiente. Aplicado na navegação robótica, o aprendizado por reforço possibilita um robô achar melhores caminhos partindo de uma aprendizado adquirido em navegações passadas. Este texto apresenta a base teórica juntamente com o conjunto de algoritmos destinados à solução do problema do aprendizado por reforço.

ABSTRACT

Reinforcement learning allows an agent learn to perform tasks optimally th-rough experiences gained from interactions with the environment. Applied in ro-botic navigation, the reinforcement learning enables a robot to find the best paths starting from a learning adiquirido in past navigations. This paper presents the theoretical basis with a set of algorithms for the solution of the problem of reinfor-cement learning.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

USP	Universidade de São Paulo
PDM	Processos de Decisão de Markov
DT	Diferença Temporal
SARSA	State-Action-Reward-State-Action
CMAC	Cerebelaar Model Articulation Controller
ROS	Robot Operating System
RFID	Radio-Frequency IDentification

LISTA DE SÍMBOLOS

s	Estado
a	Ação
π	Política de tomada de ação
r	Recompensa recebida pelo agente durante o passo executado
γ	Fator de desconto
$V^{\pi}(s)$	Função valor-estado
$V^{*}(s)$	Função valor-estado ótima
$V^{*}(s)$	Função valor-estado ótima
$Q^{\pi}(s)$	Função valor-ação
$Q^{*}(s)$	Função valor-ação ótima
λ	Retorno, parâmetro utilizado para ponderar a participação de cada uma das configurações de n passos
s_i	Entrada de índice i para o CMAC
ε	Precisão, tamanho de cada ladrilho
Q_i	Índice dos hipercubos
I_{ij}	Intervalo de subdivisão de cada fibra, onde o índice *i* corresponde à entrada e o índice *j* corresponde à fibra
Q	Distância entre fibras mucosas
act_{ij}	Função ativo para cada célula da tabela de valores

σ	Variância
a_{min}	Limite de corte para valores próximos do centro da curva gaussiana
a_v	Valor da célula
p	Valor da saída, considerando as células da tabela de memória ativas com seus pesos
w_v	Peso da célula ativa a_v

1 INTRODUÇÃO

O aprendizado por reforço constitui um paradigma que propõe o questionamento de como um agente autônomo que percebe o ambiente e executa ações sobre este pode aprender a escolher ações ótimas para alcançar seus objetivos. O aprendizado por reforço é o problema encarado por um agente que aprende o seu próprio comportamento através de interações de tentativa e erro em um ambiente dinâmico. A promessa principal deste aprendizado é a de que os agentes sejam programados através de recompensas e punições sem a necessidade de especificar-se como a tarefa será alcançada. Este problema genérico cobre tarefas como aprender a controlar um robô móvel, aprender a aperfeiçoar operações em fábricas e aprender a jogar os conhecidos jogos de tabuleiro. Cada vez que o agente realiza uma ação no ambiente, deve-se prover uma recompensa ou penalidade para indicar o desejo pelo resultado. Por exemplo, quando um aprendiz em treinamento joga um jogo qualquer, o treinador dará uma recompensa positiva caso aquele ganhe, negativa caso perca ou nula em qualquer outro caso. A tarefa do agente é aprender, a partir desta recompensa, a escolher sequências de ações que produzam o maior acúmulo de recompensas. Os algoritmos de aprendizado por reforço estão relacionados com algoritmos de programação dinâmica, frequentemente usados para solucionar problemas de otimização. Entretanto, como o aprendizado por reforço se dá por meio de repetidas interações do agente com o ambiente, o processo de aprendizado é muito lento.

Para demonstrar a eficácia desta proposta, um robô será utilizado neste pro-

jeto, como aprendiz, em tarefas de navegação em ambientes internos a edifícios. O robô, ou agente, possui um conjunto de sensores para observar o *estado* atual do ambiente e os reforços a respeito da evolução de sua execução da tarefa. Para atuar, possui um conjunto de *ações* para executar, de forma que altere o estado do ambiente. Por exemplo, o robô móvel terá sensores como câmeras, hodômetros e sonares e ações como “ir para frente” e “girar”. A tarefa do robô é aprender uma *política* de atuação para escolha de ações que atinjam as metas, maximizando os reforços que recebe.

Uma razão pela qual o aprendizado por reforço é popular é que este serve como ferramenta teórica para estudo de princípios de agentes aprendendo a atuar. Mas não é nenhuma surpresa que ele foi usado também por um grande número de pesquisadores como ferramenta computacional prática para a construção de sistemas autônomos que se auto- otimizam com a experiência. Estas aplicações têm alcançado não só robôs mas também a produção industrial e jogos de computadores.

2 APRENDIZADO POR REFORÇO

O objetivo desta seção é apresentar o problema do aprendizado por reforço que deve ser solucionado. Para isso a escolha por qual modelo a ser seguido deve ser desenvolvida a partir da estrutura matemática que envolve o problema.

O desafio principal é o aprendizado de um *agente* a partir da interação com todo o ambiente externo, atingindo uma meta final. A interação do agente com o ambiente é contínua e este sempre deve responder às *ações* daquele apresentando uma nova situação ao agente. Dessa forma, o ambiente concede *recompensas* ao agente, que serão representadas por valores numéricos que o agente deve acumular durante o tempo de uma *tarefa*.

No modelo padrão do aprendizado por reforço, o agente interage com o meio externo através de percepções e ações. A Figura 1 ilustra esta interação, mostrando que o agente recebe uma entrada, *i*, como indicação do estado atual, *s*, do ambiente; o agente em seguida toma uma ação, *a*. Esta ação então muda o estado do ambiente, e o valor obtido desta transição é comunicado ao agente através do *sinal de reforço*, *r*. O comportamento do agente, B, deve ser de escolher ações que tendem a aumentar a soma de valores a longo prazo do sinal de reforço.

Desta forma o modelo consiste de:

- Um conjunto discreto de estados, S;

- Um conjunto discreto de ações, A;
- Uma função de sinais de reforço;
- Uma função de transição de estados.

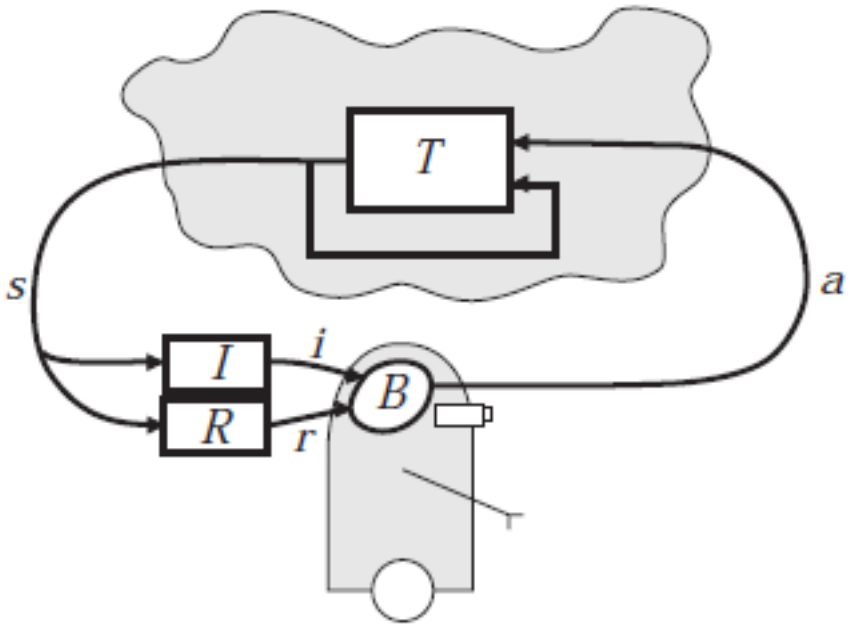

Figura 1: Modelo padrão de Aprendizado por Reforço – neste modelo tem-se o agente B interagindo com o ambiente T; o agente percebe o estado atual s através da entrada i, executa uma ação a, observa o novo estado e recebe uma recompensa r.

O objetivo do agente é encontrar a política que otimize uma medida de utilidade em função do reforço. A função alvo a ser aprendida é uma política de controle $\pi : S \rightarrow A$ que gera uma ação a apropriada do conjunto A, dado um estado atual s do conjunto S. É necessário que o agente colete experiências úteis sobre possíveis estados, ações, transições e recompensas do sistema para agir de forma ótima. Os métodos de aprendizado por reforço ditam como essa política π deve mudar conforme os resultados da experiência do agente são contabilizados.

Neste documento ainda consideramos que o agente não sabe quantas ações irá realizar até chegar ao estado final, ou seja, o agente espera eventualmente

parar, porém não sabe quando isso irá acontecer; tal abordagem é denominada *horizonte infinito*.

A seguir é apresentado o conceito de *Processos de Decisão de Markov* (PDM), que pode ser utilizado para compreensão e solução do modelo matemático para aprendizado por reforço.

2.1 Processos de Decisão de Markov

Quando um agente está em um estado *s* em um passo *t*, ele deve prever qual ação tomar caso este não seja um estado terminal. Se esta decisão sobre qual ação *a* tomar for independente dos estados anteriores, ações anteriores ou recompensas anteriores, então pode-se modelar o problema do aprendizado como um Processo de Decisão de Markov. O conceito desta abordagem é muito importante em métodos de aprendizado por reforço, pois o agente deve aprender seu comportamento com base nas informações do estado em que se encontra. Muitas vezes, mesmo que o ambiente se distancie dessa configuração de estados discretos, uma aproximação é adotada de forma a encaixar o ambiente no modelo de Markov. Uma forma de realizar essa aproximação é através da discretização dos estados do ambiente, que será apresentada em breve neste documento.

2.2 Política ótima

Os questionamentos sobre como o agente obtém a sua política π, como o agente obtém a sua política π* e como comparar políticas, ainda persistem.

A maximização do reforço acumulado ou recompensa esperada em longo prazo é uma ideia que deve ser mais bem estruturada. A sequência de recompensas, $r_t, r_{t+1}, r_{t+2}, \ldots$,recebidas pelo agente durante os passos executados nos

instantes $t, t+1, t+2, \ldots$ definem o *retorno esperado* pelo agente, e é essa função específica R_t que o agente deve maximizar. Portanto o agente pretende maximizar a esperança de reforços. A equação 2.1 apresenta esta somatória aplicada ao caso de interesse deste documento: o horizonte infinito ou tarefas contínuas; essa abordagem utiliza o caso de somatória infinita de recompensas futuras.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.1}$$

Pra compreensão total desta abordagem existe a necessidade de consideração do *fator de desconto* (γ) na equação, que é o parâmetro responsável por ponderar a participação de recompensas futuras no aprendizado do agente. Sendo $0 \leq \gamma \leq 1$, quanto mais próximo o γ de 1, mais o agente dá importância às recompensas futuras.

Ao valor esperado de R_t , para um dado s_t, dá-se o nome de *valor-estado*, que diz qual o valor de recompensa esperado para aquele estado. A função valor-estado $V^{\pi}(s)$ (2.2) é que define este valor de recompensa esperada para cada estado do *s* do ambiente seguindo a política π.

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right\}. \tag{2.2}$$

Como o valor-estado considera apenas a ação escolhida pela política π, e dado que a busca pela política ótima é um processo que considera as recompensas obtidas a partir de ações diversas, é interessante saber qual o retorno esperado ao tomar-se uma ação específica *a* em um estado *s*, retorno este conhecido como *valor-ação*. A função valor-ação $Q^{\pi}(s,a)$ é que define o valor de retorno para cada par (*s*,*a*) seguindo a política π:

$$V^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right\}. \tag{2.3}$$

Dadas tais funções, a afirmação de que a política ótima é aquela que leva ao

maior acúmulo de recompensas pode ser alterada para: a política ótima é aquela obtida a partir da função valor-estado ótima $V^*(s)$ e da função valor-ação ótima $Q^*(s,a)$. Esta ideia ainda leva ao fato de que a função valor-estado nada mais é do que a função valor-ação definida sobre a ação que atingem o valor máximo de $Q^*(s,a)$ em um estado *s*, ou seja, seguindo a política ótima:

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a). \tag{2.4}$$

Para que a política ótima $\pi^*(s)$ seja obtida devemos então obter os maiores valor- estado e valor-ação. A expressão matemática (2.5), que auxilia a compreensão desta nova definição de qual política deve ser alcançada com o aprendizado, deve então levar em consideração uma ação como argumento que maximize o valor-ação, ou seja, para um determinado estado *s*, qual ação será responsável pelo retorno máximo:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a). \tag{2.5}$$

Este artigo considera problemas onde as ações podem ter consequências não determinísticas e onde aquele que aprende não domina a teoria que descreve as consequências de suas ações. O aspecto mais atual sobre aprendizado por reforço assume que um agente não possui conhecimento da função transição de estado e função recompensa, e que ao invés de se mover baseado em um modelo ideal do estado do espaço, ele se move sobre o mundo real e observa as consequências. Neste caso, a primeira preocupação é o número de ações do mundo real que o agente deve realizar para convergir a uma política aceitável, ao invés do número de ciclos computacionais que deve gastar. A razão para isso é que em muitos domínios práticos os custos em tempo e dinheiro de realizar tais ações no mundo externo dominam os custos computacionais.

2.3 Método de solução

Existem três classes de métodos de solução para o problema do aprendizado por reforço: programação dinâmica, métodos de Monte Carlo e aprendizado por diferença temporal. Cada classe de métodos possuem suas vantagens e desvantagens. A programação dinâmica é bem desenvolvida matematicamente, porém requer um modelo completo e preciso do ambiente. Os métodos de Monte Carlo não necessitam de um modelo e é conceitualmente simples, mas não é adaptado para computação incremental passo a passo. Finalmente, os métodos de aprendizagem por diferença temporal não precisam de um modelo e são incrementais, porém são complexos para análise. O relacionamento entre os 3 métodos é um tema recorrente na teoria de aprendizado por reforço.

2.3.1 Programação dinâmica

A programação dinâmica envolve algoritmos que visam obter a política ótima em um ambiente modelado como um Processo de Decisão de Markov. Os algoritmos de programação dinâmica, embora forneçam grande base teórica, teriam pouca utilidade no problema de aprendizado por reforço que será discutido neste artigo, pois exigem um modelo perfeito do ambiente, que raramente é obtido, e um grande custo computacional. O valor teórico da programação dinâmica é tão presente que os próximos métodos buscam alcançar o mesmo efeito sem um modelo e com menor custo computacional.

2.3.2 Métodos de Monte Carlo

Os métodos de Monte Carlo já se destacam por não precisarem de conhecimento completo do ambiente. Os métodos de Monte Carlo, no entanto, precisam de certa experiência do agente dentro do ambiente, ou seja, um conjunto de

ações, estados e recompensas provenientes de interação com o ambiente, sem ser um conjunto completo com todas as transições possíveis tal como a programação dinâmica exige, realizadas em episódios ou tarefas concluídos. Os métodos de Monte Carlo entendem a experiência como uma soma de episódios que geram um retorno médio.

2.3.3 Aprendizado por diferença temporal

Aprendizado por diferença temporal (DT) é uma combinação de ideias de métodos de Monte Carlo com programação dinâmica. Assim como métodos de Monte Carlo, métodos DT podem aprender diretamente com a experiência sem um modelo perfeito do ambiente. Assim como a programação dinâmica, métodos DT atualizam suas estimativas baseados em parte em outras estimativas aprendidas, sem esperar por um resultado final.

A ideia principal dos métodos DT é que a predição do agente para o estado atual se aproxime da predição do próximo estado que será percorrido, ou seja, os métodos DT devem fazer com que o agente possa decidir qual ação tomar, em um passo *t*, de forma semelhante à mesma decisão a ser tomada em um passo *t+1*. O agente deve então aprender a prever seu comportamento futuro utilizando experiência anterior, sendo o ambiente não inteiramente conhecido. Dada essa semelhança a ser atingida, os métodos DT possuem um erro associado à diferença entre as predições realizadas, erro este que, conforme será explicitado em breve, será responsável por guiar o aprendizado.

Os métodos de aprendizado por DT descrevem como o conhecimento acumulado é utilizado em um modo *livre de modelo*, sem modelo completo do ambiente, para obter a política ótima para a *recompensa futura acumulada*. O aprendizado por DT é dividido em dois problemas principais:

- Predição: este é o problema sobre a previsão da recompensa futura acumulada para a política π a partir do estado *s*, o que significa que é o problema de previsão da *função valor-estado* $V^{\pi}(s)$.

- Controle: este é o problema de aprendizado da recompensa futura acumulada devido à escolha da ação a partindo do estado *s* e seguindo a política π, ou seja, encontrar a *função valor-ação* $Q^{\pi}(s,a)$.

2.3.3.1 Predição DT

Enquanto os métodos de Monte Carlo precisam que um episódio seja finalizado para apenas então alterar os valores-estado, os métodos por diferença temporal apenas esperam até o próximo passo $t+1$ para tal atualização; neste passo, o agente deve realizar uma atualização de seu valor-estado $V(s_t)$ a partir da recompensa observada $r_t + 1$ e o valor-estado estimado de $V(s_{t+1})$ correspondente ao estado seguinte. A forma mais simples dos métodos por DT, conhecida como DT(0), leva em consideração não a recompensa total esperada até a meta final (R_t) mas sim a recompensa devida a um único passo:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]. \tag{2.6}$$

Sendo α a *taxa de aprendizagem*, que pondera a participação do novo estado no aprendizado do agente.

Conforme citado, a abordagem de predição por métodos de Monte Carlo precisam que uma tarefa seja finalizada para que então os valores de $V(s_t)$ sejam atualizados. A principal desvantagem desta abordagem está no fato de que o agente poderia, por exemplo, entrar em um círculo durante seu trajeto e jamais alcançar o destino final, portanto não utilizando o aprendizado adquirido durante o episódio de forma eficiente.

Algumas aplicações possuem ainda episódios muito longos, de forma que o atraso de todo o aprendizado pela espera do fim do episódio torna o método por DT mais adequado que por métodos de Monte Carlo. Além de casos em que as tarefas são contínuas e não possuem nenhum episódio explícito.

A resolução de predição DT pode ser aplicada no problema de controle dos métodos DT. Em seguida, analisam-se duas classes principais, *on-policy* e *off-policy*, para solução dos problemas de controle que se diferenciam quanto ao compromisso entre exploração e desbravamento do ambiente.

2.3.3.2 Off-Policy e Q-Learning

Em geral, os algoritmos de controle de métodos DT buscam a política ótima maximizando a função valor-ação.

É importante que grande parte do conhecimento do agente sobre o ambiente seja adquirido a partir de exploração deste, ou seja, é interessante que o agente desbrave o ambiente na busca de estados com maior retorno futuro. Os algoritmos de aprendizado por reforço podem ser divididos entre aqueles que estimam π^* seguindo uma política π (aprendizado *off-policy*) e os que estimam π^* atualizando a própria política π (aprendizado *on-policy*). Basicamente isto diz que métodos *on-policy* também levam em consideração o custoda exploração na atualização de Q, diferentemente do métodos *off-policy*.

O algoritmo de *Q-Learning* busca a otimização de $Q(s,a)$ da mesma forma que a predição DT o faz com para a função valor-estado $V(s)$. Pela expressão (2.7), a política ótima é obtida a partir do valor-ação aprendido, sem atualizar a política que está sendo adotada. Observa-se claramente o aspecto incremental presente na otimização de $Q(s,a)$; o algoritmo de Q-Learning tem por base justamente a iteração dos valores-ação. Dessa forma, primeiramente os valores-ação para cada par (s,a) são inicializados arbitrariamente. Em seguida, a partir de um

estado inicial *s* qualquer do agente a tarefa se inicia, e para cada ação *a* escolhida os valores-ação devem ser atualizados, até o momento que um estado final é atingido.

$$Q^*(s_t, a_t) \leftarrow Q(s_t, a_t) + a\left[r_{t+1} + \gamma \underset{a}{max}\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \tag{2.7}$$

Algorithm 1 Q-Learning

```
Inicializa Q(s,a) arbitrariamente
for cada episódio do
    Inicializa s
    for cada passo do episódio do até s ser terminal
        Escolha a de s usando a política derivada de Q (e.g., ε-gananciosa)
        Toma ação a, observe r, s'
        Q(s,a) ← Q(s,a) + α[r + γmax_a' Q(s',a') − Q(s,a)]
        s ← s'
    end for
end for
```

2.3.3.3 On-Policy SARSA-Learning

Assim como o algoritmo Q-Learning, o algoritmo Sarsa deve aprender uma função valor-ação de forma semelhante ao processo realizado em predição por DT para a função valor-estado. A figura 2 ilustra a configuração dos pares (s_t, a_t) em diferentes passos de uma tarefa.

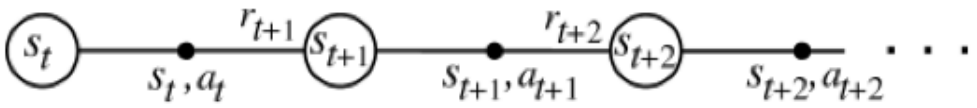

Figura 2: Abstração utilizada para compreensão de onde exatamente encontram-se o valor-estado e o valor-ação.

A atualização da função-valor $Q(s, a)$ apresentada em (2.8) difere daquela apresentada para o algoritmo Q-Learning pois o algoritmo SARSA deve atualizar a própria política que segue; em outras palavras, ao invés de escolher o melhor

valor futuro de $Q(s_{t+1}, a_{t+1})$, o algoritmo SARSA obtém a_{t+1} pela política π sendo seguida. O nome SARSA origina-se da tupla $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ utilizada para a atualização dos valores-ação.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{2.8}$$

Algorithm 2 SARSA

Inicializa $Q(s, a)$ arbitrariamente
for cada episódio **do**
    Inicializa s
    for cada passo do episódio **do** até s ser terminal
        Toma ação a, observe r, s'
        Escolha a' de s' usando a política derivada de Q (e.g., ε-gananciosa)
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'; a \leftarrow a'$
    end for
end for

Pelo conteúdo apresentado sobre o algoritmo SARSA, quando o agente chega a um estado com uma determinada recompensa, o único valor-ação $Q(s, a)$ que é atualizado é o valor referente ao par (s, a) que originou tal reforço, ou seja, o reforço será propagado para um único estado. Este fato implica que agente poderia *explorar* um mesmo caminho, de forma a distribuir um reforço significativo do estado final para os estados percorridos, ao invés de *desbravar* o ambiente na busca de novos caminhos até atingir a convergência a uma política ótima. Considerando este caso específico, os métodos de Monte Carlo aprenderiam a política ótima mais rapidamente por atualizarem todo o caminho ao fim de um episódio. O conceito de *traço de elegibilidade* será então apresentado como forma de mesclar métodos de Monte Carlo com os métodos DT.

2.3.4 Traços de elegibilidade

Os traços de elegibilidade representam um mecanismo a ser associado com os algoritmos Q-Learning e SARSA de forma a obter um método de aprendizado por reforço mais eficiente.

Existem duas formas principais de abordagem para compreensão de traços de elegibilidade. Uma dessas formas é utilizada para ligar os métodos de Monte Carlo aos métodos por Diferença Temporal, unindo as vantagens alcançadas por ambos os métodos. Esta visão será tratada neste artigo como *visão adiante*.

A outra forma de abordagem possui ênfase mecânica. Por esta visão, os traços de elegibilidade registram em memória a visita a um estado ou a tomada de uma ação com o intuito de tornar o evento ocorrido elegível de mudança devido ao resultado da análise por DT. Dessa forma, os traços de elegibilidade definem quais valores-estado ou valores-ação devem ser mais ou menos atingidos pelas atualizações do método DT. Esta abordagem será tratada neste documento como *visão retrógrada*.

É importante ressaltar que as duas visões sobre o mecanismo de traços de elegibilidade são equivalentes e são utilizadas com o objetivo de esclarecer o uso dos traços. Enquanto a visão adiante é usada para melhor compreensão do que exatamente é computado pelo algoritmo, a visão retrógrada é adotada para desenvolver a intuição utilizada pelo algoritmo de aprendizado.

Primeiramente será abordado o problema da predição de reforço com base na função valor-estado. Em seguida, as duas visões sobre os traços de elegibilidade serão detalhadas para compreensão do problema de controle, que se baseia na função valor-ação (SUTTON; BARTO, 1998).

2.3.4.1 Predição por DT de n passos

Os métodos de Monte Carlo, para o problema da predição, consideram os valores- estado de todos os estados que o agente percorre em um episódio no momento de atualizar a função valor-estado $V(s)$. Os métodos por DT, da forma como foram apresentados, atualizam a função valor-estado com base apenas em dois estados observados: s e s_{t+1}; para essa abordagem, considera-se o método por DT como sendo de 1 passo. A ideia para ligar os métodos por DT até os métodos de Monte Carlo é aumentar o número de passos dos métodos por DT até n, sendo n menor ou igual ao número de passos necessários para chegar ao fim do episódio. A figura 3 ilustra essa ideia.

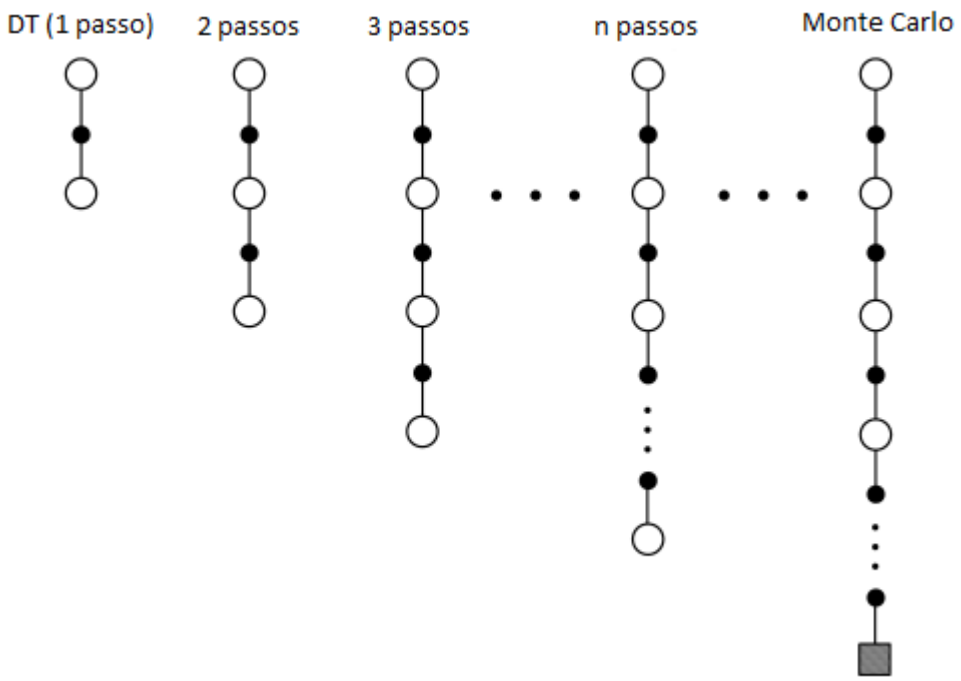

Figura 3: Modelo formado por estados e transições considerados em métodos por DT de 1 único passo até n passos para a predição por DT (SUTTON; BARTO, 1998).

Ao reforço devido ao último passo de uma configuração escolhida entre 1 passo ou n passos dá-se o nome de *alvo*. Para os métodos de Monte Carlo, o alvo escolhido é o retorno final do episódio. Para os métodos por DT de 1 passo, o

alvo é o retorno esperado considerando apenas um passo; a equação (2.9) apresenta o valor de retorno esperado (2.1) calculado em um único passo.

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}) \tag{2.9}$$

Esta ideia pode então ser generalizada chegando ao alvo decorrente de n passos (2.10).

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}). \tag{2.10}$$

Para conclusão da predição por DT de n passos, o alvo agora deve ser incluído na atualização dos valores-estado. A partir da configuração de n passos adotada, o valor de $V(s_t)$, diferentemente da expressão utilizada para 1 passo (2.6), será incrementado utilizando $R_t^{(n)}$ e não mais $R_t^{(1)}$. A expressão (2.11) representa o incremento a ser realizado no valor-estado de um estado *s*, sendo $\Delta V_t(s) = 0$ para todo $s \neq st$. Diferentemente do método de atualização on-line que ocorre na expressão (2.6), em que a atualização $V_{t+1}(s) = V_t(s) + \Delta V_t(s)$ ocorre a cada novo instante *t*, a expressão (2.11) representa a atualização *off-line* para o método de predição por DT de n passos; neste caso, a alteração do valor-estado do estado *s* apenas será efetuado ao final de um episódio, sendo que o novo valor-estado, para o próximo episódio, será $V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$ (SUTTON; BARTO, 1998).

$$\Delta V_t(s_t) = \alpha[R_t^n - V_t(s_t)]. \tag{2.11}$$

2.3.4.2 A visão diante de DT(λ)

O retorno escolhido pode também ser obtido a partir da média de retornos de n passos. Por exemplo, o alvo pode ser obtido com metade do retorno de 2 passos e metade do retorno de 4 passos, como mostra a expressão (2.12). Essa é a ideia para métodos por DT(λ), sendo λ o parâmetro utilizado para ponderar a

participação de cada uma das configurações de n passos.

$$R_t^{avg} = \frac{1}{2}R_t^2 + \frac{1}{2}R_t^4. \tag{2.12}$$

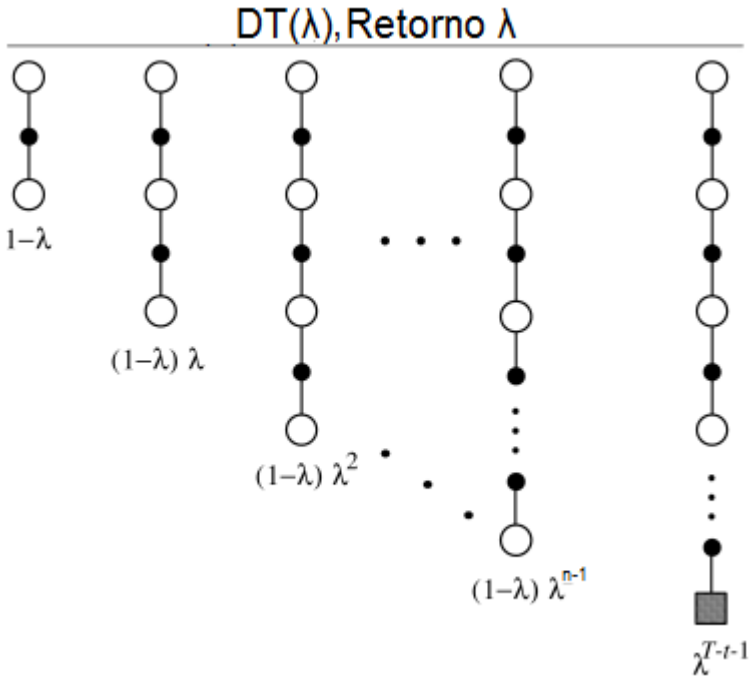

Figura 4: Diagrama de métodos por DT(λ) com atribuição de pesos a cada estrutura de n passos.

A figura 4 ilustra como os pesos são designados a cada estrutura. Cada configuração possui um peso proporcional a λ^{n-1} e o fator de normalização $(1 - \lambda)$ garante que a somatória de todos os pesos resulte em 1. A estrutura resultante visa um retorno $R_t\lambda$ chamado de *retorno* λ, definido pela expressão (2.13), que leva em conta todos os retornos de todas as configurações de n passos; T é o último passo, ou seja, o momento em que o agente atinge o estado terminal. Por esta expressão entende-se que o parâmetro λ aproxima a predição por DT dos métodos de Monte Carlo à medida que λ se aproxima de 1; se $\lambda = 1$, o retorno λ é idêntico ao retorno obtido considerando-se todos os passos de um episódio.

$$R_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t. \tag{2.13}$$

Dado esse novo conceito de retorno λ, a equação (2.11), que representa a atualização dos valores-estado, pode ser alterada para a expressão (2.14) que considera a média de retornos ao invés de um número específico n de passos.

$$\Delta V_t(s_t) = \alpha[R_t^{\lambda} - V_t(s_t)]. \tag{2.14}$$

A visão adiante por DT(λ) é assim denominada pois, o agente deve sempre olhar os passos futuros, visualizar as recompensas esperadas e atribuir pesos a elas para atualização de seu estado atual. Dessa forma, a atualização de um estado é realizada apenas uma vez, porém considerando o panorama futuro; após atualizar um estado e mover para o próximo passo, o agente não mais precisa se preocupar com o passado. Apenas estados futuros devem ser sempre verificados e reprocessados. A figura 5 ilustra a abstração aqui apresentada de como o agente observa o ambiente e eventos futuros (SUTTON; BARTO, 1998).

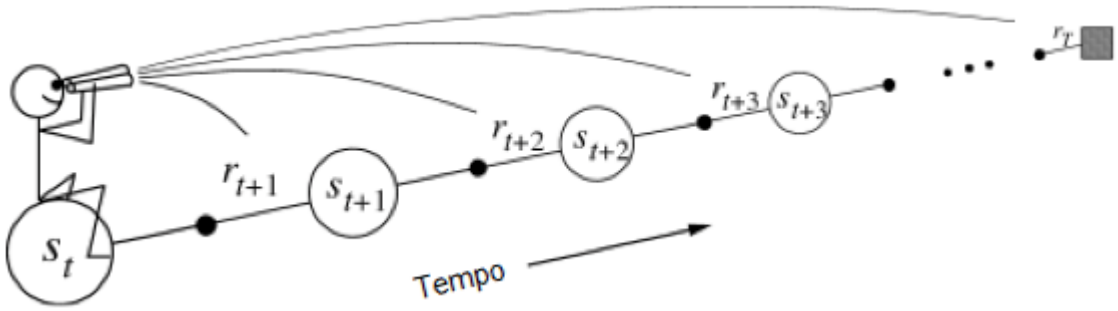

Figura 5: Ilustração de como funciona a visão adiante – o agente deve olhar apenas o panorama futuro para suas considerações atuais.

2.3.4.3 A visão retrógrada de DT(λ)

Na seção 2.3.4.2 foi apresentado como as diversas estruturas de n passos podem ser combinadas para aproximar os métodos por DT dos métodos Monte Carlo. Nesta seção será definida a visão retrógrada do algoritmo por DT(λ) que não utiliza a cada passo o que acontecerá nos passos futuros. Nesta visão, será utilizado o conceito de traço de elegibilidade introduzido na seção 2.3.4. O traço de

elegibilidade para cada estado *s* em um instante *t* é denotado por $e_t(s)$. Conforme mencionado na seção 2.3.4, o traço de elegibilidade define indiretamente quanto um valor-estado ou valor-ação deve ser incrementado após a análise por DT.

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1} & \text{se } s \neq s_t; \\ \gamma\lambda e_{t-1} + 1 & \text{se } s = s_t. \end{cases} \tag{2.15}$$

Os valores de $e_t(s)$ são definidos conforme a equação (2.15). Por esta equação, compreende-se que o traço de elegibilidade de cada estado decai por um fator $\gamma\lambda$ (sendo γ a taxa de desconto e λ o parâmetro utilizado na seção 2.3.4.2 para ponderar as estruturas de n passos) para cada passo executado pelo agente e que caso um estado seja revisitado seu traço deve ser incrementado em 1. Dessa forma, para grandes valores de λ, ainda menores do que 1, mais estados anteriores são alterados, mas cada um daqueles temporalmente mais distantes são modificados em menor escala, pois seu traço de elegibilidade é menor. Dizemos que estes últimos estados são menos creditados para o erro DT. A figura 6 mostra como o traço de elegibilidade varia para um estado com o tempo e com o número de visitas a este estado.

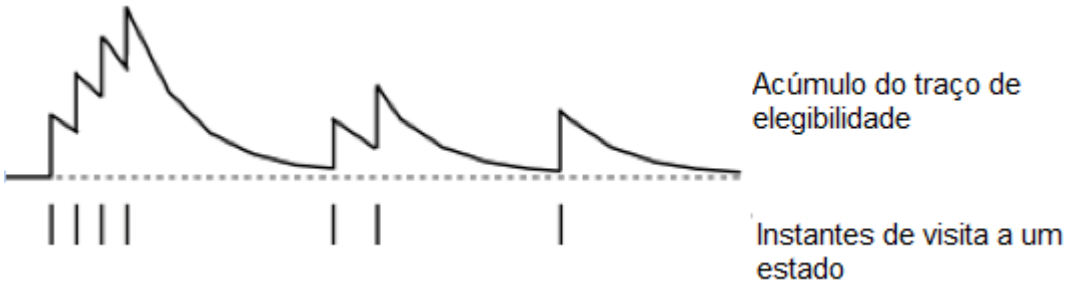

Figura 6: Gráfico explicativo sobre como o valor do traço de elegibilidade varia caso ocorram visitas aum mesmo estado.

Ainda resta associar o traço de elegibilidade com a atualização de um valor-estado. Diferentemente da visão adiante, na visão retrógrada não se utiliza a média de retornos das estruturas de n passos para a atualização do valor-estado. Para a visão retrógrada, adicionamos o traço de elegibilidade ao cálculo do incre-

mento do valor-estado. As equações (2.16) e (2.17) mostram como, a cada passo, todos os estados recentemente visitados devem ter seu respectivo valor-estado alterado em ΔV_t.

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) = R_t^{(1)} - V_t(s_t). \tag{2.16}$$

$$\Delta V_t(s) = \alpha \delta_t e_t(s). \tag{2.17}$$

O mecanismo da visão retrógrada é basicamente definido da seguinte forma: em δ_t um instante *t*, verifica-se o valor de do estado atual e calcula-se o valor de et para todos os estados até então visitados; em seguida, atualiza-se o valor-estado de todos os estados pelos quais o agente já passou e realiza um novo passo. A figura 7 ilustra como o agente se comunica com os estados já visitados para atualização de valores-estado (SUTTON; BARTO, 1998).

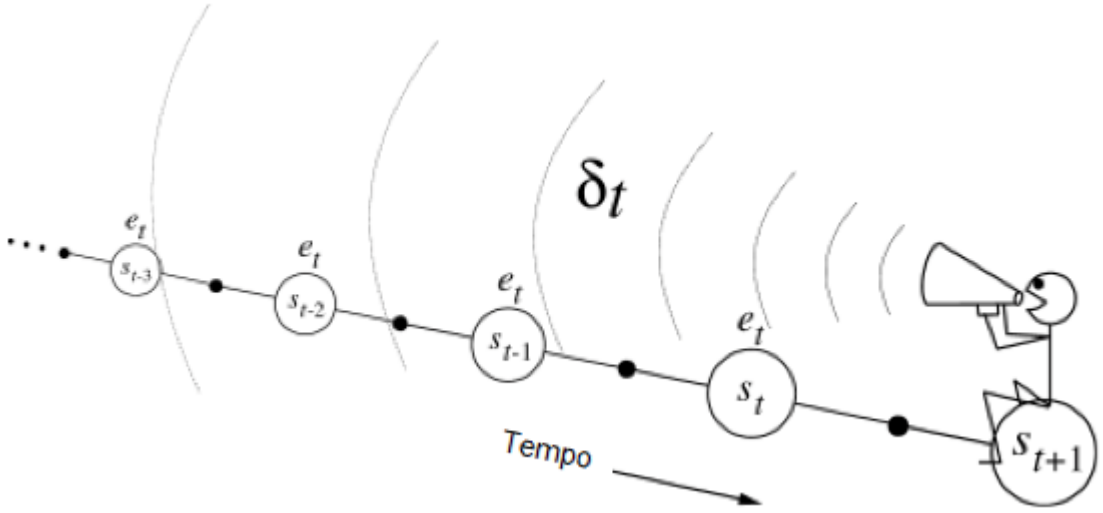

Figura 7: Ilustração de como funciona a visão retrógrada de métodos por DT – o agente deve passar aos estados anteriores a sua situação atual.

Na próxima seção será apresentado como os traços de elegibilidade podem ser combinados de forma direta com o algoritmo SARSA para resolver o problema de controle em métodos por DT(λ), ou seja, como atualizar valores-ação de forma

a atingir a função valor-ação ótima.

2.3.5 SARSA(λ)

Conforme apresentado na seção 2.3.3.3, o algoritmo SARSA é um método que soluciona o problema de controle em métodos por DT; esta versão que foi apresentada é destinada à análise a partir de um único passo. À versão do algoritmo SARSA que utiliza o traço de elegibilidade como forma de aproximar os métodos por DT dos métodos de Monte Carlo dá-se o nome de SARSA(λ).

A ideia é estender o procedimento adotado pelo problema de predição por DT(λ), que se preocupava com a atualização dos valores-estado, até os pares estado-ação (valores-ação). Dessa forma, o par estado-ação é que terá a ele associado um valor de traço de elegibilidade denotado por $e_t(s,a)$. As equações (2.19) e (2.18) se assemelham muito às equações (2.16) e (2.17), diferenciando-se apenas pela substituição necessária de valores-estado por valores- ação e troca do traço de elegibilidade referente ao estado pelo traço referente ao par estado-ação (s,a).

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a). \tag{2.18}$$

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t). \tag{2.19}$$

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{se } s = s_t \text{ e } a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{Caso contrário.} \end{cases} \tag{2.20}$$

O algoritmo SARSA(λ) busca alcançar a função valor-ação $Q^*(s,a)$ ótima a partir de atualizações sucessivas de $Q^\pi(s,a)$, portanto seguindo uma política π. Antes do início de um episódio, os valores de $Q(s,a)$ para cada par estado-ação conhecido podem ser definidos aleatoriamente, enquanto que os valores de $e_t(s,a)$

devem ser iniciados com 0 dado que nenhum estado foi visitado até então. Para que um episódio seja iniciado um par (s,a) deve ser apontado, ou seja, um estado inicial deve ser escolhido e uma ação a, definida pela política seguida, deve ser tomada. Em seguida, a partir desta ação o agente deve identificar o reforço recebido, o novo estado e a próxima ação. O algoritmo (3) apresenta o processo iterativo que deve ser realizado para atualizado dos valores de $Q(s,a)$ a cada passo de um episódio, a partir dos valores apresentados.

Algorithm 3 SARSA-λ

Inicializa $Q(s,a)$ arbitrariamente
for cada episódio **do**
    Inicializa s, a
    for cada passo do episódio **do** até s ser terminal
        Toma ação a, observe r, s'
        Escolha a' de s' usando a política derivada de Q (e.g., ε-gananciosa)
        $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
        $e(s,a) \leftarrow e(s,a) + 1$
        for todo s,a **do**
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
            $e(s,a) \leftarrow \gamma\lambda e(s,a)$
        end for
        $s \leftarrow s'; a \leftarrow a'$
    end for
end for

3 CMAC

Com a evolução das tecnologias voltadas ao estudo biológico, e principalmente do sistema nervoso, houve uma grande compreensão recente do cerebelo humano. Baseado nesses estudos, foi baseado um modelo de processamento de dados relativamente simples (primeiramente desenvolvida por Albus em 1971), porém com ótima capacidade de processamento. Tal modelo é muito utilizado na área de Inteligência Artificial e até em alguns trabalhos tratando com Aprendizado por Reforço, servindo como método para discretização de funções contínuas - como as informações são encontradas em ambientes reais. Tal modelo é conhecido como CMAC (Cerebellar Model Articulation Controller) e se baseia em um mapa de ladrilhos, montados a partir da tradução das informações de entrada em "fibras mucosas", e em seguida em "células granulares", seguindo o modelo de cerebelo biológico. Uma das razões para o uso de redes neurais para problemas com Aprendizado por Reforço é sua capacidade de generalização em situações e ações semelhantes. A generalização é consequência de, ao se ir para algum estado da matriz de valores de acordo com a entrada s, na verdade selecionamos um hipercubo com com os ladrilhos (e seus respectivos valores) que o cercam, "suavizando"a tomada de ação.

3.1 Associação no CMAC

As entradas si pertencentes ao espaço S são normalmente multidimensionais, sendo cada s_i pertencente no intervalo $[\underset{min}{s_i}, \underset{max}{s_i}]$ discretizada com uma precisão ε a ser definida (normalmente essa informação tem origem empírica) e depois cada entrada s_i é mapeadas por Q hipercubos diferentes. Cada hipercubo se situa frente ao outro, como mostra a figura 8. Dessa forma mapeamos todas as entradas s_i nesses Q hipercubos.

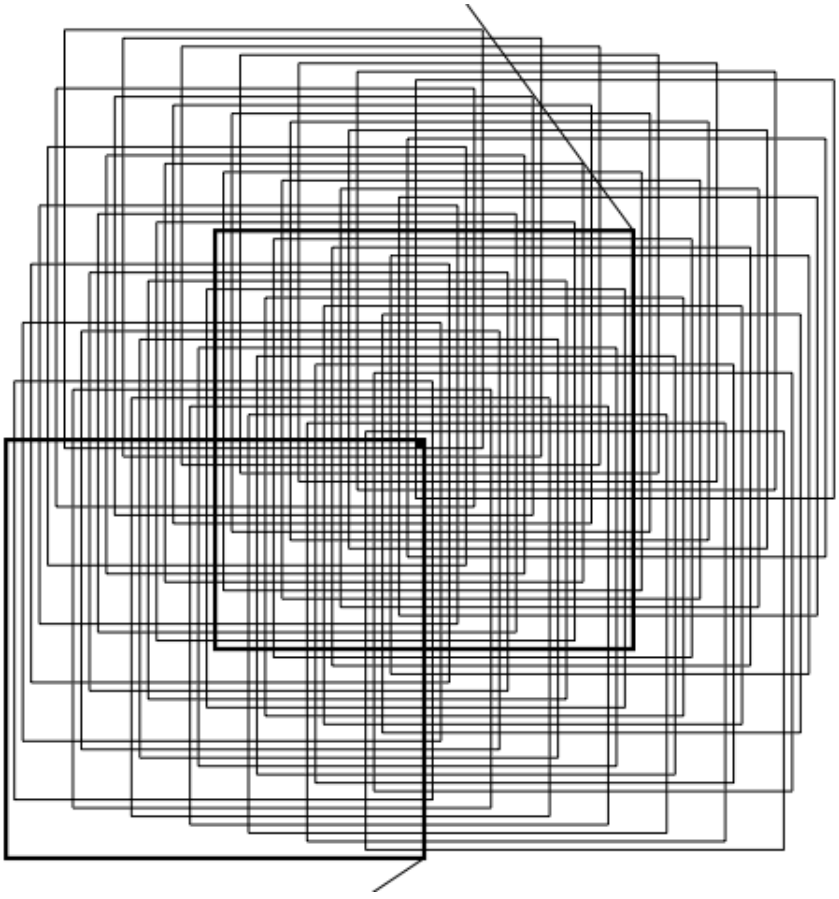

Figura 8: Exemplo da disposição de hipercubos. Com 2 entradas e Q=50.

Cada ladrilho contém o peso correspondente da entrada si no hipercubo Qi (peso sináptico), sendo que o valor total referente a tal entrada é a soma dos pesos de cada hipercubo associado a essa entrada, como exemplifica a figura 9.

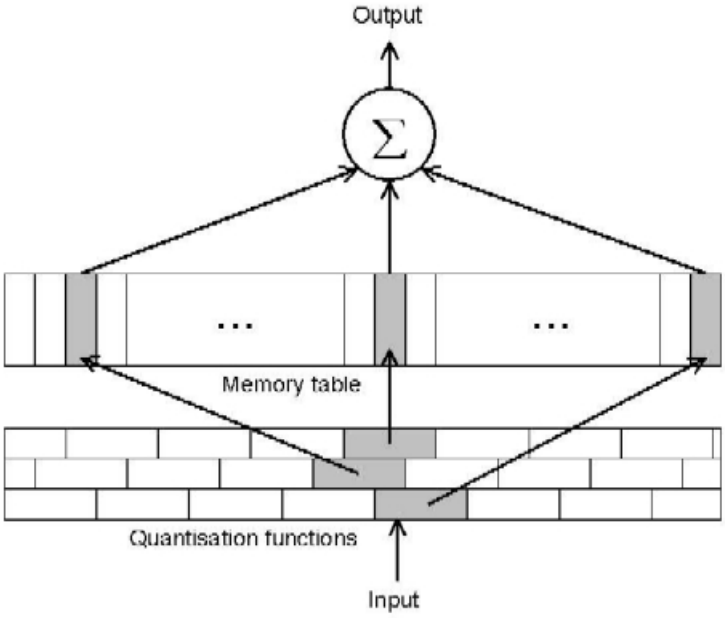

Figura 9: Visão de "cima" dos hipercubos ($Q = 3$), para o caso de uma variável unidirecional de entrada, onde cada ladrilho ativo pela entrada está mapeado em uma tabela de memória com seu peso.

3.2 Modelo geral para CMAC

3.2.1 Mapeando entradas (pertencentes a S) em fibras mucosas (pertencentes a M)

Cada entrada s^i é codificada no conjunto M de fibras, onde cada fibra é dividida em intervalos I_{ij}, onde *i* é o índice da entrada e *j* o índice da fibra. Dá-se o intervalo $I_{ij} = [j^*\varepsilon - Q^*\varepsilon; j^*\varepsilon]$, onde Q é a distância entre as fibras (distância entre os hipercubos). Já que cada intervalo I_{ij}, I_{ij+1} estão deslocados de ε e a distância dos intervalos é $Q^*\varepsilon$, cada si estará contido em exatos *Q* intervalos consecutivos, e a função de ativação da fibra será dada como:

$$act_{ij}(s_i) := \begin{cases} 1 & \text{se } s_i \in I_{ij}; \\ 0 & \text{caso contrário.} \end{cases} \tag{3.1}$$

Cada entrada s_i será codificada como o conjunto de fibras *j* pertence a M com $act_{ij} = 1$. Para se aproveitar a função de generalização, podemos alterar a

função *act* para utilizar uma curva gaussiana que reflete seu resultado para células vizinhas:

$$act_{ij}(s_i) := \begin{cases} gauss_{ij}(s_i) & \text{se } gauss_{ij}(s_i) \geq a_{min}; \\ 0 & \text{caso contrário.} \end{cases} \tag{3.2}$$

$$\textit{Com } \; gauss_{ij}(s_i) = exp\left(-\frac{1}{2}\left(\frac{s_i - \mu_{ij}}{\sigma_{ij}}\right)^2\right).$$

Onde sigma e amin devem ser satisfatoriamente escolhidos e miij é designado o centro do intervalo I_{ij}.

3.2.2 Mapeando fibras (pertencentes a M) em células (pertencentes a A)

Cada célula *v* é designada unicamente pela tupla de números *ji*, onde *j* é a fibra mucosa na dimensão *i*. A ativação a_v da célula *v* é dada como o produto de ativações das fibras referentes à célula:

$$a_v = act_v(\vec{s}) \; := \; \prod_{i=1}^{n} act_{ij_i}(s_i) = \begin{cases} 1 & \text{se } \vec{s} \in I_{1j_1} \times \ldots \times I_{nj_n}; \\ 0 & \text{caso contrário.} \end{cases} \tag{3.3}$$

4 NAVEGAÇÃO ROBÓTICA

Para implementação das técnicas de aprendizagem por reforço, o trabalho pretende utilizar robôs que procuram uma trajetória até um alvo fixo com obstáculos para as comprovações dos testes. Seu sucesso ou fracasso ao atingir o alvo é que moldará suas futuras políticas de trajetória, por isso o domínio do manuseio de robôs torna-se imprescindível no trabalho.

Antes de partir para o robô real, o desenvolvimento se passa com simuladores de navegações robóticas. Primeiro para compreender melhor o funcionamento do robô e como utilizar a metodologia de aprendizado por reforço no mundo físico, representado pela navegação robótica. Segundo, por ser é mais prático, rápido e menos custoso trabalharmos com simuladores, descartando erros e interferências do mundo real nesse primeiro modelo, economizando tempo com a flexibilização da manutenção dos códigos e evitando falhas mais graves que possam danificar o equipamento. Em um segundo momento, com um modelo melhor definido, partiremos para um robô real, no caso o já definido Pioneer 2DX.

4.1 ROS

O ROS (Robot Operating System) é um simulador de navegação robótica open-source que fornece bibliotecas e ferramentas para o desenvolvimento de aplicações para robôs. Com uma abstração do hardware, de seus drivers, bibliotecas em um alto nível (totalmente lógico), o ROS possibilita ter o total controle

do robô e do ambiente de sua atuação: criação de mapas, de robôs, definição dos sensores, scripts de políticas de interpretação de sensores e comando para atuações.

O modo de funcionamento do ROS é um grande passo para o avanço desse trabalho na prática. Ele consiste em um simulador, ou seja, uma aplicação que fornece o ambiente e ferramentas para desenvolver tudo relativo ao robô e sua navegação em um ambiente. Nesse simulador possuímos diversos pacotes que auxilia na simulação de diversos tipos de sensores também, como sonares, hodômetros, câmeras, entre outros.

Com o core da aplicação do ROS em funcionamento, podemos criar servidores e clientes virtuais, capazes de se comunicar e criar a simulação. O servidor virtual criado pelo simulador é responsável por capturar mensagens (parâmetros como magnitude, direção, velocidade, etc.) e transferir para uma atuação do robô. O cliente produz essas mensagens e as publica em um canal (chamado de tópico, de acordo com a documentação do ROS) responsável por conectar o cliente com o servidor. O ROS nos dá bastante liberdade para programar scripts (em C++ ou em Python) que fazem o papel de cliente e de servidor no simulador, assim como ferramentas para monitorar a troca dessas mensagens (como a taxa que essas mensagens são publicadas non servidor, registro de todo log da simulação, monitores gráficos das variáveis trabalhadas). Através dessa funcionalidade e desse ambiente que os algoritmos de aprendizado por reforço são aplicados.

4.1.1 Mensagens e servidores

As mensagens e servidores são descritos em arquivos de texto simples, como dito anteriormente. No caso dos arquivos de mensagens, são arquivos que descrevem os campos de uma mensagem no ROS (figura 10a). Os arquivos de servidores não são tão diferentes dos de mensagens, a diferença básica é que são

divididos em parâmetros de resquest e parâmetros de response, mas também tem como finalidade a descrição dos campos do ROS para comunicação (figura 10b).

```
1  string first_name
2  string last_name
3  uint8 age
4  uint32 score
5
```

(a) Mensagem

```
1  int64 a
2  int64 b
3  ---
4  int64 sum
5
```

(b) Servidor

Figura 10: Descrição dos campos para comunicação

4.1.2 Scripts

A criação de scripts pode ser feitas tanto em Python quanto em C++, usando os pacotes rospy e roscpp respectivamente. Essa é uma poderosa funcionalidade, já que nos possibilita a criação de programas bem estruturados manuseando as propriedades dos robôs. O exemplo mostrado na figura 11 mostra um trecho de um código em C++ que movimenta o robô através da publicação de uma mensagem. Basicamente é importada as bibliotecas básicas do ROS, o programa principal as usa para criar nós (terminais para manuseio do ROS), criar as mensagens enviadas para o robô, handlers para publicação da mensagem no nó, entre outros detalhes de parametrização, como a taxa que as mensagens serão publicadas.

Todo código feito tem que ser declarado para ser tratado como executável, assim após a compilação poderemos rodar o simulador sob o comportamento programado no código, como mostra a Figura 12.

4.2 Definição de estados

Pensando em usar a robótica para abordar o aprendizado por reforço, nos deparamos com uma problemática: como definir um estado a partir das leituras de sensores.

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"

#include <sstream>

/**
 * Esse tutorial mostra um simples envio de mensagens através do ROS.
 */
int main(int argc, char **argv)
{
  /* Inicia o ROS */
  ros::init(argc, argv, "talker");

  /* Cria um handler para um nó */
  ros::NodeHandle n;

  /* Cria um tópico, um canal para comunicação entre dois nós */
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

  /* Define a frequencia do loop */
  ros::Rate loop_rate(10);

  int count = 0;
  while (ros::ok())
  {
    /* Isto é um objeto de mensagem */
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    /* Imprime no terminal */
    ROS_INFO("%s", msg.data.c_str());

    /* Publica a mensagem */
    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }

  return 0;
}
```

Figura 11: Script de envio de mensagem para um nó.

Uma ideia para resolver essa questão é baseada no artigo (ORTIZ; GARCÍA; BORRAJO, 2008). Nele, utiliza- se marcadores RFID (Radio-Frequency IDentification) em objetos e leitores de RFID como braceletes nos pulsos de pessoas que querem ter suas ações analisadas pelo sistema. Conforme a pessoa se mexe e interage com o ambiente, gera-se um arquivo de log com registros contínuos de seus movimentos pelo bracelete leitor de RFID. Nesse log tem informações dos

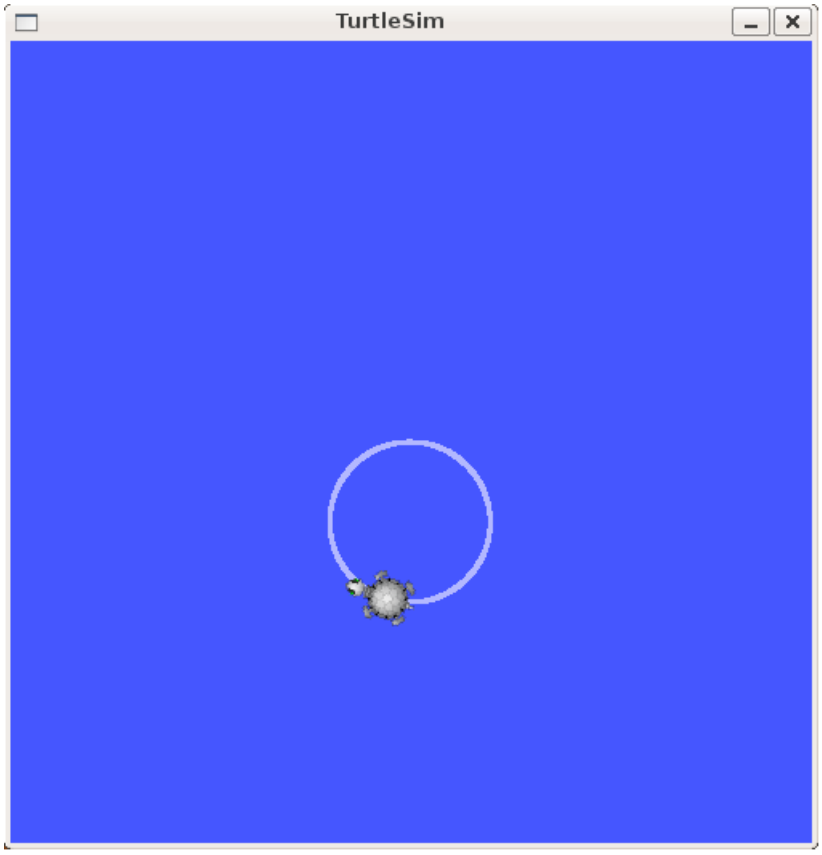

Figura 12: Simulador TurtleSim sendo executado. Aqui o rono está fazendo uma rota circular, assim como no trajeto desenhado.

sensores, como ID do sensor, a leitura do sensor e a sua duração de ativação. Esse arquivo de log gerado pelo leitor, com várias referências temporais, será parâmetro de entrada para um compilador responsável por analisá-lo e determinar possíveis ações do usuário. Essa análise se baseia na comparação de ações anteriores e posteriores de um momento, e da duração do tempo de ativação do sensor (classificando ações falsos positivas). Com essas ações bem definidas, sistemas discretos podem processar essas informações da forma que bem entender.

A abordagem escolhida não precisará necessariamente utilizar RFID. Porém a combinação de algoritmos como CMAC e a abordagem utilizada de usar marcadores, seus registros temporais, a compilação e o processamento dessas informações, será utilizada para cobrir a questão de definição de um estado do sistema

durante a navegação robótica.

4.3 Arquitetura

Temos como arquitetura base para o sistema de navegação robótica móvel chamado o sistema REACT, responsável pelo controle e manipulação de seus movimentos de acordo com seus estímulos sensoriais.

O sistema REACT tem uma arquitetura baseada em Motor-Schemas e no Método de Campos Potenciais. Ambos os métodos são movidos a comportamentos primitivos que capacitam o robô a uma navegação contínua até um alvo fixo, desviando de seus obstáculos. Inicialmente o robô terá sonares e hodômetros como sensores para o auxílio na navegação. Esses sensores tratam informações do ambiente e através de uma combinação dessas informações será determinada a movimentação do robô.

Como dito anteriormente, a arquitetura do sistema REACT é baseada em Motor-Schemas (ARKIN, 1998) que define o comportamento do robô em dois módulos principais: o módulo perceptivo, responsável por extrair os estímulos sensoriais, e o módulo de codificação que, alimentado pelo módulo perceptivo, responde aos motores os movimentos desejados.

Paralelamente, é utilizado o Método de Campos Potenciais para composição do vetor resultante da direção do robô. Esse método baseia-se na soma vetorial de comportamentos primitivos representados pelo robô, como uma movimentação contra um obstáculo, uma movimentação a favor do alvo e uma movimentação inercial referente ao seu movimento. Cada um desses comportamentos primitivos gera um vetor de acordo com seus parâmetros: um vetor que desvia o robô de acordo com a distância e direção do obstáculo, um vetor que aproxima o robô de acordo com a distância e direção do alvo a ser atingido, e um vetor que mantém

a inércia do movimento do robô. Essa soma vetorial (composta por diversos parâmetros processados do módulo de percepção) será a resultante da direção que o módulo de codificação executará nas respostas motoras do robô. O uso de conhecimento anterior a partir de tarefas similares diminui o tempo gasto pelo agente na exploração do ambiente e então o agente apresenta melhor comportamento desde o início no processo de aprendizagem.

Figura 13: Sistema robótico a ser utilizado neste projeto (Pioneer 2DX) – neste exemplo o robô já possui uma câmera auxiliar e podemos visualizar os sensores acoplados ao robô.

5 PROPOSTA

Nosso projeto trabalha com a tarefa de que um robô (Pioneer 2DX) precisa alcançar uma localização específica, um destino final. Em cenários como este, em que uma solução não é facilmente moldável dado que o robô pode não possuir todas as informações do ambiente de trabalho, a base do aprendizado por reforço foi utilizada por muitos pesquisadores. Um problema possível nessa abordagem seria a dificuldade e o tempo necessários para alcançar a convergência a um modelo ótimo devido às interações repetitivas do robô com o ambiente por tentativa e erro. Dessa forma, para acelerarmos o processo de aprendizado, buscaremos transferir o conhecimento de problemas com escopo mais reduzido para a problemática final; ou seja, a ideia é utilizar o conhecimento obtido pelo agente em uma tarefa mais simples na solução mais eficiente de um problema maior.

Nosso projeto tem como meta levar um robô (Pioneer 2DX) a alcançar uma localização específica, um destino final. Para tal, ele poderia seguir uma rota pré-estabelecida, onde seu programador simplesmente lhe passaria um conjunto de estados finitos mapeados no ambiente e uma matriz relacionando estados com ações, a partir da qual o robô se orienta para chegar ao destino, ou seja, a partir de um planejamento de rota fixa. Porém, essa abordagem se mostra muito limitada, pois não permite que o robô evolua em suas decisões, otimizando seu trabalho com relação a tempo e esforço. Para contornar essa situação, usaremos as técnicas de Aprendizado por Reforço e Transferência de conhecimento para otimizar essa tarefa: será fornecida ao robô uma política (ou biblioteca de políti-

cas) de comportamento leva ao destino certo. Porém, o robô decidirá sua rota de forma probabilística: ou ele toma aquele caminho guiado pela política fixada ou pode explorar o ambiente, julgando o quão bom ou ruins foram suas novas decisões a partir de recompensas ou punições; os algoritmos de Aprendizado por Reforço devem então ser responsáveis por aprimorar a tomada

de decisão pela política que tiver maior acúmulo de recompensas. A discretização do ambiente também será um ponto que merece atenção: a princípio devemos estudar técnicas de discretização de estados, por exemplo o modelo Cerebellar Model Arithmetic Computer(CMAC) , para definição de estados finitos baseados no ambiente contínuo.

Como primeira ação, simularemos o ambiente e o robô Pionner 2dx com o simulador open source ROS, para familiarização com comandos e ações do robô e processamento de informações recebidas por seu hodômetro e sonar. Após essa primeira etapa, montaremos um pequeno ambiente controlado (um labirinto, por exemplo) e com a integração de novos receptores (câmeras) faremos ele se locomover inicialmente com uma política determinística, para, após obtermos bons resultados, podermos aplicar técnicas de Aprendizado por reforço.

Em seguida, traremos os conceitos e técnicas desenvolvidas para um ambiente real, como um ambiente dentro de um prédio da Poli, onde a tarefa será ir de um cômodo até outro; para tal, serão necessários novos conceitos para reconhecimentos de objetos, obstáculos, presentes. Inicialmente podemos modelar os objetos de forma simples: por exemplo, ao invés de identificar uma porta, o robô identifica um quadrado vermelho como sinônimo de porta. Com êxito nessa etapa, podemos partir para métodos de interpretação de imagens mais complexas, para que o robô se adapte a ambientes mais gerais.

Por fim, buscaremos tornar a tarefa do robô mais complexa e utilizar a técnica de Transferência de Conhecimento para acelerar o processo de definição de

política “ótima” nesta nova meta, a partir da política obtida com a primeira tarefa.

REFERÊNCIAS

ARKIN, R. C. *Behavior-based robotics*. [S.l.]: MIT press, 1998.

ORTIZ, J.; GARCÍA, A.; BORRAJO, D. A relational learning approach to activity recognition from sensor readings. In: IEEE. *Intelligent Systems, 2008. IS'08. 4th International IEEE Conference*. [S.l.], 2008. v. 3, p. 19–8.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: Cambridge Univ Press, 1998.