Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

João Pedro Barros Ferreira

ÁRVORES ALEATÓRIAS DE EXPLORAÇÃO RÁPIDA COM TÉCNICAS DE APRENDIZADO DE MÁQUINAS APLICADAS À NAVEGAÇÃO DE CARROS AUTÔNOMOS

Rio de Janeiro
2022

João Pedro Barros Ferreira

ÁRVORES ALEATÓRIAS DE EXPLORAÇÃO RÁPIDA COM TÉCNICAS DE APRENDIZADO DE MÁQUINAS APLICADAS À NAVEGAÇÃO DE CARROS AUTÔNOMOS

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Controle e Automação.

Orientador: Prof. Téo Cerqueira Revoredo
Orientador: Prof. Douglas Mota Dias

Rio de Janeiro
2022

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

F383 Ferreira, João Pedro Barros.
Árvores aleatórias de exploração rápida com técnicas de aprendizado de máquinas aplicadas à navegação de carros autônomos / João Pedro Barros Ferreira. – 2022.
161f.

Orientadores: Téo Cerqueira Revoredo, Douglas Mota Dias.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia eletrônica - Teses. 2. Robótica - Teses. 3. Aprendizado do computador - Teses. 4. Controle automático - Teses. I. Revoredo, Téo Cerqueira. II. Dias, Douglas Mota. III. Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia. IV. Título.

CDU 007.52

Bibliotecária: Júlia Vieira – CRB7/6022

Assinatura

Data

João Pedro Barros Ferreira

ÁRVORES ALEATÓRIAS DE EXPLORAÇÃO RÁPIDA COM TÉCNICAS DE APRENDIZADO DE MÁQUINAS APLICADAS À NAVEGAÇÃO DE CARROS AUTÔNOMOS

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Controle e Automação.

Aprovada em 16 de Dezembro de 2022.
Banca Examinadora:

Prof. Téo Cerqueira Revoredo (Orientador)
Faculdade de Engenharia – UERJ

Prof. Douglas Mota Dias (Orientador)
Faculdade de Engenharia – UERJ

Prof. Jorge Luís Machado Do Amaral
Faculdade de Engenharia – UERJ

Prof. Alessandro Jacoud Peixoto
Universidade Federal do Rio de Janeiro

Rio de Janeiro
2022

DEDICATÓRIA

Dedico esta dissertação ao meu irmão, Gabriel.

AGRADECIMENTOS

Aos meus pais, Ricardo e Jane, pelo apoio e incentivo que serviram de alicerce para as minhas realizações.

Ao meu irmão Gabriel pela amizade e companheirismo.

Aos meu orientadores, Dr. Téo e Dr. Douglas, pelos ensinamentos, contribuições e apoio, durante todo o processo.

A todos os meus amigos da faculdade que compartilharam dos inúmeros desafios que enfrentamos.

À UERJ e o seu corpo docente, que demonstrou estar comprometido com a qualidade e excelência do ensino.

Aceita as surpresas que transformam teus planos, derrubam teus sonhos, dão rumo totalmente diverso ao teu dia e, quem sabe, à tua vida... Não há acaso. Dá liberdade ao Pai, para que ele mesmo conduza a trama dos teus dias.

D. Hélder Câmara

RESUMO

FERREIRA, João Pedro Barros. *Árvores Aleatórias de Exploração Rápida com Técnicas de Aprendizado de Máquinas Aplicadas à Navegação de Carros Autônomos*. 2022. 161 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2022.

A robótica móvel é uma área promissora da engenharia que alia a automatização de tarefas com a locomoção. Essa flexibilidade de movimento traz uma demanda por métodos que permitam a navegação autônoma do robô, evitando a colisão com qualquer outro objeto no espaço. O planejamento de trajetórias é um problema complexo que, frequentemente, exige o conhecimento profundo do robô e heurísticas particulares ao sistema em que é aplicado. Este trabalho propõe uma abordagem sistemática para este problema, considerando a união das árvores aleatórias de exploração rápida, um algoritmo clássico de planejamento de trajetórias, com diversas técnicas de aprendizado de máquinas, a fim de contornar as dificuldades associadas à obtenção de soluções analíticas. O método proposto é avaliado computacionalmente em um modelo de robô móvel similar a um automóvel de passeio, um sistema importante considerando a emergência dos carros autônomos, e atinge resultados comparáveis aos métodos analíticos, que utilizam profundo conhecimento do sistema.

Palavras-chave: robótica móvel. planejamento de trajetórias. aprendizado de máquinas.

ABSTRACT

FERREIRA, João Pedro Barros. *Rapidly-Exploring Random Trees with Machine Learning Techniques Applied to Autonomous Car Navigation*. 2022. 161 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2022.

Mobile robotics is a promising area of engineering that combines task automation with mobility. This flexibility of movement brings a demand for methods that allow autonomous navigation of the robot, avoiding collision with any other object in space. Trajectory planning is a complex problem that often requires profound knowledge of the robot and particular heuristics to the system in which it is applied. This work proposes a systematic approach to this issue, considering the union of rapidly-exploring random trees, a classic trajectory planning algorithm, with several machine learning techniques, to overcome the difficulties associated with obtaining analytical solutions. The proposed method is computationally evaluated in a mobile robot model similar to a passenger car, an important system considering the emergence of autonomous cars, and achieves results comparable to analytical methods, which uses broad knowledge of the system.

Keywords: mobile robots. trajectory planning. machine learning.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE EXEMPLOS

LISTA DE ABREVIATURAS E SIGLAS

CLMR	Car-like mobile robot
RNA	Rede neural artificial
PMC	Perceptron de múltiplas camadas
GAN	Generative adversarial network
DLVM	Deep latent variable model
VAE	Variational autoencoder
LV	log-verossimilhança
KL	Kullback–Leibler
ELBO	Evidence lower bound
CVAE	Conditional variational autoencoder
RRT	Rapidly-exploring random tree
PDM	Processo de decisão de Markov
DDPG	Deep deterministic policy gradient
HER	Hindsight experience replay
VAA	Veículo aéreo autônomo
MEGA	Maximum entropy goal achievement
RS	Reeds-Shepp
CPM	Controlador preditivo baseado em modelo

LISTA DE SÍMBOLOS

$\vec{x}$	Vetor de entrada da rede neural
$\vec{c}$	Vetor de saída de supervisão em um banco de dados
$\vec{h}^{[l]}$	Vetor de saída da camada l de uma rede neural
$N^{[l]}$	Número de neurônios artificiais na camada l de uma rede PMC
$\vec{h}_j^{[l]}$	j-ésima componente do vetor de saída da camada l de uma rede PMC
$\varphi_j^{[l]}$	Função de ativação no neurônio artificial j, da camada l, de uma rede PMC
$k_j^{[l]}$	Soma ponderada do j-ésimo neurônio artificial da camada l
$w_{jk}^{[l]}$	k-ésima componente do vetor $\vec{w}_j^{[l]}$
$b_j^{[l]}$	Constante do j-ésimo neurônio artificial da camada l de uma rede PMC
$\vec{w}_j^{[l]}$	Vetor de pesos do j-ésimo neurônio artificial da camada l
θ	Parâmetros de uma rede neural
f_θ	Função determinística representada por uma rede PMC com parâmetros θ
$\varphi^{[l]}$	Função de ativação na camada l de uma rede PMC
$\vec{x}^{(i)}$	Vetor de entrada do i-ésimo exemplo do conjunto de dados
$\vec{c}^{(i)}$	Vetor de saída de supervisão em um banco de dados
$\mathcal{N}_i$	A i-ésima distribuição normal diagonal de um conjunto de gaussianas
μ_i	A i-ésima componente do vetor de médias de uma normal diagonal
σ_i	A i-ésima componente do vetor de desvios de uma normal diagonal
D	Número de exemplos no conjunto de dados
p^*	Uma distribuição de probabilidade desconhecida que gerou os dados
E	Função de custo ou erro
p_θ	Modelo probabilístico generativo ou discriminativo com parâmetros θ
θ^*	Valores ótimos do conjunto de parâmetros θ em relação a uma otimização
θ_k	Valores do conjunto de parâmetros θ no k-ésimo ajuste
η	Taxa de aprendizado
$\vec{m}_k$	Valor do primeiro momento, na iteração k, no algoritmo ADAM
β_1	Taxa de decaimento inicial do primeiro momento no algoritmo ADAM
$\vec{v}_k$	Valor do primeiro momento, na iteração k, no algoritmo ADAM
β_2	Taxa de decaimento inicial do segundo momento no algoritmo ADAM
ξ	Um número pequeno para evitar divisões por zero no algoritmo ADAM
$\vec{\delta}^{[l]}$	Vetor de erro, na camada l, relativo ao algoritmo *backpropagation*
$\delta_j^{[l]}$	j-ésima componente do vetor $\vec{\delta}^{[l]}$
$\mathcal{D}$	Conjunto de dados
$\hat{\vec{x}}$	Saída de um modelo generativo
$\vec{z}$	Vetor latente em modelos probabilísticos latentes

$\vec{z_i}$	Um vetor latente específico
$\vec{y}$	Vetor de condições em modelos probabilísticos latentes condicionais
$\vec{Z}$	Variável aleatória associada a amostragem do espaço latente
q_i	Modelo de codificador associado ao exemplo $\vec{x}^{(i)}$ do conjunto de dados
$\mathcal{L}_i$	ELBO calculado no i-ésimo exemplo do conjunto de dados
D_{KL}	Divergência de Kullback-Leibler
ϕ	Parâmetros de uma rede neural
M	Dimensão da variável observável em modelos probabilísticos latentes
q_ϕ	Modelo probabilístico de reconhecimento ou codificador com parâmetros ϕ
$\mathcal{N}$	Distribuição normal multivariada
$\vec{\mu}_\phi$	Vetor de médias, parametrizado por ϕ, de uma gaussiana multivariada
$\vec{\sigma}_\phi$	Vetor de desvios, parametrizado por ϕ, de uma gaussiana multivariada
$\vec{\sigma}_{\phi_j}$	j-ésima componente do vetor $\vec{\mu}_\phi$
$\vec{\sigma}_{\phi_j}$	j-ésima componente do vetor $\vec{\sigma}_\phi$
w	Função de reparametrização
$\vec{\epsilon}$	Variável de reparametrização
$\mathcal{L}$	Limite inferior variacional (ELBO)
β	Fator de balanceamento de custos em um CVAE
N	Dimensão da variável latente em modelos probabilísticos latentes
β_{norm}	Fato normalizado de balanceamento de custos em um CVAE
$\mathscr{A}$	Corpo rígido que representa um robô
$\mathcal{W}$	Espaço de trabalho de um robô
$\mathcal{B}_i$	i-ésimo obstáculo em um problema de planejamento de trajetórias
τ	Trajetória entre duas configurações de um robô
$\vec{g}$	Configuração de um robô
$\mathcal{G}$	Espaço de configurações
$\mathcal{B}_i$	Obstáculo em um problema de planejamento de trajetórias
$\mathcal{GB}_i$	Espaço de configurações que colidem com o obstáculo $\mathcal{B}_i$
$\mathcal{G}_{obs}$	Espaço de configurações com colisões
$\mathcal{G}_{livre}$	Espaço de configurações sem colisões
$\vec{g}_i$	Configuração inicial do robô em um problema de planejamento de trajetórias
$\vec{g_f}$	Configuração final do robô em um problema de planejamento de trajetórias
$\mathcal{T}$	Grafo que representa a árvore aleatória de exploração rápida
$\mathcal{V}$	Conjunto de vértices em uma árvore aleatória de exploração rápida
$\mathcal{E}$	Conjunto de arestas em uma árvore aleatória de exploração rápida
ρ	Função de distância entre duas configurações de um robô
$\vec{g}_{prox}$	Configuração da árvore de exploração mais próxima à amostrada
$\vec{g}_{aleatoria}$	Configuração do robô amostrada do espaço de configurações
$\mathcal{U}$	Controlador de um robô

$\vec{g}_{nova}$	Configuração obtida ao fim da simulação nos algoritmos RRT/RRT*
C	Custo de uma trajetória entre duas configurações
$\vec{g}'$	Configuração que antecede $\vec{g}$ em uma árvore de exploração
$\mathcal{V}_{prox}$	Conjunto de vértices próximos à configuração *nova*
ϵ_g	Limiar de distância para configurações próximas
$\vec{g}_{min}$	Configuração de menor custo no algoritmo RRT*
$\vec{o}$	Vetor de codificação dos obstáculos
$\vec{y}^{(j)}$	j-ésimo vetor de condições do conjunto de dados
$\vec{g}^{(i)}$	i-ésima configuração do conjunto de dados
A_t	Variável aleatória que representa a ação, no instante t, em um PDM
S_t	Variável aleatória que representa o estado, no instante t, em um PDM
R_t	Variável aleatória que representa a recompensa, no instante t, em um PDM
$\mathcal{S}$	Espaço de estados em um PDM finito
$\mathcal{A}$	Espaço de ações em um PDM finito
$\mathcal{R}$	Conjunto de recompensas em um PDM finito
G_t	Variável aleatória que representa o retorno, no instante t, em um PDM
H	Horizonte de uma tarefa em um PDM episódico
γ	Fator de desconto em um PDM
s	Realização da variável aleatória S_t em um PDM
a	Realização da variável aleatória A_t em um PDM
s'	Realização da variável aleatória S_{t+1} em um PDM
r	Realização da variável aleatória R_{t+1} em um PDM
s_t	Realização da variável aleatória S_t em um PDM
a_t	Realização da variável aleatória A_t em um PDM
r_t	Realização da variável aleatória R_{t+1} em um PDM
p_0	Distribuição de estados iniciais em um PDM episódico
π	Política em um PDM
V	Função de valor
V_π	Função de valor associada à política π
Q_π	Função ação-valor associada à política π
π^*	Política ótima em um PDM
V^*	Função de valor ótima
Q^*	Função ação-valor ótima
$\hat{Q}$	Aproximação tabular para a função ação-valor
$\mathscr{G}$	Espaço de retornos em um PDM
Q	Função ação-valor
π_g	Política *greedy* (ambiciosa)
$\hat{Q}^*_{\pi_g}$	Aproximação tabular de $Q^*_{\pi_g}$
$Q^*_{\pi_g}$	Função ação-valor ótima de uma política determinística

π_b	Política exploratória em um algoritmo de aprendizado por reforço
J	Critério de otimização
Φ	Uma expressão na derivação do teorema do gradiente da política
s''	Realização da variável aleatória S_{t+2} em um PDM
s'''	Realização da variável aleatória S_{t+3} em um PDM
ψ^{π}	Probabilidade de transição de estado em um número de passos de tempo
$N_{x,t}$	Variável aleatória que indica se o estado no tempo t é x, em um PDM
n	Taxa de visitação dos estados em um PDM
υ_{π}	Distribuição de estados visitados em um PDM, sob ação da política π
$\hat{V}$	Aproimação para a função de valor
Q_{θ}	Aproximação da função ação-valor por uma rede PMC com parâmetros θ
E_t	Erro de Bellman no instante t
C_t	Variável aleatória de supervisão do erro de Bellman no instante t
$\mathbb{B}$	Memória de transições em um algoritmo de aprendizado por reforço
$\Omega_t^{(e)}$	Transição no instante t e episódio e do treinamento de um agente
Γ	Trajetória de um agente em um episódio de um PDM episódico
θ'	Rede neural *target* do *crítico* no algoritmo DDPG
ϕ'	Rede neural *target* do *ator* no algoritmo DDPG
ζ	Coeficiente de Polyak
ν	Ruído gaussiano
$\mathcal{M}$	Mini-lote de transições em um algoritmo de aprendizado por reforço
g	Objetivo em um PDM generalizado
r_g	Função de recompensa condicionada no objetivo g
γ_g	Fator de desconto em um PDM generalizado
$V_{g,\pi}$	Função de valor em PDM com múltiplos objetivos
m	Mapeamento do estado atual para o objetivo atingido no estado
g_b	Objetivo desejado em um PDM generalizado
g_a	Objetivo alcançado em um PDM generalizado
p_{ag}	Distribuição empírica de objetivos atingidos durante o treinamento
p_{dg}	Distribuição de objetivos desejados
q_a	Distribuição de objetivos atingidos quando a política é condicionada em g_b
$p_{ag\|g_a}$	Distribuição empírica resultante da adição de g_a a p_{ag}
η_b	Inverso do número de transições na memória $\mathbb{B}$
ϕ_a	Componente de direcionamento do sinal de controle de um CLMR
ϕ_{ref_1}	Ângulo de referência para a roda dianteira esquerda
ϕ_{ref_2}	Ângulo de referência para a roda dianteira direita
v_a	Componente de torque do sinal de controle de um CLMR
v_{ref_1}	Velocidade linear de referência para a roda traseira esquerda

v_{ref_2}	Velocidade linear de referência para a roda traseira direita
v_{ref_3}	Velocidade linear de referência para a roda dianteira esquerda
v_{ref_4}	Velocidade linear de referência para a roda dianteira direita
x_c	Coordenada x do centro do eixo traseiro de um CLMR
v_c	Velocidade escalar instantânea do CLMR
θ_c	Orientação de um CLMR
y_c	Coordenada y do centro do eixo traseiro de um CLMR
ϕ_c	Ângulo de uma roda dianteira de um CLMR
L	Distância entre os eixos traseiro e dianteiro de um CLMR
d	Diferença ponderada entre dois vetores
ϵ_r	Limiar de distância para recompensas binárias
x_g	Componente x_c de uma configuração desejada g
y_g	Componente y_c de uma configuração desejada g
θ_g	Componente θ_c de uma configuração desejada g
w_{pos}	Peso relativo à diferença absoluta entre duas posições de um CLMR
w_{orn}	Peso relativo à diferença absoluta entre duas orientações de um CLMR
f	Frequência de um controlador discreto
S_0	Variável aleatória que representa o estado inicial em um PDM
X_0	Variável aleatória que representa o valor inicial de x_c em um PDM episódico
Y_0	Variável aleatória que representa o valor inicial de y_c em um PDM episódico
Θ_0	Variável aleatória que representa o valor inicial de θ_c em um PDM episódico
v_0	Valor inicial de v_c em um PDM episódico
ϕ_0	Valor inicial de ϕ_c em um PDM episódico
α	Coeficiente que determina parcela das amostras provenientes do CVAE
ρ_π	Função ρ induzida pela atuação da política parametrizada π_ϕ
X_g	Variável aleatória que representa a componente x_c da configuração desejada
Y_g	Variável aleatória que representa a componente y_c da configuração desejada
Θ_g	Variável aleatória que representa a componente θ_c da configuração desejada
x_i	Coordenada x da configuração inicial do robô
x_f	Coordenada x da configuração final do robô
y_i	Coordenada y da configuração inicial do robô
y_f	Coordenada y da configuração final do robô
x_{pass1}	Coordenada x da primeira passagem estreita
x_{pass2}	Coordenada x da segunda passagem estreita
y_{pass1}	Coordenada y da primeira passagem estreita
y_{pass2}	Coordenada y da segunda passagem estreita

SUMÁRIO

INTRODUÇÃO

O impacto da robótica na sociedade é notável. Atualmente, os robôs são amplamente utilizados e possuem aplicações em diversos setores, que vão desde a fabricação de produtos, em geral, até a realização de tarefas no setor agrícola ou na medicina. Estima-se que o mercado da robótica continuará a crescer, impulsionado, principalmente, pela demanda progressiva por robôs industriais e o aumento de produtividade associado à automatização de processos (DUCH; ROSSETTI; HAARBURGER, 2021). A menção aos robôs industriais pode trazer à imaginação maquinas manipuladoras como as da Figura 1a, tipicamente utilizadas no setor automotivo. Este tipo de robô encontra-se fixo em um determinado local, de modo que a manipulação dos objetos está restrita a uma pequena região em torno deste ponto. Em contraste, um robô móvel, como o da Figura 1b, pode-se locomover livremente no espaço e manipular os objetos ao redor de sua posição atual. Dentre as diversas aplicações da robótica móvel pode-se citar o transporte e a manipulação de cargas e a exploração de ambientes extremos (WITZE et al., 2020). Em suma, utiliza-se a robótica móvel quando o espaço de trabalho é amplo e exige a locomoção da máquina de automação.

Superficialmente, a robótica móvel apresenta inúmeras vantagens em relação aos robôs fixos. Entretanto, a maior liberdade de locomoção traz desafios adicionais, principalmente em relação à estimação da posição e orientação do robô, já que para saber a posição de cada eixo do robô é necessário localizar um ponto de referência do mesmo em relação ao ambiente externo. Além disso, torna-se mais evidente a necessidade de desenvolver métodos para que o robô evite colisões com os diferentes objetos presentes no espaço.

A realização de trajetórias exequíveis que otimizam algum critério, como tempo ou distância percorrida, é um dos problemas centrais da robótica móvel. Como os robôs móveis possuem diferentes formas e dinâmicas de movimento, é difícil encontrar um método analítico universal que funcione para qualquer sistema. Frequentemente, utiliza-se algum tipo de heurística ou método iterativo de amostragem para resolver esse problema.

A última década mostrou que a inteligência artificial é uma ferramente essencial para resolver problemas que não podem ser abordados analiticamente. Por exemplo, o reconhecimento de imagens, uma tarefa trivial para um ser humano, é extremamente complexa para um computador e as melhores soluções atuais para este tipo de problema vêm da área de aprendizado de máquinas (RECHT et al., 2019), um dos campos de estudo da inteligência artificial.

Neste sentido, este trabalho visa explorar diversas técnicas de aprendizado de máquinas para a obtenção de trajetórias exequíveis para robôs móveis, otimizando o tempo de navegação e evitando a colisão com obstáculos no caminho. O foco do trabalho

Figura 1 - Exemplos de manipuladores e robôs móveis

(a) Robôs industriais fixos (b) Robô móvel para a agricultura

Fonte: Ben-Ari e Mondada (2018).

Figura 2 - Um veículo autônomo no *DARPA urban challenge*

Fonte: Thrun et al. (2006).

são robôs móveis similares a carros de passeio, como mostra a Figura 2. Este tipo de veículo é conhecido como CLMR (*car-like mobile robot*).

O estudo de CLMRs mostra-se relevante no cenário atual, com o crescimento da utilização de carros autônomos (HUSSAIN; ZEADALLY, 2019). Outra justificativa para explorá-los é a constatação que CLMRs autônomos atuais já utilizam extensivamente o aprendizado de máquinas (Tesla Inc, 2022), tanto para o reconhecimento de objetos quanto para a segmentação de imagens, como mostra a Figura 3.

A principal contribuição desta dissertação é a integração de trabalhos passados, nas três áreas do aprendizado de máquinas (por reforço, supervisionado e não-supervisionado) com um algoritmo clássico de planejamento de trajetórias, o RRT (*rapidly exploring random trees*). O método proposto contorna as dificuldades associadas à implementação do RRT em sistemas de maior complexidade. Outra contribuição deste trabalho é a validação computacional de algoritmos recentes na área do aprendizado de máquinas quando

Figura 3 - Reconhecimento de objetos em um veículo autônomo

Legenda: O *frame* do vídeo mostra a detecção e o reconhecimento de diversos objetos em tempo real.
Fonte: Tesla Inc (2022)

aplicados à emergente área de CLMRs autônomos.

O trabalho é estruturado da seguinte maneira: a Seção 1 é dedicada às *redes neurais artificiais* (RNA) (ABIODUN et al., 2018), em particular, às redes *perceptron de múltiplas camadas* (PMC). Esse modelo matemático, além de ser um dos principais componentes do aprendizado de máquinas, é utilizado em diversas ocasiões neste trabalho, tanto para obter aproximações de funções não-lineares complexas quanto para modelar distribuições de probabilidade. Essa seção aborda o funcionamento das redes PMCs, para os fins citados, e como estes modelos matemáticos podem ser otimizados através de uma base de dados sobre um problema.

A Seção 2 aborda o algoritmo RRT, um dos principais métodos de planejamento de trajetórias para robôs. Nesta seção, explora-se o funcionamento do algoritmo original, proposto por LaValle et al. (1998), e da principal extensão deste método, o algoritmo RRT*, criado por Karaman e Frazzoli (2010). O planejamento de trajetórias por meio de árvores de exploração é parte essencial do método proposto. Nesta seção verifica-se também o uso de modelos generativos (OUSSIDI; ELHASSOUNY, 2018) para auxiliar o tempo de execução do algoritmo RRT*, como proposto por Ichter, Harrison e Pavone (2017).

A Seção 3 é dedicada ao aprendizado por reforço (SUTTON; BARTO, 2018), uma das áreas da inteligência artificial e, neste trabalho, o componente responsável pelo controle local do robô móvel. Esta seção aborda também os algoritmos recentes que combi-

nam essa área com o aprendizado profundo (GOODFELLOW et al., 2016) e, finalmente, como os avanços recentes na abordagem de múltiplos objetivos (PLAPPERT et al., 2018) permitem a aplicação eficaz do aprendizado por reforço em problemas de robótica.

A Seção 4 apresenta a abordagem proposta, que combina os algoritmos das seções anteriores para sugerir um método de planejamento e realização de trajetórias para robôs móveis, otimizando o tempo de navegação e o custo computacional do RRT*, em ambientes cuja localização dos obstáculos é conhecida.

A Seção 5 mostra os resultados experimentais obtidos e compara a abordagem proposta com um método analítico, que exige conhecimento específico sobre os CLMRs.

1 REDES NEURAIS ARTIFICIAIS

O cérebro de um humano têm cerca de 80 bilhões de neurônios. Essas células, interconectadas através de conexões denominadas *sinapses*, formam um circuito neural biológico, talvez a estrutura mais complexa e menos compreendida pela humanidade. Um neurônio transmite informações quando a soma dos sinais recebidos atinge um determinado limiar. Este princípio básico de funcionamento inspirou o trabalho pioneiro de Rosenblatt (1958), que propôs um modelo matemático de um neurônio artificial, cujo sinal de saída dependia da aplicação de uma função de ativação sobre a soma ponderada dos sinais de entrada. Uma RNA pode ser formada por diversos neurônios agrupados de diferentes maneiras.

As RNAs são componentes essenciais do aprendizado profundo (GOODFELLOW et al., 2016), uma das áreas da inteligência artificial de maior sucesso na última década, e possuem diversas aplicações, como, por exemplo, o reconhecimento de imagens (HIJAZI et al., 2015) e o processamento de linguagem natural (FATHI; SHOJA, 2018). Esta seção aborda um tipo fundamental de rede neural, o perceptron de múltiplas camadas.

Apresenta-se o modelo matemático de uma rede PMC e como esta estrutura pode ser otimizada para representar funções determinísticas complexas ou distribuições de probabilidade. Esse processo de otimização pode ser compreendido e implementado por estruturas abstratas denominadas grafos computacionais. Por fim, esta seção aborda o funcionamento e a otimização de uma arquitetura de rede neural, composta por duas redes PMCs, capaz de gerar dados similares aos originados por um processo desconhecido.

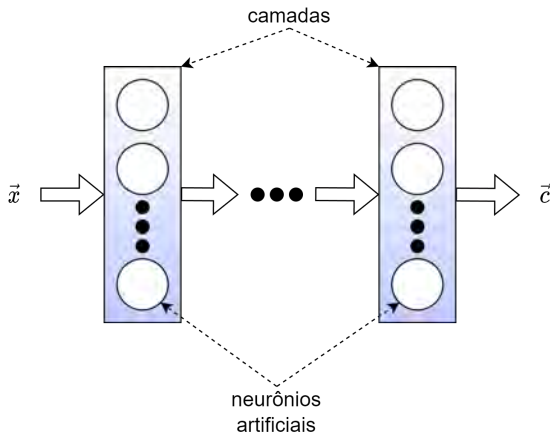
1.1 Perceptron de Múltiplas Camadas

As redes neurais PMC são funções não-lineares que mapeiam um vetor de entrada $\vec{x}$ em um vetor de saída $\vec{c}$. Uma rede é organizada em diversas camadas, cada uma contendo um determinado número de neurônios artificiais, as unidades atômicas de processamento. A Figura 4 ilustra uma rede PMC genérica.

Cada neurônio artificial recebe um vetor de entrada da camada anterior (ou o próprio vetor de entrada $\vec{x}$) e fornece como saída um único número. Portanto, pode-se dizer que uma camada com n neurônios artificiais gera um vetor com n componentes. Dessa forma, em uma rede PMC, cada camada transforma, de forma sequencial, um vetor de entrada em outro vetor de saída.

Denota-se por $\vec{h}^{[l]}$ o vetor de saída da camada l. A dimensão de $\vec{h}^{[l]}$, representando o número de neurônios na camada l, é indicada por $N^{[l]}$. Cada componente de $\vec{h}^{[l]}$, ou seja $h_j^{[l]}$, é obtida a partir do vetor de saída da camada anterior $\vec{h}^{[l-1]}$, conforme a Equação

Figura 4 - Rede neural PMC



Fonte: O autor, 2022.

1. O vetor de entrada $\vec{x}$ da rede neural pode ser representado por $\vec{h}^{[0]}$ e o vetor de saída $\vec{h}^{[L]}$. Em geral, denota-se pelo sobrescrito $^{[L]}$ os parâmetros relativos a camada de saída e por $^{(1)}$ os que se referem à primeira camada, que recebe o vetor $\vec{x}$ de entrada.

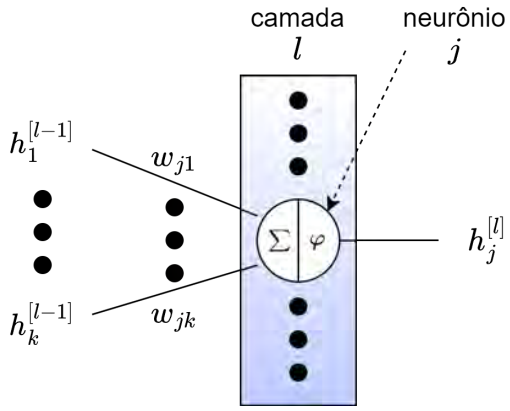
$$h_j^{[l]} = \varphi_j^{[l]} \cdot k_j^{[l]} = \varphi_j^{[l]} \left(\sum_k w_{jk}^{[l]} h_k^{[l-1]} + b_j^{[l]} \right) \tag{1}$$

A Equação 1 representa a saída do neurônio artificial j pertencente a camada l. Um vetor de *pesos* $\vec{w}_j^{[l]} = [\, w_{j1}^{[l]} \;\; w_{j2}^{[l]} \;\; \ldots]$ e uma constante, $b_j^{[l]}$, formam os parâmetros de um neurônio artificial. Generalizando esse conceito, o conjunto de pesos e constantes de todos os neurônios artificiais de uma rede PMC representam os parâmetros da rede neural. Este conjunto pode ser representado por uma única letra[1], por exemplo, θ, dessa forma, uma rede PMC é uma função parametrizada f_θ. O símbolo $\varphi_j^{[l]}$ representa a *função de ativação* do neurônio j da camada l, que atua sobre a soma ponderada $\vec{k}_j^{\,[l]}$. Um esquema mais detalhado da Equação 1 levando em conta a estrutura da rede PMC é apresentado na Figura 5.

Embora cada neurônio artificial de uma mesma camada possa apresentar uma função de ativação própria, tipicamente, em uma camada l, todos os neurônios apresentam a mesma função de ativação. Neste caso, a notação $\varphi^{[l]}$ é utilizada. A função não-linear ReLU e suas variações (Figura 6) são frequentemente utilizadas como função de ativação em redes PMC. De acordo com o teorema da aproximação universal (HOR-

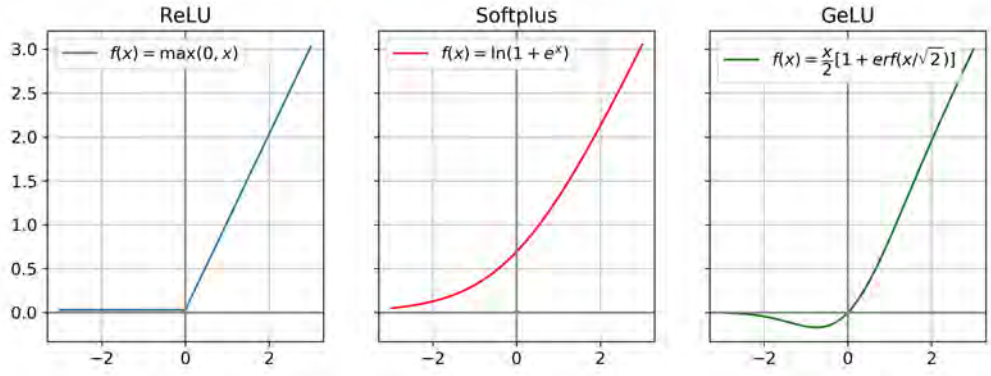
[1] Essa Dissertação não inclui sobrescritos nas letras que representam conjuntos de parâmetros, afim de evitar sobrecarga na notação.

Figura 5 - Estrutura de um neurônio artificial



Fonte: O autor, 2022

Figura 6 - Função de ativação ReLU e outras similares



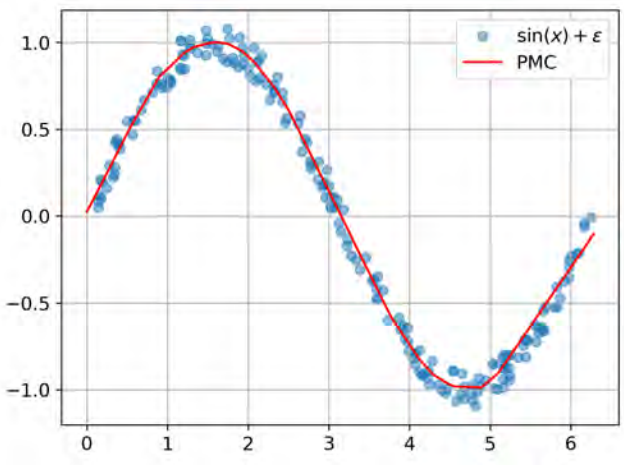
Fonte: O autor, 2022

NIK; STINCHCOMBE; WHITE, 1989), uma rede PMC com $\varphi^{(1)} = ReLU$ e $\varphi^{[L]}$ uma função linear é capaz de aproximar, com erros arbitrariamente pequenos, qualquer função contínua em um subconjunto fechado e limitado de $\mathbb{R}^n$, desde que haja uma quantidade suficiente de neurônios artificiais. Este resultado se estende para diversos outros tipos de funções não-lineares, como a função sigmoide ou a tangente hiperbólica. O Exemplo 1 mostra a aproximação da função seno com ruído por meio de uma rede PMC.

Redes PMCs podem também ser usadas para representar ou aproximar distribuições de probabilidade, como ilustra o Exemplo 2. De forma análoga ao teorema da aproximação universal de funções, redes PMC podem ser utilizadas para fornecer os parâmetros de uma mistura de distribuições normais, que, por sua vez, são aproximadores universais de densidades de probabilidades. Redes neurais, em geral, com este tipo

Exemplo 1 *Deseja-se obter um modelo que aproxime a função seno com ruído:* $f(x) = \text{sen}(x) + \epsilon$, *onde* $\epsilon \in [-1, 1]$ *é um ruído amostrado uniformemente. Suponha que* $f(x)$ *seja desconhecida, mas existem dados* $\{(x^{(i)}, c^{(i)})\}_{i=1}^{1000}$ *gerados a partir desta função. Através de um algoritmo de otimização, pode-se ajustar os parâmetros da rede PMC para obter uma aproximação de* f, *como mostra a Figura 7.*

Figura 7 - Aproximação da função seno com ruído



Fonte: O autor, 2022.

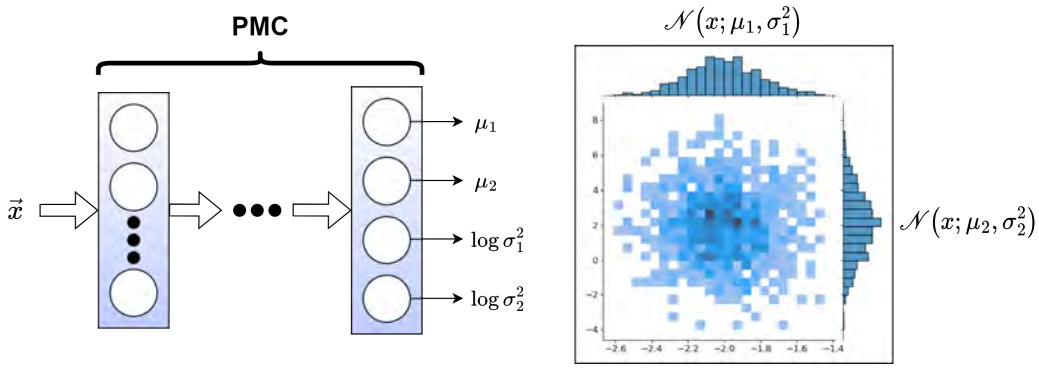
de saída são denominadas redes de mistura de densidades. Distribuições categóricas, por exemplo, podem ser obtidas com a função de ativação *softmax* (Equação 2). Neste caso, em problemas de classificação, cada componente do vetor de saída pode ser interpretada como a probabilidade do vetor de entrada pertencer à uma certa classe.

$$c_j = \frac{e^{k_j^{[L]}}}{\sum_{j=1}^{k} k_j^{[L]}} \tag{2}$$

Mesmo que seja possível aproximar diversas funções ou distribuições a partir de redes neurais, nada garante que o processo de ajuste dos parâmetros θ da rede convirja para a aproximação desejada. O ajuste dos parâmetros de uma rede neural é um problema de otimização que pode ser abordado de diversas formas, incluindo algoritmos genéticos (DING et al., 2013) ou buscas aleatórias (MASRI et al., 1999). Este trabalho aborda, nas seções seguintes, um dos métodos de otimização mais utilizados atualmente: a *descida do gradiente*.

Exemplo 2 *É possível representar uma distribuição normal n-dimensional com matriz de covariância diagonal através de uma rede PMC. Neste caso, a camada de saída deve fornecer as médias e variâncias para cada distribuição normal em $\{\mathcal{N}_i\}_{i=1}^n$, por meio de uma função de ativação linear. Como a variância deve ser maior que zero, o vetor de saída da rede pode ser representado por $[\,\mu_1\ \mu_2\ \ldots\ \mu_n\ \log\sigma_1^2\ \log\sigma_2^2\ \ldots\ \log\sigma_n^2\,]$. A Figura 8 abaixo é exemplo de uma rede PMC representando uma distribuição normal bidimensional.*

Figura 8 - Parametrização de uma gaussiana



Fonte: O autor, 2022.

1.2 **Descida do Gradiente**

A discussão da seção anterior sobre redes neurais como aproximadoras de funções é particularmente compatível com a teoria de aprendizagem estatística, em que se busca encontrar uma função a partir de dados. Neste paradigma, $\vec{x}$ e $\vec{y}$ são variáveis aleatórias distribuídas de acordo com uma distribuição de probabilidade p^* desconhecida. Dada uma sequência de pares $\{(\vec{x}^{\,(i)}, \vec{c}^{\,(i)})\}_{i=1}^D$, amostrados de p^*, busca-se construir uma função f_θ que infere $\vec{c}$ a partir de $\vec{x}$.

A otimização dos parâmetros θ da rede neural é guiada por uma função de *custo*, que mede a discrepância da previsão da rede neural, $f_\theta(\vec{x}^{\,(i)})$, em relação ao valor real $\vec{c}^{\,(i)}$. Um exemplo de função de *custo* para problemas de regressão é o erro quadrático médio:

$$E(\theta) = \frac{1}{D}\sum_{i=1}^{D}\left(\vec{c}^{\,(i)} - f_\theta(\vec{x}^{\,(i)})\right)^2 \tag{3}$$

Para problemas em que o objetivo é aproximar uma distribuição de probabilidade, o critério utilizado pode ser a função "log-verossimilhança" (LV) negativa. Dessa forma, o critério quantifica a probabilidade de ocorrência da classe $c^{(i)}$ quando a rede neural recebe

um vetor de entrada $x^{(i)}$:

$$E(\theta) = -\sum_{i=1}^{D} \log p_\theta \left(c^{(i)} | \, \vec{x}^{\,(i)} \right) \tag{4}$$

O objetivo do aprendizado *supervisionado*, apresentado na Equação 5, é encontrar o conjunto de parâmetros θ^* que minimiza a Equação 3 (ou a Equação 4, para problemas de classificação). Os valores de supervisão $\vec{c}^{\,(i)}$ são chamados de *alvos* (ou em inglês, *target*) já que $f_\theta(\vec{x}^{\,(i)})$ deve ser atualizada para fornecer uma saída próxima de $\vec{c}^{\,(i)}$.

$$\theta^* = \arg\min_\theta E(\theta) \tag{5}$$

O gradiente da Equação 3 em relação a θ fornece a direção da maior taxa de crescimento de $E(\theta)$. Dessa forma, ajustar θ em direção oposta ao gradiente tende a diminuir a função de custo. Na descida do gradiente, este ajuste é realizado de forma iterativa, sendo o parâmetro atualizado através de pequenos passos em direção oposta ao gradiente, tal como indica a Equação 6. O parâmetro que determina a magnitude da atualização é denominado *taxa de aprendizado* e é representado por η. Uma ilustração desse procedimento é apresentada na Figura 9.

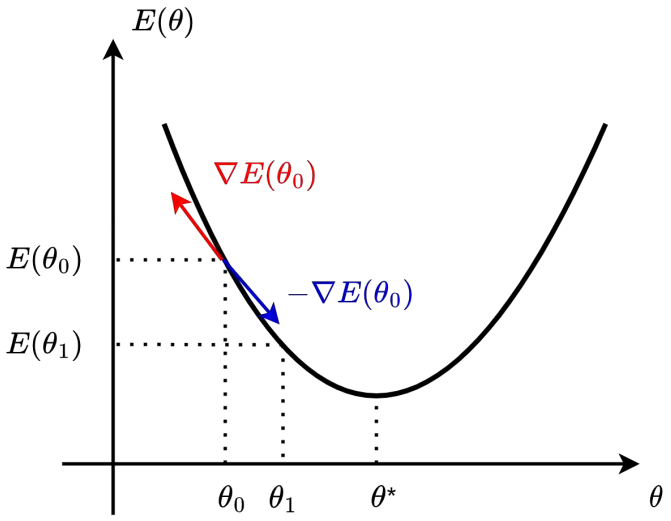
$$\theta_{k+1} = \theta_k - \eta \nabla_{\theta_k} E \tag{6}$$

A regra de atualização descrita pela Equação 6 pode utilizar uma função de custo que considera apenas uma parcela do conjunto de dados, com $\boldsymbol{M}$ pares $(\vec{x}^{\,(i)}, \vec{c}^{\,(i)})$ amostrados do conjunto total. Este processo é conhecido como *descida do gradiente estocástico em mini-lotes* (ROBBINS, 2007) e viabiliza o aprendizado em tempo real, útil quando o conjunto de dados é atualizado com frequência ou é grande demais para ser processado inteiramente (BOTTOU, 2010). Além disso, esta variação da descida do gradiente costuma ser eficiente em promover a convergência para o mínimo global (BOTTOU et al., 1991).

Na descida do gradiente estocástico em mini-lotes, a direção e magnitude do gradiente da função de custo, ao longo das iterações, apresenta alta variância, já que o mini-lote amostrado aleatoriamente é apenas uma parcela do conjunto de dados, ou seja, obtêm-se apenas uma estimativa do gradiente real. Neste caso, o processo de otimização dos parâmetros tende a apresentar um comportamento oscilatório em direção ao mínimo local, tal como mostra a Figura 10. É possível notar na ilustração que apesar das oscilações o parâmetro converge para o ponto de mínimo.

A Figura 10 busca representar outro comportamento característico da descida do gradiente em mini-lotes: utilizando o espaço de parâmetros bidimensional como exemplo, a estimativa do gradiente na iteração i pode ajustar de forma excessiva o parâmetro

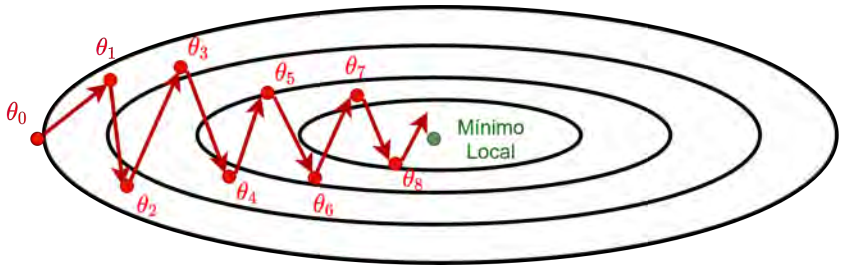
Figura 9 - Direções do gradiente da função de custo



Legenda: O ajuste de θ_0 na direção oposta ao gradiente diminui o erro de $E(\theta_0)$ para $E(\theta_1)$.

Fonte: O autor, 2022.

Figura 10 - Descida do gradiente estocástico



Legenda: Ilustração da descida do gradiente estocástico para um espaço de parâmetros bidimensional.

Fonte: O autor, 2022.

na direção vertical, enquanto o verdadeiro gradiente da função de custo encontra-se na direção horizontal. Por esta razão, o processo de otimização pode ser oscilatório e lento. Em redes PMC, normalmente, o espaço de parâmetros é, na realidade, multidimensional. A intuição do processo descrito permanece a mesma.

Variações da Equação 6 foram criadas com o objetivo de acelerar a taxa de convergência e aumentar a estabilidade do processo de otimização dos parâmetros. Este trabalho descreve o algoritmo *Adam* (KINGMA; BA, 2015), por ser uma das principais variações da descida do gradiente. O algoritmo *Adam* combina duas variações da Equação 6 para a atualização dos parâmetros, são elas: a otimização com *momento* (SUTSKEVER et al., 2013) e o algoritmo *RMSProp* (ZOU et al., 2019). Uma descrição breve desses algoritmos é apresentada a seguir.

A descida do gradiente com *momento* introduz um termo $\vec{m}_k$ representando a média móvel do gradiente, nas atualizações da Equação 6. O momento na iteração k, $\vec{m}_k$, é calculado de acordo com a expressão abaixo:

$$\vec{m}_{k+1} = \beta_1 \cdot \vec{m}_k + (1 - \beta_1) \cdot \nabla_{\theta_k} E \tag{7}$$

O termo β_1 é um hiper-parâmetro[2] que determina o peso da média móvel exponencial. A descida do gradiente com momento atualiza os parâmetros de acordo com a Equação 8:

$$\theta_{k+1} = \theta_k - \eta \cdot \vec{m_k} \tag{8}$$

O algoritmo *RMSProp*, de forma similar, mantém uma média móvel exponencial, de acordo com a Equação 9.

$$\vec{v}_{k+1} = \beta_2 \cdot \vec{v}_k + (1 - \beta_2) \cdot \left[\nabla_{\theta_k} E \odot \nabla_{\theta_k} E\right] \tag{9}$$

E os parâmetros são atualizados de acordo com a Equação 10:

$$\theta_{k+1} = \theta_k - \eta \cdot \nabla_{\theta_k} E / \sqrt{\vec{v_k} + \epsilon} \tag{10}$$

O termo ϵ é um valor arbitrariamente pequeno para evitar divisões por zero nas iterações iniciais, já que $\vec{v}_0 = \vec{0}$. A mesma inicialização nula ocorre no algoritmo de descida do gradiente com *momento*.

Tanto o algoritmo *RMSprop* quanto a descida do gradiente com *momento* tendem

[2] Um *hiper-parâmetro* é uma variável com valor determinado antes do início da otimização de um modelo de aprendizado de máquinas.

a acelerar a otimização dos parâmetros e estabilizar as oscilações ilustradas na Figura 10. O algoritmo *Adam*, descrito em seguida, combina essas duas abordagens.

Algoritmo 1 Adam

requerer: η ▷ Taxa de aprendizado
requerer: $\beta_1, \beta_2 \in [0, 1]$ ▷ Taxas de decaimento dos momentos
requerer: $E(\theta)$ ▷ Função de custo
requerer: θ_0 ▷ Valores iniciais dos parâmetros

1: **procedimento** OTIMIZAR(θ)
2: $\quad \vec{m}_0, \vec{v}_0, k \leftarrow 0$
3: $\quad$ **enquanto** θ_k não converge **faça**
4: $\quad\quad k \leftarrow k + 1$
5: $\quad\quad g_k \leftarrow \nabla_\theta E(\theta_{k-1})$ ▷ Obtenção do gradiente da função de custo
6: $\quad\quad \vec{m}_k \leftarrow \beta_1 \cdot \vec{m}_{k-1} + (1 - \beta_1) \cdot g_k$
7: $\quad\quad \vec{v}_k \leftarrow \beta_2 \cdot \vec{v}_{k-1} + (1 - \beta_2) \cdot [g_k \odot g_k]$
8: $\quad\quad \vec{m}_k \leftarrow \vec{m}_k / (1 - \beta_1^k)$ ▷ Correção de viés para o primeiro momento
9: $\quad\quad \vec{v}_k \leftarrow \vec{v}_k / (1 - \beta_2^k)$ ▷ Correção de viés para o segundo momento
10: $\quad\quad \theta_k \leftarrow \theta_{k-1} - \eta \cdot \vec{m}_k / (\sqrt{\vec{v}_k} + \xi)$ ▷ Atualização dos parâmetros. $\xi \ll 1$
11: $\quad$ **fim enquanto**
12: **fim procedimento**

Pode-se notar na linha 5 do Algoritmo 1 que o Adam exige o cálculo do gradiente da função de custo, em relação aos parâmetros da rede neural. A seção seguinte explora duas formas de obter este gradiente para redes PMCs.

1.3 Retropropagação do Gradiente

A utilização da descida do gradiente para otimizar os parâmetros de uma rede PMC necessita do cálculo do gradiente da função de custo em relação aos parâmetros de cada neurônio artificial. Redes PMCs são funções compostas, portanto, o gradiente é encontrado através da aplicação da regra da cadeia.

A aplicação da regra da cadeia para cada um dos parâmetro da rede neural é um processo computacionalmente intensivo se realizado de maneira ingênua. O algoritmo *backpropagation* (RUMELHART; HINTON; WILLIAMS, 1986)[3] fornece um método para calcular a derivada da função de custo em relação a qualquer parâmetro da rede PMC de

[3] Há um debate sobre quem inventou o algoritmo *backpropagation*. O texto de Schmidhuber (2014) discute o assunto de forma aprofundada.

forma dinâmica, estabelecendo uma relação entre as derivadas de camadas subsequentes.

O algoritmo *backpropagation* começa por uma etapa chamada de *forward*, na qual o vetor de entrada, $\vec{x} = \vec{h}^{(0)}$, é propagado através das camadas da rede neural, por meio de sucessivas aplicações da Equação 1, até que se encontre o valor de saída da rede PMC, $\vec{c} = \vec{h}^{[L]}$. A derivada da função de custo em relação à última camada é um vetor de *erro*, $\vec{\delta}^{[L]}$, que, pela regra da cadeia, é retro-propagado para as camadas anteriores. Esta etapa é chamada de *backward*. O algoritmo completo é descrito a seguir (NIELSEN, 2015), com base na notação estabelecida na Seção 1.1.

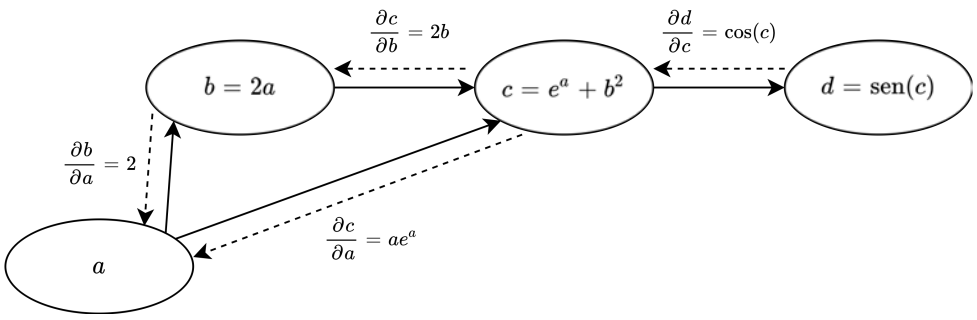
Algoritmo 2 Backpropagation

procedimento FORWARD($\vec{x}^{(i)}$)
  $\vec{h}^{[0]} \leftarrow \vec{x}^{(i)}$
  para todo $l \in \{1 \dots L\}$ **faça** ▷ Cálculo do vetor de saída
    $\vec{k}^{[l]} = \vec{w}^{[l]}\vec{h}^{[l-1]} + \vec{b}^{[l]}$
    $\vec{h}^{[l]} = \varphi(k^{[l]})$
  fim para
fim procedimento
procedimento BACKWARD($\vec{h}^{[L]}$)
  $\vec{\delta}^{[L]} \leftarrow \nabla_{\vec{h}^{[L]}} E \odot \varphi'(k^{[L]})$ ▷ Erro da última camada
  para todo $l \in \{[L-1] \dots 1\}$ **faça**
    $\vec{\delta}^{[l]} = ((w^{[l+1]})^T \vec{\delta}^{[l+1]}) \odot \varphi'(k^{[l]})$ ▷ Erro das camadas anteriores
  fim para
  para todo $w_{jk}^{[l]}, b_j^{[l]} \in \theta$ **faça**
    $\frac{\partial E}{\partial w_{jk}^{[l]}} = h_k^{[l-1]} \vec{\delta}_j^{[l]}$ ▷ Derivada do erro em relação aos pesos
    $\frac{\partial E}{\partial b_j^{[l]}} = \vec{\delta}_j^{[l]}$ ▷ Derivada do erro em relação à constante
  fim para
fim procedimento

A grande inovação trazida pelo Algoritmo 2 é o cálculo recursivo da derivada do erro, em relação às camadas da rede PMC. Isto pode ser visto como uma forma de programação dinâmica (BELLMAN, 1966), onde a derivada do erro de uma camada é utilizada para calcular a derivada do erro das camadas anteriores. Os valores $\vec{\delta}^{[1]}$ a $\vec{\delta}^{[L]}$ podem, então, ser utilizados para calcular a derivada da função de custo em relação à qualquer parâmetro da rede PMC.

As equações apresentadas no Algoritmo 2 podem ser obtidas, de forma possivelmente mais simples, ao tratar uma rede neural PMC como um *grafo computacional*. Esta abstração permite a obtenção do gradiente de funções compostas complexas em relação à um parâmetro qualquer e é implementada em diversos programas de aprendizado profundo, como o *Tensorflow* (ABADI et al., 2016) ou o *PyTorch* (PASZKE et al., 2017). Uma breve descrição desta abordagem é apresentada a seguir pois, além de ser um componente fundamental na aplicação de redes neurais atualmente, permite o entendimento

Figura 11 - Exemplo da etapa de definição do grafo computacional



Legenda: A relação entre as variáveis é estabelecida formando um grafico acíclico.

Fonte: O autor, 2022.

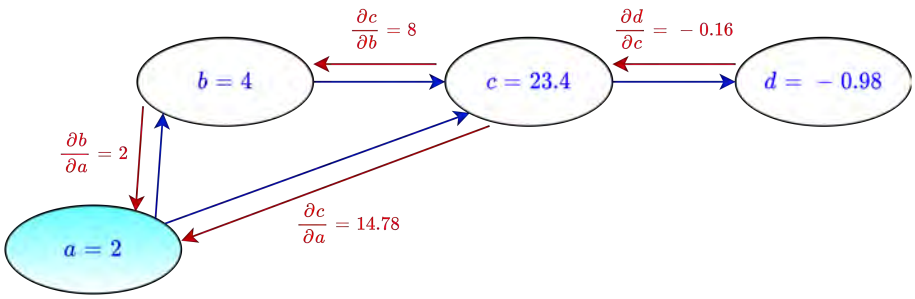
de alguns conceitos das seções seguintes.

Gráficos computacionais são estruturas acíclicas que representam as relações entre diversas variáveis. Na Figura 11, utilizada como exemplo, as setas que ligam uma variável à outra indicam a relação matemática que existe entre essas entidades. As setas tracejadas na direção oposta representam a derivada parcial de uma variável em relação à sua antecessora. Nota-se o conhecimento de uma fórmula analítica para cada derivada parcial. Em grafos computacionais estáticos, a estrutura da Figura 11 é montada sem que o valor de entrada, neste caso a variável a, seja conhecida. A partir do momento que o valor de a é fornecido, uma etapa análoga à fase *forward* do Algoritmo 2 ocorre, na qual os valores de todas as variáveis são calculados. Em seguida, é possível calcular o valor de cada derivada parcial do grafo. Esse processo é exemplificado na Figura 12.

Na etapa seguinte, a derivada parcial de uma variável qualquer, em relação à outra, é obtida ao considerar todos os caminhos reversos que ligam essas variáveis. Um valor numérico para cada caminho é calculado, a partir da multiplicação das derivadas parciais que compõe a trajetória. O somatório dos valores de todos os caminhos possíveis representa a derivada parcial entre essas variáveis. Essa representação gráfica da regra da cadeia é uma abstração que facilita o desenvolvimento de algoritmos de diferenciação automática (BAYDIN et al., 2018). Utilizando como exemplo o grafo computacional da Figura 12, para calcular a derivada de d em relação à a, deve-se considerar todas as trajetórias que levam o nó d até a, isto é: $dcba$ e dca, como indica a Figura 13. A Equação 11 apresenta a expressão da derivada, obtida ao considerar esses caminhos.

$$\frac{\partial d}{\partial a} = \underbrace{\frac{\partial d}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a}}_{\text{Caminho } dcba} + \underbrace{\frac{\partial d}{\partial c}\frac{\partial c}{\partial a}}_{\text{Caminho } dca} = \frac{\partial d}{\partial c}\left(\frac{\partial c}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial c}{\partial a}\right) \tag{11}$$

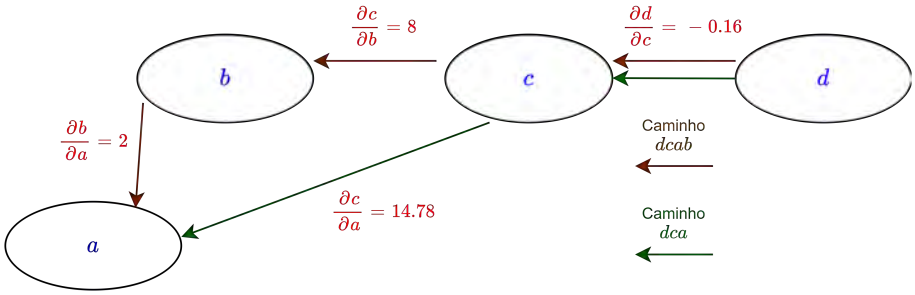
Figura 12 - Exemplo da etapa *forward* do grafo computacional



Legenda: A assimilação $a = 2$ torna possível o cálculo de todas as variáveis (indicadas em azul). As derivadas parciais assumem, então, valores numéricos, indicados em vermelho.

Fonte: O autor, 2022.

Figura 13 - Exemplo da etapa *backward* do grafo computacional



Legenda: A derivada $\partial d/\partial a$ considera todos os caminhos reversos do grafo computacional que levam de d até a.

Fonte: O autor, 2022.

Nota-se que o conceito de programação dinâmica, presente no algoritmo *backpropagation*, também se aplica nos grafos computacionais apresentados. Mais especificamente, na Figura 11, não é necessário calcular a derivada $\partial d/\partial c$ para todos os caminho, uma vez que essa quantidade é obtida em um dos caminhos e pode ser reutilizada em outras trajetórias.

Esta seção descreveu como o gradiente de funções compostas complexas, como as redes PMCs, podem ser obtidos através de grafos computacionais. Esta abstração facilita a otimização de redes neurais mais complexas, como os modelos probabilísticos abordados na seção seguinte. Além disso, a visualização dos grafos computacionais pode auxiliar o entendimento dos desafios associados à otimização desses modelos.

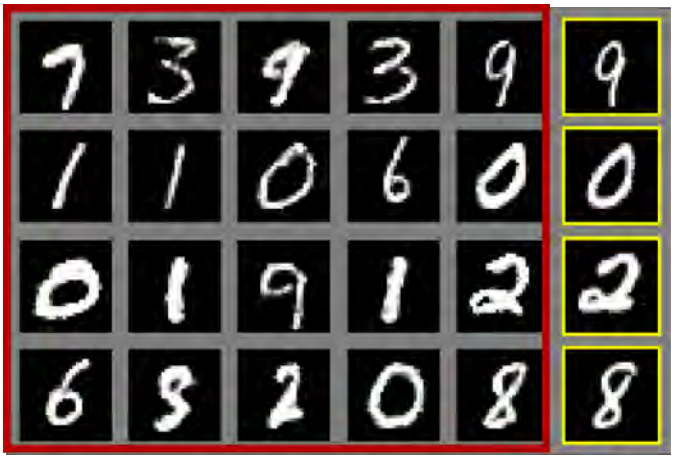
1.4 **Modelos Probabilísticos**

No aprendizado de máquinas, muitas vezes busca-se encontrar uma distribuição de probabilidades a partir de um conjunto de dados. No caso mais comum, busca-se um modelo probabilístico condicional $p_\theta(\vec{c} \mid \vec{x})$ que aproxime uma distribuição $p^*(\vec{c} \mid \vec{x})$, geradora de um conjunto de dados $\mathcal{D} = \{(\vec{x}^{(i)}, \vec{c}^{(i)})\}_{i=1}^{D}$. Por exemplo, em problemas de classificação, pode-se usar uma rede neural PMC para obter uma aproximação $p_\theta(\vec{c} \mid \vec{x}) \approx p^*(\vec{c} \mid \vec{x})$. Se a camada de saída apresenta a função de ativação *softmax*, $p_\theta(\vec{c} \mid \vec{x})$ representa uma distribuição discreta que fornece a probabilidade de $\vec{x}$ pertencer a cada classe.

Em algumas situações, busca-se apenas aproximar uma distribuição não-condicional $p_\theta(\vec{x}) \approx p^*(\vec{x})$, a partir de um conjunto de dados $\mathcal{D} = \{\vec{x}^{(i)}\}_{i=1}^{D}$. Por exemplo, na base de dados MNIST (DENG, 2012), com imagens em preto e branco de dígitos escritos à mão, cada $\vec{x}^{(i)}$ é um vetor de dimensão $28 \times 28 = 784$. Ao invés de classificar uma imagem $\vec{x}^{(i)}$, pode haver um interesse em criar novas imagens $\hat{\vec{x}}$, como mostra a Figura 14. Um modelo probabilístico deste tipo, possivelmente parametrizado por uma rede neural PMC, que gera exemplos "artificiais" é conhecido como *modelo generativo*.

Nota-se que para muitos situações em que se busca gerar novos dados, como o problema da Figura 14, apesar da variável $\vec{x}$ apresentar-se em alta dimensão (28×28), é possível admitir que um vetor $\vec{z}$, de baixa dimensão, codifique $\vec{x}$. Embora apenas a variável $\vec{x}$ seja observável, admite-se a existência de um conjunto de vetores $\{\vec{z}_i\}_{i=1\ldots}$ associados a cada $\vec{x}^{(i)}$, como uma variável "escondida". Modelos probabilísticos que assumem a existência de um vetor $\vec{z}$ não-observável são chamados de modelos *latentes* e $\vec{z}$ é conhecida como variável latente. O Exemplo 3 apresenta um conjunto de dados que admite uma representação por meio de modelos probabilísticos latentes.

Figura 14 - Imagens reais e artificiais de dígitos manuscritos

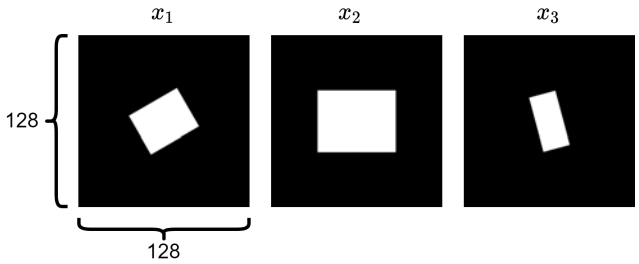


Legenda: Uma rede neural generativa (GAN - *generative adversarial networks*) gera imagens de dígitos manuscritos (à direita em amarelo). Exemplos do conjunto de dados original estão à esquerda, envolvidos por um retângulo vermelho.

Fonte: Adaptado de Goodfellow et al. (2016).

Exemplo 3 *Suponha a existência de um conjunto de dados* $\mathcal{D} = \{\vec{x}^{(i)}\}_{i=1}^{100}$*, em que cada* $\vec{x}^{(i)}$ *corresponde à uma imagem de dimensão* $128 \times 128 = 16384$*. Sabe-se que cada* $\vec{x}^{(i)}$ *é uma imagem de um retângulo de dimensões* $W \times H$ *com um ângulo de rotação* α*. A Figura 15 mostra alguns exemplos desse conjunto de dados.*

Figura 15 - Banco de dados com imagens de retângulos



Fonte: O autor, 2022.

A variável observável, $\vec{x}$*, está em alta dimensão, porém é possível reconstruir cada exemplo da figura acima se as dimensões do retângulo e sua orientação são conhecidas. Nesse caso, para cada* $\vec{x}^{(i)} \in \mathcal{D}$ *existe um conjunto de vetores latentes* $\{\vec{z}_j = [W_j, H_j, \alpha_j]\}_{j=1\ldots}$ *que contêm a informação necessária para reconstruir a imagem* $\vec{x}^{(i)}$ *em alta dimensão, ou seja, é possível elaborar um algoritmo que recebe como entrada as dimensões e orientação do retângulo e que retorne a imagem correspondente. Nota-se que para um quadrado com orientação* α_0*, e sua imagem correspondente* $\vec{x_0}$*, os vetores latentes associados têm* $\alpha_j = \alpha_0 + j \cdot 90°$*. Isto exemplifica o fato de que para cada exemplo* $\vec{x}^{(i)} \in \mathcal{D}$ *pode existir uma distribuição de vetores latentes.*

Em geral, modelos probabilísticos latentes são fatorados da seguinte forma:

$$p(\vec{x}, \vec{z}) = p(\vec{z})p(\vec{x} \mid \vec{z}) \tag{12}$$

Observa-se que $p(\vec{x})$ pode ser uma distribuição extremamente complexa mas um modelo probabilístico latente permite expressar $p(\vec{x})$ como um produto de distribuições possivelmente simples e tratáveis, por exemplo, para uma variável aleatória $\vec{z}$ discreta:

$$p(\vec{x}) = \sum_{z} p(\vec{x} \mid \vec{z})p(\vec{z}) \tag{13}$$

É possível também que o modelo probabilístico latente seja condicionado em uma outra variável $\vec{y}$:

$$p(\vec{x} \mid \vec{y}) = \sum_{z} p(\vec{x} \mid \vec{y}, \vec{z})p(\vec{z}) \tag{14}$$

A discussão da Seção 1.5 assume um modelo probabilístico latente incondicional, como o da Equação 13, mas os conceitos discutidos podem ser estendidos para distribuições condicionadas.

1.5 **Autocodificador Variacional**

Modelos probabilísticos latentes quando utilizados com redes neurais profundas, como uma rede PMC, são denominados *modelos profundos de variável latente* e abreviados por DLVM (*deep latent-variable models*). O paradigma do autocodificador variacional (*variational autoencoder* - VAE) (KINGMA; WELLING, 2013), descrito nesta seção, fornece um método eficiente para otimizar DLVMs.

A abordagem natural para otimizar modelos probabilísticos consiste em minimizar a função LV negativa (Equação 15). De forma equivalente, o objetivo de aproximar uma distribuição $p^*(\vec{x})$ com uma rede neural parametrizada por um conjunto de parâmetros θ pode ser escrito como:

$$\theta \leftarrow \arg\max_{\theta} \frac{1}{D} \sum_{i=1}^{D} \log p_\theta(\vec{x}^{(i)}) \tag{15}$$

Para modelos probabilísticos latentes a Equação 15 se decompõe da seguinte forma:

$$\theta \leftarrow \arg\max_{\theta} \frac{1}{D} \sum_{i=1}^{D} \log \left(\int_{\vec{z}} p_\theta(\vec{x}^{(i)} \mid \vec{z})\, p_\theta(\vec{z}) \right) \tag{16}$$

A integral da Equação 16 é intratável, especialmente quando $\vec{z}$ é uma variável contínua. Mesmo que seja possível calcular a integral e avaliar o objetivo, realizar o cálculo para cada exemplo do conjunto de dados é inviável. Uma alternativa é mudar o critério de otimização para o valor esperado da função LV, dessa forma é possível estimar o critério com amostras:

$$\theta \leftarrow \arg\max_{\theta} \frac{1}{D} \sum_{i=1}^{D} \mathbb{E}_{\vec{Z} \sim p_\theta(\vec{z} \mid \vec{x}^{(i)})} \left[\log p_\theta(\vec{x}^{(i)}, \vec{Z}) \right] \tag{17}$$

Conforme a discussão da Seção 1.4, para cada $\vec{x}^{(i)}$ há uma distribuição de vetores latentes, $p(\vec{z} \mid \vec{x}^{(i)})$, porém essa distribuição é desconhecida e para obter uma estimativa $p_\theta(\vec{z} \mid \vec{x})$, pelo teorema de Bayes, seria necessário obter a aproximação $p_\theta(\vec{x})$, que é o problema em questão. Portanto, não é possível realizar a amostragem indicada na Equação 17. Então, realiza-se uma aproximação da seguinte forma:

$$p_\theta(\vec{z} \mid \vec{x}^{(i)}) \approx q_i(\vec{z}) = \mathcal{N}(\vec{\mu}_i, \vec{\sigma}_i) \tag{18}$$

Na Equação 18, o subscrito i indica que distribuição $q_i(\vec{z})$ é específica ao exemplo $\vec{x}^{(i)}$ do conjunto de dados, dessa forma, um conjunto de parâmetros $\{(\vec{\mu}_i, \vec{\sigma}_i)\}_{i=1}^{D}$ deve ser utilizado para a base de dados como um todo. A distribuição normal multivariada é uma escolha comum para essa aproximação, entretanto outra distribuição simples pode ser utilizada. Com isso, é possível obter um limite inferior para a função LV. A partir da Equação 16:

$$\log\left(p_\theta(\vec{x}^{(i)})\right) = \log\left(\int_z p_\theta(\vec{x}^{(i)} \mid \vec{z})\, p_\theta(\vec{z})\right) \tag{19a}$$

$$= \log\left(\int_z p_\theta(\vec{x}^{(i)} \mid \vec{z})\, p_\theta(\vec{z}) \frac{q_i(\vec{z})}{q_i(\vec{z})}\right) \tag{19b}$$

$$= \log\left(\mathbb{E}_{\vec{Z} \sim q_i(\vec{z})}\left[\frac{p_\theta(\vec{x}^{(i)} \mid \vec{Z})\, p_\theta(\vec{Z})}{q_i(\vec{Z})}\right]\right) \tag{19c}$$

Utilizando a desigualdade de Jensen[4] (JENSEN, 1906):

$$\log\left(\mathbb{E}_{\vec{Z} \sim q_i(\vec{z})}\left[\frac{p_\theta(\vec{x}^{(i)} \mid \vec{Z})\, p_\theta(\vec{Z})}{q_i(\vec{z})}\right]\right) \geq \mathbb{E}_{\vec{Z} \sim q_i(\vec{z})}\left[\log\left(\frac{p_\theta(\vec{x}^{(i)} \mid \vec{Z})\, p_\theta(\vec{Z})}{q_i(\vec{z})}\right)\right] \tag{20}$$

[4] Em teoria de probabilidade, o teorema de Jensen costuma ser declarado como: se X é uma variável aleatória e f uma função *convexa*, então $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$. A função log é *côncava*, portanto, a relação é $\log(\mathbb{E}[X]) \geq \mathbb{E}[\log(X)]$.

Ao expandir o lado direito da desigualdade, obtêm-se:

$$\log p_\theta(\vec{x}^{\,(i)}) \geq \underbrace{\mathbb{E}_{\vec{Z}\sim q_i(\vec{z})}\left[\log p_\theta(\vec{x}^{\,(i)} \mid \vec{Z}) + \log p_\theta(\vec{Z})\right] - \mathbb{E}_{\vec{Z}\sim q_i(\vec{z})}\left[\log q_i(\vec{Z})\right]}_{\mathcal{L}_i} \tag{21}$$

O lado direito da Equação 21 é conhecido como *limite inferior variacional* ou *evidence lower bound* (ELBO), pois age como um limite inferior para a função LV de $p_\theta(\vec{x}^{\,(i)})$. Espera-se que a maximização desse limite inferior otimize o lado esquerdo da Equação 21, especialmente se $\log p_\theta(\vec{x}^{\,(i)})$ se encontra próximo desse limite. É possível mostrar que quando a aproximação $q_i(\vec{z}) \approx p_\theta(\vec{z} \mid \vec{x}^{\,(i)})$ é razoável, o lado esquerdo da Equação 21 de fato encontra-se próximo do ELBO. Isto pode ser observado ao verificar a discrepância de $q_i(\vec{z})$ em relação a $p_\theta(\vec{z} \mid \vec{x}^{\,(i)})$, por meio da divergência de Kullback-Leibler:

$$D_{KL}\left(q_i(\vec{z}) \,||\, p_\theta(\vec{z} \mid \vec{x}^{\,(i)})\right) = \mathbb{E}_{\vec{Z}\sim q_i(\vec{z})}\left[\log\left(\frac{q_i(\vec{Z})}{p_\theta(\vec{Z} \mid \vec{x}^{\,(i)})}\right)\right] \tag{22a}$$

$$= \mathbb{E}_{\vec{Z}\sim q_i(\vec{z})}\left[\log\left(\frac{q_i(\vec{Z})\, p_\theta(\vec{x}^{\,(i)})}{p_\theta(\vec{x}^{\,(i)}, \vec{Z})}\right)\right] \tag{22b}$$

$$= \mathbb{E}_{\vec{Z}\sim q_i(\vec{z})}\left[\log\left(\frac{q_i(\vec{Z})\, p_\theta(\vec{x}^{\,(i)})}{p_\theta(\vec{x}^{\,(i)} \mid \vec{Z})\, p_\theta(\vec{Z})}\right)\right] \tag{22c}$$

$$\begin{aligned} = -\mathbb{E}_{\vec{Z}\sim q_i(\vec{z})}&\left[\log p_\theta(\vec{x}^{\,(i)} \mid \vec{Z}) + \log p_\theta(\vec{Z})\right] + \\ &\mathbb{E}_{\vec{Z}\sim q_i(\vec{z})}\left[\log q_i(\vec{Z})\right] + \mathbb{E}_{\vec{Z}\sim q_i(\vec{z})}\left[\log p_\theta(\vec{x}^{\,(i)})\right] \end{aligned} \tag{22d}$$

Nas Equações 22b e 22c, as igualdade $p_\theta(\vec{x}^{\,(i)}, \vec{z}) = p_\theta(\vec{x}^{\,(i)}) p_\theta(\vec{z} \mid \vec{x}^{\,(i)}) = p_\theta(\vec{z} \mid \vec{x}^{\,(i)}) p_\theta(\vec{z})$ foram utilizadas. Na Equação 22d é possível reconhecer o ELBO, dessa forma:

$$\log p_\theta(\vec{x}^{\,(i)}) = \mathcal{L}_i + D_{KL}\left(q_i(\vec{z}) \,||\, p_\theta(\vec{z} \mid \vec{x}^{\,(i)})\right) \tag{23}$$

Como a divergência KL é sempre maior que zero, a distância entre $p_\theta(\vec{x}^{\,(i)})$ e o ELBO depende exclusivamente da aproximação $q_i(\vec{z}) \approx p_\theta(\vec{z} \mid \vec{x}^{\,(i)})$. Se a aproximação é perfeita, a divergência KL é nula e $\log(p_\theta(\vec{x}^{\,(i)}))$ é igual ao ELBO.

Nota-se na Equação 23 que a maximização do ELBO em relação aos parâmetros $\{(\vec{\mu}_i, \vec{\sigma}_i)\}_{i=1}^{D}$ irá minimizar a divergência KL, melhorando a aproximação $q_i(\vec{z}) \approx p_\theta(\vec{z} \mid \vec{x}^{\,(i)})$. Se a otimização do ELBO é realizada em relação aos parâmetros θ a tendência é que a função LV seja maximizada. Dessa forma, muda-se o objetivo apresentado na

Equação 15 para o ELBO:

$$\theta, \vec{\mu}_i\,, \vec{\sigma}_i \leftarrow \arg\max_{\theta, \vec{\mu}_i\,, \vec{\sigma}_i} \frac{1}{D} \sum_{i=1}^{D} \mathcal{L}_i \tag{24}$$

Se o número de exemplos do conjunto de dados ($\mathcal{D}$) é grande, torna-se inviável otimizar cada uma das D distribuições em $\{q_i(\vec{z})\}_{i=1}^{D}$. Com isso, pode-se realizar o processo de inferência variacional *amortizada*, onde os parâmetros $\boldsymbol{\phi}$, de uma única rede PMC, por exemplo, é utilizado para todos os D exemplos do conjunto de dados. Nesse caso, a aproximação da Equação 18 se torna:

$$p_\theta(\vec{z} \mid \vec{x}) \approx q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}) \tag{25}$$

Com esta nova aproximação, pode-se expressar o ELBO (Equação 21) como:

$$\mathcal{L}_i = \mathbb{E}_{\vec{Z} \sim q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)})} \left[\log p_\theta(\vec{x}^{(i)} \mid \vec{Z}) + \log p_\theta(\vec{Z}) \right] - \mathbb{E}_{\vec{Z} \sim q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)})} \left[\log q_{\boldsymbol{\phi}}(\vec{Z} \mid \vec{x}^{(i)}) \right] \tag{26a}$$

$$= \mathbb{E}_{\vec{Z} \sim q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)})} \left[\log p_\theta(\vec{x}^{(i)} \mid \vec{Z}) \right] - \mathbb{E}_{\vec{Z} \sim q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)})} \left[\log q_{\boldsymbol{\phi}}(\vec{Z} \mid \vec{x}^{(i)}) - \log p_\theta(\vec{Z}) \right] \tag{26b}$$

$$= \mathbb{E}_{\vec{Z} \sim q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)})} \left[\log p_\theta(\vec{x}^{(i)} \mid \vec{Z}) \right] - D_{KL} \left(q_{\boldsymbol{\phi}}(\vec{Z} \mid \vec{x}^{(i)}) \,||\, p_\theta(\vec{Z}) \right) \tag{26c}$$

Portanto, o objetivo da Equação 24 é:

$$\theta, \boldsymbol{\phi} \leftarrow \arg\max_{\theta, \boldsymbol{\phi}} \frac{1}{D} \sum_{i=1}^{D} \Bigg[\mathbb{E}_{\vec{Z} \sim q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)})} \left[\log p_\theta(\vec{x}^{(i)} \mid \vec{Z}) \right] - D_{KL} \left(q_{\boldsymbol{\phi}}(\vec{Z} \mid \vec{x}^{(i)}) \,||\, p_\theta(\vec{Z}) \right) \Bigg] \tag{27}$$

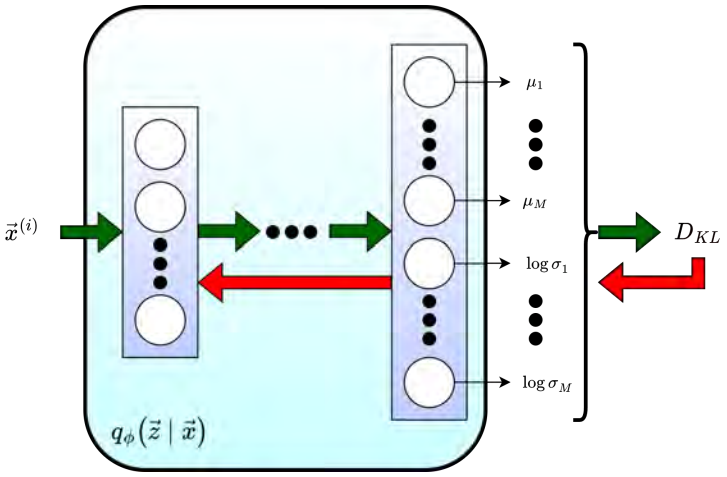
No início desta seção foi mencionado que $p_\theta(\vec{z})$ pode ser distribuição simples e tratável. Uma escolha viável e comum é $p_\theta(\vec{z}) = \mathcal{N}(\vec{0}, \vec{1})$. Se $q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x})$ é também uma distribuição normal multivariada com matriz de covariância diagonal (parametrizada por uma rede PMC), então a divergência KL da Equação 26 possui uma expressão analítica. Seja M a dimensão do vetor latente, $\vec{z}$, então:

$$D_{KL} \left(q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)}) \,||\, p(\vec{z}) \right) = D_{KL} \left(\mathcal{N} \left(\vec{\mu}_{\boldsymbol{\phi}}(\vec{x}^{(i)}), \vec{\sigma}_{\boldsymbol{\phi}}(\vec{x}^{(i)}) \right) \,||\, \mathcal{N}(\vec{0}, \vec{1}) \right) \tag{28a}$$

$$= \frac{1}{2} \sum_{j=1}^{M} \log \left(\sigma^2_{\boldsymbol{\phi}_j}(\vec{x}^{(i)}) \right) + \sigma^2_{\boldsymbol{\phi}_j}(\vec{x}^{(i)}) + \mu^2_{\boldsymbol{\phi}_j}(\vec{x}^{(i)}) + 1 \tag{28b}$$

A Figura 16 ilustra a possibilidade de otimização dos parâmetros $\boldsymbol{\phi}$, por meio de grafos computacionais, ao calcular a divergência KL da Equação 28, para um exemplo $\vec{x}^{(i)}$ no conjunto de dados.

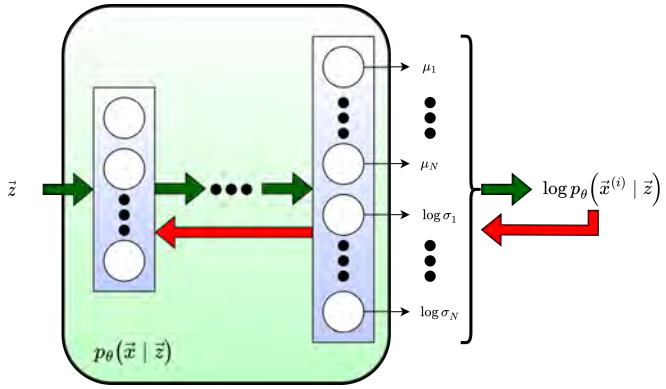
Figura 16 - Gradiente em relação aos parâmetros ϕ



Legenda: As setas verdes (que apontam para a direita) indicam a propagação direta de um vetor $\vec{x}^{(i)}$ através da rede PMC $q_{\boldsymbol{\phi}}$. A expressão analítica para a divergência KL apresentada na Equação 28 é, então, calculada para a entrada $\vec{x}^{(i)}$. As setas vermelhas (que apontam para a esquerda) indicam a propagação reversa do gradiente, tornando possível a otimização de $q_{\boldsymbol{\phi}}$.

Fonte: O autor, 2022.

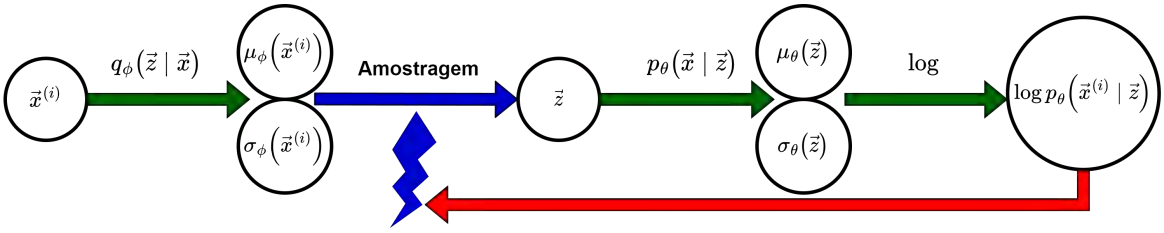
Figura 17 - Gradiente da função LV em relação à θ



Legenda: Propagação direta (setas verdes) do vetor latente $\vec{z}$ pela rede neural $p_\theta(\vec{x} \mid \vec{z})$ permite o cálculo da função LV e a obtenção do gradiente (setas vermelhas) em relação aos parâmetros θ.

Fonte: O autor, 2022.

Figura 18 - Gradiente em relação à ϕ sob operação de amostragem



Legenda: A seta azul indica a operação de amostragem que impossibilita a propagação do gradiente.

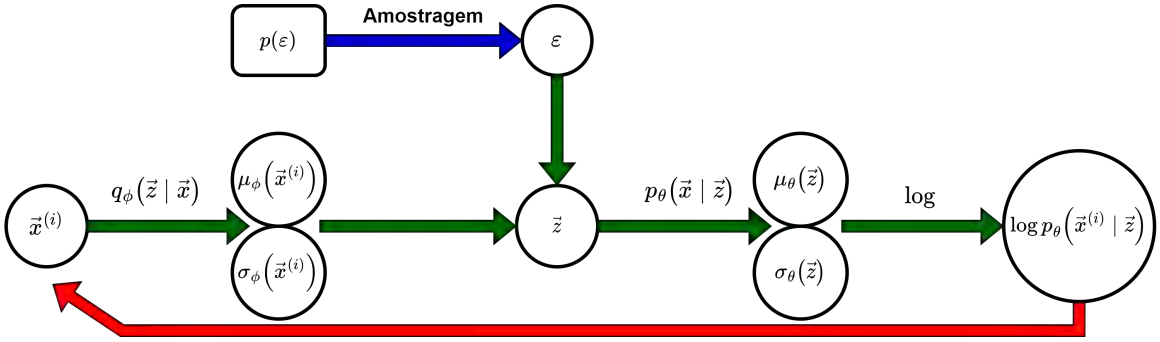
Fonte: O autor, 2022.

Uma maneira de obter o gradiente de $\mathcal{L}_i$ em relação aos parâmetros θ é tomar amostras $\vec{Z} \sim q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)})$ e estimar a função LV, $\log p_\theta(\vec{x}^{(i)} \mid \vec{z})$. A parcela da Equação 26 referente à divergência KL não depende de θ, portanto, não influencia a otimização deste conjunto de parâmetros. A obtenção de $\nabla_\theta \mathcal{L}_i$, em termos de grafos computacionais, é apresentada na Figura 17.

Resta saber se é possível estimar o gradiente em relação aos parâmetros $\boldsymbol{\phi}$, a partir do primeiro termo da Equação 26. Pode-se pensar que uma maneira de estimar esse termo seja tomar uma amostra $\vec{Z} \sim q_{\boldsymbol{\phi}}(\vec{z} \mid \vec{x}^{(i)})$ e calcular $\log p_\theta(\vec{x}^{(i)} \mid \vec{z})$. Entretanto, apesar de $\boldsymbol{\phi}$ influenciar a distribuição que origina $\vec{z}$, não é possível obter gradientes de operações que envolvam amostragem aleatória. Essa situação é representada na Figura 18.

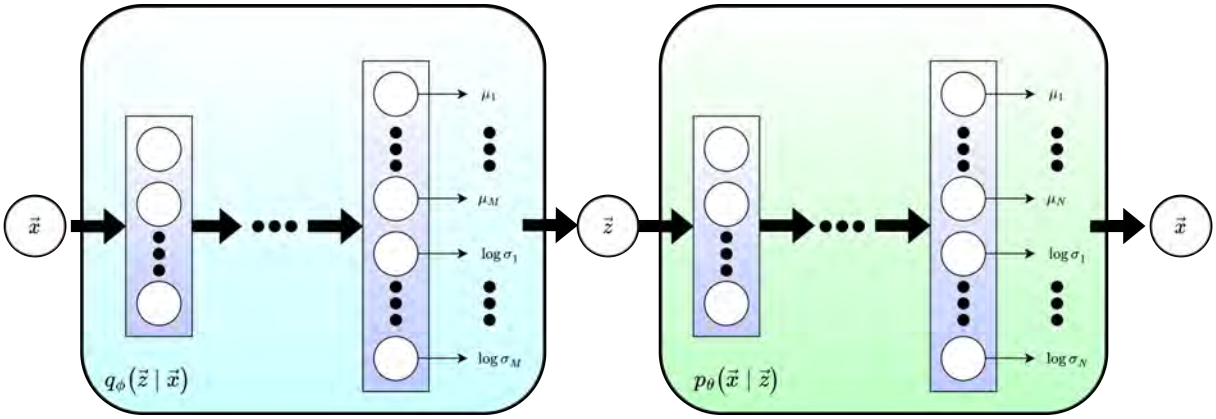
Para variáveis latentes contínuas, é possível utilizar uma reparametrização, expres-

Figura 19 - Gradiente em relação à ϕ após a reparametrização



Fonte: O autor, 2022.

Figura 20 - Modelo de variável latente com redes neurais PMC



Fonte: O autor, 2022.

sando a variável latente aleatória como uma transformação diferenciável:

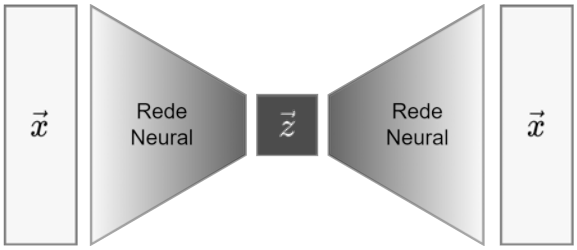
$$\vec{z} = w\left(\phi, \vec{x}, \vec{\epsilon}\right) \tag{29a}$$

$$= \mathbb{E}_{\vec{\epsilon} \sim p(\vec{\epsilon})}\left[\vec{\mu}_\phi(\vec{x}) + \vec{\sigma}_\phi(\vec{x}) \odot \vec{\epsilon}\right] \tag{29b}$$

Na Equação 29b, $\vec{\mu}_\phi(\vec{x})$ e $\vec{\sigma}_\phi(\vec{x})$ são as saídas determinísticas da rede neural $q_\phi(\vec{z} \mid \vec{x})$. Com a reparametrização, o grafo computacional da Figura 18 se torna o apresentado na Figura 19. Nota-se que a fonte de aleatoriedade passa a ser uma variável de entrada $\vec{\epsilon}$ e não há operação de amostragem no caminho entre a função objetivo e a entrada $\vec{x}^{(i)}$. Com isso, a otimização do ELBO em relação aos parâmetros ϕ é possível.

Com isso, conclui-se que a otimização de modelos de variáveis latentes que utilizam redes neurais PMC é possível, considerando a otimização do ELBO em relação aos parâmetros θ e ϕ. A arquitetura completa deste modelo, conforme a discussão desta seção, é apresentada na Figura 20:

Figura 21 - Um auto-codificador



Fonte: O autor, 2022.

Geralmente, a dimensão do vetor de entrada $\vec{x}$ é muito maior que a do vetor latente $\vec{z}$, tal como indica o Exemplo 3. Dessa forma, a rede $q_\phi(\vec{z} \mid \vec{x})$ atua como um codificador (*encoder*), que representa uma variável em alta dimensão ($\vec{x}$) por uma distribuição de possíveis vetores latentes em baixa dimensão ($\vec{z}$). De forma análoga, a rede $p_\theta(\vec{x} \mid \vec{z})$ leva o vetor latente $\vec{z}$ a uma distribuição de variáveis $\vec{x}$ em alta dimensão, por isso é chamado de decodificador (*decoder*).

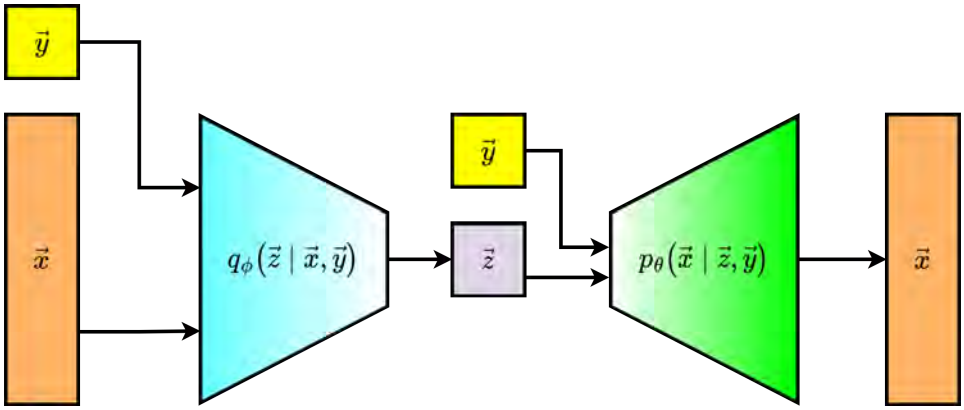
A arquitetura da Figura 20 é semelhante a um modelo conhecido como auto-codificador (*autoencoder*), que busca reconstruir uma variável $\vec{x}$ em alta dimensão passando por uma representação $\vec{z}$ intermediária, em baixa dimensão. Esse "gargalo de informação" induz a rede neural a aprender uma representação compacta do vetor de entrada. A função objetivo do auto-codificador é um erro de reconstrução, que mede a discrepância entre a entrada $\vec{x}$ e a saída $\hat{\vec{x}}$.

A semelhança do auto-codificador com o modelo de variável latente com redes neurais fez com que a arquitetura da Figura 20 ficasse conhecida como auto-codificador variacional (ou *variational autoencoder - VAE*). Outro ponto a ser destacado é que a função objetivo do VAE (Equação 26) pode ser interpretada como uma parcela relacionada à reconstrução, similar ao auto-codificador convencional, mas com a adição de um termo de regularização:

$$\mathcal{L}(\theta, \phi) = \underbrace{\mathbb{E}_{\vec{Z} \sim q_\phi(\vec{z} \mid \vec{x})} \left[\log p_\theta(\vec{x} \mid \vec{Z}) \right]}_{\text{Reconstrução}} - \underbrace{D_{KL} \left(q_\phi(\vec{z} \mid \vec{x}) \mid\mid p(\vec{z}) \right)}_{\text{Regularização}} \tag{30}$$

Em outras palavras, na otimização de um VAE busca-se não apenas reconstruir a entrada, mas também impôr uma forma à distribuição do espaço latente, por meio da divergência de KL. Utilizando como exemplo a Equação 28, o termo de regularização busca tornar $q_\phi(\vec{z} \mid \vec{x}) \approx \mathcal{N}(\vec{0}, \vec{1})$. A imposição de uma forma simples e tratável ao espaço latente, como, por exemplo, a distribuição normal $\mathcal{N}(\vec{0}, \vec{1})$, torna possível a amostragem de vetores $\vec{z}$ desta distribuição, que podem ser propagados por $p_\theta(\vec{x} \mid \vec{z})$ para gerar uma distribuição de entradas $\vec{x}$ "falsas". Tal fato explica a caracterização do VAE como um

Figura 22 - Arquitetura do CVAE



Fonte: O autor, 2022.

modelo generativo.

A definição da primeira parcela da Equação 30 como um erro de reconstrução sustenta-se por outra observação, descrita a seguir. Supondo que a saída do decodificador represente um vetor de médias de uma distribuição normal multivariada, com matriz de covariância unitária, a expressão relativa à reconstrução torna-se:

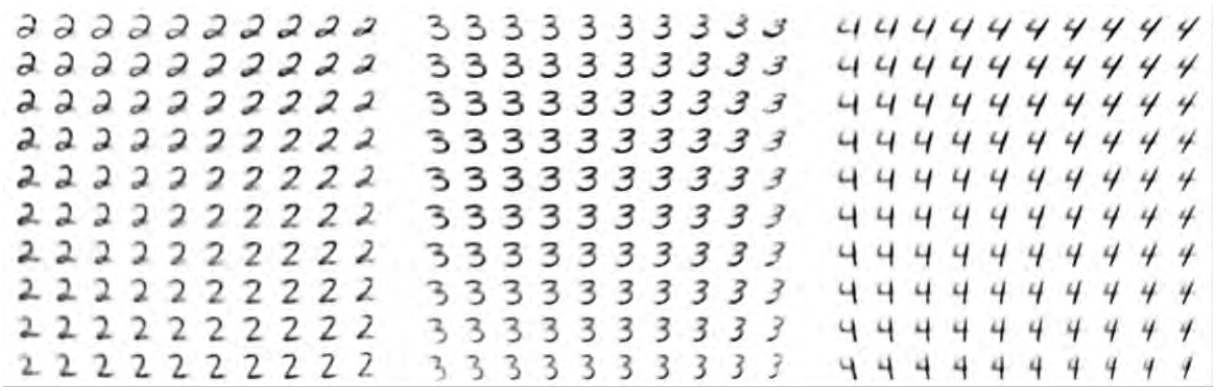
$$\mathbb{E}_{\vec{Z}\sim q_\phi(\vec{z}\mid\vec{x})}\left[\log p_\theta(\vec{x}\mid\vec{Z})\right] = \mathbb{E}_{\vec{Z}\sim q_\phi(\vec{z}\mid\vec{x})}\left[\log\left(\frac{1}{\sqrt{2\pi}}e^{-(\vec{x}-\vec{\mu}_\theta(\vec{Z}))^2/2}\right)\right] \tag{31a}$$

$$= \mathbb{E}_{\vec{Z}\sim q_\phi(\vec{z}\mid\vec{x})}\left[-\frac{\log(2\pi)}{2} - \frac{\left(\vec{x}-\vec{\mu}_\theta(\vec{Z})\right)^2}{2}\right] \tag{31b}$$

A Equação 31 mostra que a função de custo, ignorando a subtração dos valores constantes, é um erro quadrático. Esse critério, por sua vez, costuma ser utilizado em problemas de regressão supervisionada, para analisar o quão perto a predição da rede neural se encontra do valor real. No caso do VAE, o erro quadrático indica a fidelidade que o vetor de entrada $\vec{x}$ é reproduzido na saída.

Foi mencionado na Seção 1.4 que a discussão sobre modelos probabilísticos de variáveis latentes, como o VAE, se estende para modelos condicionados. O VAE condicional ou (*conditional variational autoencoder* - CVAE) é um modelo generativo capaz de gerar variáveis $\vec{x}$ "artificiais", a partir de um vetor $\vec{y}$ de condições. Supondo que no exemplo da Figura 14 seja necessário gerar dígitos manuscritos específicos, então, as duas redes neurais q_ϕ e p_θ representariam distribuições também condicionadas em um vetor $\vec{y}$, como indica a Figura 22. A variável $\vec{y}$ seria utilizada para indicar qual dígito (de 0 a 9) o modelo generativo iria produzir, a partir de amostras do espaço latente. Um exemplo de dígitos manuscritos gerados por um CVAE é apresentado na Figura 23.

Figura 23 - Geração condicional de dígitos manuscritos



Legenda: Um CVAE gera dígitos com base no vetor $\vec{y}$ de entrada (com codificação *one-hot*). Diferentes formas para os dígitos são obtidas ao considerar diversos valores de $\vec{z}$: $\{\vec{z} \in \mathbb{R}^2 \mid z_1, z_2 \in [-5, 5]\}$.

Fonte: Adaptado de Kingma et al. (2014).

Outra importante extensão do VAE está relacionada à forma em que esses modelos são otimizados. Higgins et al. (2016) mostrou que a adição de um hiper-parâmetro β à função objetivo do VAE (Equação 32) é crucial para que o espaço latente torne-se "desemaranhado". Isso significa, para o Exemplo 3, que cada componente de $\vec{z}$ seria responsável por apenas uma mudança específica na imagem $\vec{x}$. Dessa forma, alterar uma componente $\vec{z}$ iria causar apenas uma mudança na orientação do retângulo, sem afetar o seu comprimento ou largura, por exemplo.

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{\vec{Z} \sim q_\phi(\vec{z} \mid \vec{x})} \Big[\log p_\theta(\vec{x} \mid \vec{Z}) \Big] - \beta D_{KL} \left(q_\phi(\vec{z} \mid \vec{x}) \mid\mid p(\vec{z}) \right) \tag{32}$$

A Figura 24 apresenta uma comparação entre o VAE tradicional e o β-VAE. Nessa figura, em cada linha, todas as componentes do vetor latente $\vec{z}$ são mantidas fixas, exceto uma, responsável por codificar a orientação das faces humanas. Nota-se que no VAE convencional, a alteração desta componente causa mudanças mais significativas, se comparadas com o β-VAE, em outros aspectos, como a tonalidade da pele ou as características do cabelo.

Ao observar a Equação 28 é possível notar que a magnitude da divergência KL depende da dimensão do vetor latente $\vec{z}$, enquanto o termo de reconstrução da Equação 30 depende da dimensão da variável $\vec{x}$. Para uniformizar a magnitude de cada parcela da função objetivo em problemas com diferentes dimensões de $\vec{z}$ e $\vec{x}$, utiliza-se um fator β normalizado. Seja N e M as dimensões de $\vec{x}$ e $\vec{z}$, respectivamente, o fator β normalizado

Figura 24 - Comparação do β-VAE com o VAE

Legenda: Comparação qualitativa entre o VAE e o β-VAE, em termos de entrelaçamento do espaço latente. Destaca-se a segunda linha, onde no VAE a alteração da componente responsável pela orientação resulta no aparecimento de faces com óculos. No β-VAE as características dos rostos são semelhantes com as diferentes orientações.

Fonte: Higgins et al. (2016).

é:

$$\beta_{norm} = \frac{\beta N}{M} \tag{33}$$

Esta seção abordou as redes neurais PMCs e como estes modelos podem ser otimizados e utilizados como aproximadores de funções ou modelos generativos. A seção seguinte é dedicada apenas ao problema de planejamento de trajetórias e como algoritmos clássicos podem ser aperfeiçoados com modelos generativos, em particular.

2 PLANEJAMENTO DE TRAJETÓRIAS

O problema de planejamento de trajetórias é central para a robótica móvel. Frequentemente, deseja-se obter não apenas um caminho livre de colisões com os possíveis obstáculos presentes no ambiente de movimentação, mas também otimizar algum critério como a distância total percorrida ou o consumo de energia. Uma das fontes de complexidade do problema é que geralmente as dimensões do robô não podem ser desprezadas, ou seja, não é possível, de imediato, tratar o robô como um ponto no espaço tri-dimensional.

É interessante observar a diferença entre *caminho* e *trajetória*: enquanto o primeiro representa apenas uma sequência de posições e orientações do robô no espaço, o segundo deve associar um instante de tempo a cada membro dessa sequência. Uma vez que o movimento do robô é gerado por atuadores, sistemas com limitações físicas, o planejamento de trajetórias é mais complexo, já que estas limitações devem ser consideradas na associação do tempo à sequencia de posições e orientações.

Esta seção aborda os conceitos fundamentais para a formulação do problema de planejamento de caminhos e trajetórias. Em seguida, os principais métodos de resolução baseados em *árvores de exploração* são apresentados. Finalmente, discute-se o uso de modelos generativos, como o CVAE, exposto na Seção 1.5, para auxiliar a efetividade dos algoritmos de planejamento.

2.1 Formulação do Problema

Seja $\mathscr{A}$ um corpo rígido, representando um robô, que se move em um espaço euclidiano $\mathcal{W} = \mathbb{R}^N$, com $N = 2$ ou $N = 3$. Um conjunto de obstáculos $\{\mathcal{B}_i\}_{i=1\ldots m}$, também representados como corpos rígidos, estão distribuídos em $\mathcal{W}$. Considera-se que as geometrias do robô e dos obstáculos são conhecidas, assim como a posição exata de cada obstáculo. Dada uma *pose* (posição e orientação) inicial do robô, busca-se a geração de uma trajetória τ, composta por uma sequência de poses, ao longo de um intervalo de tempo, de forma que o mesmo alcance uma pose final, evitando colisões com qualquer obstáculo.

Uma *configuração* $\vec{g}$ é um vetor que especifica completamente a posição de cada ponto do robô. O espaço N-dimensional que contém todas as configurações possíveis do robô é representado por $\mathcal{G}$. Denota-se por $\mathscr{A}(\vec{g})$ o subconjunto de $\mathcal{W}$ ocupado pelo robô $\mathscr{A}$ quando o mesmo se encontra na configuração $\vec{g}$. O número de *graus de liberdade* de um robô equivale ao número mínimo de coordenadas necessárias para representar a configuração do robô.

A definição do espaço de configurações $\mathcal{G}$ permite representar o robô como um

ponto, em seu próprio espaço (LYNCH; PARK, 2017). Além disso, é possível projetar cada um dos obstáculos nesse mesmo espaço $\mathcal{G}$. Com isso, um problema de planejamento potencialmente complexo, ao considerar a geometria de cada corpo rígido, traduz-se em encontrar um caminho para um robô pontual.

Formalmente, pode-se projetar cada obstáculo $\mathcal{B} \in \{\mathcal{B}_i\}_{i=1}^{m}$ em $\mathcal{W}$ para o espaço de configurações $\mathcal{G}$, conforme a Equação 34.

$$\mathcal{GB}_i = \{\vec{g} \in \mathcal{G} \mid \mathcal{A}(\vec{g}) \cap \mathcal{B}_i \neq \emptyset\} \tag{34}$$

Desta forma, define-se $\mathcal{G}_{obs}$ como a união de todos os obstáculos no espaço de configurações:

$$\mathcal{G}_{obs} = \bigcup_{i=1}^{m} \mathcal{GB}_i \tag{35}$$

Intuitivamente, $\mathcal{G}_{obs}$ representa o conjunto de configurações do robô que implicam em colisão com algum obstáculo. O Exemplo 4 mostra um procedimento para descobrir $\mathcal{G}_{obs}$ em problemas cuja geometria do robô e dos obstáculos são simples. De forma análoga, o espaço de configurações *livre*, $\mathcal{G}_{livre}$, contém todas as configurações que não resultam em colisões:

$$\mathcal{G}_{livre} = \mathcal{G} \setminus \mathcal{G}_{obs} \tag{36}$$

A partir da definição do espaço de configurações e seus subconjuntos, é possível formalizar o objetivo do planejamento de trajetórias: encontrar uma trajetória contínua que pertença ao espaço de configurações *livre* e que leve o robô de uma configuração inicial $\vec{g_i}$, em um instante $t = 0$ à uma final $\vec{g_f}$, em $t = t_f$.

$$\tau : \left[0, t_f\right] \rightarrow \mathcal{G}_{livre}, \qquad \text{onde } \begin{cases} \tau(0) = \vec{g_i} \\ \tau(t_f) = \vec{g_f} \end{cases} \tag{37}$$

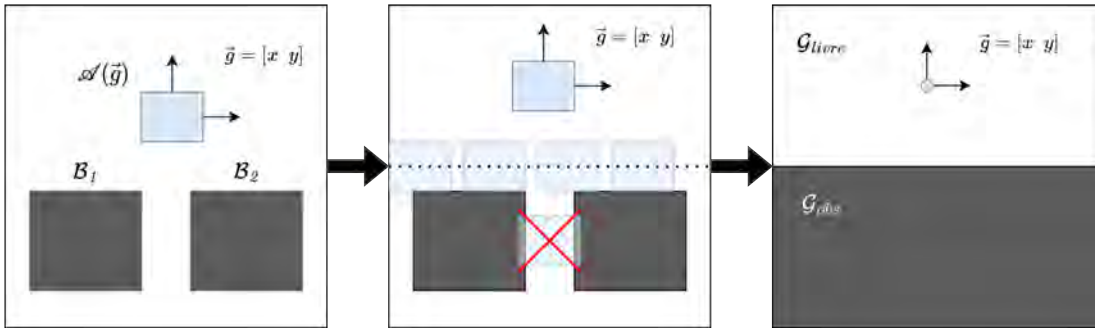
Nas seções seguintes, a notação $\tau\langle \vec{g_i}, \vec{g_f} \rangle$ é utilizada para denotar uma trajetória que inicia em uma configuração $\vec{g_i}$ e termina em $\vec{g_f}$.

2.2 Árvores de Exploração

As árvores aleatórias de exploração rápida (*rapidly exploring random tree - RRT*), introduzidas por LaValle et al. (1998), são métodos que resolvem o problema de planejamento de trajetórias de forma probabilística, através de estruturas gráficas acíclicas.

Exemplo 4 *Um robô de forma retangular se move em um ambiente com dois obstáculos. Supondo que o robô se movimente apenas por translações, sua configuração é o par de coordenadas* $\vec{g} = [x \;\; y]$. *O robô tem dois graus de liberdade. O conjunto ocupado em* $\mathcal{W} = \mathbb{R}^2$ *pelo robô e pelos obstáculos são indicados por* $\mathscr{A}(\vec{g})$ *e* $\mathcal{B}_{1,2}$, *respectivamente, conforme apresentado na Figura 25. Já que as geometrias do robô e dos obstáculos são simples, é possível descobrir* $\mathcal{G}_{obs}$ *"deslizando" o robô ao longo da fronteira dos obstáculos e verificando a ocorrência de colisões. No espaço de configuração* $\mathcal{G}$, *o robô é representado por um ponto enquanto os obstáculos aparentam-se "inflados".*

Figura 25 - Espaço de obstáculos para um robô retangular



Fonte: O autor, 2022.

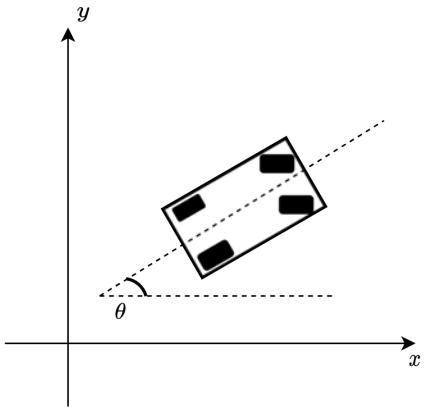
Diversas extensões do algoritmo foram propostas desde sua criação (KARAMAN; FRAZZOLI, 2010; PETIT; DESBIENS, 2021; NASIR et al., 2013; LAI; RAMOS; FRANCIS, 2019; BRUNNER; BRÜGGEMANN; SCHULZ, 2013). Apesar da existência de métodos probabilísticos na época (AMATO; WU, 1996; KAVRAKI et al., 1996), esses se mostram ineficazes em sistemas robóticos não-holonômicos, como os CLMRs, que apresentam equações não-integráveis envolvendo os parâmetros de sua configuração ou suas respectivas derivadas. Por consequência, há uma limitação no espaço de direções possíveis para as quais o robô pode se locomover, como ilustra o Exemplo 5. O planejamento de trajetórias de sistemas não-holonômicos é mais complexo para a maior parte dos algoritmos de planejamento, porém é abordado de forma natural com o algoritmo RRT (KARAMAN; FRAZZOLI, 2013).

Nos algoritmos baseados em árvores de exploração é comum denotar por $\vec{x}$ o *estado* do robô, generalizando o conceito de configuração para representar não só a posição e a orientação, mas também, possivelmente, as velocidades e acelerações das juntas do robô. Da mesma forma, substitui-se o espaço de configurações $\mathcal{G}$ por um *espaço de estados* $\mathcal{X}$. Este trabalho, contudo, denota os *estados* das árvores de exploração por *configurações*[5]

[5] Em seções posteriores, este trabalho emprega a letra x para representar uma coordenada do eixo cartesiano. Da mesma forma, o termo *estado* é utilizado com frequência no tratamento de processos de decisões Markovianos.

Exemplo 5 *Um CLMR, que se movimenta como um carro de passeio, é um sistema não-holonômico. Se a configuração do robô é representada por* $\vec{g} = [x \ y \ \theta]$*, onde* θ *representa a orientação do veículo, é impossível que haja um deslocamento infinitesimal* $\delta d = [0 \ \delta y \ 0]$*, ou seja, um movimento lateral, quando* $\theta = 0$*, por exemplo.*

Figura 26 - Um sistema robótico não-holonômico



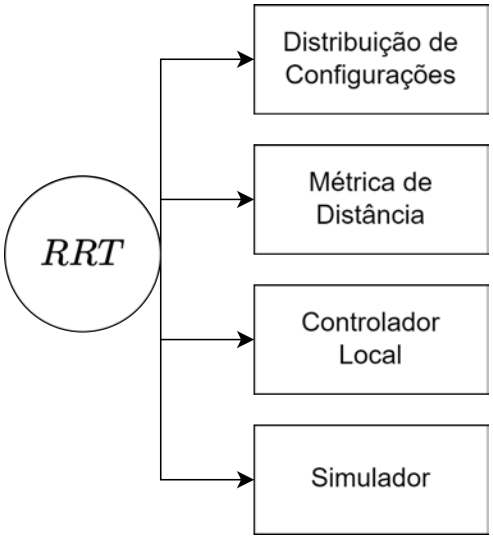
Fonte: O autor, 2022.

(g), tal qual a formulação apresentada na Seção 2.1. Além de lidar com sistemas não-holonômicos, o algoritmo RRT não assume conhecimento de $\mathcal{G}_{obs}$, e exige apenas que seja possível checar se uma determinada configuração implica em colisão. Isto se mostra conveniente pois, ao contrário do Exemplo 4, robôs e obstáculos com geometrias complexas dificultam a obtenção exata de $\mathcal{G}_{obs}$. Além disso, como o conceito de estado é mais geral que a definição de configuração, $\mathcal{G}_{obs}$ pode representar não só obstáculos, mas também restrições nas velocidades e acelerações do robô.

O termo RRT pode se referir à estrutura de dados do algoritmo: um grafo direcionado onde cada vértice define uma configuração e cada aresta, uma trajetória entre duas configurações. O algoritmo recebe como entrada uma configuração inicial $\vec{g}_i$ e busca expandir a estrutura de dados $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, onde $\mathcal{V}$ representa o conjunto de configurações e $\mathcal{E}$ o conjunto de arestas. Em cada iteração do algoritmo, amostra-se uniformemente uma configuração $\vec{g}_{aleatoria}$ de $\mathcal{G}_{livre}$. De acordo com uma métrica ρ, que calcula a distância entre duas configurações, busca-se, em $\mathcal{V}$, o vértice $\vec{g}_{prox}$ mais próximo de $\vec{g}_{aleatoria}$. Simula-se a ação de um controlador $\mathcal{U}$, que busca diminuir a distância entre $\vec{g}_{prox}$ e $\vec{g}_{aleatoria}$, por um intervalo de tempo Δt. Limitado pelo tempo de simulação, o robô chega a uma configuração $\vec{g}_{nova}$, tal que $\rho(\vec{g}_{nova}, \vec{g}_{aleatoria}) < \rho(\vec{g}_{prox}, \vec{g}_{aleatoria})$. Finalmente, caso a trajetória até $\vec{g}_{nova}$ não implique em colisão com algum obstáculo, esse novo vértice é adicionado ao conjunto $\mathcal{V}$ e a trajetória $\tau\langle\vec{g}_{prox}, \vec{g}_{nova}\rangle$ é incluída em $\mathcal{E}$. Com base no procedimento descrito neste parágrafo, pode-se notar a necessidade de quatro componentes fundamentais para o algoritmo RRT, como mostra a Figura 27.

O pseudocódigo do RRT é apresentado no Algoritmo 3. Para facilitar a descrição

Figura 27 - Principais componentes
do algoritmo RRT



Fonte: O autor, 2022.

do procedimento, assume-se que a função de simulação recebe um controlador $\mathcal{U}$, que calcula o sinal de controle a cada passo de simulação. A função "simular" retorna a trajetória entre duas configurações, após um determinado intervalo de tempo Δt. Assume-se também que a estrutura de dados que representa a trajetória obtida após a simulação armazena algumas informações, tais como, os sinais de controle aplicados, um indicador de colisão com algum obstáculo, entre outras. A descrição detalhada das funções presentes no pseudocódigo encontram-se no Apêndice B.

O algoritmo RRT dá origem a um grafo direcionado que rapidamente explora o espaço de configurações, como ilustra o Exemplo 6. A distribuição dos vértices em $\mathcal{V}$ se aproxima da distribuição que gera $\vec{g}_{aleatoria}$. Desta forma, quando a amostragem é uniforme, o algoritmo é probabilisticamente completo, isto é, à medida que o número de iterações tende ao infinito, a probabilidade de se encontrar uma solução tende a 1.

Diversas variações do algoritmo RRT foram propostas desde a sua criação. Talvez a extensão mais relevante seja o algoritmo RRT*. Enquanto o RRT garante que uma solução seja encontrada, quando o número de iterações tende ao infinito, não há garantia de que a trajetória encontrada seja ótima, em relação à métrica escolhida. Karaman e Frazzoli (2010) propõe o algoritmo RRT* para resolver esse problema, após mostrar que o RRT tradicional, na realidade, nunca converge para a solução ótima. Para entender o algoritmo RRT* mostra-se necessário a definição de alguns termos e funções:

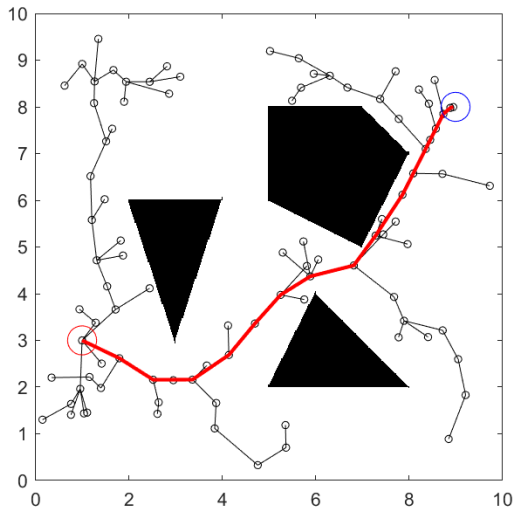
a) O *custo de uma trajetória*, $C(\tau\langle\vec{g}_1, \vec{g}_2\rangle)$, é um número real positivo calculado com base em algum critério de otimalidade. A solução ótima para um problema de planejamento consiste na trajetória de custo *mínimo* que liga uma configuração

Algoritmo 3 RRT

procedimento GERAR RRT($\vec{g}_i, K, \Delta t$)
  $\mathcal{V} \leftarrow \{\vec{g}_i\}$
  $\mathcal{E} \leftarrow \emptyset$
  $i \leftarrow 0$
  enquanto $i < K$ **faça**
    $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E})$
    $\vec{g}_{aleatoria} \leftarrow$AMOSTRARCONFIG()
    $\vec{g}_{prox} \leftarrow$CONFIGMAISPROXIMA($\mathcal{V}, \vec{g}_{aleatoria}, \rho$)
    $\tau\langle \vec{g}_{prox}, \vec{g}_{nova} \rangle \leftarrow$SIMULAR($\vec{g}_{prox}, \mathcal{U}, \Delta t$)
    se LIVREDECOLISOES($\tau\langle \vec{g}_{prox}, \vec{g}_{nova} \rangle$) **então**
      $\mathcal{V} \leftarrow \mathcal{V} \cup \{\vec{g}_{nova}\}$
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{\tau\langle \vec{g}_{prox}, \vec{g}_{nova} \rangle\}$
      $i \leftarrow i + 1$
    fim se
  fim enquanto
  retorna G
fim procedimento

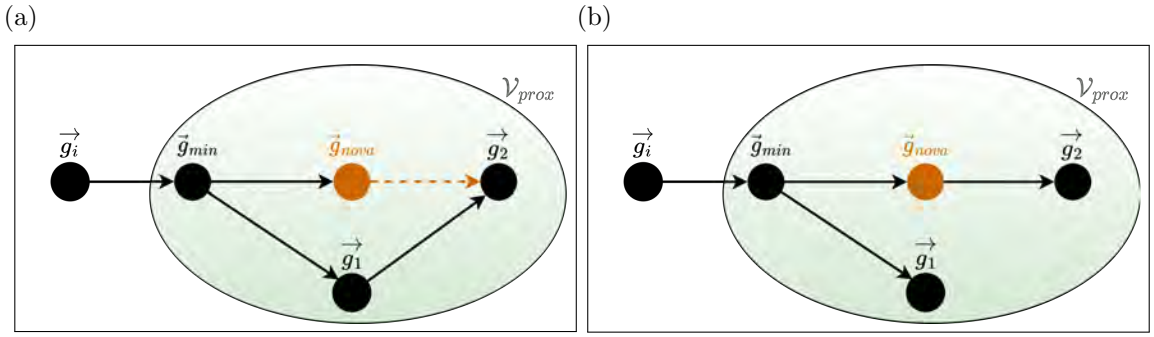
Exemplo 6 *Um robô móvel do tipo carro, como o do Exemplo 5, deve navegar entre duas configurações. Na Figura abaixo, o círculo vermelho indica o estado inicial. Os círculos pretos menores representam os estados de $\mathcal{V}$, enquanto as arestas pretas indicam as trajetórias entre esses estados. A estrutura de dados gerada pelo algoritmo permite encontrar um caminho (indicado em vermelho) até o estado representado pelo círculo azul. Nota-se a semelhança da estrutura gerada pelo algoritmo com uma árvore. Um estado em $\mathcal{V}$ têm apenas um vértice precedente, porém pode apresentar diversos estados sucessores, dando origem a diversas ramificações.*

Figura 28 - Planejamento de trajetórias com RRT



Fonte: O autor, 2022.

Figura 29 - Procedimento de religação



Legenda: (a) - a configuração $\vec{g}_{nova}$, recém adicionada, permite diminuir o custo (distância euclidiana na ilustração) de $\vec{g}_i$ até $\vec{g}_2$; (b) - a aresta entre $\vec{g}_2$ e sua configuração antecessora, $\vec{g}_1$, é removida e a antecessora de $\vec{g}_2$ passa a ser $\vec{g}_{nova}$.
Fonte: O Autor, 2022.

inicial $\vec{g}_i$ a uma final $\vec{g}_f$, sem a ocorrência de colisões;

b) A métrica de distância entre dois estados, $\rho(\vec{g}_1, \vec{g}_2)$, é definida como o custo da trajetória ótima entre estes estados, sem considerar a presença de obstáculos;

c) Seja $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ uma árvore de exploração, $\vec{g} \in \mathcal{V}$ uma configuração e $\vec{g}' \in \mathcal{V}$ o nó que antecede $\vec{g}$, isto é, $\tau\langle\vec{g}', \vec{g}\rangle \in \mathcal{E}$. Então, o *custo da configuração* $\vec{g}$, $C(\vec{g})$, é igual ao custo da configuração antecessora somado ao custo da trajetória que une $\vec{g}'$ a $\vec{g}$; ou seja, $C(\vec{g}) = C(\vec{g}') + C(\tau\langle\vec{g}', \vec{g}\rangle)$. Considerando o custo da origem nulo, $C(\vec{g}_i) := 0$, é possível, então, obter o custo de qualquer configuração em $\mathcal{V}$ recursivamente.

O RRT* (Algoritmo 4) difere do RRT em dois aspectos. Primeiro, no RRT*, a configuração antecessora de $\vec{g}_{nova}$ é a que minimiza o custo dessa configuração, enquanto no algoritmo RRT, o nó antecessor é sempre $\vec{g}_{prox}$. Segundo, o algoritmo RRT* conta com uma etapa de religação das arestas em $\mathcal{E}$: após a adição de $\vec{g}_{nova}$ à árvore verifica-se, para cada configuração $\vec{g}_j \in \mathcal{V}_{prox}$, onde $\mathcal{V}_{prox} \subseteq \mathcal{V}$ é o conjunto de vértices próximos de $\vec{g}_{nova}$ (de acordo um limiar ϵ_g), se o custo de $\vec{g}_j$ é menor caso o nó antecessor de $\vec{g}_j$ passe a ser $\vec{g}_{nova}$. Nessa condição, $\vec{g}_{nova}$ torna-se a configuração antecessora de $\vec{g}_j$. A Figura 29 detalha o procedimento descrito nesse parágrafo.

Algoritmo 4 RRT*

procedimento GERAR RRT*($\vec{g}_i, K, \Delta t$)
    $\mathcal{V} \leftarrow \{\vec{g}_i\}$
    $\mathcal{E} \leftarrow \emptyset$
    $i \leftarrow 0$
    enquanto $i < K$ **faça**
        $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E})$
        $\vec{g}_{aleatoria} \leftarrow$ AMOSTRARCONFIG($\mathcal{G}_{livre}$)
        $\vec{g}_{prox} \leftarrow$ CONFIGMAISPROXIMA($\vec{g}_{aleatoria}$)
        $\tau\langle \vec{g}_{prox}, \vec{g}_{nova} \rangle \leftarrow$ SIMULAR($\vec{g}_{prox}, \vec{g}_{aleatoria}, \mathcal{U}, \Delta t$)
        se LIVREDECOLISOES($\tau\langle \vec{g}_{prox}, \vec{g}_{nova} \rangle$) **então**
            $\mathcal{V}_{prox} \leftarrow$ CONFIGPROXIMAS($\vec{g}_{nova}, \mathcal{V}, \rho, \epsilon_g$)
            $\vec{g}_{min} \leftarrow$ CONFIGMENORCUSTO($\vec{g}_{nova}, \mathcal{V}_{prox}, C$)
            $\mathcal{V} \leftarrow \mathcal{V} \cup \{\vec{g}_{nova}\}$
            $\mathcal{E} \leftarrow \mathcal{E} \cup \{\tau\langle \vec{g}_{min}, \vec{g}_{nova} \rangle\}$
            $\mathcal{E} \leftarrow$ RELIGAR($\vec{g}_{nova}, \mathcal{E}, \mathcal{V}_{prox}$)
        fim se
        $i \leftarrow i + 1$
    fim enquanto
    $\mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E})$
    retorna $\mathcal{T}$
fim procedimento

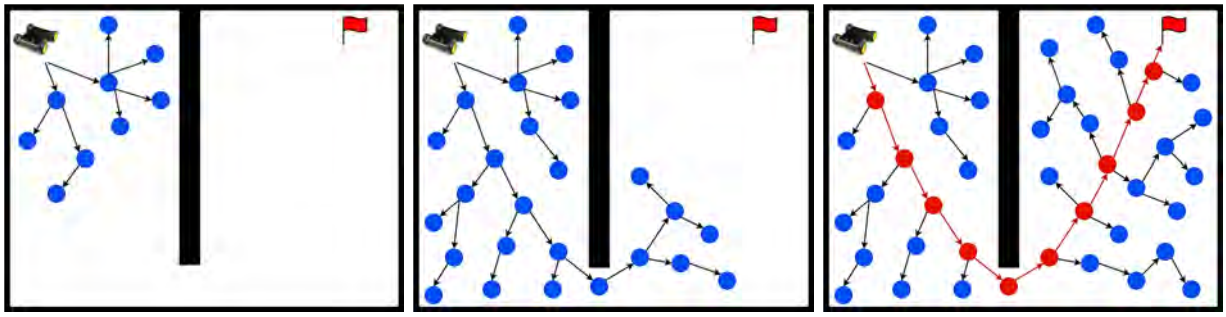
O procedimento de religação permite que o algoritmo seja assintoticamente ótimo, isto é, à medida que o número de iterações tende ao infinito, a trajetória que leva de $\vec{g}_i$ a $\vec{g}_f$ tende a apresentar o menor custo possível. Já que a qualidade da trajetória encontrada depende do número de iterações do algoritmo, tipicamente a solução obtida será sub-ótima, considerando as limitações práticas computacionais associadas à execução do algoritmo. A seção seguinte aborda uma forma de otimizar o custo computacional dos algoritmos RRT e RRT*, diminuindo consideravelmente o tempo necessário para que uma solução seja encontrada.

2.3 Aprendizado de Distribuições de Amostras

Para ilustrar um dos problemas do algoritmo RRT*, considere que este método seja aplicado com a finalidade de encontrar a trajetória ótima para um robô móvel. Como indica a Figura 30, à medida que ocorrem as iterações do algoritmo, a estrutura de dados cresce até que, eventualmente, uma solução é encontrada. Com atenção à Figura 30c,

Figura 30 - Crescimento da árvore de exploração rápida ao longo das iterações

(a) (b) (c)



Legenda: (a) e (b) - as configurações da árvore de exploração são indicadas por círculos azuis e os vértices, por setas pretas. A bandeira vermelha indica a configuração final desejada; (c) - a solução encontrada pelo algoritmo apresenta-se em vermelho.
Fonte: O Autor, 2022.

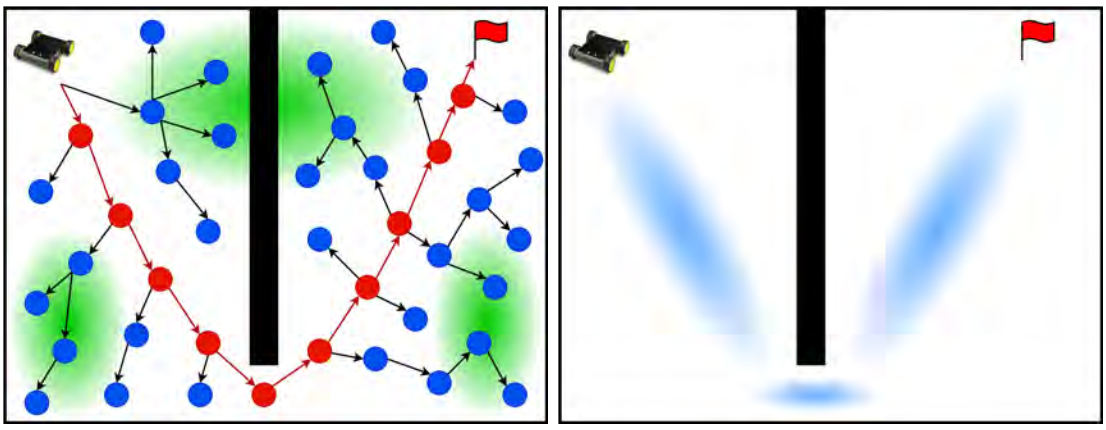
pode-se notar que a maior parte das configurações não pertencem à trajetória encontrada. A Figura 31a mostra algumas regiões que claramente não devem pertencer à solução ótima, mas é inevitável que a árvore seja composta por tais configurações, já que estas são realizações de uma amostragem uniforme do espaço de configurações livres. A adição de cada configuração à árvore de exploração é precedida de uma etapa de verificação do melhor sinal de controle, simulação do sistema dinâmico e checagem de colisões. Dessa forma, há uma perda de tempo computacional com configurações que não farão parte da solução final.

Pode-se pensar em introduzir um viés na amostragem de configurações, para que exista uma probabilidade maior de amostrar uma configuração que pertença a uma região promissora. Porém, há semelhanças evidentes entre o problema de descobrir quais são essas regiões e a própria questão de encontrar a trajetória ótima. Vale notar que uma das razões do sucesso de métodos baseados em amostragem, como RRT e mapas probabilísticos (KAVRAKI et al., 1996), reside na capacidade de contornar a dificuldade de resolver o problema do planejamento de trajetórias analiticamente. Enquanto o exemplo das Figuras 30 e 31 assume que o robô tem dimensão desprezível e dinâmica simples, um robô CLMR apresenta restrições complexas em sua movimentação, o que torna difícil a busca por uma solução matemática precisa.

A última década mostrou que tarefas complexas podem ser abordadas por métodos de aprendizado de máquinas, desde que exista uma quantidade suficiente de dados. Ichter, Harrison e Pavone (2017) mostraram que é possível aprender uma distribuição de configurações promissoras, condicionada às características do problema de planejamento. Especificamente, a arquitetura do CVAE, abordada no fim da Seção 1.5, foi utilizada para

Figura 31 - Comparação entre regiões no espaço de estados livre.

(a) (b)



Legenda: (a) - as regiões em verde indicam espaços de amostragem com baixa qualidade; (b) - as regiões em azul são promissoras e provavelmente farão parte da solução ótima para o problema.

Fonte: O Autor, 2022.

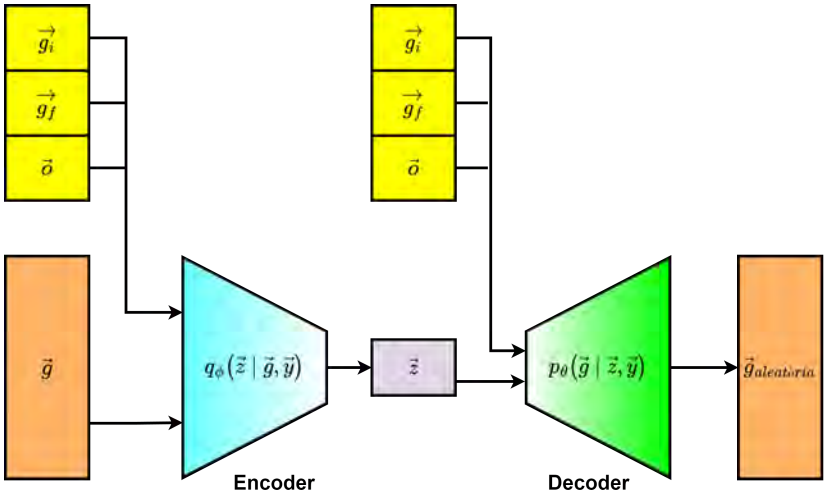
gerar amostras de configurações aleatórias no algoritmo RRT*. A Figura 32 mostra as características do CVAE, quando utilizados para este fim.

Tomando a perspectiva de modelos probabilísticos latentes, pode-se considerar que as configurações pertencentes à trajetória ótima são originadas a partir de um algoritmo, que recebe um vetor latente $\vec{z}$ de entrada e outro vetor de condições $\vec{y}$. Dessa forma, o problema do planejamento é semelhante à geração de imagens falsas, de forma que tal algoritmo é desconhecido, dada a complexidade do problema. Diferentes valores de $\vec{z}$, para um mesmo vetor $\vec{y}$ de condições, podem gerar configurações aleatórias distintas, que formam uma região promissora no espaço de configurações. Este processo é ilustrado na Figura 33 para o exemplo proposto na Figuras 30.

A utilização de configurações promissoras provenientes do CVAE acelera consideravelmente o tempo necessário para que uma solução seja encontrada no RRT*, porém, com a amostragem uniforme de configurações, existem garantias teóricas de que a solução encontrada seja ótima, à medida que o número de iterações tende ao infinito. Tal garantia pode ser mantida se uma parcela das configurações aleatórias vier de uma distribuição uniforme (ICHTER; HARRISON; PAVONE, 2017). Dessa forma, é possível introduzir um parâmetro α, que indica qual a fração das amostras aleatórias de configurações são originadas do modelo generativo CVAE.

O treinamento do CVAE para o planejamento de trajetórias exige uma base de dados em que cada exemplo de treinamento é composto por um vetor de condições $\vec{y}$, que especifica as características do problema, e um estado pertencente à solução ótima

Figura 32 - Arquitetura do CVAE para auxiliar o planejamento de trajetórias

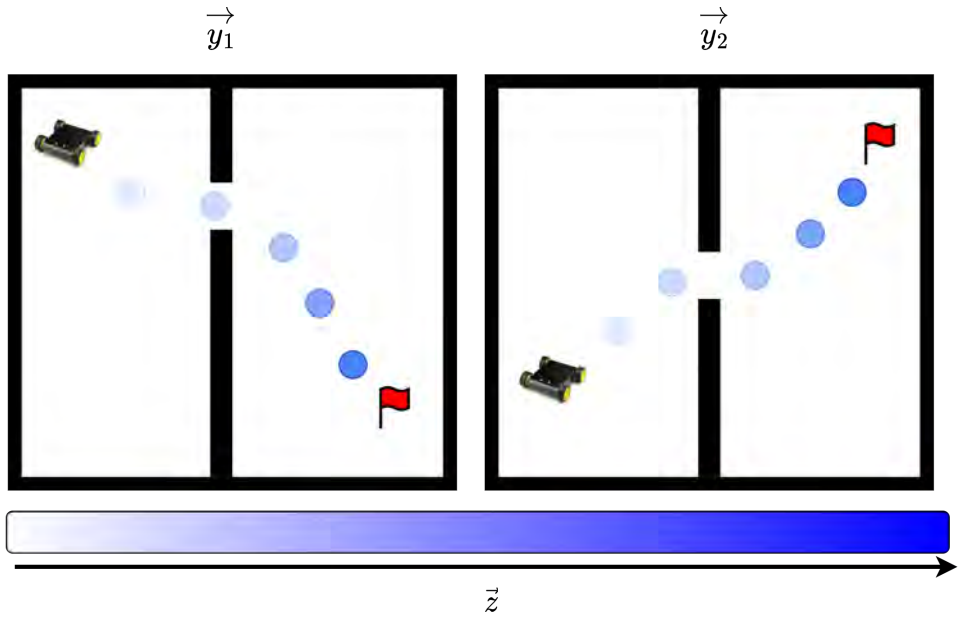


Legenda: A entrada e saída são configurações $\vec{g}_{aleatoria}$. O vetor $\vec{y}$ de variáveis condicionantes é composto pelas configurações iniciais e finais, junto à um vetor $\vec{y}_{obs}$ que codifica a localização dos obstáculos.

Fonte: O autor, 2022.

encontrada por algum algoritmo de planejamento. As condições são compostas pelas configurações iniciais e finais, além de um vetor $\vec{o}$, que codifica a localização dos obstáculos. Supondo que o RRT* seja utilizado para gerar os exemplos da base de dados, a trajetória encontrada para um determinado problema com características $\vec{y}^{(j)}$ será um conjunto de configurações $\{\vec{g}^{(i)}\}_{i=1}^{k}$. Dessa forma, k exemplos de treinamento podem ser obtidos, $\{(\vec{g}^{(i)}, \vec{y}^{(j)})\}_{i=1}^{k}$, para uma única execução do RRT*. Diversas execuções do algoritmo de planejamento podem ser necessárias até que o número de exemplos seja suficiente para a otimização do CVAE. O processo descrito é ilustrado na Figura 34.

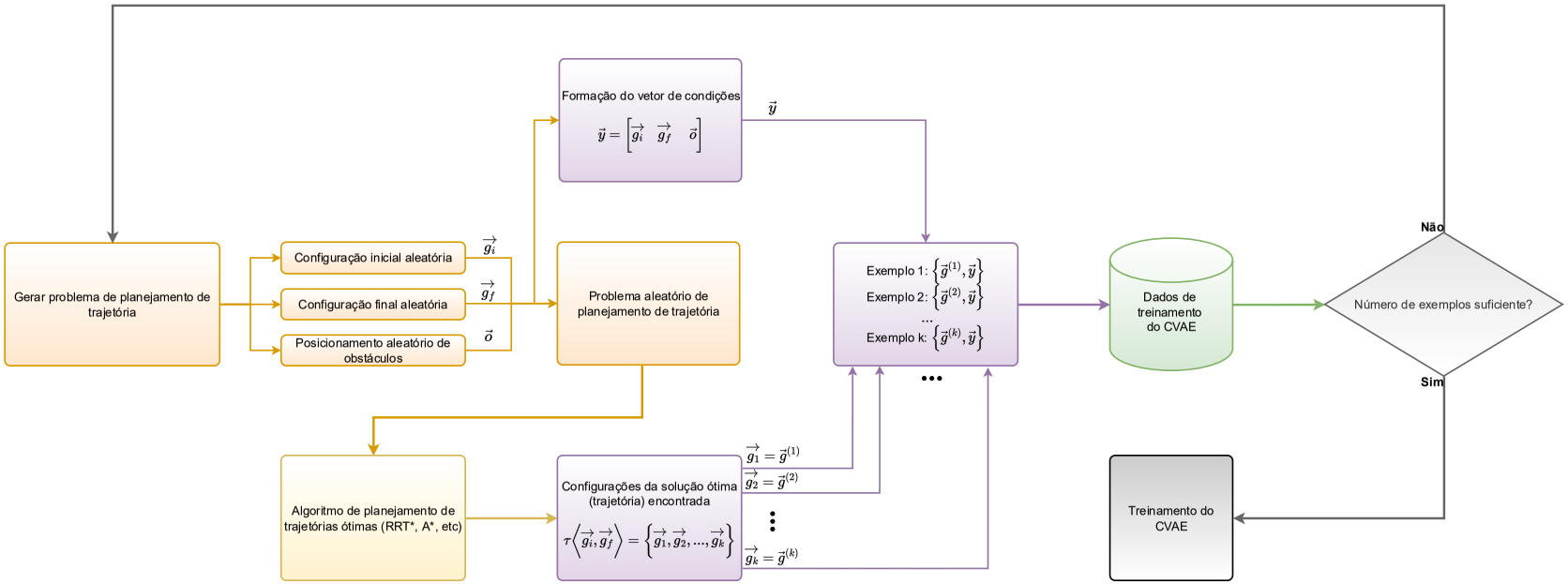
Figura 33 - Ilustração das configurações geradas pelo CVAE



Legenda: A ilustração exemplifica dois problemas de planejamentos distintos, com diferentes configurações iniciais, finais e disposição dos obstáculos. Os vetores de condições $\vec{y}_1$ e $\vec{y}_2$ codificam estas características. A amostragem de vetores latentes e a subsequente propagação destes vetores pelo decodificador do CVAE, junto ao vetor de condições $\vec{y}$, particular a cada situação, geram configurações aleatórias promissoras para o RRT*.

Fonte: O autor, 2022.

Figura 34 - Geração de dados para treinamento do CVAE



Fonte: O autor, 2022.

3 APRENDIZADO POR REFORÇO

A seção anterior abordou o planejamento de trajetórias, por meio de algoritmos baseados em árvores de exploração. O uso do aprendizado de máquinas, mais especificamente, através de modelos generativos parametrizados por redes PMCs, permite otimizar estes algoritmos. Contudo, a implementação do RRT* exige alguns componentes fundamentais, como, por exemplo, um controlador e uma função de custo. Considerando o problema de navegação autônoma de um CLMR, por exemplo, como definir uma função de custo que caracterize a distância entre configurações? Como desenvolver um controlador para o robô? Este trabalho busca solucionar esses problemas por meio do *aprendizado por reforço*.

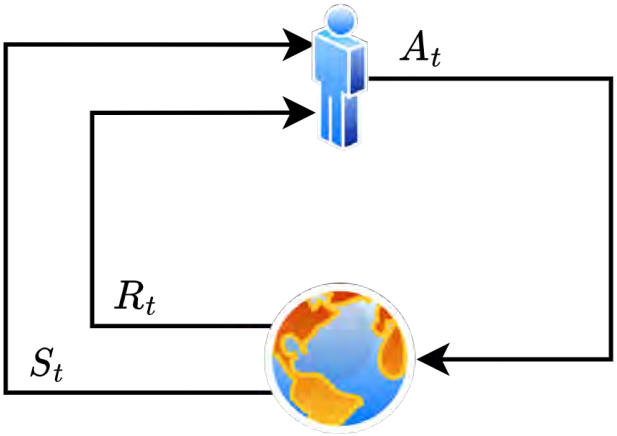
O aprendizado por reforço é uma do campo da inteligência artificial que se concentra em como entidades inteligentes devem executar ações dentro de um ambiente para maximizar a noção de recompensa cumulativa. Naturalmente, alguns problemas se adequam de imediato a esse paradigma, por exemplo, jogos de *video-game* costumam apresentar um sistema de pontuação e o *agente*, aquele que toma as decisões no jogo, deve escolher as ações adequadas a maximização dessa pontuação. Para outras tarefas, em geral, há um objetivo que deve ser alcançado em um sistema e o sinal de recompensa deve, de alguma forma, caracterizar a obtenção desse objetivo. Na maior parte dos problemas, as ações que o agente realiza afetam não só a recompensa imediata recebida, mas também todas as recompensas futuras. A conciliação das consequências de curto e longo prazo é um dos problemas fundamentais do aprendizado por reforço.

O formalismo matemático para o aprendizado por reforço vêm dos *processos de decisão de Markov* (PDM). Apesar da existência de diversos tipos de PDMs (PUTERMAN, 1990), esta seção é dedicada aos PDMs finitos, discretos e completamente observáveis, por serem mais intuitivos. As Seções 3.1 e 3.2 apresentam os conceitos básicos dos PDMs. Duas abordagens clássicas para a resolução de PDMs são discutidas nas Seções 3.3 e 3.4. A integração do aprendizado profundo (ARULKUMARAN et al., 2017) nos algoritmos clássicos é apresentada na Seção 3.5. O aprendizado por reforço com *múltiplos objetivos* é introduzido na Seção 3.6. Finalmente, um dos algoritmos de estado da arte é abordado na Seção 3.7.

3.1 Processos de Decisão Markov

Um PDM é uma formulação matemática para a tomada de decisão em situações cujos acontecimentos estão sob controle parcial de um agente. Esta formulação é adequada para diversos problemas reais de otimização (WHITE, 1993) e serve como base teórica

Figura 35 - Interação entre um agente e um ambiente externo



Fonte: O autor, 2022.

para o estudo do aprendizado por reforço.

Um problema de aprendizado por reforço é caracterizado pela interação de um agente com um ambiente externo, em instantes discretos de tempo. O agente exerce controle parcial sobre a dinâmica do ambiente através de *ações* (A_t), enquanto o ambiente apresenta consequências ao agente, sob a forma de representações de seu *estado* interno (S_t) e *recompensas* (R_t). Essa interação é ilustrada na Figura 35. Em um PDM finito:

a) $S_t \in \mathcal{S}$, onde $\mathcal{S}$ é um conjunto finito de estados;

b) $A_t \in \mathcal{A}(S_t)$, onde $\mathcal{A}$ é um conjunto finito de ações admissíveis no estado S_t;

c) $R_t \in \mathcal{R}$, onde $\mathcal{R}$ é um conjunto finito de recompensas.

O objetivo do agente, em um problema de aprendizado por reforço, é maximizar o acúmulo de recompensas obtidas ao longo do tempo. Essa soma é denominada *retorno* (G_t):

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_H \tag{38}$$

A formulação de retorno da Equação 38 é viável para tarefas em que a interação do agente com o ambiente é representada naturalmente por episódios independentes, de duração H (horizonte), nesse caso o PDM é caracterizado como *episódico*[6]. Entretanto, para interações contínuas de tempo ilimitado, a Equação 38 pode ser indefinida. Esta é

[6] Um PDM pode ser episódico ainda que o horizonte seja infinito, desde que existam estados *terminais* na tarefa. No xadrez, por exemplo, uma situação de cheque-mate é um estado terminal, já que implica na terminação da tarefa.

uma das razões para a adoção do *fator de desconto*, $0 < \gamma < 1$. Desta forma, é possível definir o retorno como a soma acumulada descontada de recompensas:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{39}$$

Em um PDM, o estado S_t contém toda a informação relevante sobre o processo. Isto significa que, S_t e A_t são informações suficientes para que seja possível prever S_{t+1}, ou seja, as informações do passado, $\{S_0, A_0, \ldots, S_{t-1}, A_{t-1}\}$, são irrelevantes quando os estados e ações atuais são conhecidos. Esta característica do processo é denominada *propriedade de Markov*. Matematicamente, isto significa que a probabilidade condicional de estados futuros depende apenas dos valores atuais:

$$\Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \Pr\{S_{t+1} \mid S_t = s, A_t = a, \ldots, S_0 = s_0, A_0 = a_0\} \tag{40}$$

Com a propriedade de Markov, é possível definir toda a dinâmica do ambiente a partir de uma distribuição de probabilidade, condicionada apenas no estado e ação atuais:

$$p(s', r \mid s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t = s, A_t = a\} \tag{41}$$

A distribuição da Equação 41 pode ser utilizada para definir duas entidades: uma distribuição de transição de estados (Equação 42) e uma função de recompensa (Equação 43).

$$p(s' \mid s, a) = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \tag{42}$$

$$r(s, a) = \mathbb{E}\left[R_{t+1} \,\middle|\, S_t = s, A_t = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{43}$$

A Equação 42 descreve a probabilidade de transição para um estado s_{t+1}, quando o agente se encontra em um estado s_t e realiza uma ação a_t. Em um ambiente determinístico, essa probabilidade é igual a 1 para um único estado e 0 para os outros. A função de recompensa, indicada na Equação 43, representa o valor esperado para a recompensa no próximo instante de tempo, dado que o estado e a ação atuais são s_t e a_t, respectivamente. A utilização do valor esperado é necessária já que, no caso mais geral possível, o estado e a ação no instante t condicionam as distribuição de probabilidades dos estados e recompensas futuras. Naturalmente, para sistemas determinísticos, a função de recompensa é determinística e o conceito de expectativa pode ser descartado. A for-

mulação de um PDM para problemas episódicos exige a definição de uma distribuição de estados iniciais. Denota-se por $p_0(s)$ a probabilidade do agente iniciar o episódio no estado s. O Exemplo 7 mostra como um problema pode ser formulado como um PDM, conforme a discussão desta Seção.

Como os PDMs são apenas modelos matemáticos para abordar problemas que exigem a tomada de decisão, há liberdade na escolha de diversos parâmetros, dentre os quais se destaca a função de recompensa que, para a maior parte dos problemas, não é especificada e exige conhecimento do problema em questão. A escolha da representação do estado influencia a caracterização ou não do processo como Markoviano, ou seja, se é possível descartar o histórico da trajetória do agente quando o estado atual é conhecido. Finalmente, há liberdade na escolha do que constitui a *ação* do agente: em problemas de robótica, pode-se escolher o torque nas juntas do robô ou uma decisão em alto nível, como, por exemplo, "locomover-se para frente". Naturalmente, essa escolha, por sua vez, afeta a distribuição de transição de estados (Equação 42).

3.2 Política e Função de Valor

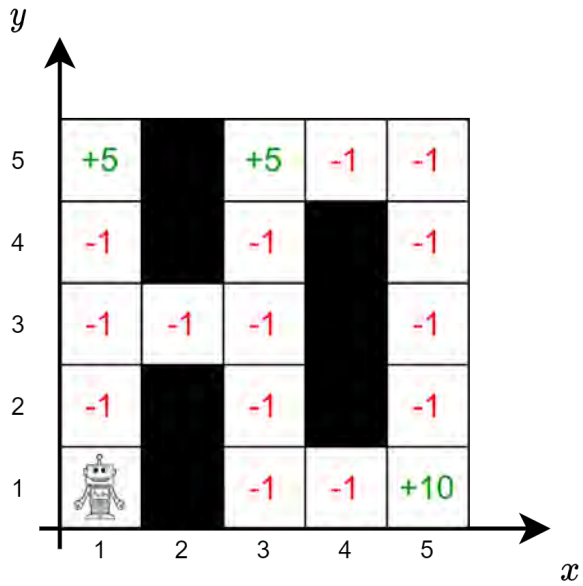
O objetivo dos algoritmos de aprendizado por reforço é resolver PDMs. Para entender o significado dessa afirmação é preciso introduzir o conceito de *política* e *função de valor*.

A política, geralmente denotada por π, é uma característica do agente que determina o seu modo de agir no ambiente. Supondo que o agente se encontre em um estado s_t, a política determina qual será a ação a_t realizada neste estado. Uma política é caracterizada como *determinística* se, em um estado s_t, apenas uma única ação possa ser selecionada. Em uma política *estocástica*, para cada s_t há uma distribuição indicando a probabilidade de tomar cada uma das ações possíveis. Em termos matemáticos, uma política determinística é uma função que seleciona a ação, tomando o estado atual como argumento, ou seja, $a_t = \pi(s_t)$. Nesses termos, uma política estocástica é uma distribuição de probabilidade condicionada no estado atual: $a_t = \pi(a_t \mid s_t)$.

É de se esperar que políticas diferentes resultem em retornos (Equação 39) distintos. Em outras palavras, em um PDM, o modo de agir (a política), influencia a expectativa de recompensa acumulada. Para um agente ótimo, que sempre toma as melhores decisões, a expectativa de retorno é alta. Em contrapartida, para um agente inapto, a expectativa de retorno é baixa. Este raciocínio sugere que cada política, ou modo de agir, induz uma expectativa de retorno diferente. A *função valor*, denotada por V, é uma quantidade que representa essa expectativa de retorno. Como esta função depende da política é comum utilizar a notação V_π. A função valor depende também do estado em que o agente se encontra: no Exemplo 7, mesmo que o robô aja de forma completamente aleatória, é de

Exemplo 7 *Um robô deve navegar em um labirinto e coletar recompensas, mas só possui combustível para realizar 8 movimentos. A Figura 36 apresenta o problema. O estado do robô são as coordenadas nos eixos x e y, ou seja, o par ordenado $s = (x, y)$. O robô inicia a tarefa sempre na mesma posição, isto é, $s_0 = (1, 1)$. Os números dentro de cada quadrado indicam a recompensa que o robô recebe ao transitar para o local. Os quadrados preenchidos indicam obstáculos. O robô pode se locomover em 4 direções: $\{\leftarrow, \rightarrow, \downarrow, \uparrow\}$, desde que não colida com algum obstáculo.*

Figura 36 - Navegação em um labirinto



Fonte: O autor, 2022.

Este problema pode ser caracterizado por um PDM, representado pela tupla $\langle \mathcal{S}, \mathcal{A}, p, r, p_0, \gamma, H \rangle$:

a) o espaço de estados, $\mathcal{S}$, contém todas as coordenadas em que não há obstáculos;

b) o espaço de ações, $\mathcal{A}(S_t)$, são todas as direções de movimento que não causam colisão com os obstáculos, quando o agente se encontra no estado S_t;

c) a transição de estados, p, é determinística. Por exemplo, caso o agente realize a ação $\uparrow$ no estado $(1, 1)$ o próximo estado será $(1, 2)$. Matematicamente, têm-se que $\Pr\{S_{t+1} = (1, 2) \mid S_t = (1, 1), A_t = \uparrow\} = 1$ e $\Pr\{S_{t+1} \neq (1, 2) \mid S_t = (1, 1), A_t = \uparrow\} = 0$;

d) a função de recompensa r também é determinística e, por exemplo, para $s_t = (1, 1)$ e $a_t = \uparrow$ têm-se que $r(s_t, a_t) = -1$;

e) γ pode ser considerado um hiper parâmetro que determina o quão importante são as recompensas esperadas em um futuro distante, em relação às imediatas. Além disso, o fator de desconto busca modelar um fenômeno biológico em que recompensas imediatas costumam ser priorizadas em relação às distantes;

f) por fim, o robô pode se movimentar 8 vezes, logo o número máximo de instantes de tempo, para cada episódio da tarefa, é $H = 8$.

se esperar que o valor esperado para o retorno seja maior para o estado $s = (4, 1)$, se comparado com a posição inicial, $s = (1, 1)$. Informalmente, a função de valor representa o quão bom é estar em um estado, sob uma certa política. Matematicamente, a função de valor é a expectativa do retorno, condicionada no estado atual[7]:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \,\middle|\, S_t = s \right] \tag{44}$$

De forma similar, pode-se perguntar o quão bom é tomar uma certa ação, em um estado, e seguir uma determinada política em seguida. A função *ação-valor* representa a expectativa de retorno condicionada no estado e ação atuais, com o subscrito π indicando que o agente segue a política π após tomar a ação no estado:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \,\middle|\, S_t = s, A_t = a \right] \tag{45}$$

No início desta seção foi mencionado que o objetivo dos algoritmos de aprendizado por reforço é resolver PDMs. Este objetivo pode ser caracterizado pela busca por uma política *ótima*, denotada por π^*. Tal objetivo pode ser entendido como a busca pela melhor maneira de atuar no ambiente, visando a maximização da recompensa acumulada. Já que cada política induz uma função valor (ou função ação-valor), a política ótima determina a existência de uma função valor ótima, ou função ação-valor ótima, denotadas por V^* ou Q^*, respectivamente.

Observa-se que a política ótima maximiza a expectativa de retorno e as funções de valor ou ação-valor são caracterizadas por essa expectativa, tal como indica as Equações 44 e 45. Dessa forma, a política ótima, por definição, maximiza essas duas funções:

$$V^*(s) = \max_\pi V_\pi(s) \tag{46a}$$

$$Q^*(s, a) = \max_\pi Q_\pi(s, a) \tag{46b}$$

Supondo que $Q^*(s, a)$ seja conhecida e o agente se encontra em um determinado estado s_1. Ao calcular $Q^*(s_1, a_i)$ para todas as ações $a_i \in \mathcal{A}(s_1)$, obtêm-se a expectativa de retorno para cada forma de agir no estado s_1. Basta, então, escolher a ação que maximize essa expectativa. Em outras palavras, a tarefa de encontrar a melhor forma de agir no ambiente pode ser simplificada pela obtenção de $Q^*(s, a)$ ou $V^*(s)$. Para problemas simples, em que o número de estados e ações são pequenos, um algoritmo de aprendizado por reforço pode armazenar em uma tabela o valor de $Q(s, a)$ para todos os estados e ações possíveis.

[7] Este trabalho utiliza a notação $\mathbb{E}_\pi[.]$ para explicitar a dependência do valor esperado na política.

Alguns algoritmos de aprendizado por reforço buscam obter uma aproximação $\hat{Q}(s,a) \approx Q^*(s,a)$ através de uma importante relação, conhecida como equação de Bellman (BELLMAN; KALABA, 1965), que estabelece uma forma recursiva para a função valor ou ação-valor. Dada a importância da equação de Bellman, é interessante verificar sua derivação.

Utilizando a propriedade de linearidade do operador de expectativa na definição de função valor (Equação 44):

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s\right] \tag{47a}$$
$$= \mathbb{E}_\pi\left[R_{t+1} \,\middle|\, s\right] + \gamma \mathbb{E}_\pi\left[G_{t+1} \,\middle|\, s\right] \tag{47b}$$

A derivação consiste em aplicar a lei da probabilidade total sequencialmente para as duas parcelas da soma na Equação 47:

$$\mathbb{E}_\pi\left[R_{t+1} \,\middle|\, s\right] = \sum_{r\in\mathcal{R}} r \Pr\{r \mid s\} \tag{48a}$$
$$= \sum_{r\in\mathcal{R}} \sum_{a\in\mathcal{A}(s)} r \Pr\{r \mid s, A_t = a\} \Pr\{A_t = a \mid s\} \tag{48b}$$

Utiliza-se a definição de política para substituir uma das probabilidades condicionais da Equação 48b:

$$\mathbb{E}_\pi\left[R_{t+1} \,\middle|\, S_t = s\right] = \sum_{r\in\mathcal{R}} \sum_{a\in\mathcal{A}(s)} r \Pr\{r \mid s, a\} \pi(a \mid s) \tag{49a}$$
$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{r\in\mathcal{R}} r \Pr\{r \mid s, a\} \tag{49b}$$
$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} r \Pr\{r \mid s, a, S_{t+1} = s'\} \Pr\{S_{t+1} = s' \mid s, a\} \tag{49c}$$

Utilizando a fatoração $\Pr\{r, s' \mid s, a\} = \Pr\{r \mid s, a, s'\} \Pr\{s' \mid s, a\}$, têm-se que:

$$\mathbb{E}_\pi\left[R_{t+1} \,\middle|\, s\right] = \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} r \Pr\{r, s' \mid s, a\} \tag{50a}$$
$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} p(r, s' \mid s, a) r \tag{50b}$$

Para a outra parcela da soma na Equação 47:

$$\mathbb{E}_\pi\left[G_{t+1} \,\middle|\, s\right] = \sum_{g\in\mathcal{G}} g \Pr\{G_{t+1} = g \mid s\} \tag{51a}$$

$$= \sum_{g\in\mathcal{G}} \sum_{a\in\mathcal{A}(s)} g \Pr\{g \mid s, A_t = a\} \Pr\{A_t = a \mid s\} \tag{51b}$$

$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{g\in\mathcal{G}} g \Pr\{g \mid s, a\} \tag{51c}$$

$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{g\in\mathcal{G}} \sum_{s'\in\mathcal{S}} g \Pr\{g \mid s, a, S_{t+1} = s'\} \Pr\{S_{t+1} = s' \mid s, a\} \tag{51d}$$

$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{s'\in\mathcal{S}} \Pr\{s' \mid s, a\} \sum_{g\in\mathcal{G}} g \Pr\{g \mid s, a, s'\} \tag{51e}$$

$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{s'\in\mathcal{S}} \Pr\{s' \mid s, a\} \sum_{g\in\mathcal{G}} \sum_{r\in\mathcal{R}} g \Pr\{g \mid s, a, s', r\} \Pr\{r \mid s, a\} \tag{51f}$$

$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} \Pr\{r \mid s, a\} \Pr\{s' \mid s, a\} \sum_{g\in\mathcal{G}} g \Pr\{g \mid s, a, s', r\} \tag{51g}$$

$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} p(r, s' \mid s, a) \underbrace{\sum_{g\in\mathcal{G}} g \Pr\{g \mid s, a, s', r\}}_{\mathbb{E}_\pi\left[G_{t+1} \,\middle|\, s,a,s',r\right]} \tag{51h}$$

Com atenção à expectativa na Equação 51h:

$$\mathbb{E}_\pi\left[G_{t+1} \,\middle|\, s, a, s', r\right] = \mathbb{E}_\pi\left[G_{t+1} \,\middle|\, S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r\right] \tag{52a}$$

$$= \mathbb{E}_\pi\left[R_{t+2} + \gamma G_{t+2} \,\middle|\, S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r\right] \tag{52b}$$

Considerando a propriedade de Markov, a recompensa R_{t+2} e a expectativa de retorno G_{t+2} dependem apenas de S_{t+1} e A_{t+1}, portanto:

$$\mathbb{E}_\pi\left[G_{t+1} \,\middle|\, s, a, s', r\right] = \mathbb{E}_\pi\left[G_{t+1} \,\middle|\, s'\right] \tag{53a}$$

$$= \mathbb{E}_\pi\left[G_{t+1} \,\middle|\, S_{t+1} = s'\right] \tag{53b}$$

$$= V_\pi(s') \tag{53c}$$

Substituindo na Equação 51:

$$\mathbb{E}_\pi\left[G_{t+1} \,\middle|\, s\right] = \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} p(r, s' \mid s, a) V_\pi(s') \tag{54}$$

Por fim, substituindo as Equações 50b e 54 no somatório da Equação 47:

$$V_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} \,|\, s\right] + \gamma \mathbb{E}_\pi\left[G_{t+1} \,|\, s\right] \tag{55a}$$

$$= \sum_{a \in \mathcal{A}(s)} \pi(a \,|\, s) \sum_{\substack{r \in \mathcal{R} \\ s' \in \mathcal{S}}} p(r, s' \,|\, s, a) r + \gamma \sum_{a \in \mathcal{A}(s)} \pi(a \,|\, s) \sum_{\substack{r \in \mathcal{R} \\ s' \in \mathcal{S}}} p(r, s' \,|\, s, a) V_\pi(s') \tag{55b}$$

Colocando em evidência alguns termos, obtêm-se a equação de Bellman para a função valor:

$$V_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a \,|\, s) \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} p(r, s' \,|\, s, a) \left[r + \gamma V_\pi(s')\right] \tag{56}$$

A equação de Bellman relaciona o valor de um estado com duas quantidades: uma expectativa de recompensa imediata r e um valor esperado para o estado seguinte, s'. Cada uma dessas quantidades levam em conta as diferentes probabilidades associadas ao PDM. Uma maneira de entender a Equação 56 é a seguinte: qual seria o valor médio do estado s, considerando todas as possibilidades de ações, recompensas e próximos estados, cada qual multiplicado por um peso, associado à sua probabilidade de ocorrência?

A equação de Bellman para a função ação-valor pode ser obtida com um procedimento similar ao adotado na demonstração da Equação 56:

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s, A_t = a\right] \tag{57a}$$

$$= \mathbb{E}_\pi\left[R_{t+1} \,\middle|\, S_t = s, A_t = a\right] + \gamma \mathbb{E}_\pi\left[G_{t+1} \,\middle|\, S_t = s, A_t = a\right] \tag{57b}$$

A primeira parcela da soma representa a função de recompensa (Equação 43):

$$Q_\pi(s, a) = r(s, a) + \gamma \sum_{g \in \mathcal{G}} \Pr\{g \,|\, s, a\} g \tag{58a}$$

$$= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \Pr\{s' \,|\, s, a\} \sum_{g \in \mathcal{G}} \Pr\{g \,|\, s, a, s'\} g \tag{58b}$$

$$= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \Pr\{s' \,|\, s, a\} \sum_{a' \in \mathcal{A}(s')} \pi(a' \,|\, s') \sum_{g \in \mathcal{G}} \Pr\{g \,|\, s, a, s', a'\} g \tag{58c}$$

$$= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \,|\, s, a) \sum_{a' \in \mathcal{A}(s')} \pi(a' \,|\, s') \underbrace{\sum_{g \in \mathcal{G}} \Pr\{g \,|\, s', a'\} g}_{Q_\pi(s', a')} \tag{58d}$$

Com isso, obtêm-se a equação de Bellman para a função ação-valor:

$$Q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \,|\, s, a) \sum_{a' \in \mathcal{A}(s')} \pi(a' \,|\, s') Q_\pi(s', a') \tag{59}$$

3.3 Q-Learning

A criação do algoritmo *Q-learning* (WATKINS; DAYAN, 1992) foi um marco para a teoria de aprendizado por reforço. Este algoritmo busca uma aproximação para a função ação-valor ótima associada a uma política determinística.

Pode-se aplicar a equação de Bellman para a função ação-valor ótima (Equação 46b):

$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a) \tag{60a}$$

$$= \max_{\pi} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \sum_{a' \in \mathcal{A}(s')} \pi(a' \mid s') Q_\pi(s',a') \right\} \tag{60b}$$

Como a função de recompensa e a distribuição de estados seguintes não dependem da política, têm-se que:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{\pi} \left\{ \sum_{a' \in \mathcal{A}(s')} \pi(a' \mid s') Q_\pi(s',a') \right\} \tag{61a}$$

Já que, por definição, uma política ótima π^* está associada à Q^*, pode-se escrever:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \sum_{a' \in \mathcal{A}(s')} \pi^*(a' \mid s') Q^*(s',a') \tag{62}$$

Nota-se que:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \sum_{a' \in \mathcal{A}(s')} \pi^*(a' \mid s') Q^*(s',a') \tag{63a}$$

$$= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \max_{a'} \left\{ Q^*(s',a') \right\} \tag{63b}$$

A igualdade entre as Equações 63a e 63b é verificada com a expansão do somatório sobre o espaço de ações:

$$\sum_{a' \in \mathcal{A}(s')} \pi^*(a' \mid s') Q^*(s',a') = \pi^*(a_1 \mid s') Q^*(s',a_1) + \pi^*(a_2 \mid s') Q^*(s',a_2) + \dots \tag{64}$$

Supondo que $Q^*(s',a_1) \geq Q^*(s',a_i)$, para qualquer ação $\{a_i \in \mathcal{A}(s') \mid a_i \neq a_1\}$, o somatório atinge o valor máximo se $\pi^*(a_1|s') = 1$, ou seja, se a política π^* seleciona a ação mais promissora deterministicamente, em relação à expectativa de recompensa Q^*. Ainda que duas ações tenham o mesmo valor para um estado, por exemplo, se $Q^*(s',a_1) = Q^*(s',a_2)$, selecionar a_1 ou a_2 com probabilidade 1 maximiza a Equação 64.

Essa política que seleciona ações deterministicamente, com base no maior valor de Q, é conhecida como *greedy* (ambiciosa) e é denotada por π_g neste trabalho. O algoritmo *Q-learning* busca construir uma aproximação $\hat{Q}^*_{\pi_g}(s,a) \approx Q^*_{\pi_g}(s,a)$, isto é, busca-se a função ação-valor ótima associada à política *greedy*. Pode-se reescrever a Equação 63b para enfatizar essa afirmação:

$$Q^*_{\pi_g}(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \max_{a'} \left\{ Q^*_{\pi_g}(s',a') \right\} \tag{65}$$

Como π_g é uma política determinística, $a' = \pi_g(s')$:

$$Q^*_{\pi_g}(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) Q^*_{\pi_g}(s', \pi_g(s')) \tag{66}$$

Escrevendo a Equação 66 em termos de expectativa:

$$\begin{aligned} Q^*_{\pi_g}(s,a) = & \mathbb{E}_{R_{t+1} \sim p(r \mid s,a)} \left[R_{t+1} \mid S_t = s, A_t = a \right] + \\ & \gamma \mathbb{E}_{S_{t+1} \sim p(s' \mid s,a)} \left[Q^*_{\pi_g} \left(S_{t+1}, \pi_g(S_{t+1}) \right) \mid S_t = s, A_t = a \right] \end{aligned} \tag{67}$$

A primeira parcela da soma na Equação 67 é a definição de função de recompensa (Equação 43), enquanto a segunda parcela é o valor esperado da função $Q^*_{\pi_g}$, considerando os possíveis próximos estados.

As expectativas da Equação 67 podem ser estimadas com amostras da interação do agente com o ambiente. Supondo que exista uma estimativa $\hat{Q}^*_{\pi_g}$ para a função $Q^*_{\pi_g}$ e que o agente se encontre em um determinado estado: após realizar uma ação a_t, uma recompensa $R_{t+1} = r_{t+1}$ é obtida e o sistema transita para o próximo estado $S_{t+1} = s_{t+1}$. A soma da recompensa obtida com o valor estimado para o próximo estado, considerando uma ação da política *greedy*, isto é, $Q^*_{\pi_g}(S_{t+1}, \pi_g(S_{t+1}))$, podem ser usados para atualizar a aproximação $\hat{Q}^*_{\pi_g}$, através da multiplicação do erro por uma constante η, similar à taxa de aprendizado introduzida na Seção 1:

$$\hat{Q}^*_{\pi_g}(s_t,a_t) \leftarrow \overbrace{\hat{Q}^*_{\pi_g}(s_t,a_t)}^{\text{Valor atual}} + \eta \Bigg(\underbrace{\overbrace{r_{t+1} + \gamma \hat{Q}^*_{\pi_g}(s_{t+1}, \pi_g(s_{t+1}))}^{\text{Nova estimativa}} - \hat{Q}^*_{\pi_g}(s_t,a_t)}_{\text{Erro}} \Bigg) \tag{68}$$

Uma propriedade interessante do algoritmo *Q-learning* pode ser observada na Equação 67: a aleatoriedade das expectativas vêm da dinâmica do PDM (qual recompensa R_{t+1} o ambiente fornece e qual é o próximo estado S_{t+1}), isto é, as estimativas dos valores esperados podem ser obtidas com interações que envolvam a tomada de qualquer ação no ambiente. Em suma, é possível utilizar uma outra política π_b para explorar o ambiente e utilizar as informações obtidas para aproximar a função ação-valor ótima de uma política *greedy*. Por esta constatação, o *Q-learning* é considerado um algoritmo

off-policy.

Como na maior parte dos problemas de aprendizado por reforço o agente não tem conhecimento da dinâmica do ambiente, é fundamental que o mesmo seja capaz de explorar o espaço de estados. Uma maneira de auxiliar este processo exploratório é empregar uma política heurística que seleciona uma ação aleatória, com probabilidade ϵ, ou a ação $a = \pi_g(s)$, com probabilidade $(1 - \epsilon)$. Essa política é chamada de ϵ-*greedy*.

3.4 **Policy Gradient**

O algoritmo *Q-learning* aproxima a função ação-valor ótima e essa estimativa é utilizada para selecionar a melhor ação, em termos de expectativa de retorno. Pode-se pensar em otimizar diretamente a política, com base nas recompensas obtidas na interação do agente com o ambiente. Suponha que uma política, denotada por $\pi_{\boldsymbol{\phi}}$, é parametrizada por um vetor de parâmetros $\boldsymbol{\phi}$, representando, por exemplo, os pesos e as constantes de uma rede neural. Caso a política seja probabilística, isto é, $\pi_{\boldsymbol{\phi}} = \pi_{\boldsymbol{\phi}}(a \mid s)$, a rede neural pode receber o estado s como entrada e fornecer como saída a probabilidade de cada ação, utilizando a função *softmax*, por exemplo. Se a política é determinística, a rede neural recebe o estado s como entrada e a saída é um valor real. Neste caso, a camada de saída pode apresentar uma ativação linear ou não-linear.

Enquanto o algoritmo *Q-learning* busca uma aproximação da função valor e usa essa estimativa para selecionar a ação, os métodos caracterizados como *policy gradient* otimizam diretamente os parâmetros $\boldsymbol{\phi}$ da política. Supondo que as interações são episódicas e que não haja desconto ($\gamma = 1$), a ideia fundamental é capturar o gradiente do retorno, obtido seguindo a política $\pi_{\boldsymbol{\phi}}$, a partir do estado inicial. Dessa forma, a função objetivo desse problema de otimização é o valor do estado inicial, sob a política $\pi_{\boldsymbol{\phi}}$:

$$J(\boldsymbol{\phi}) = V_{\pi_{\boldsymbol{\phi}}}(s_0) \tag{69}$$

Com a obtenção do gradiente da Equação 69, é possível atualizar os parâmetros da política, visando a maximização da função valor:

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \eta \nabla J(\boldsymbol{\phi}) \tag{70}$$

Um dos teoremas de maior importância para o aprendizado por reforço é o *teorema do gradiente da política* (SUTTON et al., 1999), cuja derivação é apresentada a seguir:

$$\nabla V_{\pi_{\boldsymbol{\phi}}}(s) = \nabla_{\boldsymbol{\phi}} \left[\sum_{a \in \mathcal{A}(s)} \pi_{\boldsymbol{\phi}}(a \mid s) Q_{\pi_{\boldsymbol{\phi}}}(s, a) \right] \tag{71}$$

Aplicando a regra da cadeia:

$$\nabla V_{\pi_\phi}(s) = \sum_{a\in\mathcal{A}(s)} \left[\nabla_\phi \pi_\phi(a \mid s) Q_{\pi_\phi}(s,a) + \pi_\phi(a \mid s)\nabla_\phi Q_{\pi_\phi}(s,a)\right] \tag{72a}$$

$$= \sum_{a\in\mathcal{A}(s)} \Bigg[\nabla_\phi \pi_\phi(a \mid s) Q_{\pi_\phi}(s,a) \tag{72b}$$

$$+ \pi_\phi(a \mid s)\nabla_\phi \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s', r \mid s, a)\left(r + V_{\pi_\phi}(s')\right) \Bigg]$$

Como a recompensa r não depende de ϕ e $\sum_{r\in\mathcal{R}} p(s', r|s, a) = p(s'|s, a)$:

$$\nabla V_{\pi_\phi}(s) = \underbrace{\sum_{a\in\mathcal{A}(s)} \left[\nabla_\phi \pi_\phi(a \mid s) Q_{\pi_\phi}(s,a)}_{\Phi(s)} + \pi_\phi(a \mid s) \sum_{s'\in\mathcal{S}} p(s' \mid s, a)\nabla_\phi V_{\pi_\phi}(s') \right] \tag{73}$$

Para simplificar a notação, denota-se por $\Phi(s)$ a parcela indicada na Equação 73. Rearranjando os termos:

$$\nabla V_{\pi_\phi}(s) = \Phi(s) + \sum_{a\in\mathcal{A}(s)} \pi_\phi(a \mid s) \sum_{s'\in\mathcal{S}} p(s' \mid s, a)\nabla_\phi V_{\pi_\phi}(s') \tag{74a}$$

$$= \Phi(s) + \sum_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}} \pi_\phi(a \mid s) p(s' \mid s, a)\nabla_\phi V_{\pi_\phi}(s') \tag{74b}$$

$$= \Phi(s) + \sum_{s'\in\mathcal{S}} \underbrace{\sum_{a\in\mathcal{A}(s)} \pi_\phi(a \mid s) p(s' \mid s, a)}_{\Pr\{s' \mid s\}} \nabla_\phi V_{\pi_\phi}(s') \tag{74c}$$

Uma parte da Equação 74c representa a probabilidade de transição de um estado para outro, como é demonstrado abaixo:

$$\Pr\{s' \mid s\} = \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) \Pr\{s' \mid s, a\} \tag{75a}$$

$$= \sum_{a\in\mathcal{A}(s)} \pi(a \mid s) p(s' \mid s, a) \tag{75b}$$

Com isso, a partir da Equação 74c:

$$\nabla V_{\pi_\phi}(s) = \Phi(s) + \sum_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}} \pi_\phi(a \mid s) p(s' \mid s, a)\nabla_\phi V_{\pi_\phi}(s') \tag{76a}$$

$$= \Phi(s) + \sum_{s'\in\mathcal{S}} \Pr\{s' \mid s\}\nabla_\phi V_{\pi_\phi}(s') \tag{76b}$$

A Equação 76b tem um significado intuitivo: o gradiente da função valor em um es-

tado depende do gradiente do valor dos possíveis próximos estados, cada um multiplicado pela probabilidade de ocorrência. Esta forma recursiva do gradiente pode ser expandida:

$$\nabla V_{\pi_\phi}(s) = \Phi(s) + \sum_{s' \in \mathcal{S}} \Pr\{s' \mid s\} \nabla_\phi V_{\pi_\phi}(s') \tag{77a}$$

$$= \Phi(s) + \sum_{s' \in \mathcal{S}} \Pr\{s' \mid s\} \Bigg[\Phi(s') + \sum_{s'' \in \mathcal{S}} \Pr\{s'' \mid s'\} \nabla_\phi V_{\pi_\phi}(s'') \Bigg] \tag{77b}$$

$$= \Phi(s) + \sum_{s' \in \mathcal{S}} \Pr\{s' \mid s\} \Bigg[\Phi(s') + \sum_{s'' \in \mathcal{S}} \Pr\{s'' \mid s'\} \Bigg[\Phi(s'') + \sum_{s''' \in \mathcal{S}} \Pr\{s''' \mid s''\} \nabla_\phi V_{\pi_\phi}(s''') \Bigg] \Bigg] \tag{77c}$$

$$= \Phi(s) + \sum_{s' \in \mathcal{S}} \Pr\{s' \mid s\} \Phi(s') + \sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \Pr\{s'' \mid s'\} \Pr\{s' \mid s\} \Phi(s'') + \sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \sum_{s''' \in \mathcal{S}} \Pr\{s''' \mid s''\} \Pr\{s'' \mid s'\} \Pr\{s' \mid s\} \Phi(s''') + \dots \tag{77d}$$

Nota-se na Equação 77d que:

$$\Pr\{s'' \mid s\} = \sum_{s' \in \mathcal{S}} \Pr\{s'' \mid s', s\} \Pr\{s' \mid s\} \tag{78a}$$

$$= \sum_{s' \in \mathcal{S}} \Pr\{s'' \mid s'\} \Pr\{s' \mid s\} \tag{78b}$$

Onde $\Pr\{s'' \mid s', s\} = \Pr\{s'' \mid s'\}$ devido à propriedade de Markov. Da mesma forma:

$$\Pr\{s''' \mid s\} = \sum_{s' \in \mathcal{S}} \Pr\{s''' \mid s', s\} \Pr\{s' \mid s\} \tag{79a}$$

$$= \sum_{s' \in \mathcal{S}} \Pr\{s''' \mid s'\} \Pr\{s' \mid s\} \tag{79b}$$

$$= \sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \Pr\{s''' \mid s'', s'\} \Pr\{s' \mid s'', s\} \Pr\{s'' \mid s', s\} \tag{79c}$$

$$= \sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \Pr\{s''' \mid s''\} \Pr\{s' \mid s\} \Pr\{s'' \mid s'\} \tag{79d}$$

$$= \sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \Pr\{s''' \mid s''\} \Pr\{s'' \mid s'\} \Pr\{s' \mid s\} \tag{79e}$$

O termo $\Pr\{s'' \mid s\}$ (Equação 78) representa a probabilidade de transitar de um determinado estado no instante t para outro no instante $(t+2)$. Da mesma forma, $\Pr\{s''' \mid s\}$ (Equação 79) representa a probabilidade de transitar de um estado para outro em três instantes de tempo. Considerando a notação $\psi^{\pi}(s \to x, n)$ para indicar a probabilidade de transitar de um estado s para outro x, em n instantes de tempo, quando uma política π é seguida, a Equação 77d pode ser escrita como:

$$\nabla V_{\pi_\phi}(s) = \Phi(s) + \sum_{s' \in \mathcal{S}} \Pr\{s' \mid s\} \Phi(s') \tag{80a}$$
$$\sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \Pr\{s'' \mid s'\} \Pr\{s' \mid s\} \Phi(s'') +$$
$$\sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \sum_{s''' \in \mathcal{S}} \Pr\{s''' \mid s''\} \Pr\{s'' \mid s'\} \Pr\{s' \mid s\} \Phi(s''') + \ldots$$

$$= \Phi(s) + \sum_{s' \in \mathcal{S}} \underbrace{\Pr\{s' \mid s\}}_{\psi^{\pi_\phi}(s \to x, 1)} \Phi(s') \tag{80b}$$
$$\sum_{s'' \in \mathcal{S}} \underbrace{\sum_{s' \in \mathcal{S}} \Pr\{s'' \mid s'\} \Pr\{s' \mid s\}}_{\Pr\{s'' \mid s\} = \psi^{\pi_\phi}(s \to x, 2)} \Phi(s'') +$$
$$\sum_{s''' \in \mathcal{S}} \underbrace{\sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \Pr\{s''' \mid s''\} \Pr\{s'' \mid s'\} \Pr\{s' \mid s\}}_{\Pr\{s''' \mid s\} = \psi^{\pi_\phi}(s \to x, 3)} \Phi(s''') + \ldots$$

$$= \Phi(s) + \sum_{x \in \mathcal{S}} \psi^{\pi_\phi}(s \to x, 1) \Phi(x) + \tag{80c}$$
$$\sum_{x \in \mathcal{S}} \psi^{\pi_\phi}(s \to x, 2) \Phi(x) +$$
$$\sum_{x \in \mathcal{S}} \psi^{\pi_\phi}(s \to x, 3) \Phi(x) + \ldots$$

Com isso, obtêm-se a seguinte expressão para o gradiente da função valor:

$$\nabla V_{\pi_\phi}(s) = \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \psi^{\pi_\phi}(s \to x, k) \Phi(x) \tag{81a}$$

$$= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \psi^{\pi_\phi}(s \to x, k) \sum_{a \in \mathcal{A}} \nabla_\phi \pi_\phi(a \mid x) Q_{\pi_\phi}(x, a) \tag{81b}$$

Para o caso episódico, o gradiente da função objetivo é calculado para estado inicial, dessa forma, a Equação 81b se torna:

$$\nabla_\phi J(\phi) = \nabla_\phi V_{\pi_\phi}(s_0) \tag{82a}$$

$$= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \psi^{\pi_\phi}(s_0 \to x, k) \sum_{a \in \mathcal{A}} \nabla_\phi \pi_\phi(a \mid x) Q_{\pi_\phi}(x, a) \tag{82b}$$

Com atenção à seguinte parcela da Equação 82:

$$\sum_{k=0}^{\infty} \psi^{\pi_\phi}(s_0 \to x, k) \tag{83}$$

Cada termo do somatório fornece a probabilidade de que o estado x seja visitado no instante de tempo k, dado um estado inicial s_0. O somatório dessas probabilidades pode ser interpretado como uma taxa de visitação para o estado x ou o tempo médio que o agente permanece no estado x, em um episódio. Esta constatação é verificada ao considerar uma variável aleatória indicadora $N_{x,t}$, que assume valor 1 caso o estado x seja alcançado no instante t e 0 caso contrário. Dessa forma, o valor esperado da soma de $N_{x,t}$, para cada instante de tempo do episódio, corresponde ao número de vezes que o estado é visitado. A Equação 84 mostra que esta expectativa corresponde, de fato, à parcela da Equação 83:

$$n(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} N_{x,t}\right] \tag{84a}$$

$$= \sum_{t=0}^{\infty} \mathbb{E}\left[N_{x,t}\right] \tag{84b}$$

$$= \sum_{t=0}^{\infty} \Pr\{S_t = x\} \tag{84c}$$

$$= \sum_{t=0}^{\infty} \psi^{\pi_\phi}(s_0 \to x, t) \tag{84d}$$

A taxa de visitação $n(x)$ pode ser normalizada para constituir uma distribuição de probabilidade sobre os estados, indicando o valor esperado da fração do tempo que o

agente permanece em um determinado estado:

$$\upsilon_{\pi_\phi}(x) = \frac{n(x)}{\sum_{s \in \mathcal{S}} n(s)} \tag{85a}$$

Com isso, o teorema do gradiente da política pode ser escrito como:

$$\nabla_\phi J(\phi) \propto \sum_{s \in \mathcal{S}} \upsilon_{\pi_\phi}(s) \sum_{a \in \mathcal{A}} \nabla_\phi \pi_\phi(a \mid s) Q_{\pi_\phi}(s, a) \tag{86a}$$

Um importante algoritmo baseado na Equação 86 é o REINFORCE (WILLIAMS, 1992; WILLIANMS, 1988). Observa-se que a distribuição de estados visitados depende da política que está sendo seguida. Desde que o agente siga a política π em um episódio, é de se esperar que os estados sejam visitados de acordo com a distribuição $\upsilon_\pi(s)$[8]. É possível, então, expressar o gradiente como uma expectativa:

$$\nabla_\phi J(\phi) \propto \sum_{s \in \mathcal{S}} \upsilon_{\pi_\phi}(s) \sum_{a \in \mathcal{A}} \nabla_\phi \pi_\phi(a \mid s) Q_{\pi_\phi}(s, a) \tag{87a}$$

$$= \mathbb{E}_{S_t \sim \upsilon_{\pi_\phi}(s)} \left[\sum_{a \in \mathcal{A}} \nabla_\phi \pi_\phi(a_t \mid S_t) Q_{\pi_\phi}(S_t, a_t) \right] \tag{87b}$$

Multiplica-se a Equação 87b por $\frac{\pi_\phi(a_t \mid S_t)}{\pi_\phi(a_t \mid S_t)}$ com o intuito de substituir o somatório sobre todas as ações por uma expectativa:

$$\nabla_\phi J(\phi) \propto \mathbb{E}_{S_t \sim \upsilon_{\pi_\phi}(s)} \left[\sum_{a \in \mathcal{A}} \pi_\phi(a_t \mid S_t) \frac{\nabla_\phi \pi_\phi(a_t \mid S_t)}{\pi_\phi(a_t \mid S_t)} Q_{\pi_\phi}(S_t, a_t) \right] \tag{88a}$$

$$= \mathbb{E}_{S_t \sim \upsilon_{\pi_\phi}(s),\, A_t \sim \pi_\phi(a \mid s)} \left[\frac{\nabla_\phi \pi_\phi(A_t \mid S_t)}{\pi_\phi(A_t \mid S_t)} Q_{\pi_\phi}(S_t, A_t) \right] \tag{88b}$$

Utiliza-se a relação $\nabla_\phi \log f(\phi) = \frac{\nabla_\phi f(\phi)}{f(\phi)}$ para simplificar a Equação 88b:

$$\nabla_\phi J(\phi) \propto \mathbb{E}_{S_t \sim \upsilon_{\pi_\phi}(s),\, A_t \sim \pi_\phi(a \mid s)} \left[Q_{\pi_\phi}(S_t, A_t) \nabla_\phi \log \pi_\phi(A_t \mid S_t) \right] \tag{89a}$$

Substituindo a definição de função ação-valor na Equação 89 têm-se que:

$$\nabla_\phi J(\phi) \propto \mathbb{E}_{S_t \sim \upsilon_{\pi_\phi}(s),\, A_t \sim \pi_\phi(a \mid s)} \left[\mathbb{E}\left[G_t \,\middle|\, S_t, A_t \right] \nabla_\phi \log \pi_\phi(A_t \mid S_t) \right] \tag{90a}$$

$$= \mathbb{E}_{S_t \sim \upsilon_{\pi_\phi}(s),\, A_t \sim \pi_\phi(a \mid s)} \left[\mathbb{E}\left[G_t \nabla_\phi \log \pi_\phi(A_t \mid S_t) \,\middle|\, S_t, A_t \right] \right] \tag{90b}$$

$$= \mathbb{E}_{S_t \sim \upsilon_{\pi_\phi}(s),\, A_t \sim \pi_\phi(a \mid s)} \left[G_t \nabla_\phi \log \pi_\phi(A_t \mid S_t) \right] \tag{90c}$$

[8] O subscrito em $\upsilon_\pi(s)$ é utilizado para tornar explícita a influência da política π na distribuição de estados visitados.

A Equação 90 sugere um algoritmo que interaja com o ambiente seguindo a política $\pi_{\boldsymbol{\phi}}$ e, ao fim do episódio, calcule o retorno G_t, para cada instante de tempo. Este valor é multiplicado pelo gradiente da função LV da política, que pode ser prontamente obtido através dos grafos computacionais descritos na Seção 1.2, quando a política é parametrizada por uma rede neural, por exemplo. Os parâmetros $\boldsymbol{\phi}$ são, então, atualizados com base na Equação 70. Um dos problemas deste algoritmo é a alta variância já que o gradiente é ponderado pelo retorno e, intuitivamente, pode-se esperar uma alta variação dessa quantidade, dado que cada ação pode mudar por completo a trajetória de estados e recompensas futuras. A alta variância do algoritmo REINFORCE pode ser mitigada com a inclusão de uma função $b(s)$ dependente do estado:

$$\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) \propto \sum_{s \in \mathcal{S}} \upsilon(s) \sum_{a \in \mathcal{A}} \nabla_{\boldsymbol{\phi}} \pi_{\boldsymbol{\phi}}(a \mid s)(Q_{\pi_{\boldsymbol{\phi}}}(s,a) - b(s)) \tag{91}$$

Este fator pode ser incluído já que seu valor é nulo:

$$\begin{aligned}
\sum_{s \in \mathcal{S}} \upsilon(s) \sum_{a \in \mathcal{A}} \nabla_{\boldsymbol{\phi}} \pi_{\boldsymbol{\phi}}(a \mid s) b(s) &= \sum_{s \in \mathcal{S}} \upsilon(s) b(s) \sum_{a \in \mathcal{A}} \nabla_{\boldsymbol{\phi}} \pi_{\boldsymbol{\phi}}(a \mid s) && \text{(92a)} \\
&= \sum_{s \in \mathcal{S}} \upsilon(s) b(s) \nabla_{\boldsymbol{\phi}} \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\phi}}(a \mid s) && \text{(92b)} \\
&= \sum_{s \in \mathcal{S}} \upsilon(s) b(s) \nabla_{\boldsymbol{\phi}} 1 && \text{(92c)} \\
&= \sum_{s \in \mathcal{S}} \upsilon(s) b(s) 0 && \text{(92d)} \\
&= 0 && \text{(92e)}
\end{aligned}$$

Na Equação 92a, $b(s)$ pode ser deslocado do somatório já que não depende das ações, apenas do estado. O gradiente, por ser um operador linear, também pode ser deslocado. Por fim, já que $\pi_{\boldsymbol{\phi}}(a \mid s)$ é uma distribuição de probabilidade o somatório sobre o seu suporte vale 1.

Como $b(s)$ deve depender apenas do estado, pode-se utilizar uma estimativa da função valor. Dessa forma, dada a correlação entre o retorno obtido, a partir de um estado, e o valor estimado, é de se esperar que a variância do gradiente seja menor. Com essa proposição, a atualização dos parâmetros da política se torna:

$$\boldsymbol{\phi}_{t+1} = \boldsymbol{\phi}_t + \eta \left(G_t - \hat{V}(S_t) \right) \log \pi_{\boldsymbol{\phi}}(A_t \mid S_t) \tag{93}$$

É possível ainda introduzir uma aproximação para o retorno, com base na função

valor estimada:

$$G_t \approx R_{t+1} + \gamma \hat{V}(S_{t+1}) \tag{94}$$

Os parâmetros são então atualizados com base na expressão abaixo:

$$\phi_{t+1} = \phi_t + \eta \underbrace{\left(R_{t+1} + \gamma \hat{V}(S_{t+1}) - \hat{V}(S_t) \right)}_{\delta_t} \log \pi_\phi(A_t \mid S_t) \tag{95}$$

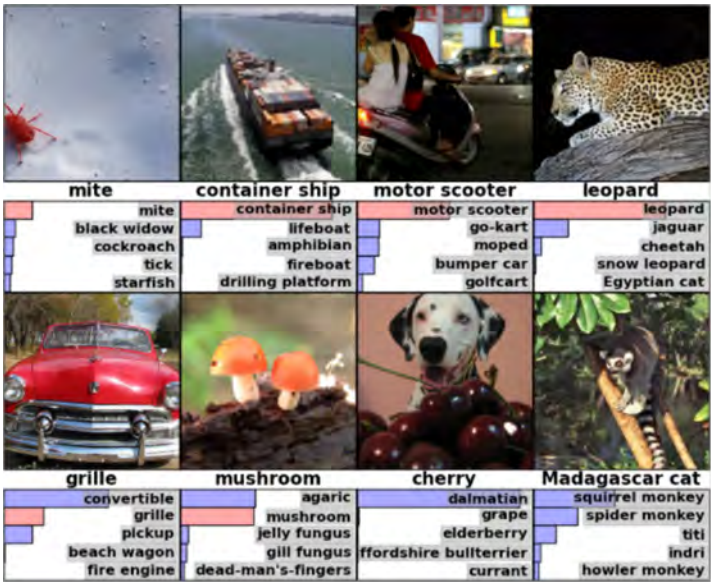
Nota-se que essa atualização introduz um viés, já que uma aproximação da função valor é utilizada, e não o valor real. Algoritmos que utilizam expressões semelhantes a Equação 95 são conhecidos como *ator-crítico* (LILLICRAP et al., 2015; FUJIMOTO; HOOF; MEGER, 2018; SCHULMAN et al., 2017; HAARNOJA et al., 2018; SCHULMAN et al., 2015). A política seleciona as ações e atua no ambiente, por isso é chamada de *ator*, enquanto a aproximação da função valor "julga" a qualidade das ações escolhidas. Se a parcela indicada por δ_t é positiva, de acordo com o valor indicado pelo "crítico", a ação selecionada tende a se tornar preponderante, caso contrário, a ação escolhida terá probabilidade menor de ocorrer após a atualização. É possível notar a semelhança das atualizações de parâmetros tratadas nesta Seção com o objetivo de máxima log-verossimilhança, abordado na Seção 1, com a particularidade que a direção e magnitude da atualização dos parâmetros é ponderada pelo retorno.

3.5 **Aprendizado por Reforço Profundo**

Um dos fatores que contribuíram para a popularidade do aprendizado por reforço nos últimos anos foi a união entre esse campo e o aprendizado profundo. Tipicamente, utilizavam-se redes neurais em problemas de aprendizado por reforço em dados estruturados, com algum tipo de pré-processamento. Por exemplo, supondo um problema em que um robô navega em um ambiente desconhecido, o estado seria composto pela posição e orientação do robô. Em contraste, um ser humano é capaz de navegar em ambientes complexos utilizando apenas a visão, sem necessitar de informações precisas sobre sua localização no ambiente. Em problemas de classificação de imagens, o uso de redes neurais convolucionais (GU et al., 2018) com diversas camadas permitiu um avanço extraordinário na área de visão computacional. Seria possível obter o mesmo sucesso com algoritmos de aprendizado por reforço através do processamento direto de dados não estruturados, como imagens?

Um dos trabalhos mais importantes nos últimos anos culminou com a criação do algoritmo *deep Q-learning* (MNIH et al., 2015) que respondeu parcialmente esses questionamentos. Mnih et al. (2015) mostraram que era possível treinar agentes em jogos de Atari

Figura 37 - Classificação de imagens pela rede neural convolucional "AlexNet"



Fonte: Krizhevsky, Sutskever e Hinton (2017).

(video-game), processando imagens através de redes neurais convolucionais, de forma a obter desempenho similar a humanos. Além disso, o método proposto lidou com diversos problemas decorrentes da integração de redes neurais profundas com o aprendizado por reforço. Os parágrafos seguintes detalham o funcionamento do algoritmo.

O algoritmo *Q-learning* clássico aproxima a função ação-valor ótima supondo uma política *greedy*. A equação de Bellman, como visto na Seção 3.3, estabelece que:

$$\begin{aligned} Q^*_{\pi_g}(s,a) =& \mathbb{E}_{R_{t+1}\sim p(r\,|\,s,a)}\left[R_{t+1} \;\middle|\; S_t = s, A_t = a\right] + \\ & \gamma \mathbb{E}_{S_{t+1}\sim p(s'\,|\,s,a)}\left[Q^*_{\pi_g}\left(S_{t+1}, \pi_g(S_{t+1})\right) \;\middle|\; S_t = s, A_t = a\right] \end{aligned} \tag{96}$$

Se as expectativas são obtidas a partir de amostras da interação do agente com o ambiente, a cada instante de tempo, a atualização da função ação-valor aproximada é:

$$\hat{Q}^*_{\pi_g}(s_t,a_t) \leftarrow \hat{Q}^*_{\pi_g}(s_t,a_t) + \eta\Big(\underbrace{r_{t+1} + \gamma \hat{Q}^*_{\pi_g}(s_{t+1}, \pi_g(s_{t+1})) - \hat{Q}^*_{\pi_g}(s_t,a_t)}_{\text{Erro}}\Big) \tag{97}$$

A atualização da Equação 97 é apropriado para problemas em que o valor da aproximação $\hat{Q}^*_{\pi_g}$ é armazenado em tabelas, de forma que a atualização da aproximação em um estado e ação particulares não altera o valor de $\hat{Q}^*_{\pi_g}$ em outros estados e ações. O mesmo não ocorre quando $\hat{Q}^*_{\pi_g}$ é uma função parametrizada, como, por exemplo, uma rede neural. Nesse caso, a atualização dos parâmetros da rede afeta o valor previsto para todos os estados e ações, tornando o processo de otimização problemático. Porém,

o poder expressivo de modelos não-lineares, como as redes PMCs, é fundamental para que a aproximação generalize a estimativa para estados e ações que o agente não visitou durante a fase de treinamento, o que ocorre frequentemente em problemas complexos, com espaços de estados e ações amplos. Supondo que a função ação-valor seja representada por uma rede neural, com parâmetros θ, pode-se utilizar o erro quadrático como critério de otimização, onde o valor de supervisão é o mesmo da Equação 97:

$$E_t(\theta) = \mathbb{E}\left[(C_t - Q_\theta(S_t, A_t))^2\right] \tag{98a}$$

$$= \mathbb{E}\left[\left(R_{t+1} + \gamma Q_\theta(S_{t+1}, \pi_g(S_{t+1})) - Q_\theta(S_t, A_t)\right)^2\right] \tag{98b}$$

Apesar das aparentes semelhanças da Equação 98b com o critério de regressão no aprendizado supervisionado, discutido brevemente na Seção 1.2, algumas particularidades podem ser notadas:

a) Enquanto no aprendizado supervisionado as amostras $\{(\vec{x}_i, \vec{c}_i)\}_{i=1}^{D}$ são fixadas antes do treinamento, na Equação 98b os valores de supervisão C_t mudam ao longo do treinamento, uma vez que dependem de θ, o conjunto de parâmetros atualizados a cada iteração;

b) Na maioria das vezes, o aprendizado supervisionado considera que as amostras são independentes, o que não ocorre na Equação 98b. Supondo que o agente percorra uma trajetória $\Gamma = \{S_0, A_0, S_1, R_1, A_1, \ldots, S_H, R_H\}$ em um episódio, é de se esperar que haja correlação entre as amostras $C_1 = R_2 + \gamma Q_\theta\left(S_2, \pi_g(S_2)\right)$ e $C_2 = R_3 + \gamma Q_\theta(S_3, \pi_g(S_3))$, uma vez que o estado e a recompensa em $t = 3$ dependem do estado visitado anteriormente, em $t = 2$.

Para diminuir a correlação entre as amostras, Mnih et al. (2015) utilizam uma técnica conhecida como *experience replay* (LIN, 1992), em que as experiências do agente com o ambiente são armazenadas em uma memória ($\mathbb{B}$), conhecida como *replay buffer*. Denota-se por $\Omega_t^{(e)}$ a *transição* no instante de tempo t e episódio e, um conjunto com as informações da interação do agente com o ambiente[9]:

$$\Omega_t^{(e)} = \{s_t^{(e)}, a_t^{(e)}, r_{t+1}^{(e)}, s_{t+1}^{(e)}\} \tag{99}$$

Durante a fase de treinamento, em que o agente interage com o ambiente, todas as transições (de todos os episódios) são armazenadas no *replay buffer*. A atua-

[9] Geralmente, um sinal $D_t^{(e)}$, que indica se $S_{t+1}^{(e)}$ é um estado terminal, também é incluído como componente da transição, já que os valores de supervisão (Equação 98b) devem incluir apenas a recompensa imediata quando $S_{t+1}^{(e)}$ é um estado terminal.

lização dos parâmetros da rede neural é realizada através da descida do gradiente estocástico em mini-lotes, onde cada exemplo de treinamento é uma transição amostrada dessa memória. Como as informações armazenadas são de diferentes episódios e instantes de tempo, diminui-se a correlação entre as amostras.

Para resolver o problema em que os valores de supervisão mudam ao longo do treinamento, Mnih et al. (2015) utilizam uma outra rede neural, com uma versão antiga dos parâmetros θ. Se Q_{θ_i} representa a rede neural na iteração i, o valor do alvo C_t (Equação 98) é calculado a partir de $Q_{\theta_{i-1}}$. Dessa forma, o valor de supervisão permanece fixo durante algumas iterações da descida do gradiente estocástico. Essa rede neural secundária é denominada *target* (alvo), já que sua função é apenas prover um valor de supervisão para guiar a atualização de Q_θ. Essas mudanças no critério da Equação 98 fundamentam o algoritmo *deep Q-learning*, cuja função de custo, para uma única amostra, é dada por:

$$E(\theta) = \mathbb{E}_{\Omega_t^{(e)} \sim \mathbb{B}} \left[\left(R_{t+1}^{(e)} + \gamma Q_{\theta_{i-1}} \left(S_{t+1}^{(e)}, \pi_g(S_{t+1}^{(e)}) \right) - Q_{\theta_i} \left(S_t^{(e)}, A_t^{(e)} \right) \right)^2 \right] \tag{100}$$

A diferenciação dessa função em relação aos parâmetros θ fornece o gradiente deste novo critério de otimização:

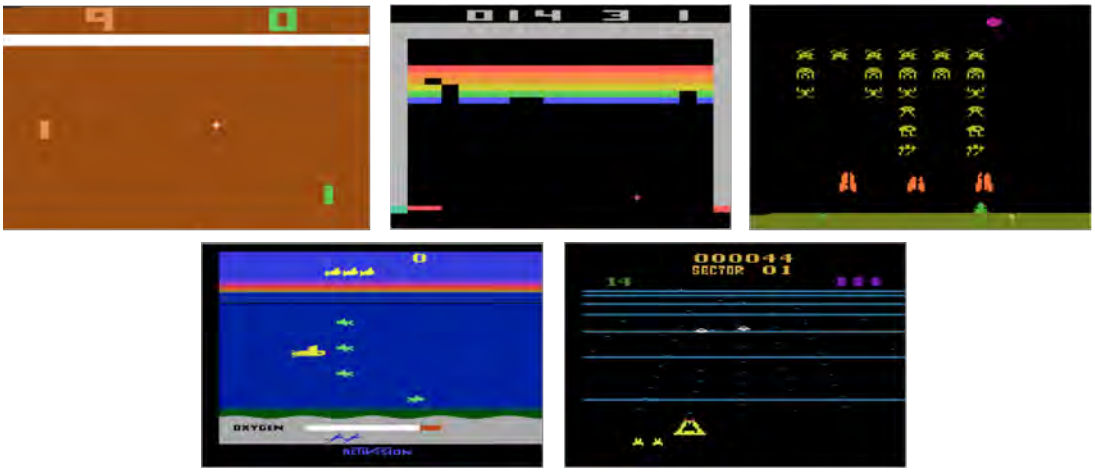
$$\nabla_\theta E(\theta) = \mathbb{E}_{\Omega_t^{(e)} \sim \mathbb{B}} \Bigg[\Bigg(R_{t+1}^{(e)} + \gamma Q_{\theta_{i-1}} \left(S_{t+1}^{(e)}, \pi_g(S_{t+1}^{(e)}) \right) - Q_{\theta_i} \left(S_t^{(e)}, A_t^{(e)} \right) \Bigg) \nabla_\theta Q_\theta \left(S_t^{(e)}, A_t^{(e)} \right) \Bigg] \tag{101}$$

Vale notar que, assim como o algoritmo *Q-learning* clássico, o *deep Q-learning* é *off-policy*. Portanto, as experiências obtidas e armazenadas em $\mathbb{B}$ podem ser geradas por uma política exploratória. Embora a função ação-valor dependa do estado e da ação, a rede neural do algoritmo *deep Q-learning* utilizada por Mnih et al. (2015) recebe apenas uma observação do estado (sob a forma de imagens) como entrada mas têm múltiplas saídas, uma para cada ação. Dessa forma, é possível calcular o valor de todas as ações com uma única fase de propagação direta na rede neural.

O algoritmo *deep Q-learning* obteve sucesso em diversos jogos de Atari (Figura 38), atingindo pontuações similares às obtidas por humanos. Entretanto, sua aplicação é restrita à problemas em que o espaço de ações é discreto. Os jogos de Atari têm um total de 16 ações admissíveis mas um problema de robótica pode admitir ações em um intervalo contínuo, por exemplo, se a ação representa o torque a ser aplicado em uma junta do robô. Nessa situação, verificar a ação que maximiza Q_θ (equivalente a calcular $\pi_g(s)$, a ação da política *greedy*), torna-se uma tarefa computacionalmente intensiva.

O algoritmo *deep deterministic policy gradient* (DDPG) (LILLICRAP et al., 2015)

Figura 38 - Jogos de Atari



Legenda: O algoritmo *Deep Q-learning* processa diretamente dados não-estruturados, sob a forma de imagens, através de redes neurais convolucionais.

Fonte: O autor, 2022.

é similar em muitos aspectos ao *deep Q-learning* mas possibilita a abordagem de problemas em que o espaço de ações é contínuo. Dentre as similaridades destaca-se a utilização de uma rede neural para aproximar a função ação-valor ótima de uma política *greedy*, o uso de um *replay buffer* para armazenar as experiências e a utilização de redes neurais *target* para guiar a atualização de Q_θ.

Como o nome sugere, o algoritmo DDPG se baseia principalmente no teorema do gradiente da política determinística (SILVER et al., 2014), que pode ser visto como um caso especial do gradiente da política estocástica, quando a variância da distribuição de ações tende a zero. Pode ser provado que o gradiente de uma política determinística π_ϕ é dado por:

$$\nabla_\phi J(\phi) = \mathbb{E}_{S\sim\upsilon_{\pi_\phi}}\left[\nabla_\theta \pi_\phi(S)\nabla_a Q_{\pi_\phi}(S,a)\Big|_{a=\pi_\phi(S)}\right] \tag{102}$$

A derivação da Equação 102 é similar à apresentada na Seção 3.4. Assim como o caso estocástico, não é necessário conhecimento de derivadas que envolvem a dinâmica do ambiente. Observa-se na Equação 102 que, assim como o gradiente da política estocástica, a expectativa ocorre sobre a distribuição de estados visitados pela política cujo gradiente deseja-se conhecer. Portanto, é natural que os algoritmos que utilizam o gradiente da política sejam *on-policy*, ou seja, o comportamento do agente é ditado pela política que está sendo aprimorada. Seria possível obter o gradiente de uma política a partir de amostras de uma outra política exploratória?

A Equação 103 é chamada de *off-policy deterministic policy gradiente* e estabelece

uma aproximação para o gradiente de uma política determinística π_ϕ quando os estados visitados são induzidos por uma política β:

$$\nabla_\phi J(\phi) \approx \mathbb{E}_{S\sim\upsilon_\beta}\left[\nabla_\phi Q_{\pi_\phi}(S,a)\Big|_{a=\pi_\phi(S)}\right] \tag{103}$$

A Equação 103 mostra que é possível otimizar uma política determinística a partir da derivada da função ação-valor desta mesma política. A ideia básica do algoritmo DDPG (LILLICRAP et al., 2015) é, portanto, otimizar uma política *greedy* determinística através da Equação 103, utilizando uma estimativa $Q_\theta \approx Q^*_{\pi_g}$. O algoritmo *deep Q-learning* fornece, justamente, uma forma de obter esta aproximação, através da Equação 101. Com isso, Lillicrap et al. (2015) propõe otimizar paralelamente uma política *greedy* determinística π_ϕ e a função ação-valor desta política, Q_θ, por meio da Equação 104:

$$\nabla_\theta E(\theta) = \mathbb{E}_{\Omega_t^{(e)}\sim\mathbb{B}}\Bigg[\Bigg(R_{t+1}^{(e)} + \gamma Q_{\theta'}\left(S_{t+1}^{(e)}, \pi_\phi(S_{t+1}^{(e)})\right) - \tag{104a}$$

$$Q_{\theta'}\left(S_t^{(e)}, A_t^{(e)}\right)\Bigg)\nabla_\theta Q_{\theta'}\left(S_t^{(e)}, A_t^{(e)}\right)\Bigg]$$

$$\nabla_\phi J(\phi) \approx \mathbb{E}_{\Omega_t^{(e)}\sim\mathbb{B}}\left[\nabla_\phi Q_\theta(S_t^{(e)}, \pi_\phi(S_t^{(e)}))\right] \tag{104b}$$

Enquanto no *deep Q-learning* os parâmetros da rede *target* são cópias passadas da rede principal, no algoritmo DDPG as redes *target*, representadas por θ' e ϕ' na Equação 104, são atualizadas para acompanhar lentamente as redes principais, através da Equação 105:

$$\theta' \leftarrow \theta\zeta + (1-\zeta)\theta' \qquad 0 < \zeta < 1 \tag{105a}$$

$$\phi' \leftarrow \phi\zeta + (1-\zeta)\phi' \tag{105b}$$

Pode-se observar na Equação 104 que o algoritmo DDPG calcula o valor de supervisão, C_t na Equação 98, utilizando duas redes *target*, enquanto o algoritmo *deep Q-learning* usa apenas uma. Isso porque o valor de supervisão depende da política *greedy*, agora parametrizada também por uma rede neural e atualizada na mesma frequência que a função ação-valor. Dessa forma, os mesmos argumentos que motivam a utilização de redes *target* para Q_θ no *deep Q-learning* são apropriados para o DDPG, isto é, busca-se a estratificação dos valores de supervisão, visando estabilizar o processo de otimização.

Por ser um algoritmo *off-policy*, é possível utilizar uma política exploratória enquanto a política *greedy* é otimizada. Tipicamente, utiliza-se a própria política *greedy* com a adição de ruído gaussiano para ajudar exploração do ambiente. O DDPG (Algoritmo 5) obteve sucesso em diversos experimentos, muitas vezes superando o desempenho de

Algoritmo 5 DDPG

procedimento TreinarAgenteDDPG
requerer: M: número de amostras do mini-lote de transições
requerer: N: número de episódios de treinamento
    $\theta \leftarrow$ InicializarRedeNeuralFuncaoAcaoValor
    $\phi \leftarrow$ InicializarRedeNeuralPolitica
    $\theta' \leftarrow \theta$
    $\phi' \leftarrow \phi$
    $\mathbb{B} \leftarrow \emptyset$
    para episódio $e = 0 \dots N$ **faça**
        $s_0 \sim p(s_0)$
        para $t = 0 \dots H$ **faça**
            $\nu \sim \mathcal{N}(\mu, \sigma^2)$ ▷ Adição de ruído para exploração
            $a_t \leftarrow \pi_\phi(s_t) + \nu$
            $s_{t+1}, r_{t+1} \sim p(s_{t+1}, r_{t+1} \mid s_t, a_t)$ ▷ Interação agente com o ambiente
            $\Omega_t^{(e)} \leftarrow \{s_t, a_t, s_{t+1}, r_{t+1}\}$
            $\mathbb{B} \leftarrow \mathbb{B} \cup \{\Omega_t^{(e)}\}$ ▷ Adiciona transição à memória
            $\mathcal{M} \leftarrow \emptyset$
            para amostra $m = 1 \dots M$ **faça** ▷ Mini-lote $\mathcal{M}$ de transições
                $\Omega \sim \mathbb{B}$
                $\mathcal{M} \leftarrow \mathcal{M} \cup \{\Omega\}$
            fim para
            $E_\theta, J_\phi \leftarrow 0$
            para todo transição $\Omega \in \mathcal{M}$ **faça**
                $s_t, a_t, s_{t+1}, r_{t+1} \leftarrow \Omega$
                $c \leftarrow r_{t+1} + \gamma Q_{\theta'}(s_{t+1}, \pi_{\phi'}(s_{t+1}))$
                $E(\theta) \leftarrow E_\theta + \left(r_{t+1} + \gamma Q_{\theta'}(s_{t+1}, \pi_{\phi'}(s_{t+1})) - Q_\theta(s_t, a_t)\right)^2$
                $J(\phi) \leftarrow J_\phi + Q_\theta(s_t, \pi_\phi(s_t))$
            fim para
            $E(\theta) \leftarrow E_\theta / |\mathcal{M}|$ ▷ Valor médio dos critérios para cada amostra
            $J(\phi) \leftarrow J_\phi / |\mathcal{M}|$
            $\theta \leftarrow \theta + \eta \nabla_\phi E_\phi$ ▷ Atualização de Q_θ e π_ϕ
            $\phi \leftarrow \phi + \eta \nabla_\phi J_\phi$
            $\theta' \leftarrow \zeta\theta + (1 - \zeta)\theta'$ ▷ Atualização das redes *target*
            $\phi' \leftarrow \zeta\phi + (1 - \zeta)\phi'$
        fim para
    fim para
    retorna G
fim procedimento

métodos que tinham acesso à dinâmica completa do problema (LILLICRAP et al., 2015). Assim como o *deep Q-learning*, o algoritmo pôde lidar com problemas que envolviam o processamento direto de imagens.

3.6 **Aprendizado por Reforço com Múltiplos Objetivos**

A aproximação de funções é essencial para o sucesso dos algoritmos de aprendizado por reforço. Problemas reais dificilmente podem ser abordados por funções de valor ou políticas tabulares, que não utilizam algum tipo de parametrização, isso porque, frequentemente, o número de estados ou ações é incontável nesses casos. Por esta razão, para muitos problemas é impossível que o agente visite todos os estados possíveis durante a fase de treinamento. Entretanto, parametrizar uma política torna possível que o agente tome ações parecidas em estados semelhantes. Em outras palavras, a aproximação de funções ou distribuições pode permitir que o agente generalize para estados nunca visitados o que foi aprendido durante a fase de treinamento.

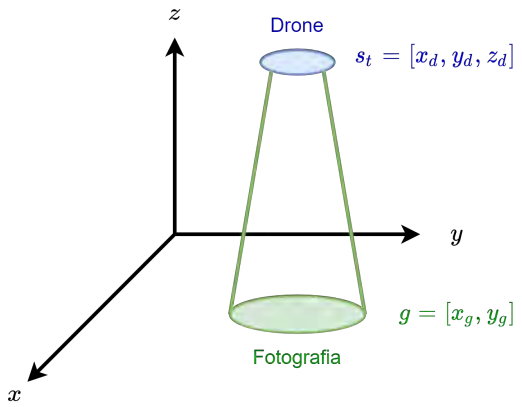
Uma rede neural utilizada para aproximar uma função valor (ou ação-valor) permite generalizar a expectativa de retorno para estados ou ações que nunca foram experimentados. E se essa generalização pudesse ser obtida para diferentes objetivos? Suponha um problema em que um robô deve navegar até um certo ponto no espaço. Um agente treinado por um algoritmo de aprendizado por reforço pode ser especialista em chegar até esse ponto, a partir de qualquer estado inicial, porém não pode-se esperar que o agente seja capaz de atingir outro ponto qualquer. Isto ocorre porque a função de recompensa utilizada no treinamento é específica em guiar a locomoção a apenas um ponto. Contudo, a dinâmica do processo, ou seja, como as ações do agente afetam o estado do sistema, independem do objetivo desejado, definido implicitamente na função de recompensa. Isso sugere que há similaridades entre as diferentes tarefas de locomoção a pontos específicos. Seria possível, então, definir uma função de valor que permita a generalização da expectativa de retorno para diferentes objetivos ou funções de recompensa? Ou uma política que seja otimizada para alguns objetivos mas que generalize para outros? Isso permitiria que um robô especialista em chegar nos ponto P_1 e P_2 seja capaz de atingir qualquer ponto próximo.

O paradigma do aprendizado por reforço com múltiplos objetivos permite que essa generalização seja alcançada. Quando um problema é formulado com base em múltiplos objetivos apenas a função de recompensa e o fator de desconto são alterados: cada objetivo g ("goal") define uma função de recompensa própria r_g e um fator de desconto γ_g. O fator de desconto passa a ser uma função que depende do estado, de forma que $\gamma_g(s) = 0$ se o objetivo g é alcançado no estado s, ou seja, este parâmetro é utilizado para a terminação antecipada do episódio. Formalmente, cada objetivo g induz um PDM dife-

rente $\langle \mathcal{S}, \mathcal{A}, \mathcal{G}, p, r_g, \gamma_g \rangle$, onde $\mathcal{S}$ e $\mathcal{A}$ são os espaços de estados e ações, respectivamente. $\mathcal{G}$ é o espaço de objetivos e p é a distribuição de transição de estados. Assume-se que para qualquer estado s existe um objetivo g correspondente, ou seja, há um mapeamento $m : \mathcal{S} \rightarrow \mathcal{G}$.

Exemplo 8 *Um veículo autônomo aéreo (VAA) é utilizado para tirar fotografias áreas de diversos pontos no solo, a partir de qualquer altitude. Para manter a simplicidade, considera-se que o estado do agente são as coordenadas do VAA ($s_t = [\,x_d \;\; y_d \;\; z_d\,]$) e que a ação define o deslocamento ao longo dos três eixos ($a_t = [\,dx \;\; dy \;\; dz\,]$. Cada objetivo representa as coordenadas $g = [\,x_g \;\; y_g\,]$ do ponto que deve ser fotografado, de forma que se o VAA está no estado $s = [\,x \;\; y \;\; z\,]$ o ponto fotografado (objetivo atingido) é $g = [\,x \;\; y\,]$. Com isso, o mapeamento de estados para objetivos é: $m : [\,x \;\; y \;\; z\,] \rightarrow [\,x \;\; y\,]$. A função de recompensa pode ser definida a partir da distância euclidiana entre o ponto a ser fotografado e a projeção da posição do VAA no solo: $r_g(s, a) = -\sqrt{(x_d - x_g)^2 + (y_d - y_g)^2}$.*

Figura 39 - Um VAA realiza fotografias aéreas



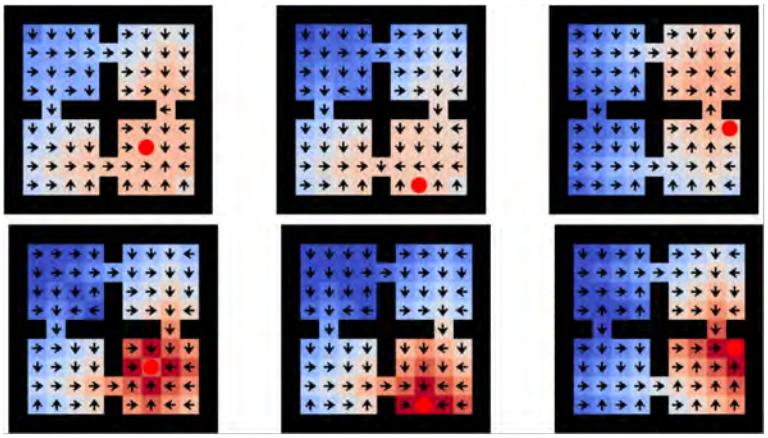
Fonte: O autor, 2022.

Enquanto a função valor aproximada tradicional $V_\pi(s)$ denota uma função parametrizada por θ que representa a expectativa de retorno, quando o agente se encontra em um estado s e segue uma política π, uma função valor universal $V_{g,\pi}(s)$ (Equação 106) representa a expectativa de retorno condicionada também em um objetivo g. Por exemplo, uma rede neural utilizada para aproximar uma função valor universal recebe como entrada também um objetivo g. O mesmo conceito vale para políticas *universais* que dependem do estado e do objetivo. Dessa forma, o algoritmo de aprendizado por reforço visa obter uma única política $\pi^*(a \,|\, s, g)$, que seja ótima para todos os PDMs, cada qual relacionado a um objetivo $g \in \mathcal{G}$.

$$V_{g,\pi}(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} r_g(s_t, a_t) \prod_{k=0}^{t} \gamma_g(s_k) \,\middle|\, s_0 = s \right] \tag{106}$$

Schaul et al. (2015) mostraram que uma função valor universal era capaz de generalizar a expectativa de retorno para objetivos que não faziam parte dos dados de

Figura 40 - Generalização da função valor para diferentes tarefas



Legenda: Um agente deve navegar no labirinto e capturar o círculo vermelho. Os quadrados vermelhos indicam estados com maior valor, em relação aos azuis. Na fileira *inferior*, o valor dos estados para a função valor ótima. Na fileira *superior*, o valor dos estados para a função valor aproximada, obtida após o treinamento. O agente não é treinado em objetivos localizados no quadrante inferior direito, mas é capaz de generalizar, razoavelmente, o valor dos estados para objetivos nesta região.

Fonte: Schaul et al. (2015)

treinamento (Figura 40). O resultado deste trabalho abriu espaço para a resolução de um problema fundamental no aprendizado por reforço relacionado às funções de recompensas esparsas (NAIR et al., 2018; VECERÍK et al., 2017). Os parágrafos seguintes detalham esse problema e a solução encontrada.

Suponha que deseja-se aplicar um algoritmo de aprendizado por reforço no jogo de Xadrez. Como especificar uma função de recompensa para essa tarefa? Seria a solução buscar uma fórmula matemática que incentive o agente a imitar um jogador profissional? Mesmo que isso seja possível, não há garantias de que esta forma de jogar seja ótima, ou ainda próximo de ótima. Em muitos casos, desenvolver uma função de recompensa ingênua pode levar o agente a apresentar um comportamento inesperado, que não realiza o objetivo desejado.

Para evitar esse tipo de problema, a função de recompensa deve especificar o objetivo e não o comportamento do agente. No jogo de Xadrez o objetivo é ganhar a partida, logo a função de recompensa poderia ser binária: o agente recebe 1 se obtêm a vitória e 0 caso contrário. As recompensas binárias, ou esparsas, trazem um desafio para a maior parte dos algoritmos de aprendizado por reforço: tendo em vista o exemplo do Xadrez, como receber uma recompensa pela vitória se o agente não "sabe" como obtê-la? Em geral, os algoritmos utilizam ações aleatórias e aprendem por tentativa e erro, nesse

caso, é improvável que o agente obtenha a vitória. Dessa forma, a única recompensa obtida é 0 e, por consequência, todos os estados apresentarão o mesmo valor ou expectativa de retorno. Neste cenário, o aprendizado não ocorre e a tarefa não é resolvida.

Um progresso significativo foi obtido para problemas de aprendizado por reforço com múltiplos objetivos formulados em termos de recompensas binárias, graças a uma técnica conhecida como *Hindsight Experience Replay* (HER) (ANDRYCHOWICZ et al., 2017). A intuição básica desse algoritmo é descrita a seguir.

Supondo que o algoritmo DDPG seja selecionado para resolver o problema do Exemplo 8, mas que a função de recompensa seja esparsa:

$$r_g(s,a) = \begin{cases} 0 & m(s) \neq g \\ 1 & m(s) = g \end{cases} \tag{107}$$

Isto significa que, no início do episódio, se deseja-se alcançar um objetivo g, uma recompensa unitária obtida somente se um estado s, que corresponde ao objetivo g, seja alcançado. É improvável que o agente consiga atingir o objetivo g durante a fase de treinamento, pelos motivos citados nos parágrafos anteriores. Dessa forma, o episódio contém apenas recompensas nulas e o aprendizado não ocorre. Porém, durante sua trajetória, o agente visita outros estados $\{s_1, s_2, s_3, \ldots\}$ e cada estado visitado têm um objetivo correspondente que foi alcançado $\{g_1, g_2, g_3, \ldots\}$. Se no início do episódio o objetivo desejado fosse $g \in \{g_1, g_2, g_3, \ldots\}$ o objetivo seria alcançado e uma recompensa unitária obtida, permitindo o aprendizado do agente.

Considerando que a trajetória original foi armazenada no *replay buffer*, é possível adicionar à memória outras transições, supondo que o objetivo desejado é um dos que foram alcançados. Isso pode ser feito pois a dinâmica do ambiente não depende do objetivo que o agente escolhe perseguir no início da trajetória. Nesse sentido, a palavra *hindsight* (retrospectiva) representa a pergunta: qual seria a recompensa se o objetivo perseguido fosse outro?

Esse método possui algumas analogias com o aprendizado humano. Um jogador de golfe busca direcionar a bola em direção ao buraco. Mesmo que esse objetivo não seja alcançado, a trajetória da bola carrega informações úteis sobre a dinâmica do processo, ou seja, o aprendizado não ocorre apenas quando o objetivo desejado é alcançado. Sob outra perspectiva, o HER pode ser visto como uma forma de "aumento" da base de dados, um procedimento bastante utilizado no aprendizado supervisionado. Entretanto, as novas trajetórias incluídas na memória não são enviesadas ou falsas, mas informações tão plausíveis quanto à trajetória original.

Dada a descrição intuitiva do HER, uma atenção é dada à terminologia adotada na literatura. No algoritmo DDPG a ação utilizada para explorar o ambiente é uma versão ruidosa da política π_ϕ. Nos problemas com múltiplos objetivos, π_ϕ depende também do

objetivo g que o agente deseja atingir. Como o objetivo sendo perseguido influencia a ação tomada pelo agente, o objetivo desejado é chamado de *comportamental* e denotado por g_b (*behavioural goal*). Durante a fase de treinamento, no início de cada episódio, um objetivo comportamental é escolhido e a política π_ϕ estará condicionada em g_b, ao longo de toda a duração do episódio. No algoritmo HER, g_b vêm de uma distribuição p_b, escolhida de acordo com a tarefa que deseja-se realizar. Conforme a discussão dos parágrafos anteriores, enquanto o agente persegue um determinado objetivo g_b, diversos objetivos são atingidos, cada um correspondendo a um estado visitado ao longo da trajetória. Os objetivos atingidos são denotados por g_a (*achieved goal*).

O HER é compatível com qualquer algoritmo de aprendizado por reforço *off-policy* que use *replay buffer*. O Algoritmo 6 descreve a função que deve ser chamada após cada episódio de treinamento, a fim de incrementar a memória $\mathbb{B}$ com transições retrospectivas[10].

Algoritmo 6 HER

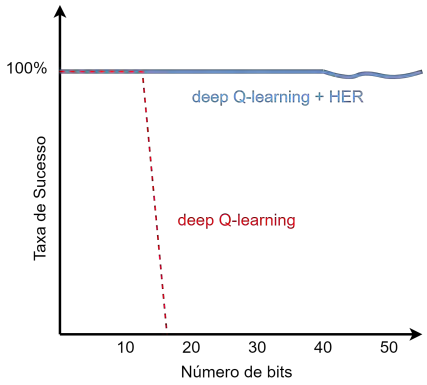
requerer: $\mathbb{B}$: memória (*replay buffer*)
requerer: $\Gamma = \{s_0, a_0, s_1, r_1, a_1, \ldots, s_T, R_T\}$: trajetória de um episódio

1: **procedimento** AdicionarTransiçõesRetrospectivas($\mathbb{B}, \Gamma$)
2: $\quad G \leftarrow \emptyset$
3: $\quad$ **para** $t = 0 \ldots H$ **faça**
4: $\quad\quad \bar{s} \sim \{s_t, \ldots, s_H\}$ ▷ Amostra um objetivo atingido após instante t
5: $\quad\quad \bar{g} \leftarrow m(\bar{s})$
6: $\quad\quad \bar{r} \leftarrow r_{\bar{g}}(s_t, a_t)$ ▷ Recompensa supondo objetivo desejado $\bar{g}$
7: $\quad\quad \bar{\Omega} \leftarrow \{s_t, a_t, r_t, s_{t+1}\}$
8: $\quad\quad \mathbb{B} \leftarrow \mathbb{B} \cup \{\bar{\Omega}\}$
9: $\quad$ **fim para**
10: $\quad$ **retorna** $\mathbb{B}$
11: **fim procedimento**

Um dos experimentos realizados por Andrychowicz et al. (2017) para verificar o desempenho do HER envolve uma tarefa em que o agente altera individualmente um bit dentre uma sequência de n bits, buscando igualar uma outra sequência, selecionada no início do episódio. A recompensa é esparsa e ocorre apenas quando a sequência alvo é atingida. Para $n > 20$, algoritmos como o *deep Q-learning* falham, pois o espaço de estados é amplo e o agente obtém apenas recompensas nulas, dada a incapacidade de explorar o espaço de estados. Quando o algoritmo *deep Q-learning* é combinado com o HER, a tarefa pode ser resolvida para valores maiores de n, como indica a Figura 41.

[10] A estratégia de amostragem de objetivos atingidos durante o episódio, descrita na Linha 4 do Algoritmo 6, é chamada de *future*, pois seleciona apenas objetivos atingidos após o instante t (ANDRYCHOWICZ et al., 2017).

Figura 41 - Algoritmo HER no problema *bit-flip*



Fonte: Adaptado de Andrychowicz et al. (2017).

O ganho de desempenho com a utilização do algoritmo HER foi significativo em problemas de robótica (Figura 42), já que tipicamente esse tipo de tarefa demanda funções de recompensa esparsas, dada a dificuldade de projetar uma função de recompensa informativa, especialmente em sistemas não-holonômicos.

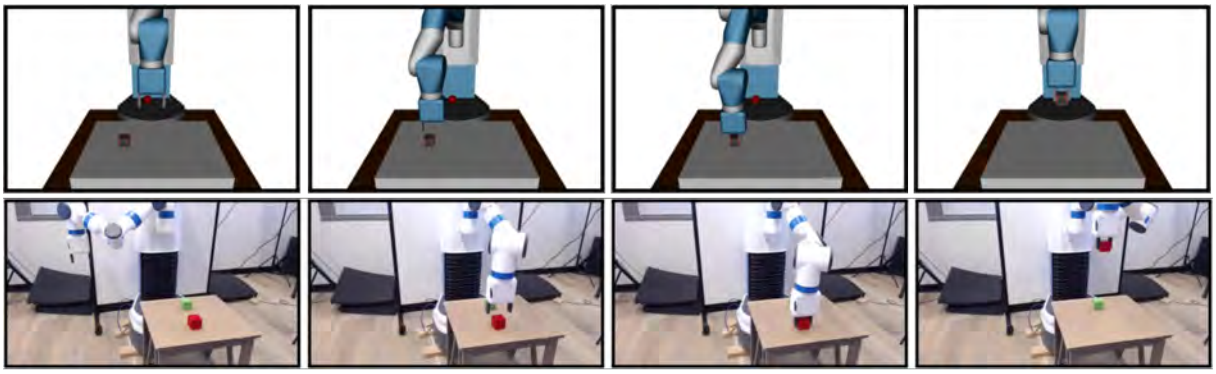
Apesar dos resultados promissores obtidos com o algoritmo HER, a taxa de sucesso diminui à medida que a distância entre o estado inicial do agente e o objetivo desejado aumenta. Esta limitação é conhecida como o problema do horizonte longo. A seção seguinte apresenta o método proposto por Pitis et al. (2020) para lidar com essa dificuldade.

3.7 **Maximização da Entropia de Objetivos Atingidos**

Observa-se que para algoritmos *off-policy* as transições contidas no *replay buffer* determinam o aprendizado do agente, pois o processo de otimização é realizado com amostras dessa memória. Com isso, para que o agente aprenda a alcançar um determinado objetivo g_1, é necessário atingir, em algum momento do treinamento, objetivos próximos a g_1. Isso fica evidente no algoritmo HER, pois as transições criadas artificialmente são compostas apenas por objetivos alcançados durante o episódio. O problema do horizonte longo ocorre pois objetivos distantes não são encontrados por meio ações exploratórias simples, como ilustra o Exemplo 9, e, portanto, o agente não aprende a alcançar estes objetivos.

Seja p_{ag} a distribuição empírica de objetivos alcançados durante toda a fase de treinamento, isto é, p_{ag} representa o histórico do agente. Denota-se por p_{dg} a distribuição de objetivos distantes desejados. Para resolver o problema do horizonte longo, é necessário que haja intersecção entre essas duas distribuições, já que apenas alcançando os objetivos

Figura 42 - Transferência de política com o algoritmo HER

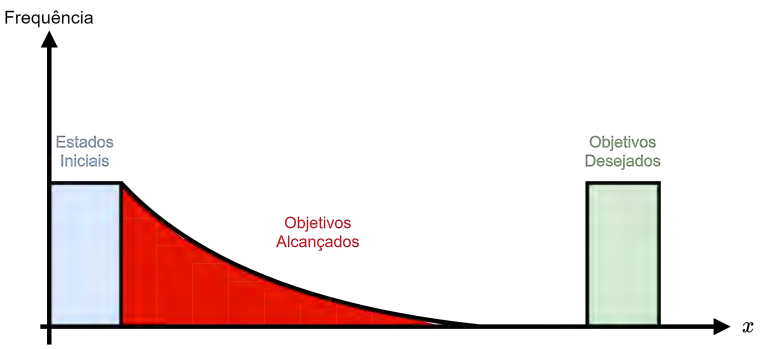


Legenda: Na tarefa *pick-and-place*, o robô deve capturar o bloco e levá-lo até uma posição específica. Na fileira *superior*, a fase de treinamento do agente em um simulador. A política é transferida para o robô real, como mostra a fileira *inferior*. A tarefa só pôde ser concluída com sucesso quando o algoritmo DDPG é utilizado junto com o HER.

Fonte: Adaptado de Andrychowicz et al. (2017).

Exemplo 9 *Suponha um problema em que tanto o estado quanto o objetivo são unidimensionais, de forma que* $s = g = x$*. O agente pode realizar deslocamentos instantâneos ao longo do semi-eixo* $x > 0$*. Deseja-se utilizar o algoritmo HER para alcançar os objetivos situados na região verde, indicada na Figura 43. O agente inicia um episódio em algum ponto da região azul e explora o ambiente a partir desta posição. Quanto maior a distância do objetivo em relação aos estados iniciais, menor a frequência de visitação do agente. Isto ocorre pois é improvável visitar um estado tão distante através de ações exploratórias, que são selecionadas aleatoriamente. Pode-se explicar essa afirmação por meio de uma analogia: suponha que a ação exploratória do agente seja escolhida ao jogar uma moeda, de forma que o agente se movimenta para direita em uma unidade quando coroa ou para a esquerda caso contrário. A chance de visitar o estado* $x = 10$*, partindo de* $x = 0$*, em* 10 *instantes de tempo, é* $0,5^{10} \approx 0,1\%$*.*

Figura 43 - Problema de horizonte longo



Fonte: O autor, 2022.

distantes é possível aprender a obtê-los. O objetivo do algoritmo MEGA é, portanto, tornar a distribuição p_{ag} similar a p_{dg}. Esse propósito pode ser alcançado ao minimizar a divergência KL entre p_{ag} e p_{dg}, porém esta medida de dispersão é assimétrica e, em geral, $D_{KL}\left(p_{dg} \mid\mid p_{ag}\right) \neq D_{KL}\left(p_{ag} \mid\mid p_{dg}\right)$. Resta, portanto, definir qual expressão deve ser utilizada.

Nota-se que a expressão $J(p_{ag}) = D_{KL}\left(p_{ag} \mid\mid p_{dg}\right)$, chamada de divergência *reversa*, não pode ser utilizada como função de custo em um processo de otimização, uma vez que o seu valor é sempre infinito. Isto ocorre pois p_{dg} não pode cobrir o suporte de p_{ag}, uma vez que os estados iniciais são desconectados dos objetivos desejados, em problemas de horizonte longo. Dessa forma, para qualquer objetivo g_0 atingido no início do episódio, têm-se que $p_{ag}(g_0) > 0$ e $p_{dg}(g_0) = 0$, logo:

$$J(p_{ag}) = D_{KL}\left(p_{ag} \mid\mid p_{dg}\right) = \sum_{g \in \mathcal{G}} p_{ag}(g) \log\left(\frac{p_{ag}(g)}{p_{dg}(g)}\right) \tag{108a}$$

$$= \sum_{g \in \mathcal{G}} p_{ag}(g) \log\left(\frac{p_{ag}(g)}{0}\right) \tag{108b}$$

$$= \infty^{11} \tag{108c}$$

O mesmo ocorre para a divergência *direta*, $D_{KL}\left(p_{dg} \mid\mid p_{ag}\right)$, uma vez que p_{ag} não cobre o suporte de p_{dg}, como ilustra a Figura 43. O algoritmo MEGA propõe expandir o suporte de p_{ag} até o ponto em que a divergência *direta* apresente um valor finito, quando há sobreposição entre as distribuições p_{ag} e p_{dg}. A expansão do suporte de p_{ag} pode ser obtida ao maximizar a entropia desta distribuição, como ilustra a Figura 44. Intuitivamente, para uma quantidade fixa de objetivos atingidos em p_{ag}, quanto maior o número de possibilidades, isto é, o suporte da distribuição, maior a incerteza (entropia) acerca da variável aleatória correspondente. Esse critério é exposto na Equação 109:

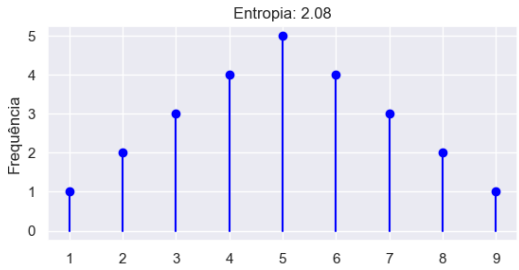
$$J_{\text{MEGA}}(p_{ag}) = -H\left[p_{ag}\right] \tag{109}$$

O algoritmo MEGA propõe escolher o objetivo comportamental g_b de forma que os objetivos atingidos durante o episódio, quando adicionados ao histórico p_{ag}, minimizem a Equação 109. Existe uma distribuição $q_a(g_a|g_b)$ desconhecida, que indica a probabilidade de atingir um objetivo g_a durante o episódio, quando a política do agente está condicionada em g_b. Dessa forma, a função objetivo pode ser escrita em termos do valor esperado da entropia de $p_{ag|g_a}$, onde $p_{ag|g_a}$ representa a concatenação dos objetivos atingidos no

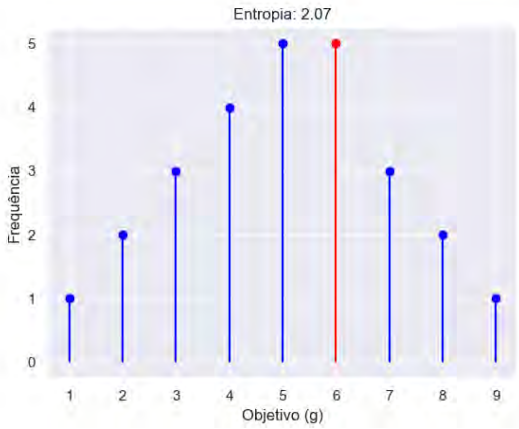
[11] No sentido estrito, $\log(0)$ é indefinido, mas a definição de divergência KL utiliza a convenção (THOMAS; JOY, 2006): $p(x)\log(p(x)/0) = \infty$.

Figura 44 - Comparação do ganho de entropia

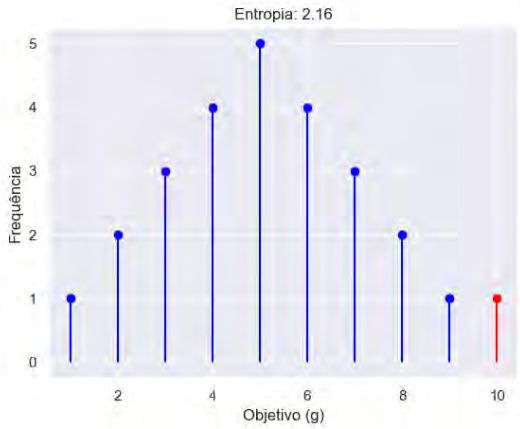
(a)



(b)



(c)



Legenda: (a) - a distribuição p_{ag} com 25 objetivos alcançados; (b) e (c) - apresentam a entropia de p_{ag} quando um determinado objetivo (indicado em vermelho) é adicionado ao histórico do agente. O objetivo atingido em (c) garante o maior ganho de entropia e, de fato, estende o suporte de p_{ag}.

Fonte: O Autor, 2022.

episódio com a distribuição empírica p_{ag}:

$$J_{\text{MEGA}}(p_{ag|g_a}) = \mathbb{E}_{g_a \sim q_a(g_a|g_b)} - H\left[p_{ag|g_a}\right] \tag{110}$$

A melhor escolha para o objetivo comportamental é, então:

$$g_b^* = \arg\max_{g_b \in \mathcal{B}} \mathbb{E}_{g_a \sim q_a(g_a|g_b)} H\left[p_{ag|g_a}\right] \tag{111}$$

Pitis et al. (2020) mostram que a maximização apresentada na Equação acima é equivalente a maximização do ganho de entropia:

$$g_b^* = \arg\max_{g_b \in \mathbb{B}} \mathbb{E}_{g_a \sim q_a(g_a|g_b)} \Delta H(g_a) \tag{112a}$$

$$= \arg\max_{g_b \in \mathbb{B}} \mathbb{E}_{g_a \sim q_a(g_a|g_b)} \left[p_{ag}(g_a) \log p_{ag}(g_a) - (p_{ag}(g_a) + \eta_b) \log(p_{ag}(g_a) + \eta_b)\right] \tag{112b}$$

Outro resultado importante é que conforme $\eta_b = (1/|\mathbb{B}|) \to 0$, ou seja, o número de experiências do *replay buffer* tende ao infinito, a Equação 112 se torna:

$$\lim_{\eta_b \to 0} g_b^* = \arg\max_{g_b \in \mathbb{B}} D_{KL}\left(q_a(g_a \mid g_b) \mid\mid p_{ag}\right) + H[q_a(g_a \mid g_b)] \tag{113a}$$

A Equação 113 tem um sentido intuitivo, pois busca-se um objetivo comportamental g_b que:

a) Induza uma distribuição de objetivos atingidos diferente do histórico do agente (parcela da divergência KL);

b) Leve o agente a atingir objetivos diversificados (entropia de $q_a(g_a|g_b)$).

Em outras palavras, busca-se um g_b que condicione a política e leve o agente à regiões nunca exploradas. Além disso, g_b deve ser tal que os objetivos atingidos nessas regiões sejam diversos, caracterizando um comportamento exploratório em que o espaço de objetivos é experimentado igualmente em todas as direções.

A distribuição $q_a(g_a|g_b)$ é desconhecida, porém, é possível obter uma aproximação através de dois métodos:

a) utilizar um modelo probabilístico, por exemplo, KDE (*kernel density estimator* ou janelas de *Parzen*) para estimar a distribuição;

b) empregar uma heurística para definir uma distribuição aproximada, sob determinadas condições.

Os melhores resultados experimentais obtidos por Pitis et al. (2020) utilizam um método heurístico, descrito a seguir. Supondo que a distribuição q_a seja tal que o objetivo desejado (comportamental) sempre é atingido, têm-se que:

$$q_a(g_a|g_b) = \mathbb{1}\left[g_a = g_b\right] \tag{114}$$

A suposição acima necessita que o agente seja capaz de atingir o objetivo g_a. Isso pode ser garantido por dois mecanismos: os objetivos comportamentais candidatos são amostrados do *replay buffer* (o agente já atingiu esse objetivo no passado) e apenas objetivos com valor (de acordo com a função Q estimada) maior que um limiar dinâmico são selecionados. Este limite é calculado com base na taxa de sucesso em alcançar objetivos durante o treinamento. Substituindo a suposição da Equação 114 na Equação 113:

$$\lim_{\eta_b \to 0} g_b^* = \arg\max_{g_b \in \mathbb{B}} D_{KL}\left(q_a(g_a \mid g_b) \,||\, p_{ag}\right) + H[q_a(g_a|g_b)] \tag{115a}$$

$$= \arg\max_{g_b \in \mathbb{B}} D_{KL}\left(\mathbb{1}\left[g_a = g_b\right] \,||\, p_{ag}\right) + H[\mathbb{1}\left[g_a = g_b\right]] \tag{115b}$$

$$= \arg\max_{g_b \in \mathbb{B}} D_{KL}\left(\mathbb{1}\left[g_a = g_b\right] \,||\, p_{ag}\right) + 0 \tag{115c}$$

$$= \arg\max_{g_b \in \mathbb{B}} \sum_{g_a \in G} \mathbb{1}\left[g_a = g_b\right]\left(\log \mathbb{1}\left[g_a = g_b\right] - \log p_{ag}(g_a)\right) \tag{115d}$$

$$= \arg\max_{g_b \in \mathbb{B}} \log 1 - \log p_{ag}(g_b) \tag{115e}$$

$$= \arg\min_{g_b \in \mathbb{B}} \log p_{ag}(g_b) \tag{115f}$$

A Equação acima sugere a escolha de um objetivo comportamental que apresente a menor probabilidade de ocorrência, sob a perspectiva do histórico de objetivos atingidos, p_{ag}. Com isso, torna-se necessário obter um modelo para estimar a distribuição p_{ag}. Pitis et al. (2020) utilizam o KDE para obter este modelo.

Quando o objetivo selecionado é atingido, os mecanismos de exploração (como, por exemplo, a adição de ruído às ações) são aumentados, com o intuito de expandir a fronteira de objetivos atingidos no passado. Essa estratégia heurística é similar ao algoritmo *Go Explore* (ECOFFET et al., 2019), que atingiu o estado da arte em alguns problemas complexos de aprendizado por reforço.

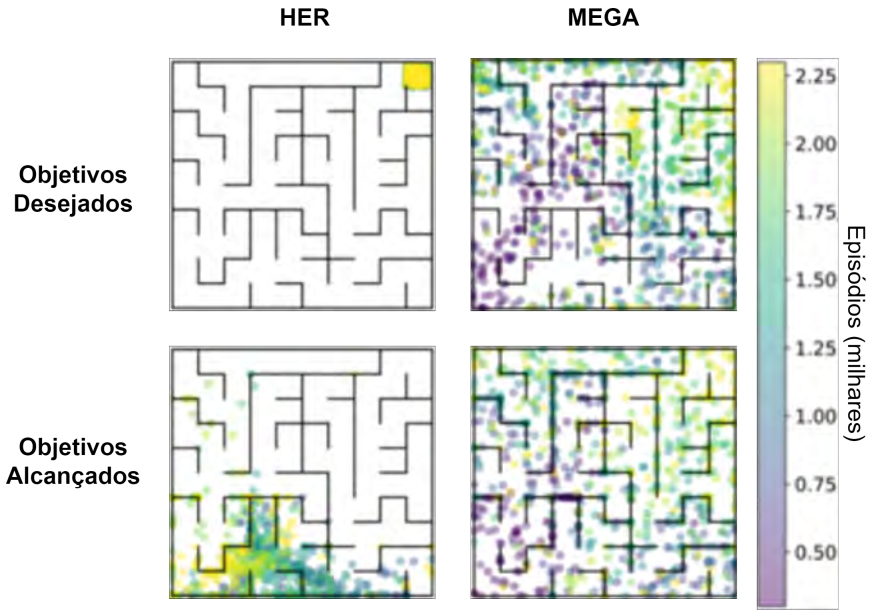
O algoritmo MEGA também utiliza um mecanismo heurístico diferenciado para obter as transições "artificiais", de forma que, para cada transição real são armazenadas no *replay buffer*:

a) 4 transições com objetivos alcançados durante o episódio (como no algoritmo HER);

b) 3 transições com alguns objetivos desejados, amostrados de p_{dg};

c) 1 transição com objetivo alcançado durante todo o histórico do agente, isto é, amos-

Figura 45 - Comparação dos algoritmos HER e MEGA



Legenda: Objetivos *comportamentais* e *atingidos* ao longo dos episódios, em um problema de navegação em um labirinto. O robô não tem informações sobre os obstáculos, o que dificulta a exploração do espaço. O algoritmo HER utiliza uma distribuição externa de objetivos desejados e não atinge posições distantes. Em contrapartida, o algoritmo MEGA escolhe os próprios objetivos comportamentais e gradualmente atinge regiões mais distantes.

Fonte: Adaptado de Pitis et al. (2020).

trado de p_{ag};

d) 1 transição aleatória contendo um objetivo comportamental aleatório g_b, utilizado nos episódios passados.

Outras quantidades, para cada tipo de transição artificial, podem ser utilizadas, porém, os melhores resultados experimentais em (PITIS et al., 2020) foram obtidos com a configuração apresentada na alínea anterior. A Figura 45 apresenta um desses resultados.

4 MÉTODOS

Esta seção apresenta a metodologia aplicada nesta Dissertação para resolver o problema de navegação autônoma de robôs móveis, em ambientes cuja localização dos obstáculos é conhecida. É importante observar que o conhecimento da posição dos obstáculos é uma etapa fundamental anterior ao planejamento e seguimento da trajetória. Neste trabalho, assume-se que o critério de eficiência para as trajetórias é o tempo total do percurso entre a configurações inicial e a final. O método proposto busca unificar diversas técnicas de aprendizado de máquinas para otimizar o algoritmo RRT* e prover uma maneira sistemática de aplicar esse algoritmo em sistemas não-holonômicos como CLMRs, evitando heurísticas que exigem conhecimento profundo do sistema.

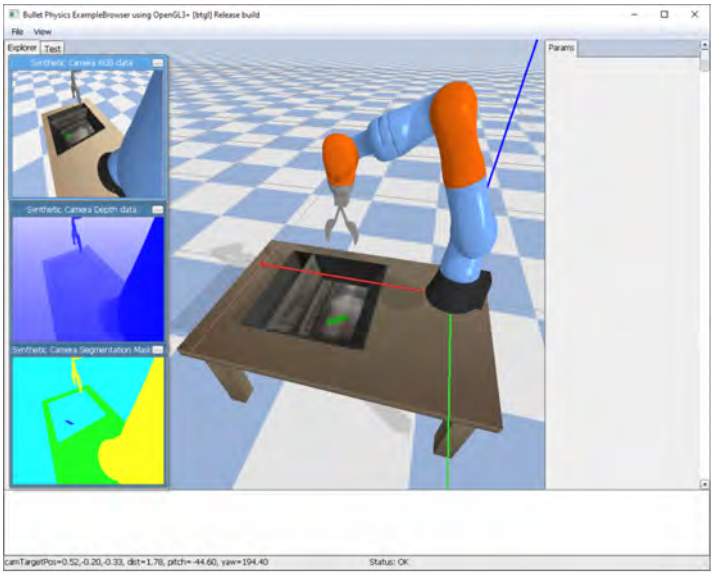
A Seção 4.1 aborda a simulação do sistema dinâmico, um componente fundamental do algoritmo RRT*. A Seção 4.2 discute a elaboração do *módulo de controle*, isto é, um controlador local responsável por guiar o robô de uma configuração à outra. A Seção 4.3 aborda o que neste trabalho é chamado de *módulo de planejamento* e discute a implementação do modelos generativo β-CVAE para a geração de amostras de configurações, visando aumentar a eficiência do RRT*. A Seção 4.4 apresenta um procedimento sistemático para implementação do método proposto.

4.1 Ambiente de Simulação

Este trabalho utiliza o programa PyBullet (COUMANS; BAI, 2016) como simulador do sistema dinâmico. Este software é caracterizado como um simulador de física, pois busca reproduzir os diversos fenômenos mecânicos, como inércia e atrito, que ocorrem no mundo real. *A s*imulação do robô é fundamental para o RRT* mas também é utilizada, nesta Dissertação, para o desenvolvimento do controlador, por meio do aprendizado por reforço. O PyBullet foi utilizado em diversos trabalhos recentes que aplicam o aprendizado por reforço em problemas de robótica (TAN et al., 2018; PENG et al., 2018; MATAS; JAMES; DAVISON, 2018). Dentre as vantagens desse simulador destacam-se a utilização da linguagem de programação *Python*, uma das mais populares no campo de aprendizado de máquinas atualmente (RASCHKA; PATTERSON; NOLET, 2020), o fato de ser gratuito e a capacidade de simular robôs que captam imagens (Figura 46).

O uso de simulações em problemas de aprendizado por reforço é fundamental. Mesmo que seja possível treinar agentes em plataformas robóticas reais, diversas limitações dificultam essa alternativa, dentre as quais pode-se citar as restrições de segurança que devem ser impostas durante o treinamento e a maior dificuldade em obter uma grande quantidade de dados (Dulac-Arnold et al., 2021). É possível contornar essas dificulda-

Figura 46 - Pybullet e a simulação de captura de imagens.



Fonte: O autor, 2022.

Tabela 1 - Parâmetros do CLMR no PyBullet

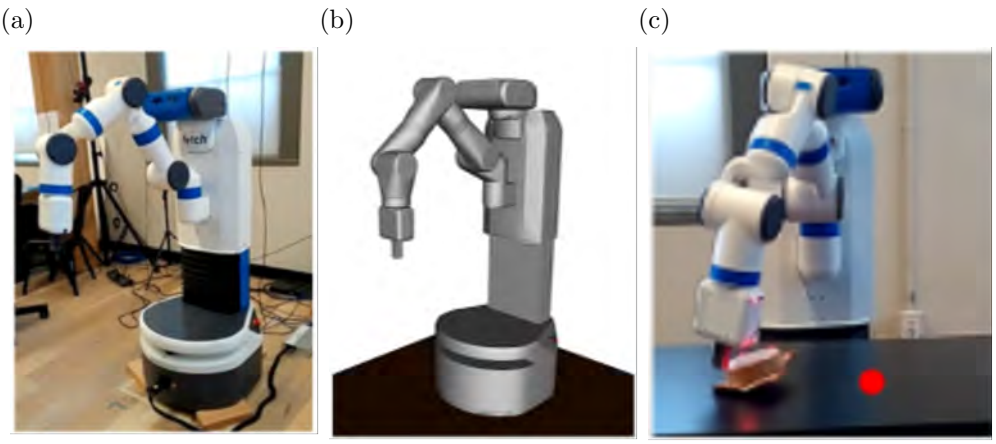
Parâmetro	Valor
Comprimento (m)	0,46
Largura (m)	0,30
Velocidade máxima (m/s)	1,06
Aceleração máxima (m/s^2)	0,11
Angulação máxima das rodas dianteiras (rad)	0,70

Fonte: O autor, 2022

des ao realizar a fase de treinamento em simuladores. Evidentemente, o ambiente de simulação não corresponderá, com extrema fidelidade, ao ambiente real em que deseja-se utilizar o robô, dessa forma, haverá incoerências na atuação do agente no domínio real. Porém, algoritmos como *domain randomization* auxiliam na correspondência coerente entre os ambientes de simulação e o real. Essa transferência de política foi realizada com sucesso por Peng et al. (2018), com um robô treinado exclusivamente através do simulador MuJoCo (TODOROV; EREZ; TASSA, 2012), como mostra a Figura 47.

O robô móvel utilizado como estudo de caso neste trabalho é um CLMR, cujo modelo no PyBullet é apresentado na Figura 48. Algumas características adicionais desse modelo são apresentadas na Tabela 1. Todas as rodas do veículo possuem um motor de tração. Cada uma das rodas dianteiras conta com um motor adicional, responsável pela alteração do ângulo da mesma em relação ao eixo longitudinal do robô, o que permite a mudança de direção do veículo. O PyBullet simula o funcionamento de motores de corrente contínua regulados por controladores de ação proporcional derivativa (JOHNSON;

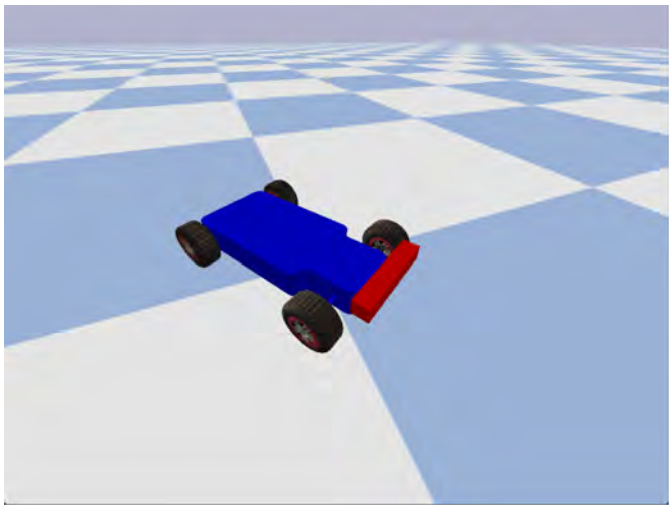
Figura 47 - Transferência de política entre domínios



Legenda: Braço robótico *fetch*, com 7 graus de liberdade. (a) - o robô real; (b) - o modelo do robô no simulador MuJoCo. (c) - O robô em uma tarefa real cujo objetivo era empurrar um disco até uma certa posição. A taxa de sucesso da política foi similar entre o domínio real (91%) e a simulação (89%).

Fonte: Peng et al. (2018).

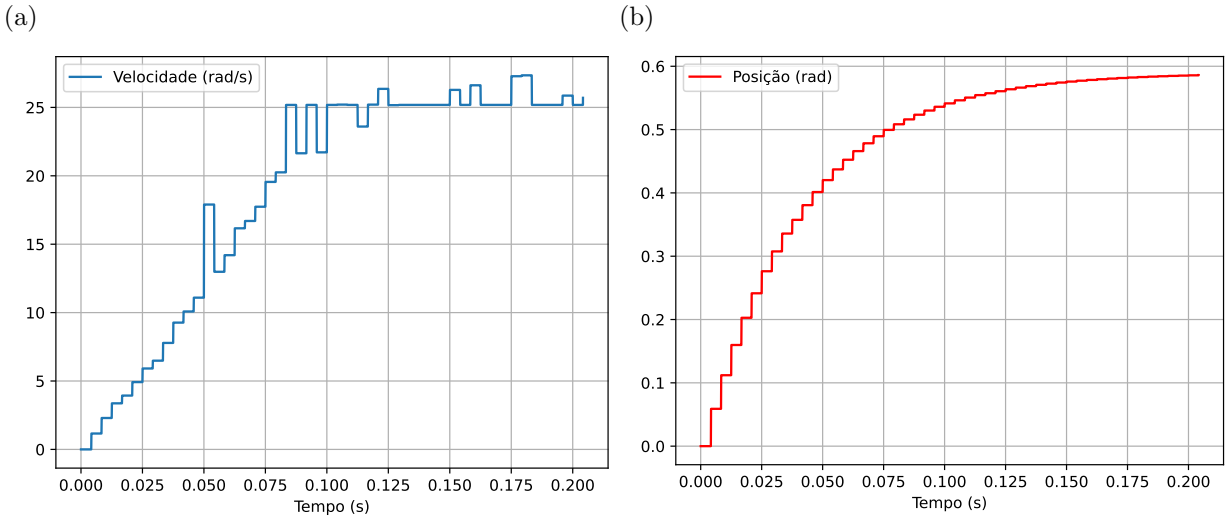
Figura 48 - Modelo do CLMR no PyBullet



Legenda: A parte frontal do veículo é indicada pelo retângulo vermelho.

Fonte: O autor, 2022.

Figura 49 - Resposta dos motores a um sinal degrau



Legenda: (a) - resposta do motor de tração. (b) - resposta do motor de direcionamento.
Fonte: O autor, 2022.

MORADI, 2005), de modo que a referência do controlador (valor desejado) é fornecida pelo usuário. Dois tipos de referência podem ser utilizados: velocidade ou posição. Isto significa que o usuário pode definir um valor desejado de velocidade (linear ou angular) para a junta ou uma determinada posição (linear ou angular) para a mesma. Neste trabalho, um sinal de controle ϕ_a determina as posições angulares de referência (ϕ_{ref_1} e ϕ_{ref_2}) dos motores das rodas dianteiras responsáveis pelo direcionamento do veículo. Um outro sinal de controle, v_a, é utilizado para determinar as velocidades angulares de referência (v_{ref_1}, v_{ref_2}, v_{ref_3} e v_{ref_4}) para os motores de tração em cada uma das rodas. Todos os valores de referência são calculados com base na geometria de Ackermann (MITCHELL; STANIFORTH; SCOTT, 2006). A Figura 49 apresenta as respostas dos motores a um sinal de referência com máxima amplitude.

O PyBullet utiliza um intervalo de tempo $\Delta t = 4.16\,\text{ms}$ para simular as diversas forças que atuam no robô, por exemplo, o torque do motor, as forças de atrito e a gravidade. Observa-se na Figura 49 que os valores de referência são atingidos rapidamente.

Os experimentos com o RRT* são realizados em uma arena quadrada com $100\,\text{m}^2$. O robô deve percorrer de um lado ao outro, através de duas passagens estreitas, como indica a Figura 50. Considerando que a largura da passagem estreita não muda, é possível caracterizar qualquer problema de planejamento aleatório, neste cenário, através de um

vetor de condições $\vec{y}$, tal que:

$$\vec{y} = [\,\vec{g_i} \;\|\; \vec{g_f} \;\|\; \vec{o}\,] \tag{116a}$$

$$\vec{g_i} = [\,x_i \;\; y_i \;\; \text{sen}(\theta_i) \;\; \cos(\theta_i)\,] \qquad \begin{cases} 0{,}8 \leq x_i \leq 1{,}2 \\ 2{,}0 \leq y_i \leq 7{,}0 \\ 0 \leq \theta_i \leq 2\pi \end{cases} \tag{116b}$$

$$\vec{g_f} = [\,x_f \;\; y_f \;\; \text{sen}(\theta_f) \;\; \cos(\theta_f)\,] \qquad \begin{cases} 8{,}8 \leq x_f \leq 9{,}2 \\ 2{,}0 \leq y_f \leq 9{,}0 \\ 0 \leq \theta_f \leq 2\pi \end{cases} \tag{116c}$$

$$\vec{o} = [\,x_{pass1} \;\; y_{pass1} \;\; x_{pass2} \;\; y_{pass2}\,] \qquad \begin{cases} 6{,}0 \leq x_{pass1} \leq 7{,}0 \\ 2{,}0 \leq y_{pass1} \leq 7{,}0 \\ 6{,}0 \leq x_{pass2} \leq 7{,}0 \\ 2{,}0 \leq y_{pass2} \leq 7{,}0 \end{cases} \tag{116d}$$

A Figura 51 apresenta uma ilustração das faixa de valores possíveis, descritas na Equação 116, para as configurações iniciais e finais, além das posições permitidas para as passagens estreitas. Observa-se que a codificação escolhida para os obstáculos contém apenas as coordenadas x e y de cada passagem e que não há restrição na orientação das configurações iniciais e finais.

4.2 **Módulo de Controle**

A implementação de um controlador e a definição de uma métrica de distância para o RRT* podem ser etapas complexas, especialmente para sistemas não-holonômicos como CLMRs. Para exemplificar essas dificuldades, dá-se atenção aos passos relacionados à adição de uma nova configuração $\vec{g}_{nova}$ à árvore de exploração:

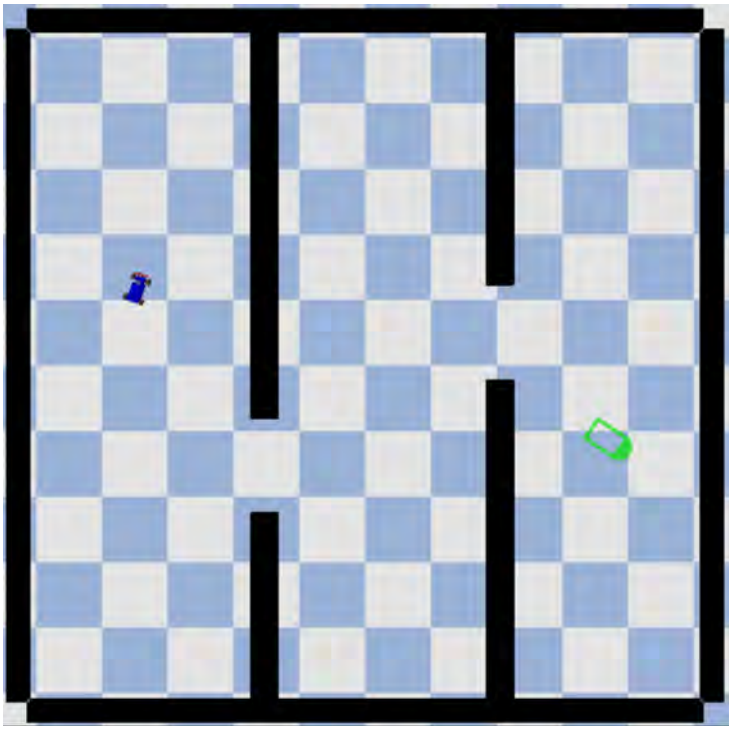
a) uma configuração $\vec{g}_{aleatoria}$ é amostrada uniformemente (ou com a utilização de um modelo generativo);

b) verifica-se a configuração $\vec{g}_{prox} \in \mathcal{V}$ mais próxima de $\vec{g}_{aleatoria}$, de acordo com uma métrica ρ, que representa o custo da trajetória ótima entre duas configurações, supondo que não há obstáculos;

c) uma simulação é realizada com um controlador $\mathcal{U}$, visando minimizar essa distância entre $\vec{g}_{prox}$ e $\vec{g}_{aleatoria}$;

d) $\vec{g}_{nova}$ é a configuração final da simulação, após Δt instantes de tempo;

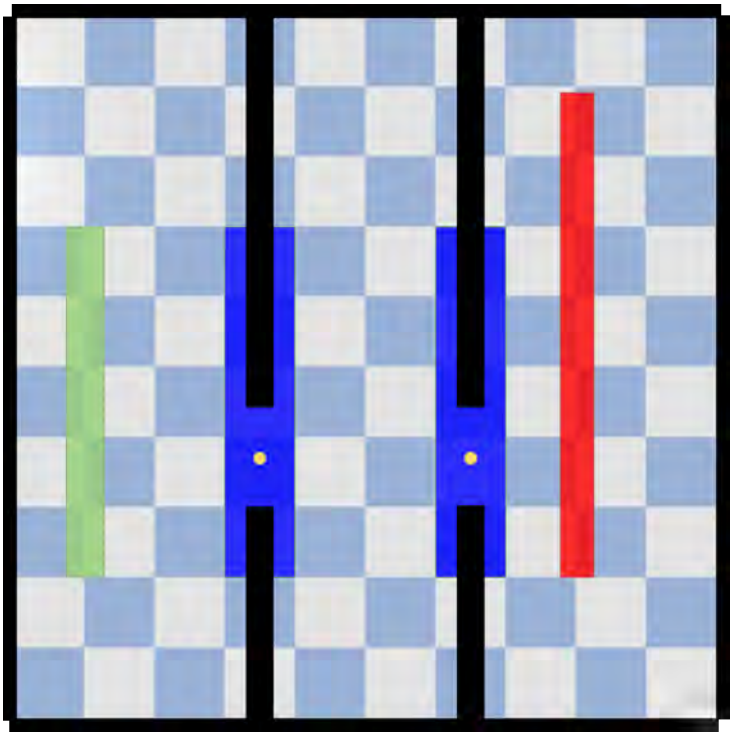
Figura 50 - Mapa com duas passagens estreitas



Legenda: O robô deve alcançar o objeto verde, que representa uma determinada posição e orientação desejada, evitando a colisão com os obstáculos, representados pelos retângulos pretos.

Fonte: O autor, 2022.

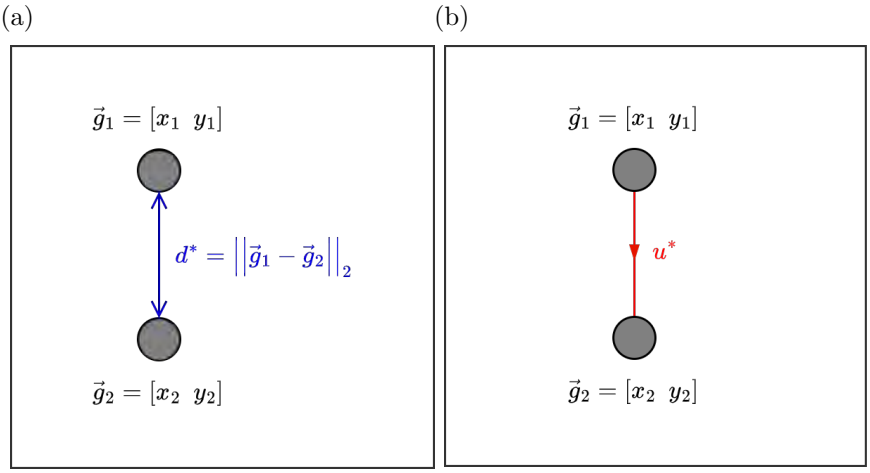
Figura 51 - Características do problema de planejamento



Legenda: A região em *verde* indica a posição inicial do robô. As regiões *azuis* representam as possíveis posições do centro da passagem estreita (indicado pelo círculo *amarelo*). A região *vermelha* indica a posição do estado que deseja-se alcançar. Cada quadrado branco e azul possui $1\,\mathrm{m}^2$.

Fonte: O autor, 2022.

Figura 52 - Métrica e sinal de controle para um robô pontual



Fonte: O autor, 2022.

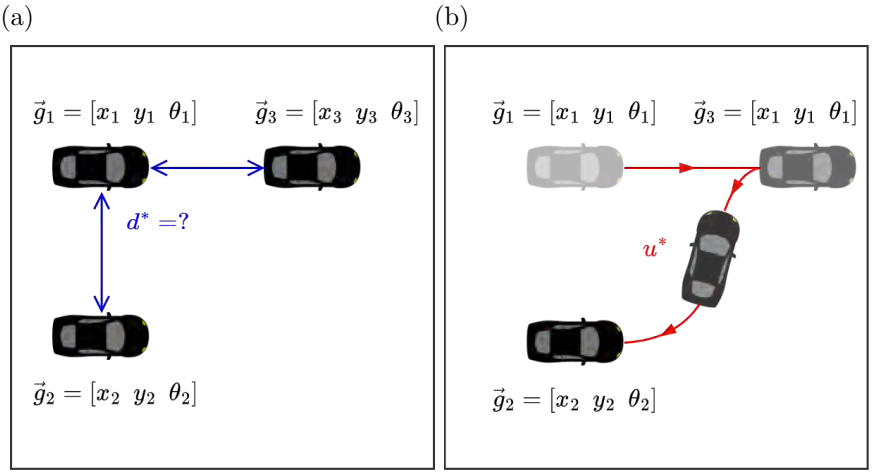
Os seguintes questionamentos podem surgir com relação às etapas enunciadas: qual é a métrica relacionada à trajetória ótima? Como obter um controlador que minimiza esta métrica?

Supondo que o custo de uma trajetória é a distância total percorrida, a métrica ρ deve ser tal que o sinal de controle que minimize ρ induza um percurso ótimo em relação ao comprimento do caminho. Nessas condições, a distância euclidiana é apropriada para o robô da Figura 52a, que pode se locomover em qualquer direção. De fato, a minimização da distância é obtida com a movimentação na direção e sentido indicados na Figura 52b, o que claramente constitui a melhor forma de atuação.

Em contrapartida, para a situação da Figura 53a, onde o robô é um CLMR, a métrica euclidiana sugere que a distância de $\vec{g_1}$ até $\vec{g_2}$ e $\vec{g_3}$ são iguais. Considerando as restrições usuais de um automóvel de passeio, é possível notar que alcançar a configuração $\vec{g_2}$ demanda um percurso mais extenso. Pode-se imaginar que o melhor percurso é algo similar ao que mostra a Figura 53b e, neste caso, um controlador ótimo que realiza esse percurso não minimiza a métrica euclidiana e sim uma outra função desconhecida. De certa forma, esta métrica desconhecida, mas que está associada à um controlador ótimo que a minimiza, é a função de distância "real" do problema. Para o robô da Figura 52 a norma euclidiana representa a distância real entre as configurações, considerando o custo como o comprimento do percurso. Para sistemas não-holonômicos, como o CLMR da Figura 53, a distância "real" é desconhecida e depende dos diversos parâmetros do robô, como o ângulo máximo de orientação das rodas, por exemplo.

É possível notar a similaridade do problema de encontrar uma função de distância e o controlador ótimo correspondente com o problema de aprendizado por reforço. Mais especificamente, a distância "real" tem relação direta com a função valor ótima, V^*, assim

Figura 53 - Métrica e sinal de controle para um CLMR



Fonte: O autor, 2022.

como o controlador ótimo associado a essa distância representa a política ótima π^*. Da mesma forma, o controlador ótimo seleciona as ações que minimizam a distância "real", enquanto a política ótima seleciona a ação que maximiza o valor do próximo estado.

Assim sendo, este trabalho propõe encontrar um controlador local para o CLMR a partir do aprendizado por reforço com múltiplos objetivos. É importante observar, contudo, que esse controlador não considera a existência de obstáculos e busca apenas atingir a configuração próxima desejada de maneira eficiente.

É possível descrever o movimento de CLMRs em baixas velocidades (POLACK et al., 2017) a partir do modelo cinemático de bicicleta, apresentado na Equação 117. As variáveis deste modelo podem ser observadas na Figura 54.

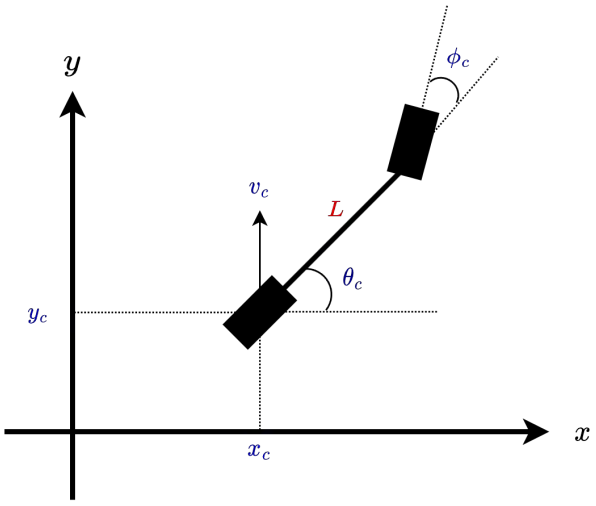
$$\dot{x}_c = v_c \cos(\theta_c) \tag{117a}$$

$$\dot{y}_c = v_c \sin(\theta_c) \tag{117b}$$

$$\dot{\theta}_c = \frac{v_c \tan(\phi_c)}{L} \tag{117c}$$

No modelo cinemático de bicicleta as variáveis de controle v e ϕ indicam os valores desejados de velocidade e a orientação do eixo frontal do veículo, respectivamente. Dessa forma, as variáveis de entrada v e ϕ podem ser modeladas como ações $a = [\,v_a\ \phi_a\,]$ que o agente realiza no ambiente e que correspondem, por sua vez, aos sinais utilizados para calcular os valores de referência fornecidos a cada motor. Se o sistema obedece a Equação 117, e o estado s observado pelo agente contém as variáveis x_c, y_c, θ_c, v_c e ϕ_c; esse estado contempla a propriedade de Markov. Já que os estados e objetivos são entradas para as redes neurais, a componente θ desses vetores é transformada em duas entradas adicionais, correspondendo ao seno e cosseno desse ângulo, dessa forma, evita-se

Figura 54 - Variáveis do modelo cinemático de bicicleta



Legenda: No modelo cinemático de bicicleta, as rodas dianteiras e traseiras, representadas pelos retângulos, têm os seus centros unidos em um único ponto no espaço. A constante L indica a distância entre esses pontos. Assume-se também, neste trabalho, que a distância do centro de gravidade às rodas traseiras e dianteiras são iguais.

Fonte: O autor, 2022.

a utilização de entradas cíclicas. Em suma[12]:

$$s = [\, x_c \;\; y_c \;\; \text{sen}(\theta_c) \;\; \cos(\theta_c) \;\; v_c \;\; \phi_c \,] \tag{118a}$$

$$a = [\, v_a \;\; \phi_a \,] \tag{118b}$$

É importante notar que o agente deve ser capaz de atingir qualquer configuração (posição e orientação) próxima de sua localização atual. A configuração que deseja-se alcançar corresponde a um objetivo g_b, de forma que, o mapeamento de estados para objetivos alcançados é definido como:

$$m : [\, x_c \;\; y_c \;\; \text{sen}(\theta_c) \;\; \cos(\theta_c) \;\; v_c \;\; \phi_c \,] \rightarrow [\, x_c \;\; y_c \;\; \text{sen}(\theta_c) \;\; \cos(\theta_c) \,] \tag{119}$$

Conforme a discussão da Seção 3.6, a formulação do problema com múltiplos objetivos viabiliza a utilização de funções de recompensa esparsas, o que é desejável, pois a criação de métricas de recompensa informativas podem resultar em políticas sub-ótimas. Já que o robô deve alcançar o objetivo no menor tempo possível, pode-se utilizar a seguinte função de recompensa binária:

$$r_g(s,a) = \begin{cases} -1 & d(s',g) > \epsilon_r \\ 0 & d(s',g) \leq \epsilon_r \end{cases} \tag{120}$$

Onde d é uma norma calculada a partir da diferença ponderada entre as componentes do estado atingido e do objetivo desejado. Sejam x_g, y_g e θ_g as componentes x_c, y_c e θ_c de uma configuração g desejada; então, a distância d é calculada a partir da Equação 121. O parâmetro ϵ_r representa uma distância mínima para que se possa considerar que o objetivo desejado foi alcançado. O valor exato destes parâmetros e outros apresentados nessa seção encontram-se na Tabela 2.

$$\begin{aligned} d(s,g) = \Big(& w_{pos}\left(|x_c - x_g|\right) + w_{pos}\left(|y_c - y_g|\right) + \\ & w_{orn}\left(|\,\text{sen}(\theta_c) - \text{sen}(\theta_g|)\right) + w_{orn}\left(|\,\cos(\theta_c) - \cos(\theta_g|)\right) \Big)^{1/2} \end{aligned} \tag{121}$$

O fator de desconto, dependente do estado, é utilizado para terminação antecipada do episódio, assim que o objetivo é atingido:

$$\gamma_g(s) = \begin{cases} \gamma & d(s',g) > \epsilon_r \\ 0 & d(s',g) \leq \epsilon_r \end{cases} \tag{122}$$

[12] A variável ϕ_t na Equação 118 representa o ângulo da roda frontal esquerda.

Como a recompensa é uma constante negativa, fornecida a cada instante de tempo, a política ótima é aquela que atinge o objetivo no menor tempo possível, já que isto causa a terminação antecipada do episódio. O parâmetro γ, na Equação 122, representa o fator de desconto padrão em PDMs. A função de recompensa e o fator de desconto dependem do parâmetro ϵ_r e dos pesos w_{pos} e w_{orn}. Os valores desses parâmetros foram escolhidos de forma que o objetivo é considerado atingido quando o robô se encontra bem próximo do alvo, mas não necessariamente na posição e orientação exata, como mostra a Figura 55.

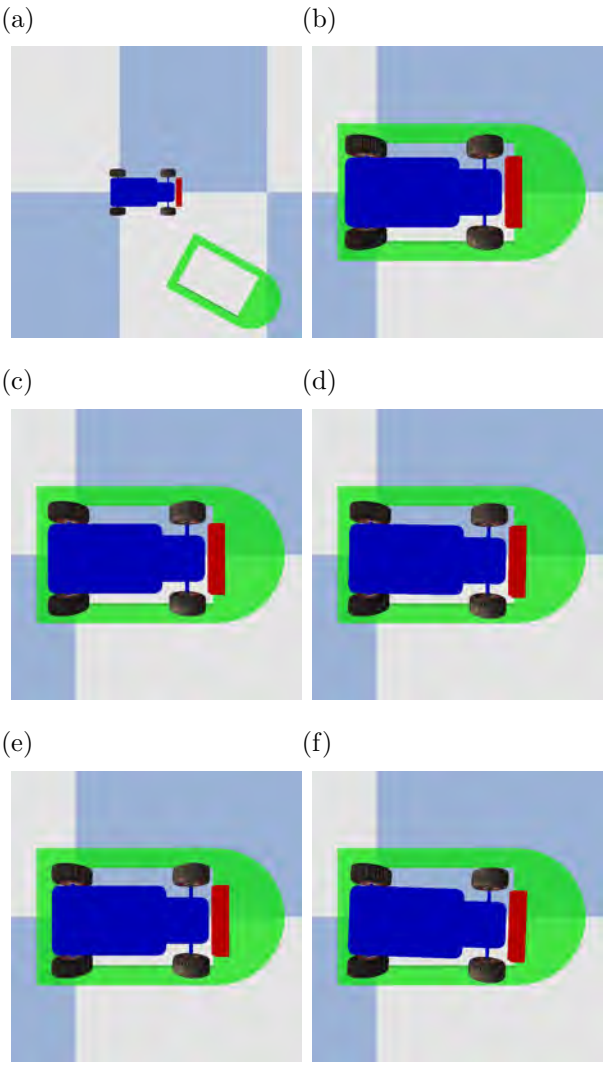
Algoritmos de aprendizado por reforço tendem a apresentar problemas em tarefas de robótica quando a frequência em que as ações são tomadas é muito grande (REDA; TAO; van de Panne, 2020), em relação ao horizonte da tarefa. Uma possível causa para este problema é a situação descrita no Exemplo 9. Por esta razão, cada ação que o agente escolhe é repetida pelos próximos n_{rep} passos de tempo. Isso pode ser visto como um controlador discreto, que funciona em uma determinada frequência f. Para PDMs episódicos, deve-se também definir a distribuição de estados iniciais. No início de cada episódio, as variáveis de s_0 vêm de uma distribuição uniforme, com exceção da velocidade e orientação do eixo iniciais:

$$S_0 = [\, X_0 \;\; Y_0 \;\; \text{sen}(\Theta_0) \;\; \cos(\Theta_0) \;\; v_0 \;\; \phi_0 \,] \quad \begin{cases} X_0 \sim \mathcal{U}\left(-\frac{d}{2}, \frac{d}{2}\right) \\ Y_0 \sim \mathcal{U}\left(-\frac{d}{2}, \frac{d}{2}\right) \\ \Theta_0 \sim \mathcal{U}\left(0, 2\pi\right) \\ v_0 = 0 \\ \phi_0 = 0 \end{cases} \tag{123}$$

O algoritmo DDPG com o MEGA, descrito na Seção 3.7, é escolhido para resolver o problema de aprendizado por reforço descrito nesta Seção. Como no algoritmo MEGA o agente gradualmente atinge objetivos mais distantes da distribuição de estados iniciais, é possível que o agente seja capaz de atingir objetivos fora da região estipulada na Equação 123. É importante observar que o cenário de simulação não contém obstáculos.

Os valores de diversos parâmetros citados nesta seção e outros que fazem parte do algoritmo de aprendizado por reforço são apresentados na Tabela 2. Com esses parâmetros mantidos fixos, realiza-se uma busca em grade pelos hiper-parâmetros ideais. Um total de 4 configurações são experimentadas com a permutação dos valores apresentados na Tabela 3. O conjunto de parâmetros escolhido é o que apresenta o melhor desempenho, em termos de recompensa acumulada, em 50 episódios de teste, realizados a cada 10000 instantes de tempo de treinamento. Os episódios de teste são gerados com as mesmas características para os diferentes conjuntos de hiper-parâmetros.

Figura 55 - Limiar de realização do objetivo



Legenda: A ilustração busca representar a proximidade necessária do CLMR, em relação à posição e orientação alvos, para que o objetivo seja considerado atingido. (a) - o retângulo com face arredondada representa a configuração desejada. Em (b), (c), (d), (e), (f) a configuração desejada é $[x_g \;\; y_g \;\; \theta_g] = [0 \;\; 0 \;\; 0]$. Em (b) o CLMR atinge o objetivo com exatidão, isto é, $[x_c \;\; y_c \;\; \theta_c] = [0 \;\; 0 \;\; 0]$ e $d(s,g) = 0 \leq \epsilon_r$, logo $r_g = 0$, de acordo com a Equação 120; (c) - para $[x_c \;\; y_c \;\; \theta_c] = [0{,}01 \;\; 0{,}01 \;\; 0] \implies d(s,g) = -0{,}087 \leq \epsilon_r \implies r_g = 0$; (d) - para $[x_c \;\; y_c \;\; \theta_c] = [0{,}01 \;\; 0{,}01 \;\; 1°] \implies d(s,g) = -0{,}094 \leq \epsilon_r \implies r_g = 0$; (e) - para $[x_c \;\; y_c \;\; \theta_c] = [0{,}02 \;\; 0{,}01 \;\; 0] \implies d(s,g) = -0{,}106 \leq \epsilon_r \implies r_g = -1$; (f) - para $[x_c \;\; y_c \;\; \theta_c] = [0{,}01 \;\; 0{,}01 \;\; 0] \implies d(s,g) = -0{,}101 \leq \epsilon_r \implies r_g = -1$.

Fonte: O Autor, 2022.

Tabela 2 - Parâmetros do módulo de controle

Parâmetro	Valor
w_{pos}	0,35
w_{orn}	0,15
ϵ_r	0,10
γ	0,99
f (Hz)	12
Passos de tempo de treinamento	10^6
Número máximo de transições no *replay buffer*	10^6
Função de ativação das redes neurais	GeLU
Otimizador das redes neurais	Adam

Fonte: O autor, 2022.

Tabela 3 - Busca por hiper-parâmetros no módulo de controle

Parâmetro	Valores
Neurônios nas camadas escondidas	$\{[512, 512, 512]; [1024, 1024]\}$
Tamanho dos mini-lotes de transições	$\{1024; 2048\}$

Fonte: O autor, 2022.

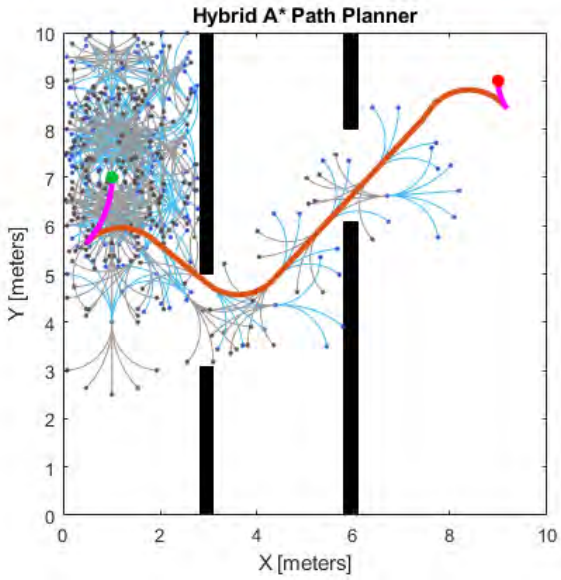
4.3 **Módulo de Planejamento**

Conforme a abordagem das Seções 2.3 é possível obter um modelo generativo que cria amostras de configurações promissoras, de acordo com as características do problema de planejamento, auxiliando a convergência do algoritmo RRT*. Para o modelo de CLMR da Seção 4.1 pode-se definir a *configuração* da mesma forma que o *objetivo*, estabelecido na Seção 4.2:

$$\vec{g} = [\, x_c \;\; y_c \;\; \text{sen}(\theta_c) \;\; \cos(\theta_c) \,] \tag{124}$$

A base de dados é formada por configurações de caminhos ótimos gerados pelo algoritmo HybridA* (DOLGOV et al., 2008; PETEREIT et al., 2012), através do software MATLAB (MATLAB, 2022). Esse algoritmo realiza o planejamento de caminhos para CLMRs, com base no modelo cinemático de bicicleta, de forma eficiente, permitindo que as soluções sejam obtidas rapidamente. A primeira etapa do HybridA* consiste em expandir uma árvore a partir da configuração inicial, de modo que cada ramificação corresponda à simulação do modelo cinemático de bicicleta (Equação 117) com ângulos de direcionamento $\phi_c \in \{-40°, -35°, \ldots, 35°, 40°\}$. Outras ramificações são geradas com base nos caminhos de Reed-Shepps (REEDS; SHEPP, 1990). O custo do caminho envolve um balanceamento entre dois critérios: a distância percorrida e a proximidade dos obstáculos.

Figura 56 - Solução com o HybridA* no MATLAB



Fonte: O autor, 2022.

O caminho que liga a configuração inicial à final tipicamente exibe características subótimas. Com isso, aplica-se um pós-processamento, visando suavizar o caminho e diminuir sua distância total.

As configurações pertencentes ao caminho final encontrado pelo algoritmo e o vetor de condições são armazenados em arquivos *csv*. A base de dados é formada por aproximadamente 5000 amostras $(\vec{g}^{\,(i)}, \vec{y}^{\,(i)})$ para treinamento e 500 para validação. Os parágrafos seguintes elaboram o critério de otimização do β-CVAE.

Como aborda a Seção 1.5, assume-se que $p(\vec{z})$ é uma distribuição simples e tratável. A escolha deste trabalho é uma gaussiana com média nula e matriz de covariância identidade, como mostra a Equação 125. A dimensão da distribuição é igual à M, onde M é a dimensão do vetor latente, um dos hiper-parâmetros do β-CVAE. O vetor $\vec{\epsilon}$, utilizado na técnica de reparametrização, também é amostrado de uma distribuição normal padrão.

$$p(\vec{z}) = \mathcal{N}\left(\vec{0}, \vec{1}\right) \tag{125}$$

A função de custo a ser otimizada é similar a Equação 32, porém deve-se notar que os modelos probabilísticos dependem também do vetor de condições $\vec{y}$, composto pela concatenação das configurações inicial e final, além de um vetor $\vec{o}$, que representa a localização dos obstáculos:

$$\vec{y} = [\vec{g}_i \parallel \vec{g}_f \parallel \vec{o}] \tag{126}$$

A Equação 127 apresenta a função de custo do β-CVAE para uma amostra $(\vec{g}^{\,(i)}, \vec{y}^{\,(i)})$ do conjunto de dados, com a consideração que a variável observável $\vec{x}$ é agora representada por uma configuração $\vec{g}$. As redes PMC decodificadora e codificadora são parametrizadas por θ e ϕ, respectivamente.

$$\mathcal{L}_i(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) = \mathbb{E}_{\vec{Z}\sim q_\phi(\vec{z}\mid \vec{g}^{\,(i)}, \vec{y}^{\,(i)})}\left[\log p_\theta\left(\vec{g}^{\,(i)} \mid \vec{Z}, \vec{y}^{\,(i)}\right)\right] - \frac{N}{M}\beta D_{KL}\left(q_\phi(\vec{z}\mid \vec{g}^{\,(i)}, \vec{y}^{\,(i)}) \,||\, p(\vec{z})\right) \tag{127}$$

Neste trabalho, o codificador q_ϕ parametriza uma distribuição normal multivariada com matriz de covariância diagonal. Dessa forma, a divergência KL têm a forma analítica da Equação 28:

$$\mathcal{L}_i(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) = \mathbb{E}_{\vec{Z}\sim q_\phi(\vec{z}\mid \vec{g}^{\,(i)}, \vec{y}^{\,(i)})}\left[\log p_\theta\left(\vec{g}^{\,(i)} \mid \vec{Z}, \vec{y}^{\,(i)}\right)\right] - \frac{N\beta}{2M}\sum_{j=1}^{M}\left[\log(\sigma^2_{\phi_j}(\vec{g}^{\,(i)}, \vec{y}^{\,(i)})) + \sigma^2_{\phi_j}(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) + \mu^2_{\phi_j}(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) + 1\right] \tag{128}$$

Considera-se também que o decodificador p_θ é uma rede PMC que parametriza o vetor de médias de uma distribuição normal multivariada, com matriz de covariância identidade. Dessa forma, pode-se substituir a primeira parcela da Equação 128 por uma expressão similar à um erro de reconstrução, como mostra a Equação 31:

$$\mathcal{L}_i(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) = \sum_{j=1}^{M}\mathbb{E}_{\vec{Z}\sim q_\phi(\vec{z}\mid \vec{g}^{\,(i)}, \vec{y}^{\,(i)})}\left[-\frac{\log(2\pi)}{2} - \frac{\left(\vec{g_j}^{\,(i)} - \mu^{(i)}_{\theta_j}(\vec{Z})\right)^2}{2}\right] - \frac{N\beta}{2M}\sum_{j=1}^{M}\left[\log(\sigma^2_{\phi_j}(\vec{g}^{\,(i)}, \vec{y}^{\,(i)})) + \sigma^2_{\phi_j}(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) + \mu^2_{\phi_j}(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) + 1\right] \tag{129}$$

Onde a expectativa da Equação 129 está relacionada à obtenção do vetor latente após a reparametrização:

$$\vec{Z} = \mu_\phi(\vec{g}) + \sigma_\phi(\vec{g}) \odot \vec{\epsilon} \qquad \vec{\epsilon} \sim \mathcal{N}(\vec{0}, \vec{1}) \tag{130}$$

Pode-se simplificar o critério da Equação 129 eliminando a soma das constantes, já que não influenciam no gradiente. Além disso, os fatores de escalonamento são removidos, de forma a concentrar o balanceamento dos termos no hiper-parâmetro β. Substituindo N pela dimensão da variável observável (Equação 124), pode-se obter o critério de oti-

Tabela 4 - Parâmetros do módulo de planejamento

Parâmetro	Valor
Tamanho do mini-lote	64
Número de épocas	50
Função de ativação das redes neurais	ReLU

Fonte: O autor, 2022.

Tabela 5 - Busca por hiper-parâmetros no módulo de planejamento

Parâmetro	Valores
Dimensão do espaço latente (M)	$\{2; 3\}$
Fator de balanceamento (β)	$\{0{,}001; 0{,}005; 0{,}01; 0{,}05; 0{,}1; 0{,}5; 1; 5\}$
Neurônios nas camadas escondidas	$\{[256, 256]; [512, 512]\}$
Taxa de aprendizado	$\{0{,}001; 0{,}01\}$

Fonte: O autor, 2022.

mização:

$$\mathcal{L}_i(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) = \sum_{j=1}^{M} \mathbb{E}_{\vec{\epsilon}\sim\mathcal{N}(\vec{0},\vec{1})}\left[-\left(\vec{g}_j^{\,(i)} - \mu_{\theta_j}^{(i)}(\mu_\phi(\vec{g}) + \sigma_\phi(\vec{g}) \odot \vec{\epsilon})\right)^2\right] - \frac{4\beta}{M}\sum_{j=1}^{M}\left[\log(\sigma_{\phi_j}^2(\vec{g}^{\,(i)}, \vec{y}^{\,(i)})) + \sigma_{\phi_j}^2(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) + \mu_{\phi_j}^2(\vec{g}^{\,(i)}, \vec{y}^{\,(i)}) + 1\right] \tag{131}$$

O grafo computacional da Equação 131 é implementado no software PyTorch (PASZKE et al., 2019) e os parâmetros de θ e ϕ otimizados com o Adam (Algoritmo 1). O treinamento do β-CVAE é interrompido quando o erro total no conjunto de validação aumenta, em relação à iteração anterior. Outras características do β-CVAE são apresentadas na Tabela 4. A busca por hiper-parâmetros é realizada com todas as permutações dos valores da Tabela 5.

4.4 **RRT* com Aprendizado de Máquinas**

As Seções 4.2 e 4.3 abordaram a obtenção dos módulos de controle e planejamento, respectivamente. Esta seção elabora no procedimento de integração desses módulos no algoritmo RRT*. Os dois módulos implementam redes neurais PMC, e portanto, se beneficiam da busca pelos melhores hiper-parâmetros. A princípio, verifica-se como selecionar os melhores modelos após a realização dessa busca. A obtenção de uma métrica de distância para o RRT* é abordada em seguida.

A escolha dos melhores parâmetros para o módulo de controle é trivial. Considerando que cada conjunto de hiper-parâmetro dá origem a uma política, após a fase de treinamento, pode-se testar cada controlador em um determinado número de episódios e selecionar o que atinge maior acúmulo de recompensas. Neste trabalho, o desempenho do agente, em episódios de teste, é monitorado a cada 10000 passos de tempo de treinamento e os parâmetros são armazenados em um arquivo, sempre que a recompensa acumulada aumenta, em relação à iteração anterior. Com isso, é possível comparar os hiper-parâmetros com base no melhor desempenho de teste, ao longo de todo o treinamento.

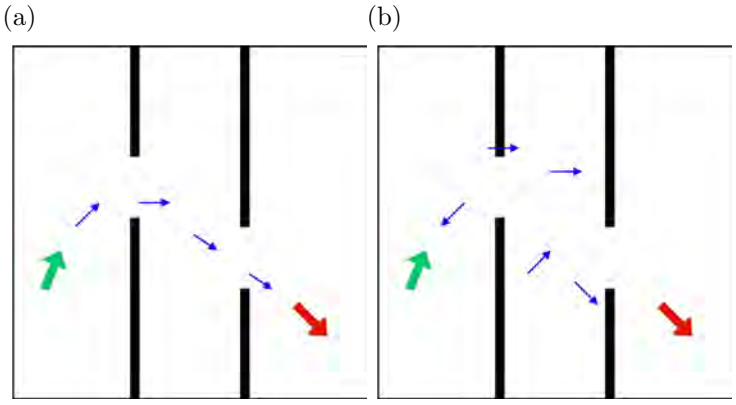
Selecionar o melhor conjunto de parâmetros para o β-CVAE é uma tarefa mais complexa. Os resultados experimentais obtidos por Theis, Oord e Bethge (2015) demonstram que a metodologia de avaliação do modelo generativo depende contexto em que esses modelos são aplicados. Por exemplo, a avaliação subjetiva é apropriada para aplicações que envolvam a sintetização de imagens (THEIS; OORD; BETHGE, 2015). Como o movimento de CLMRs é conhecido, a seleção do modelo pode ser realizada por meio de inspeções visuais das amostras geradas, como demonstra a Figura 57. Para realizar a seleção do modelo de forma automática, pode-se pensar em um critério que caracterize a qualidade da solução de um algoritmo de planejamento, quando sob auxílio de um determinado modelo generativo.

Como sugerido por Ichter, Harrison e Pavone (2017) para trabalhos futuros, esta Dissertação usa um fator α, que determina a probabilidade de que a amostra aleatória seja originada do β-CVAE, adaptativo, de modo que a distribuição de amostras se torne mais uniforme, à medida em que aumenta o número de vértices na árvore de exploração. A Figura 58 mostra o decaimento de α em razão do número de iterações do RRT*.

Para obter a métrica de distância, observa-se inicialmente que um controlador ótimo minimiza a função de distância "real", da mesma forma que a política ótima maximiza a função valor ótima, conforme a discussão no início da Seção 4.2. Se a política ótima é obtida no desenvolvimento do módulo de controle, a função ação-valor Q_θ tem relação direta com a métrica de distância "real" do problema. Contudo a política $\pi_{\boldsymbol{\phi}}$ apenas aproxima π^* e a utilização de Q_θ como métrica de distância é problemática, uma vez que algoritmos de aprendizado por reforço baseados no *Q-learning*, quando usam aproximações da função ação-valor, tendem superestimar a expectativa de retorno em algumas regiões do espaço de estados (THRUN; SCHWARTZ, 1993). Além disso, se a política não consegue atingir um determinado objetivo, a expectativa de retorno associada é $-\infty$, o que não pode representado por redes neurais.

Supondo que uma política aproximadamente ótima $\pi_{\boldsymbol{\phi}} \approx \pi^*$ seja obtida após a obtenção do módulo de controle, a função de distância associada a esta política, $\rho_{\pi_{\boldsymbol{\phi}}}$, uma aproximação da distância "real", pode ser obtida de maneira exata ao simular $\pi_{\boldsymbol{\phi}}$ em um episódio. Contudo, essa abordagem exigiria um grande número de simulações, se utilizada como maneira de calcular a distância entre configurações, a cada iteração do RRT*. Para

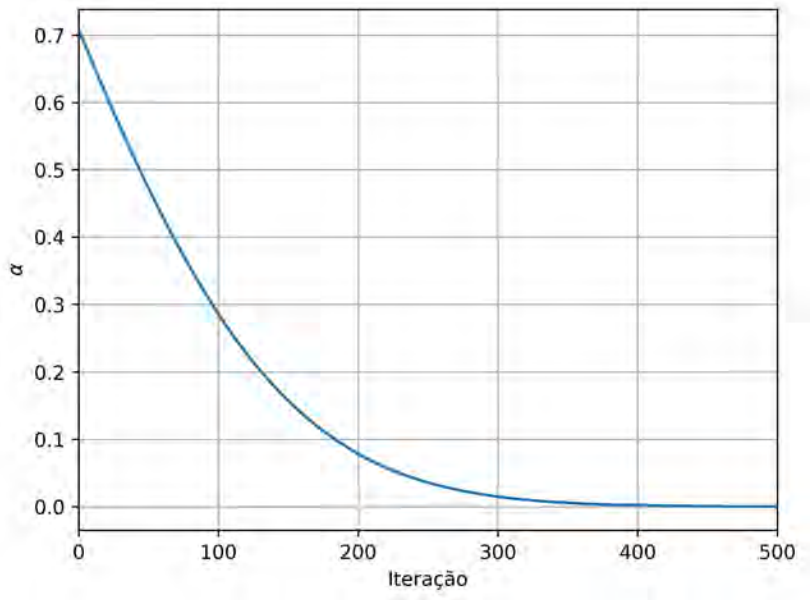
Figura 57 - Comparação de modelos generativos



Legenda: As setas indicam a configuração, isto é, a posição e a orientação do veículo. Em *verde*, a configuração inicial. Em *vermelho*, a configuração final (objetivo). Em *azul*, as configurações aleatórias geradas pelo β-CVAE. (a) - o modelo gera configurações coerentes com uma trajetória de CLMR; (b) - este outro modelo gera configurações que colidem com os obstáculos ou que não são coerentes.

Fonte: O autor, 2022.

Figura 58 - Probabilidade de amostragem do β-CVAE



Fonte: O autor, 2022.

contornar essa limitação, este trabalho propõe utilizar o aprendizado supervisionado para obter uma aproximação $\rho \approx \rho_{\pi_\phi}$. É importante observar que em algumas sub-rotinas do RRT*, em que a métrica de distância ρ é utilizada, o algoritmo necessita de uma simulação para verificar a ocorrência de colisões. Pode-se aproveitar essas simulações para calcular o custo da trajetória, ao invés de considerar a estimativa do modelo de aprendizado supervisionado. Os parágrafos seguintes detalham a implementação desse modelo.

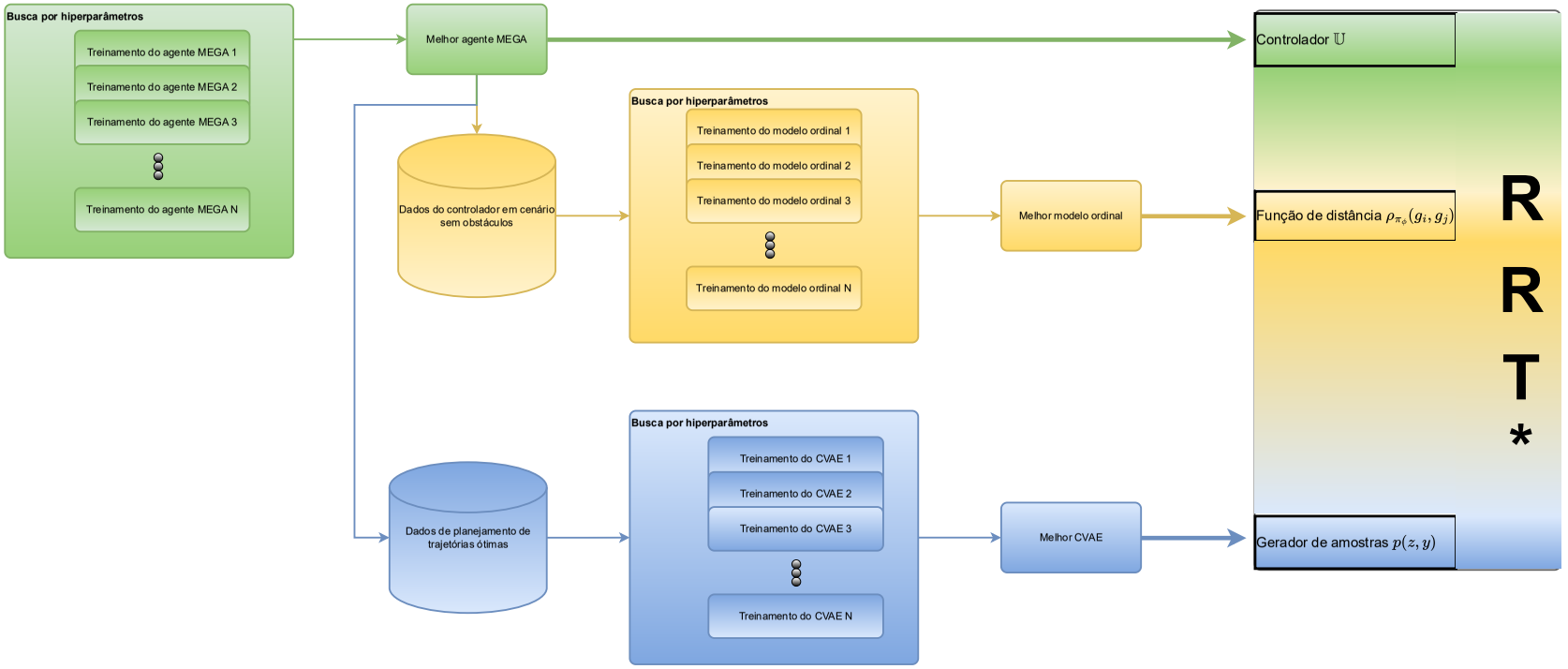
Após a obtenção do módulo de controle, a política π_ϕ é simulada em diversos episódios de teste, em um cenário sem obstáculos, de forma a alcançar um objetivo aleatório. Caso o agente atinja o objetivo, o custo da trajetória é armazenado. Ao repetir este procedimento diversas vezes é possível montar uma base de dados com amostras $(\vec{x}\,, y)$, onde $\vec{x}$ é formado pela concatenação da configuração atingida no início do episódio com a configuração aleatória desejada e y é o custo da trajetória realizada no episódio. Para o caso abordado neste trabalho, isto é, a navegação autônoma de CLMRs através de trajetórias ótimas em relação ao tempo, esse custo é um número inteiro positivo, representando os passos de tempo de simulação, até o momento em que a configuração final é atingida. Com isso, o problema de aprendizado supervisionado é *ordinário* (GUTIÉRREZ et al., 2015).

Este trabalho utiliza o modelo de regressão ordinária CORN (SHI; CAO; RASCHKA, 2021), que divide o aprendizado em sub-tarefas de classificação binária. Experimentos preliminares mostraram que o erro de predição pode ser diminuído significativamente, sem custo adicional de recursos computacionais, ao utilizar um "*ensemble* de parâmetros" (HUANG et al., 2017): durante o treinamento do modelo diversos mínimos locais são visitados, com o auxílio de um agendamento cíclico na taxa de aprendizado (LOSHCHILOV; HUTTER, 2016), e os parâmetros do modelo são armazenados. A predição do *ensemble* é o valor médio das predições do modelo, com os diferentes parâmetros registrados. Os detalhes da implementação do modelo de regressão ordinária são apresentados nos Apêndices A.2 e C.3.

O processo de busca por hiper-parâmetros e treinamento dos diversos modelos de aprendizado de máquinas, como proposto neste trabalho, pode demandar um custo computacional intenso. Parte deste processo é simplificada pela utilização do algoritmo de planejamento HybridA*, que utiliza heurísticas próprias a veículos CLMRs e é capaz de gerar dados com eficácia, o que não necessariamente ocorre para qualquer sistema não-holonômico. Neste caso, este trabalho propõe a obtenção do módulo de controle, com a busca por hiper-parâmetros, como a primeira etapa da abordagem. O controlador local desenvolvido pode então ser utilizado para a geração de dados da métrica de distância. O módulo de planejamento é concebido por meio de trajetórias ótimas, que, por sua vez, podem ser criadas a partir da utilização do RRT* com o controlador e a métrica, obtidos na etapa anterior. Os diferentes módulos desenvolvidos neste processo são integrados ao

algoritmo RRT*, como ilustra a Figura 59.

Figura 59 - Método para integração do RRT* com o aprendizado de máquinas



Fonte: O autor, 2022.

5 VALIDAÇÃO COMPUTACIONAL

Esta seção apresenta os experimentos realizados e os resultados obtidos. Inicialmente, o funcionamento dos módulos de controle e planejamento são verificados, nesta ordem, de forma independente. Os resultados do algoritmo RRT* com as modificações propostas são apresentados em seguida e comparadas com um método analítico, que exige conhecimento específico sobre CLMRs. O mapa com obstáculos escolhido para os experimentos do módulo de planejamento são utilizados também para testar o RRT*.

5.1 Módulo de Controle

Esta seção aborda os resultados obtidos no treinamento e teste do agente de aprendizado por reforço. A princípio, o melhor conjunto de hiper-parâmetros da Tabela 3 é apresentado, de acordo com os critérios de teste estabelecidos. Em seguida, diversos resultados são apresentados para o agente obtido com tais parâmetros.

A Tabela 6 mostra o melhor desempenho na fase de teste, em qualquer etapa do treinamento, para todos os conjuntos de hiper-parâmetros. A seguinte notação é utilizada para referir-se aos diferentes conjuntos de hiper-parâmetros: com base na Tabela 3, o conjunto *h_xyz* indica que o primeiro parâmetro (de cima para baixo na Tabela 3) assume o x-to valor, o segundo parâmetro assume o y-to valor, e assim por diante. O *desempenho* na Tabela 6 refere-se à melhor média de recompensa acumulada nos 50 episódios de teste, ao longo da fase de treinamento.

A Figura 60 apresenta a recompensa acumulada nos episódios de teste para o melhor conjunto de hiper-parâmetros da Tabela 6. Os próximos resultados apresentados nesta seção referem-se à este conjunto. Na Figura 60, pode-se notar que após aproximadamente 150000 passos de tempo há um progresso limitado no desempenho do agente, portanto, o treinamento do agente pode ser realizado com uma quantidade menor de iterações. A melhoria da recompensa acumulada ao longo do treinamento pode também ser visualizada na Figura 61, que compara a trajetória do veículo em um episódio de teste em diferentes épocas do treinamento. É possível notar que nas duas primeiras épocas, a trajetória do veículo é ineficiente e longa, contudo, a partir da quarta época os caminhos tornam-se mais eficientes.

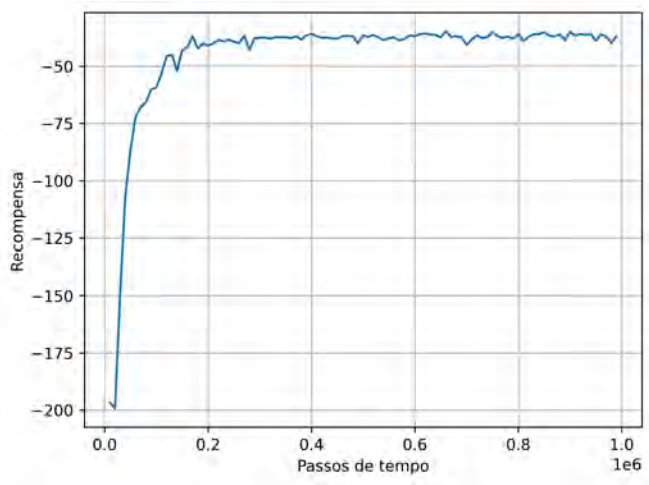
A principal característica do algoritmo MEGA é a capacidade de lidar com problemas de robótica de longo horizonte, ao buscar a maximização da entropia dos objetivos atingidos pelo agente, durante toda a fase de treinamento. A Figura 62 apresenta a entropia da distribuição empírica p_{ag}, em função dos passos de tempo de treinamento. É possível notar um crescimento significativo da entropia nos instantes iniciais, indicando a

Tabela 6 - Comparação dos hiper-parâmetros para o módulo de controle

Conjunto de Hiperparâmetros	Desempenho
h_11	$-34,84$
*h_12**	$-34,72^*$
h_21	$-35,78$
h_22	$-35,54$

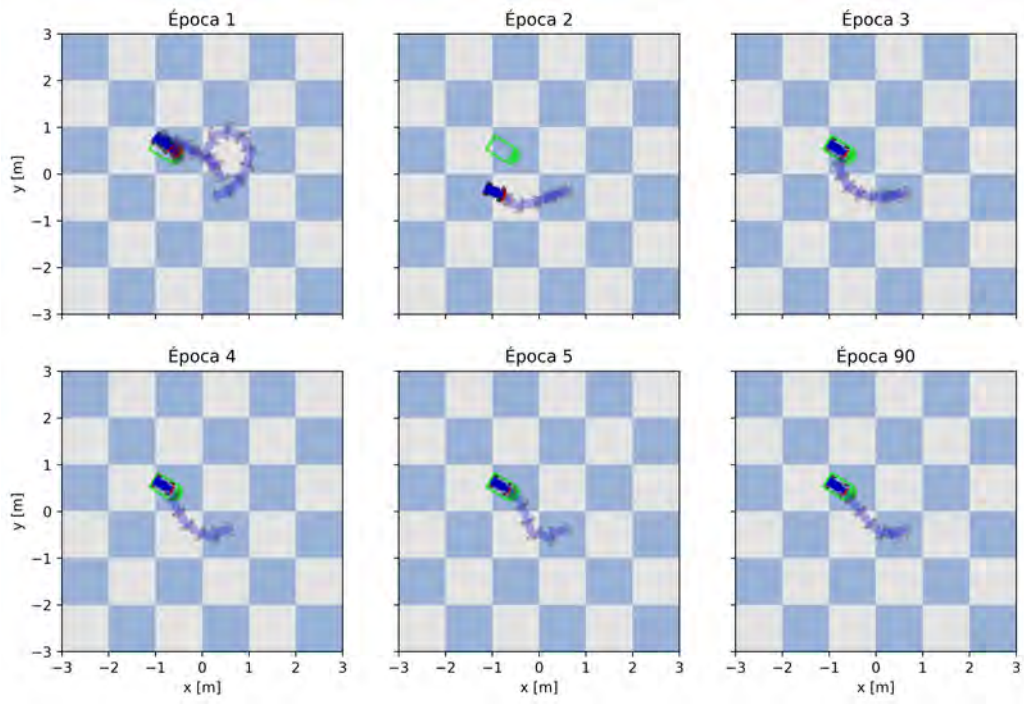
Legenda: O asterisco indica o melhor valor.
Fonte: O autor, 2022.

Figura 60 - Desempenho do agente nos episódios de teste



Fonte: O autor, 2022.

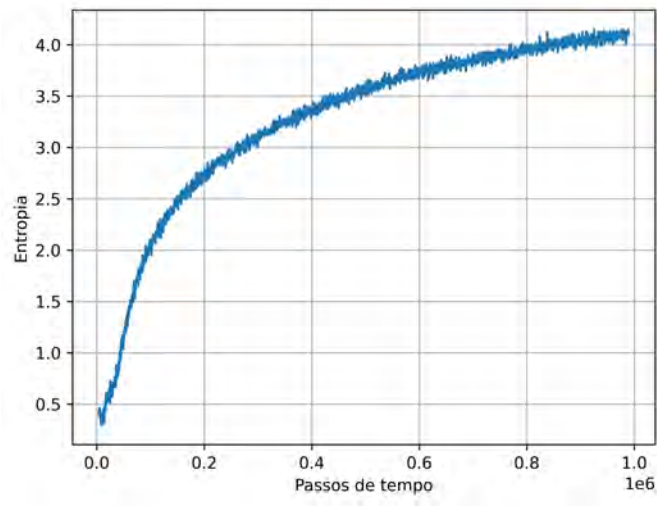
Figura 61 - Progressão do agente em um episódio de teste.



Legenda: Considera-se que uma época equivale a 10000 passos de simulação de treinamento.

Fonte: O autor, 2022.

Figura 62 - Entropia da distribuição de objetivos atingidos durante o treinamento.



Fonte: O autor, 2022.

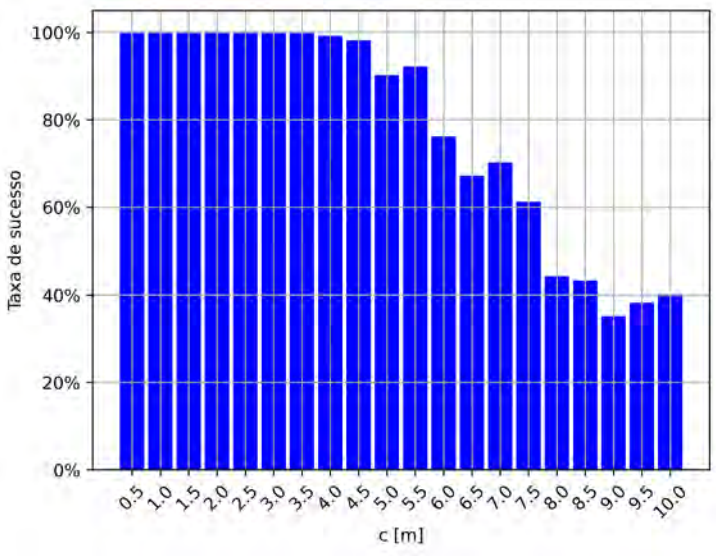
otimização da Equação 110 pela seleção adequada de objetivos comportamentais.

Já que o procedimento de religação do RRT* demanda que a configuração desejada seja atingida após a simulação, é interessante observar se o controlador é capaz de alcançar objetivos distantes, considerando que o algoritmo MEGA busca aumentar o horizonte de alcance desses objetivos. A Figura 63 apresenta a taxa de sucesso quando o agente inicia um episódio na origem e deve alcançar uma configuração $\vec{g} = [\, X_g \;\; Y_g \;\; \Theta_g \,]$, onde $X_g \sim U(-c,\, c)$, $Y_g \sim U(-c,\, c)$ e $\Theta_g \sim U(0,\, 2\pi)$. Um total de 100 episódios são gerados para cada valor de c.

Como pode ser visto na Figura 63, a taxa de sucesso é 100% para $c \leq 3.5\,\mathrm{m}$ e diminui para menos de 50% quando $c > 8.0\,\mathrm{m}$, o que evidência a dificuldade associada aos problemas com horizonte longo. A possibilidade de alcançar objetivos distantes permite o uso de intervalos de simulações maiores no RRT*, o que pode ser desejável em algumas aplicações, a fim de possivelmente reduzir o número de iterações necessárias para encontrar uma solução.

A Figura 64 apresenta uma comparação entre as trajetórias do agente, em diversos episódios de teste, e uma abordagem analítica que combina caminhos de *Reed-Shepps* (RS) (REEDS; SHEPP, 1990) com um controlador preditivo baseado em modelo (CPM) (GARCIA; PRETT; MORARI, 1989). Pode-se notar o melhor desempenho do agente de aprendizado por reforço, porém, vale notar que a implementação do controlador CPM depende da especificação de diversos parâmetros, tais como: o horizonte de predição, o peso das variáveis de estado na definição do custo, entre outros. É interessante observar que o controlador preditivo foi implementado com base no modelo cinemático de bicicleta

Figura 63 - Taxa de sucesso em função da distância



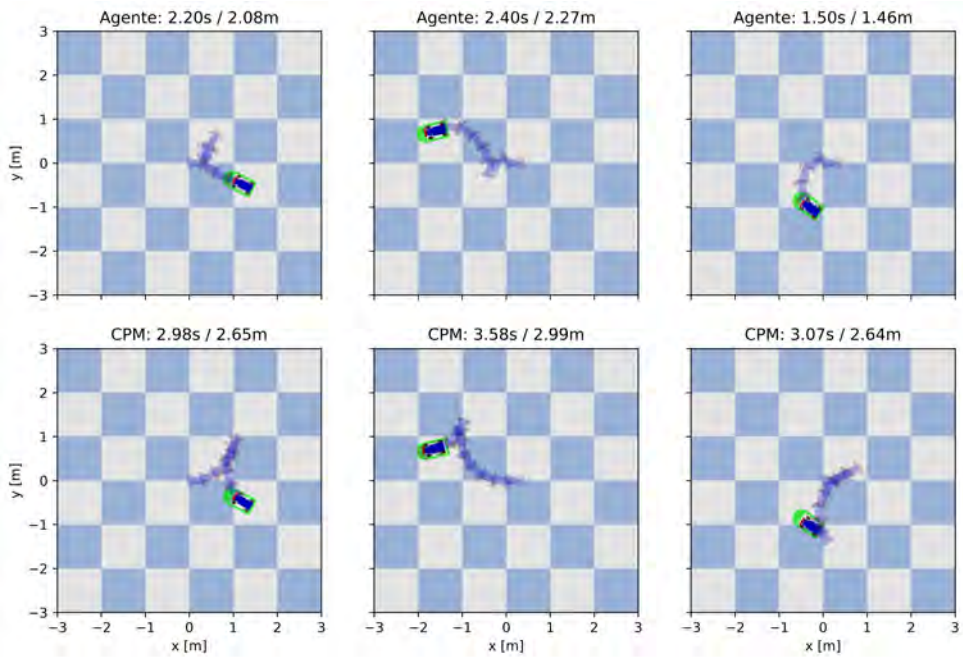
Fonte: O autor, 2022.

(Equação 117) que, possivelmente, não corresponde com absoluta fidelidade às simulações mais complexas. Além disso, os caminhos de RS são compostos por curvas de menor raio possível e mudanças repentinas de direção, entretanto, a referência do sinal de controle não é atingida imediatamente, como mostra a Figura 49. Através de diversos experimentos preliminares, que incluíam a modificação do modelo de bicicleta ao acrescentar uma aproximação de primeira ordem para as respostas ao sinal de referência, o melhor resultado para o CPM foi obtido ao considerar caminhos RS com raios maiores. Contudo, a utilização de trajetórias de referência com raios maiores certamente implica em caminhos mais longos, o que explicaria a diferença de desempenho entre as abordagens.

Para contornar essas dificuldades, realizou-se um experimento para comparar o comprimento da trajetória do agente de aprendizado por reforço com o comprimento do caminho de RS. Os resultados para alguns cenários podem ser observados na Figura 65. A comparação do tempo de navegação e da distância percorrida para todas as abordagens expostas nesta seção é apresentada na Figura 66, em 20 episódios de teste, gerados com as mesmas características para cada um dos métodos. Note que o objetivo original deste trabalho é a realização de trajetórias ótimas em relação ao tempo, porém, a comparação do comprimento da trajetória com a abordagem analítica pode servir para auxiliar a avaliação do agente.

De acordo com os resultados apresentados nas Figuras 65 e 66, é possível notar que a distância percorrida pelo agente é similar ao comprimento dos caminhos RS. É importante observar que o agente de aprendizado por reforço está sendo comparado com uma abordagem que fornece os caminhos mais curtos possíveis, considerando o modelo cinemático de bicicleta.

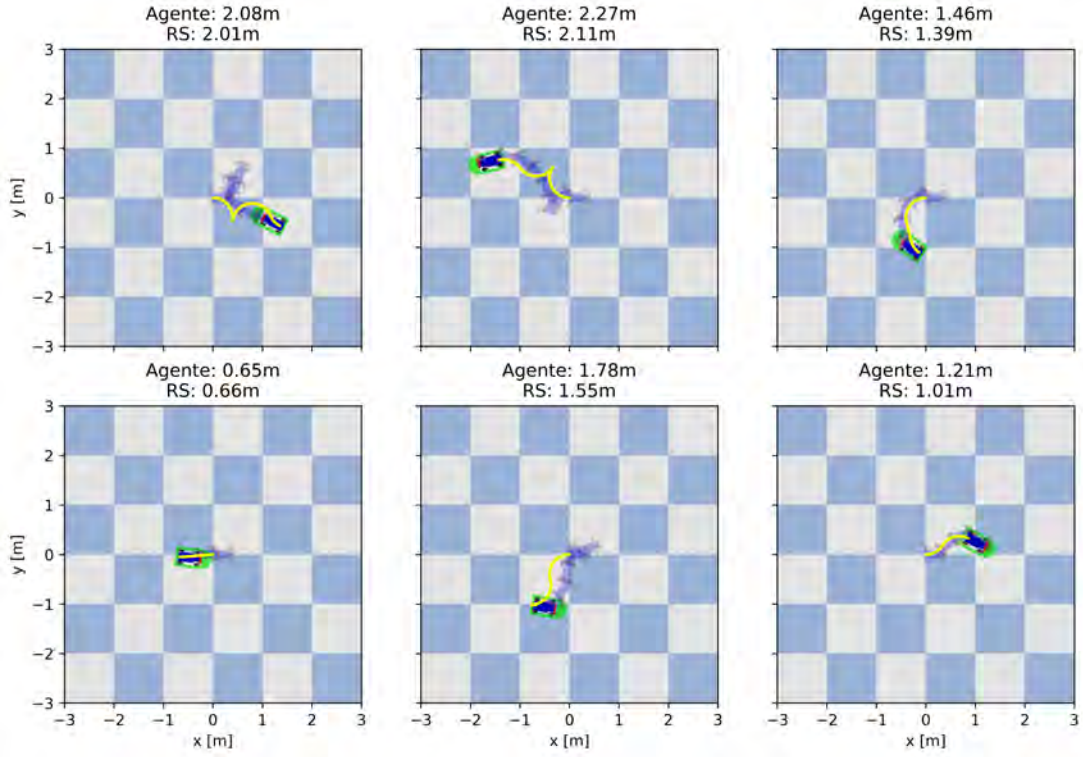
Figura 64 - Comparação entre o agente e a abordagem analítica



Legenda: A linha *acima* apresenta os caminhos realizados pelo agente MEGA. *Abaixo*, as trajetória do CPM quando as referências são caminhos de RS. Os problemas são iguais em uma mesma coluna. O tempo de realização da trajetória a distância percorrida encontram-se no topo de cada imagem.

Fonte: O autor, 2022.

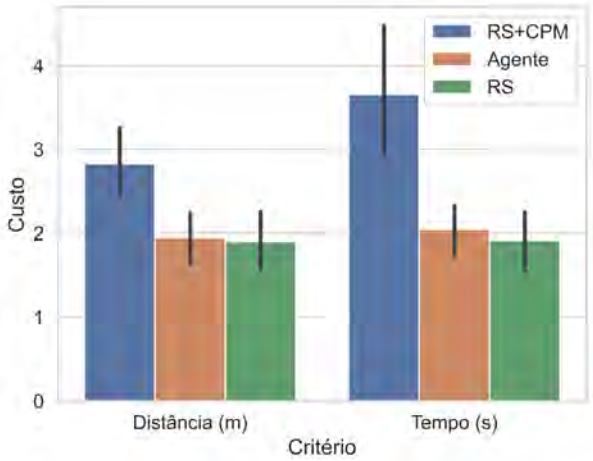
Figura 65 - Comparação dos caminhos do agente com as curvas de RS



Legenda: A trajetória do agente é representada pela sequência de configurações. Em *amarelo*, os caminhos correspondentes às curvas de RS. A distância total percorrida, em metros, é indicada no topo de cada imagem.

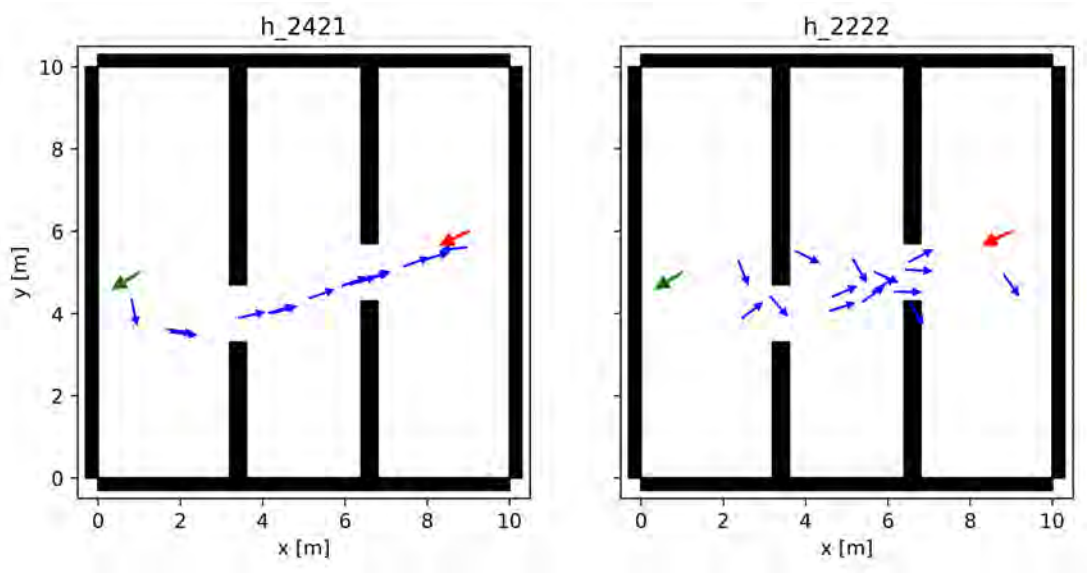
Fonte: O autor, 2022.

Figura 66 - Comparação entre as abordagens



Fonte: O autor, 2022.

Figura 67 - Comparação entre dois modelos com diferentes hiper-parâmetros



Legenda: A seta *verde* representa a configuração inicial, enquanto a seta *vermelha* indica a configuração final desejada. As setas *azuis* representam as configurações geradas pelo modelo.

Fonte: O autor, 2022.

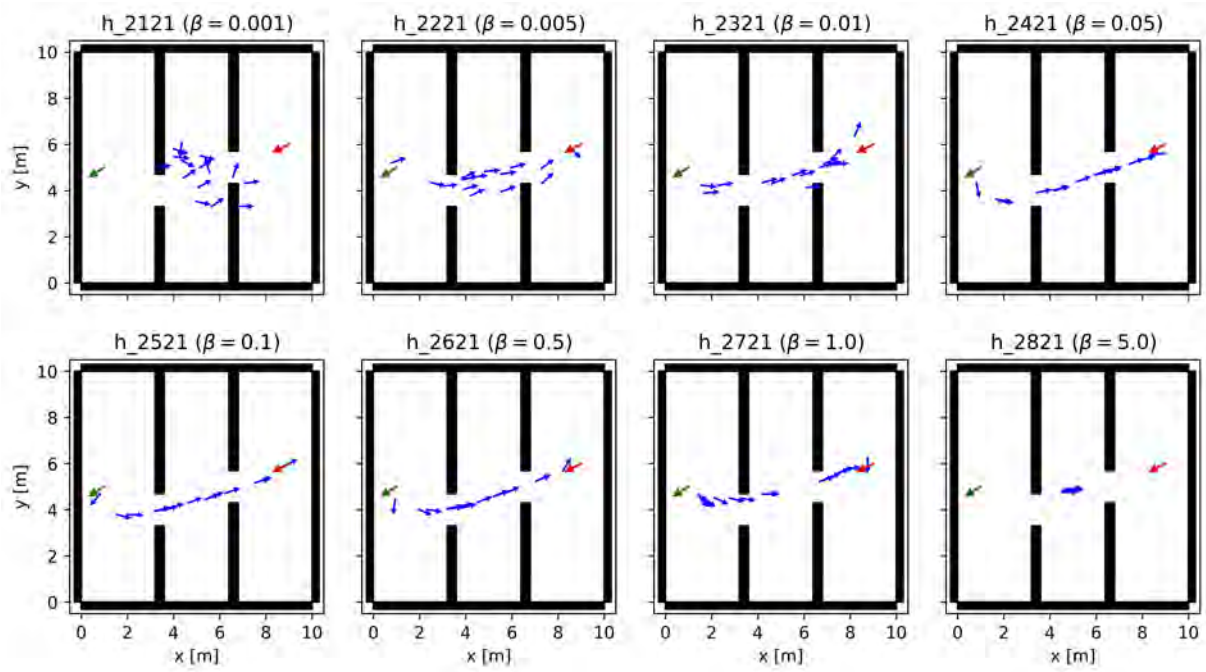
5.2 **Módulo de Planejamento**

Esta seção apresenta os resultados relacionados à obtenção do módulo de planejamento. O principal objetivo nesta etapa é verificar se o modelo β-CVAE é capaz de gerar amostras de configurações coerentes e que possam auxiliar o tempo de convergência do algoritmo RRT*.

Após a geração de dados e a subsequente busca por hiper-parâmetros, o modelo *h_2421* é escolhido, com base na inspeção visual das configurações geradas por todos os outros modelos treinados, com seus respectivos parâmetros. A Figura 67 apresenta as configurações geradas por dois modelos treinados com diferentes hiper-parâmetros. É possível notar que um desses modelos gera configurações coerentes com a solução que espera-se encontrar. Neste trabalho, o modelo foi escolhido com base nas amostras geradas em cinco problemas diferentes, contudo, a inspeção visual em um número maior de cenários é propícia à confiabilidade no processo de seleção do modelo. Exemplos de configurações geradas pelos diferentes conjuntos de parâmetros experimentados podem ser vistos no Apêndice C.2.

Através da extensiva busca por hiper-parâmetros foi possível notar o impacto do fator β de balanceamento das parcelas da função de custo. Com base na Figura 68, pode-se

Figura 68 - Influência do fator β



Legenda: Comparação entre as amostras geradas por modelos com hiper-parâmetros iguais, mas treinados com diferentes fatores de balanceamento de custo.

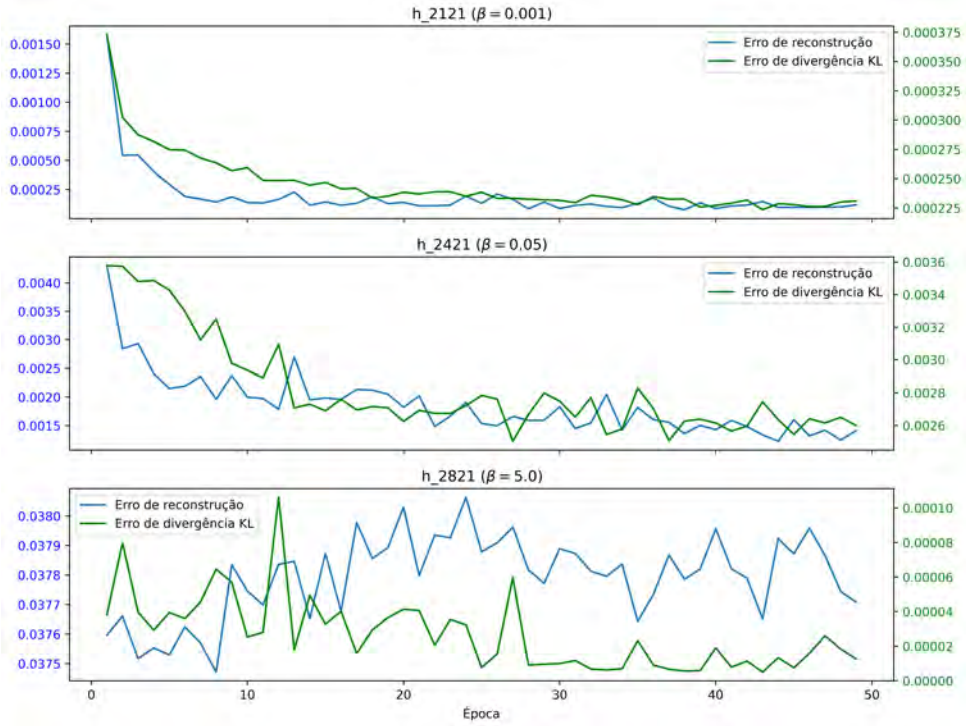
Fonte: O autor, 2022.

observar que para $\beta < 0{,}01$ a distribuição de amostras é esparsa. Em contrapartida, para $\beta > 1$, as amostras se concentram em uma pequena região do espaço de configurações. Em ambos os casos, a distribuição de amostras não reflete a trajetória correta para solucionar o problema. Os melhores resultados são obtidos com $0{,}05 \leq \beta \leq 0{,}1$. As curvas dos erros para três modelos da Figura 68 são apresentadas na Figura 69, onde é possível notar que os erros reconstrução e divergência se encontram na mesma ordem de magnitude, no início do treinamento, para o melhor conjunto de hiper-parâmetros (h_2421), o que não ocorre para os outros casos.

A Figura 70 apresenta as configurações geradas pelo modelo *h_2421* ao longo das épocas de treinamento. Inicialmente, as amostras de configurações são esparsas e não caracterizam a trajetória ótima do problema. Já nas últimas iterações, o modelo é capaz de gerar amostras que evitariam a colisão com os obstáculos e que capturam a dinâmica esperada de um robô móvel similar a um carro de passeio. Os erros de treino e validação ao longo das épocas podem ser vistos na Figura 71.

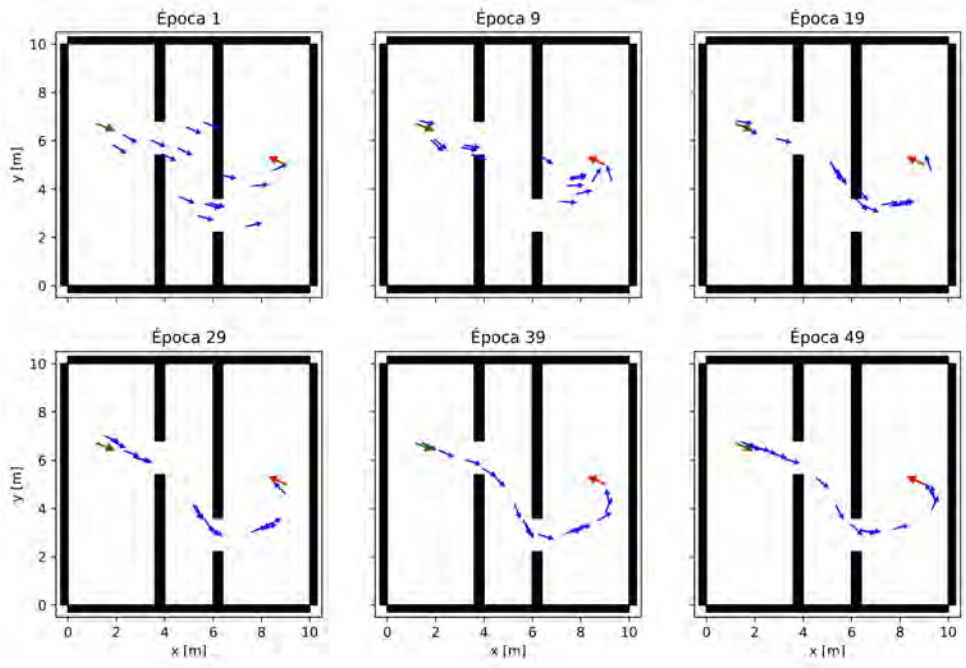
As amostras geradas pelo modelo após o treino podem ser visualizadas em seis problemas diferentes, na Figura 72. Pode-se verificar que a distribuição de configurações geradas nesses problemas são coerentes e podem, possivelmente, diminuir o tempo ne-

Figura 69 - Erro no conjunto de validação para diferentes valores de β.



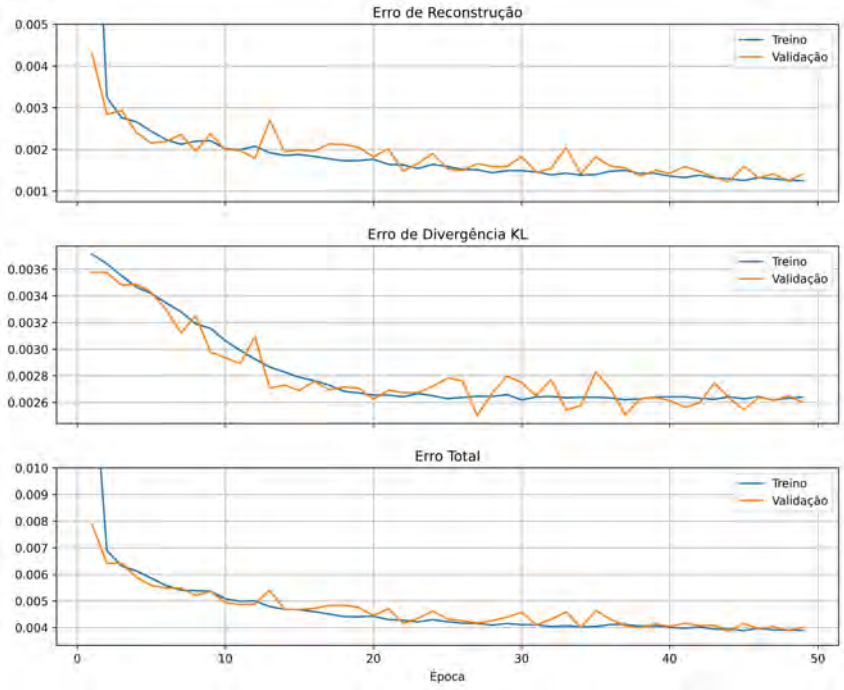
Fonte: O autor, 2022.

Figura 70 - Amostras geradas pelo melhor modelo ao longo treinamento



Fonte: O autor, 2022.

Figura 71 - Erro de treino e validação do melhor modelo obtido.



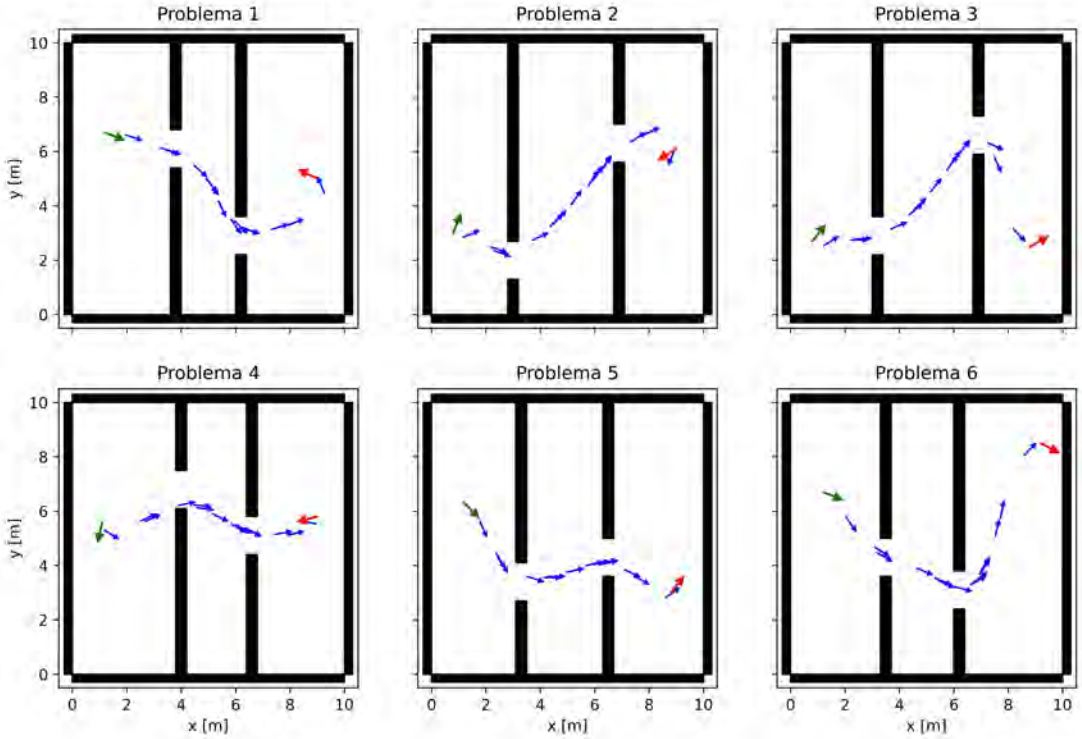
Fonte: O autor, 2022.

cessário para que o RRT* encontre uma solução.

5.3 **RRT***

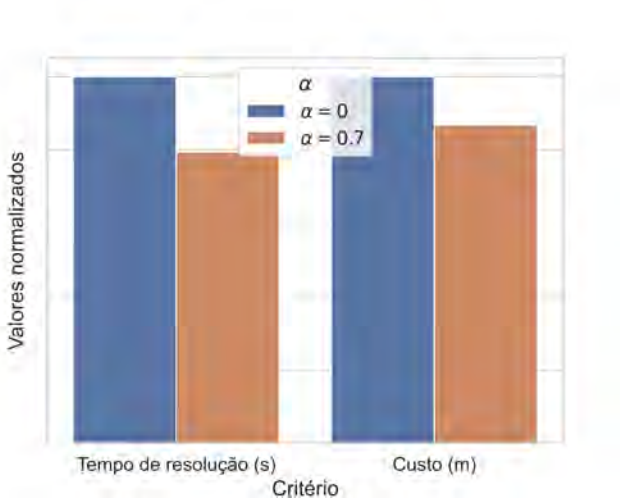
Esta seção apresenta os resultados obtidos nos experimentos relacionados ao método proposto, que combina o algoritmo RRT* com o aprendizado por reforço, a geração inteligente de configurações para o robô e o aprendizado supervisionado para obtenção de uma métrica de distância. Os problemas de planejamento de trajetórias envolvem o cenário com passagens estreitas, descrito na Seção 5.2. Dado o caráter estocástico do algoritmo RRT*, os resultados apresentados nesta seção são realizados com sementes (*seeds*) geradoras de números aleatórios que influenciam todos os processos probabilísticos do RRT*. Para facilitar a exposição dos resultados, o método proposto é chamado de RRT*+ApM (RRT* com o **A**prendizado de **M**áquinas). Essa abordagem é comparada com o procedimento analítico, denominado RRT*+RS+CPM, que utiliza caminhos de RS com o CPM. Vale observar que os caminhos de RS servem dois propósitos nesta abordagem: fornecer a trajetória de referência para o CPM e servir como métrica de distância entre configurações (equivalente ao comprimento total do caminho). Esta seção considera uma simplificação na execução do algoritmo RRT*+RS+CPM: as simulações

Figura 72 - Configurações geradas pelo melhor modelo em diversos problemas.



Fonte: O autor, 2022.

Figura 73 - Influência do modelo generativo no RRT*



Fonte: O autor, 2022.

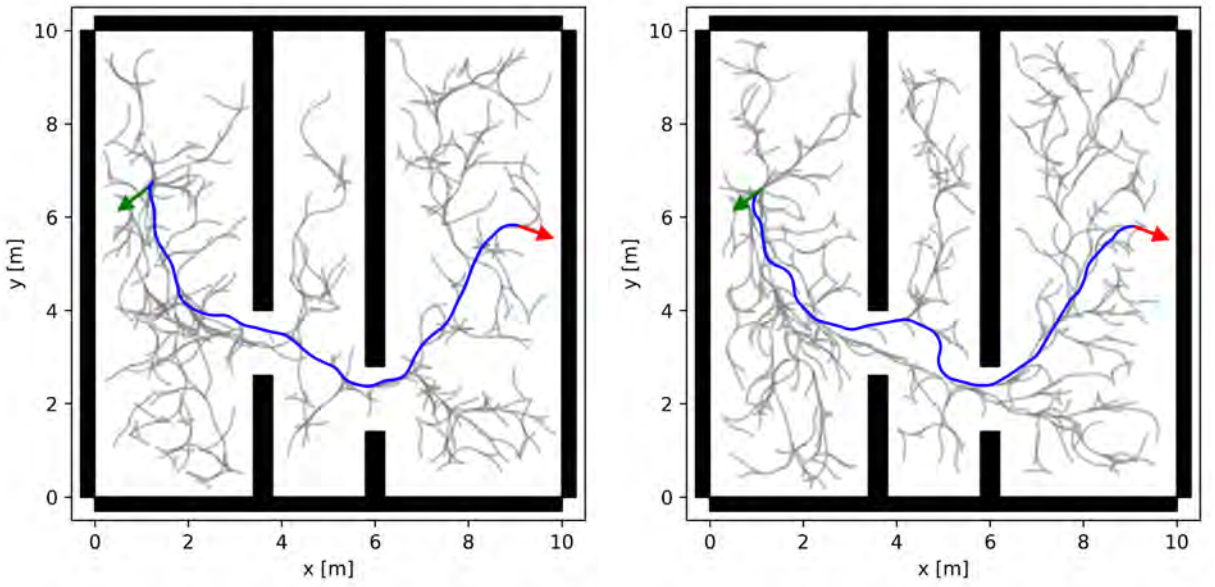
do sistema, que fazem parte do RRT*, são realizadas com a suposição que o CPM segue perfeitamente o caminho de RS. Com essa consideração, não é necessário executar o CPM nas iterações do RRT*, uma tarefa computacionalmente intensa, especialmente à medida que o número de vértices da árvore de exploração cresce.

O primeiro experimento verifica se, de fato, a utilização de amostras geradas pelo módulo de planejamento diminui o tempo necessário para encontrar uma solução. Para diminuir o custo computacional, considera-se a execução do RRT*+RS+CPM sem a utilização do procedimento de religação. O tempo médio de execução do algoritmo, até que a solução seja encontrada, em 5 experimentos com sementes diferentes, é comparado entre a execução do RRT* com o β-CVAE ($\alpha = 0{,}7$) e o algoritmo que utiliza apenas amostragem uniforme. Como mostra a Figura 73, a utilização do β-CVAE diminui o tempo necessário para encontrar uma solução em aproximadamente 20% e a trajetória obtida apresenta um custo total menor, em média.

Os próximos resultados referem-se à um experimento que compara o RRT*+RS+CPM com o RRT*+ApM em 9 problemas de planejamento gerados com condições aleatórias, respeitando as faixas de valores descritas na Equação 116. Cada problema é gerado com uma semente diferente. As duas variações do RRT* são executadas por 500 iterações, em cada um das condições aleatórias, e ambas utilizam o módulo de planejamento. Vale destacar que a semente geradora do problema também dita a escolha das configurações aleatórias, incluindo as provenientes do β-CVAE. A Figura 74 apresenta a solução encontrada pelas duas abordagens e a árvore de exploração, em um dos problemas.

Figura 74 - Árvores de exploração nas duas abordagens

(a) RRT*+ApM (b) RRT*+RS+CPM



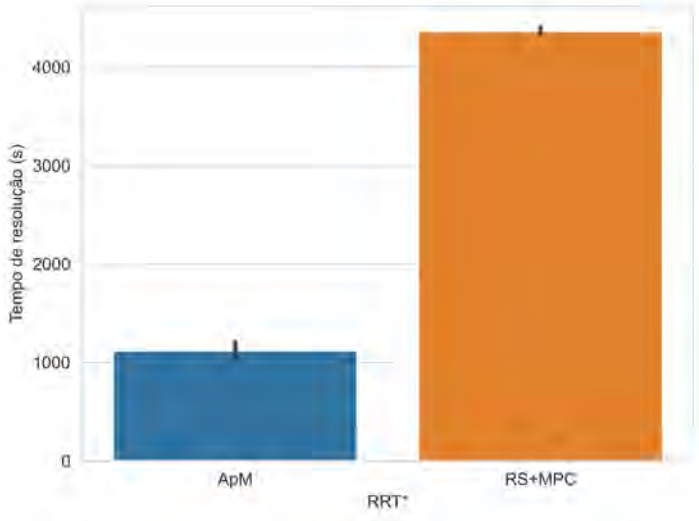
Fonte: O Autor, 2022.

As duas abordagens são capazes de resolver todos os problemas gerados de forma eficiente e geralmente encontram a solução antes da iteração de número 200. O custo computacional associado ao método proposto é consideravelmente menor, como mostra a Figura 75, pois, apesar do método analítico considerar apenas o modelo cinemático, definido implicitamente na geração das curvas de RS, as funções de criação dos caminhos são implementadas em uma linguagem interpretada. O tempo de inferência do sinal de controle é menor no controlador desenvolvido pelo aprendizado por reforço, de acordo com a Figura 76, o que é relevante sob a perspectiva de implementação do controlador em sistemas embarcados, geralmente limitados na capacidade computacional. A diferença no tempo de inferência ocorre porque, para o controlador proposto, a obtenção do controle limita-se à propagação direta de um vetor na rede PMC, enquanto o CPM deve resolver um problema de otimização não-linear com restrições.

É importante enfatizar que o caminho apresentado na Figura 74a representa a atuação do controlador de aprendizado por reforço, enquanto a Figura 74b mostra apenas o caminho de referência para o CPM. Uma comparação das trajetórias realizadas pelos controladores, nas soluções da Figura 74, é apresentada na Figura 77.

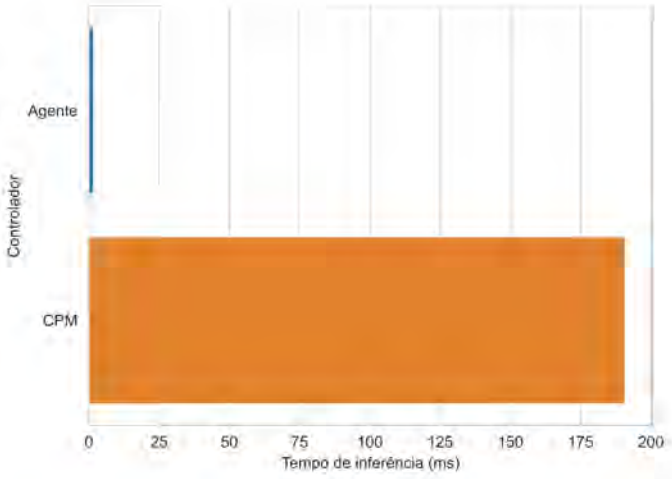
A Figura 78 apresenta a taxa de sucesso dos controladores na realização das trajetórias obtidas pelo RRT*, nos 9 problemas de navegação. Pode-se notar que o agente MEGA é capaz atingir a configuração final em todas as situações, enquanto o CPM falha em algumas, apesar da busca pelos melhores parâmetros considerar a taxa de sucesso do controlador em diversos cenários de teste. É possível que o modelo cinemático seja

Figura 75 - Comparação do custo computacional em 500 iterações do RRT*



Fonte: O autor, 2022.

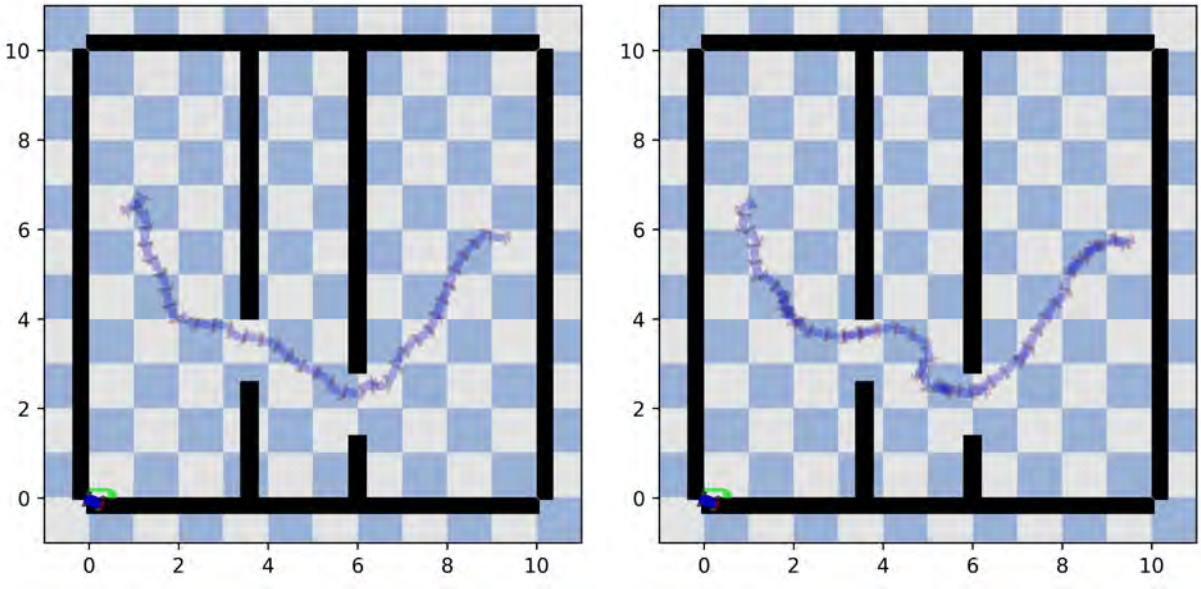
Figura 76 - Tempo de inferência do sinal de controle



Fonte: O autor, 2022.

Figura 77 - Trajetórias realizadas pelos controladores

(a) RRT*+ApM (b) RRT*+RS+CPM

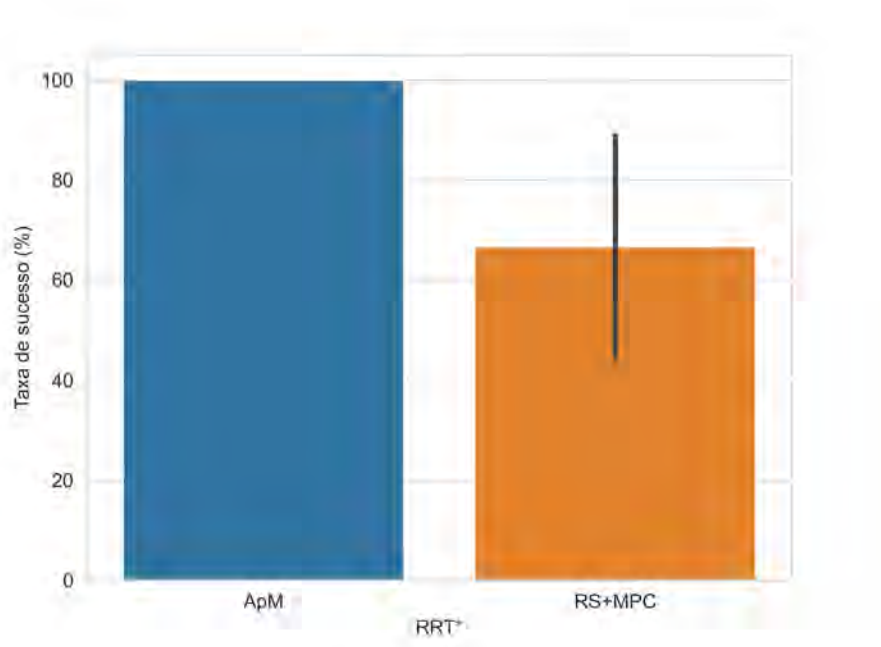


Fonte: O Autor, 2022.

uma aproximação insuficiente para o robô do PyBullet quando os caminhos de RS são utilizados como referência.

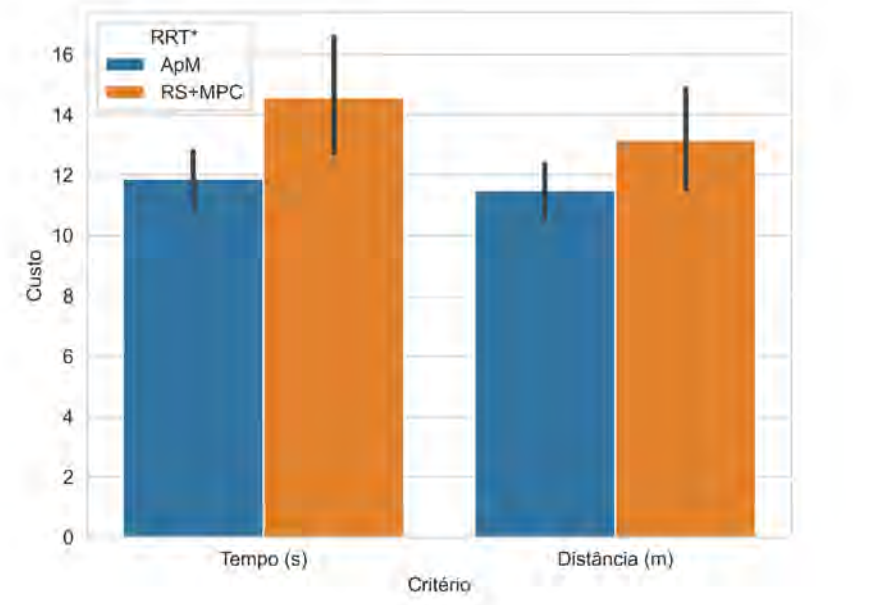
A Figura 79 compara a qualidade das trajetórias encontradas e realizadas pelos algoritmos RRT*+ApM e RRT*+RS+CPM, em termos de dois critérios: distância percorrida e tempo de navegação. Este experimento considera o valor médio dos critérios apenas nos problemas em que o CPM foi capaz de seguir o caminho encontrado. Pode-se notar que o método proposto obtém o melhor resultado, nos dois critérios. É importante enfatizar que o critério otimizado no aprendizado por reforço é o tempo de percurso, enquanto o CPM segue uma trajetória de referência que têm a distância percorrida como custo. Entretanto, é de se esperar que esses critérios estejam relacionados.

Figura 78 - Taxa de sucesso dos controladores



Fonte: O autor, 2022.

Figura 79 - Qualidade das trajetórias realizadas



Fonte: O autor, 2022.

CONCLUSÃO

Este trabalho aplicou técnicas de aprendizado por reforço, supervisionado e não-supervisionado no seguimento e planejamento de trajetórias, realizado por meio de árvores de exploração rápida. O método proposto não assume conhecimento específico sobre o sistema, como acontece com frequência na aplicação do algoritmo RRT*, e é, portanto, uma alternativa para a abordagem de sistemas não-holonômicos complexos.

Considerando o interesse crescente da sociedade por CLMRs autônomos, este trabalho contribui para validar computacionalmente o uso de modelos generativos para este tipo de veículo. Enquanto o trabalho de Ichter, Harrison e Pavone (2017) utiliza apenas o CVAE, os resultados experimentais contribuem para mostrar a importância de considerar o fator β, durante o treinamento desses modelos. O controlador local elaborado por meio do aprendizado profundo por reforço com múltiplos objetivos também se mostrou eficaz para este tipo de veículo e o desempenho obtido foi comparável com métodos analíticos que geram caminhos de referência comprovadamente ótimos. O algoritmo RRT*, quando implementado com as variações propostas, obtém resultados que superam o RRT* com métricas e controladores que utilizam a modelagem do sistema.

A variação do RRT* proposta nesse trabalho é vantajosa sob a perspectiva do planejamento e controle em tempo real de CLMRs, em virtude do pequeno tempo de inferência do sinal de controle e da geração de amostras promissoras. Este método pode ser aplicado, por exemplo, em estacionamentos inteligentes, onde as condições do problema, como as vagas ocupadas, podem ser representadas por vetores de números binários. Pode-se considerar ainda o treinamento de diversas políticas para diferentes tipos de CLMRs.

Trabalhos futuros devem buscar a validação da abordagem proposta em experimentos com CLMRs reais, seja para verificar a transferência de política entre domínios ou para estudar a viabilidade da implementação do método em sistemas embarcados. Outros experimentos podem ser realizados com modelos mais complexos de CLMRs no PyBullet ou em outros simuladores de física. Nesse sentido, outra possibilidade é considerar um controlador preditivo baseado no modelo dinâmico do CLMR, a fim de melhorar a comparação da abordagem proposta com o método analítico. Outras avaliações computacionais são oportunas, principalmente para verificar o impacto dos diversos parâmetros que guiam os algoritmos de aprendizado. O presente trabalho também pode ser estendido por meio de análises teóricas acerca da utilização de aproximações, tanto para a função de distância quanto para o controlador local.

REFERÊNCIAS

ABADI, M. et al. TensorFlow: A system for large-scale machine learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. [S.l.: s.n.], 2016. p. 265–283.

ABIODUN, O. I. et al. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, Elsevier, v. 4, n. 11, p. e00938, 2018.

AKIBA, T. et al. Optuna: A next-generation hyperparameter optimization framework. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. [S.l.: s.n.], 2019. p. 2623–2631.

AMATO, N. M.; WU, Y. A randomized roadmap method for path and manipulation planning. In: *Proceedings of IEEE International Conference on Robotics and Automation*. [S.l.: s.n.], 1996. v. 1, p. 113–120.

ANDRYCHOWICZ, M. et al. Hindsight Experience Replay. *CoRR*, abs/1707.01495, 2017.

ARULKUMARAN, K. et al. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, v. 34, n. 6, p. 26–38, 2017.

BA, J. L.; KIROS, J. R.; HINTON, G. E. Layer normalization. *ArXiv e-prints*, p. arXiv–1607, 2016.

BAYDIN, A. G. et al. Automatic differentiation in machine learning: A survey. *Journal of Marchine Learning Research*, Microtome Publishing, v. 18, p. 1–43, 2018.

BELLMAN, R. Dynamic programming. *Science (New York, N.Y.)*, American Association for the Advancement of Science, v. 153, n. 3731, p. 34–37, 1966.

BELLMAN, R.; KALABA, R. E. *Dynamic Programming and Modern Control Theory*. [S.l.]: Citeseer, 1965. v. 81.

Ben-Ari, M.; MONDADA, F. Robots and Their Applications. In: Ben-Ari, M.; MONDADA, F. (Ed.). *Elements of Robotics*. Cham: Springer International Publishing, 2018. p. 1–20. ISBN 978-3-319-62533-1.

BOTTOU, L. Large-scale machine learning with stochastic gradient descent. In: *Proceedings of COMPSTAT'2010*. [S.l.]: Springer, 2010. p. 177–186.

BOTTOU, L. et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, v. 91, n. 8, p. 12, 1991.

BRUNNER, M.; BRÜGGEMANN, B.; SCHULZ, D. Hierarchical rough terrain motion planning using an optimal sampling-based method. *2013 IEEE International Conference on Robotics and Automation*, p. 5539–5544, 2013.

COUMANS, E.; BAI, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

DENG, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, IEEE, v. 29, n. 6, p. 141–142, 2012.

DING, S. et al. Evolutionary artificial neural networks: A review. *Artificial Intelligence Review*, Springer, v. 39, n. 3, p. 251–260, 2013.

DOLGOV, D. et al. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, v. 1001, n. 48105, p. 18–80, 2008.

DUCH, B. N.; ROSSETTI, F.; HAARBURGER, R. *Evolution of the EU Market Share of Robotics: Data and Methodology*. 2021. Https://publications.jrc.ec.europa.eu/repository/handle/JRC124114.

Dulac-Arnold, G. et al. Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis. *Machine Learning*, v. 110, n. 9, p. 2419–2468, 2021.

ECOFFET, A. et al. Go-explore: A new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

FATHI, E.; SHOJA, B. M. Deep neural networks for natural language processing. In: *Handbook of Statistics*. [S.l.]: Elsevier, 2018. v. 38, p. 229–316.

FUJIMOTO, S.; HOOF, H.; MEGER, D. Addressing function approximation error in actor-critic methods. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2018. p. 1587–1596.

GARCIA, C. E.; PRETT, D. M.; MORARI, M. Model predictive control: Theory and practice—A survey. *Automatica*, Elsevier, v. 25, n. 3, p. 335–348, 1989.

GOODFELLOW, I. et al. *Deep Learning*. [S.l.]: MIT press Cambridge, 2016. v. 1.

GU, J. et al. Recent advances in convolutional neural networks. *Pattern recognition*, v. 77, p. 354–377, 2018.

GUTIÉRREZ, P. A. et al. Ordinal regression methods: Survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, v. 28, n. 1, p. 127–146, 2015.

HAARNOJA, T. et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: *International Conference on Machine Learning*. [S.l.]: PMLR, 2018. p. 1861–1870.

HIGGINS, I. et al. Beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

HIJAZI, S. et al. Using convolutional neural networks for image recognition. *Cadence Design Systems Inc.: San Jose, CA, USA*, v. 9, 2015.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989.

HUANG, G. et al. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

HUSSAIN, R.; ZEADALLY, S. Autonomous cars: Research results, issues, and future challenges. *IEEE Communications Surveys & Tutorials*, v. 21, n. 2, p. 1275–1313, 2019.

ICHTER, B.; HARRISON, J.; PAVONE, M. Learning sampling distributions for robot motion planning. set. 2017.

JENSEN, J. L. W. V. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, Institut Mittag-Leffler, v. 30, n. none, p. 175–193, 1906.

JOHNSON, M. A.; MORADI, M. H. *PID Control*. [S.l.]: Springer, 2005.

KARAMAN, S.; FRAZZOLI, E. Incremental Sampling-based Algorithms for Optimal Motion Planning. *CoRR*, abs/1005.0416, 2010.

KARAMAN, S.; FRAZZOLI, E. Sampling-based optimal motion planning for non-holonomic dynamical systems. In: IEEE. *2013 IEEE international conference on robotics and automation*. [S.l.], 2013. p. 5041–5047.

KAVRAKI, L. et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, v. 12, n. 4, p. 566–580, 1996.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

KINGMA, D. P. et al. Semi-supervised learning with deep generative models. *Advances in neural information processing systems*, v. 27, 2014.

KINGMA, D. P.; WELLING, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, v. 60, n. 6, p. 84–90, 2017.

LAI, T.; RAMOS, F.; FRANCIS, G. Balancing global exploration and local-connectivity exploitation with rapidly-exploring random disjointed-Trees. In: *2019 International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2019. p. 5537–5543.

LAVALLE, S. M. et al. Rapidly-exploring random trees: A new tool for path planning. Ames, IA, USA, 1998.

LILLICRAP, T. P. et al. *Continuous Control with Deep Reinforcement Learning*. [S.l.]: arXiv, 2015.

LIN, L.-J. *Reinforcement Learning for Robots Using Neural Networks*. [S.l.]: Carnegie Mellon University, 1992.

LOSHCHILOV, I.; HUTTER, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

LYNCH, K. M.; PARK, F. C. *Modern robotics*. [S.l.]: Cambridge University Press, 2017.

MASRI, S. et al. Training neural networks by adaptive random search techniques. *Journal of Engineering Mechanics*, American Society of Civil Engineers, v. 125, n. 2, p. 123–132, 1999.

MATAS, J.; JAMES, S.; DAVISON, A. J. Sim-to-real reinforcement learning for deformable object manipulation. In: *Conference on Robot Learning*. [S.l.: s.n.], 2018. p. 734–743.

MATLAB. *Version 9.12.0 (R2022a)*. Natick, Massachusetts: The MathWorks Inc., 2022.

MITCHELL, W. C.; STANIFORTH, A.; SCOTT, I. *Analysis of Ackermann Steering Geometry*. [S.l.], 2006.

MNIH, V. et al. Human-level control through deep reinforcement learning. *nature*, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

NAIR, A. et al. Overcoming exploration in reinforcement learning with demonstrations. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2018. p. 6292–6299.

NASIR, J. et al. RRT*-SMART: A rapid convergence implementation of RRT*. *International Journal of Advanced Robotic Systems*, v. 10, n. 7, p. 299, 2013.

NIELSEN, M. A. *Neural Networks and Deep Learning*. [S.l.]: Determination press San Francisco, CA, USA, 2015. v. 25.

OUSSIDI, A.; ELHASSOUNY, A. Deep generative models: Survey. In: *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*. [S.l.]: IEEE, 2018. p. 1–8.

PASZKE, A. et al. Automatic differentiation in pytorch. 2017.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, v. 12, p. 2825–2830, 2011.

PENG, X. B. et al. Sim-to-real transfer of robotic control with dynamics randomization. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.]: IEEE, 2018. p. 3803–3810.

PETEREIT, J. et al. Application of hybrid A* to an autonomous mobile robot for path planning in unstructured outdoor environments. In: *ROBOTIK 2012; 7th German Conference on Robotics*. [S.l.: s.n.], 2012. p. 1–6.

PETIT, L.; DESBIENS, A. L. RRT-Rope: A deterministic shortening approach for fast near-optimal path planning in large-scale uncluttered 3D environments. In: *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.: s.n.], 2021. p. 1111–1118.

PITIS, S.; CHAN, H.; ZHAO, S. *Mrl: Modular RL*. [S.l.]: GitHub, 2020.

PITIS, S. et al. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In: *International Conference on Machine Learning*. [S.l.]: PMLR, 2020. p. 7750–7761.

PLAPPERT, M. et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, abs/1802.09464, 2018.

POLACK, P. et al. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. [S.l.: s.n.], 2017. p. 812–818.

PUTERMAN, M. L. Markov decision processes. *Handbooks in operations research and management science*, Elsevier, v. 2, p. 331–434, 1990.

RASCHKA, S.; PATTERSON, J.; NOLET, C. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information-an International Interdisciplinary Journal*, v. 11, n. 4, p. 193, 2020.

RECHT, B. et al. Do ImageNet Classifiers Generalize to ImageNet? In: *Proceedings of the 36th International Conference on Machine Learning*. [S.l.]: PMLR, 2019. p. 5389–5400. ISSN 2640-3498.

REDA, D.; TAO, T.; van de Panne, M. Learning to locomote: Understanding how environment design matters for deep reinforcement learning. In: *Motion, Interaction and Games*. [S.l.: s.n.], 2020. p. 1–10.

REEDS, J.; SHEPP, L. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, v. 145, n. 2, p. 367–393, 1990.

ROBBINS, H. E. A Stochastic Approximation Method. *Annals of Mathematical Statistics*, v. 22, p. 400–407, 2007.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

SCHAUL, T. et al. Universal value function approximators. In: *International Conference on Machine Learning*. [S.l.]: PMLR, 2015. p. 1312–1320.

SCHMIDHUBER, J. Who invented backpropagation. *More in [DL2]*, 2014.

SCHULMAN, J. et al. Trust region policy optimization. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2015. p. 1889–1897.

SCHULMAN, J. et al. Proximal policy optimization algorithms. *arXiv e-prints*, p. arXiv–1707, 2017.

SHI, X.; CAO, W.; RASCHKA, S. Deep neural networks for rank-consistent ordinal regression based on conditional probabilities. *arXiv preprint arXiv:2111.08851*, 2021.

SILVER, D. et al. Deterministic policy gradient algorithms. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2014. p. 387–395.

SUTSKEVER, I. et al. On the importance of initialization and momentum in deep learning. In: DASGUPTA, S.; MCALLESTER, D. (Ed.). *Proceedings of the 30th International Conference on Machine Learning*. Atlanta, Georgia, USA: PMLR, 2013. (Proceedings of Machine Learning Research, v. 28), p. 1139–1147.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. [S.l.]: MIT press, 2018.

SUTTON, R. S. et al. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, v. 12, 1999.

TAN, J. et al. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Tesla Inc. *Artificial Intelligence & Autopilot*. 2022. Https://www.tesla.com/AI.

THEIS, L.; OORD, A. van den; BETHGE, M. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

THOMAS, M.; JOY, A. T. *Elements of Information Theory*. [S.l.]: Wiley-Interscience, 2006.

THRUN, S. et al. Stanley: The robot that won the DARPA grand challenge. *Journal of field Robotics*, Wiley Online Library, v. 23, n. 9, p. 661–692, 2006.

THRUN, S.; SCHWARTZ, A. Issues in using function approximation for reinforcement learning. In: *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*. [S.l.: s.n.], 1993. v. 6, p. 1–9.

TODOROV, E.; EREZ, T.; TASSA, Y. Mujoco: A physics engine for model-based control. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.: s.n.], 2012. p. 5026–5033.

VECERÍK, M. et al. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR*, abs/1707.08817, 2017.

WATKINS, C. J.; DAYAN, P. Q-learning. *Machine learning*, Springer, v. 8, n. 3-4, p. 279–292, 1992.

WHITE, D. J. A Survey of Applications of Markov Decision Processes. *The Journal of the Operational Research Society*, Palgrave Macmillan Journals, v. 44, n. 11, p. 1073–1096, 1993. ISSN 01605682, 14769360.

WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, v. 8, n. 3, p. 229–256, 1992.

WILLIANMS, R. Toward a theory of reinforcement-learning connectionist systems. *Technical Report NU-CCS-88-3, Northeastern University*, 1988.

WITZE, A. et al. NASA has launched the most ambitious Mars rover ever built: Here's what happens next. *Nature*, Nature, v. 584, n. 7819, p. 15–16, 2020.

ZOU, F. et al. A sufficient condition for convergences of adam and rmsprop. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2019. p. 11127–11135.

APÊNDICE A – Implementações

Esta seção apresenta detalhes da implementação de diversos componentes abordados neste trabalho. O código utilizado está contido no seguinte repositório:

GitHub: RRT-ML

A.1 Aprendizado por Reforço

A implementação do algoritmo de aprendizado por reforço utilizou uma série de técnicas tradicionais na literatura. Os valores de alguns parâmetros também foram escolhidos com base nos artigos estudados.

Todas as observações recebidas pelo agente são normalizadas, de forma que, os componentes do vetor de estado armazenados no *replay buffer* apresentam média nula e variância unitária. Utilizou-se também *layer normalization* (BA; KIROS; HINTON, 2016) e regularização L_2, com fator de multiplicação igual a 0,1; em todas as redes neurais do algoritmo. O valor máximo para o gradiente foi estabelecido como 5, uma prática comum em diversas implementações. O valor do coeficiente τ de atualização das redes neurais *target* foi definido como 0,95. O ruído adicionado às ações para auxiliar a exploração do ambiente é gaussiano com média nula e variância unitária. O valor amostrado desta distribuição é então multiplicado por $\alpha = 0{,}1$, antes de ser somado ao valor de saída da política. O espaço de ações é $\{a_t \in \mathbb{R}^2 | -1 \leq a_t \leq 1\}$, de modo que $a_t = [1 \;\; 1]$ corresponde à aplicação do valor máximo de referência nos motores de tração e direcionamento.

Nos primeiros 2500 instantes de tempo do treinamento não ocorre otimização das redes neurais, isto porque não há um número grande de transições no *replay buffer* e, portanto, há grande correlação entre as informações presentes na memória. Até o instante de tempo 5000 o agente seleciona as ações de forma aleatória e uniforme, uma forma de facilitar o acúmulo de experiências nos momentos iniciais. O mecanismo de exploração do algoritmo MEGA, que atua no momento que o objetivo de mínima densidade é atingido, modifica a constante α de ruído para 0,2, visando auxiliar a exploração do espaço de estados nesta região. O processo de seleção deste objetivo de mínima densidade consiste em amostrar 500 objetivos atingidos de p_{ag} e então verificar o valor de $\hat{p}_{ag}(g_a)$ para cada um. O modelo $\hat{p}_{ag}$ é um estimador KDE com largura de banda 0,2, otimizado a cada 100 instantes de tempo com 10000 amostras do histórico de objetivos atingidos, p_{ag}. As transições artificiais criadas e adicionadas ao *replay buffer* seguem a proporção enunciada no fim da Seção 3.7.

O código usado nos experimentos têm como base a biblioteca MRL (PITIS; CHAN; ZHAO, 2020), que, por sua vez, implementa as redes neurais através do software de apren-

dizado profundo PyTorch (PASZKE et al., 2019). O estimador KDE utiliza a biblioteca Scikit-learn (PEDREGOSA et al., 2011).

A.2 **Modelo Ordinal**

O modelo de regressão ordinal é uma rede PMC implementada no PyTorch e otimizada com o algoritmo Adam. Todas as camadas da rede neural utilizam a função de ativação GeLU, com exceção da camada de saída, que usa apenas a função identidade. A diferença prática de implementação entre o modelo de regressão usual e o ordinal (CORN) reside num critério de otimização particular (SHI; CAO; RASCHKA, 2021), que exige um vetor de saída com dimensão igual ao número de classes. A busca por hiper-parâmetros é realizada por meio da biblioteca *Optuna* (AKIBA et al., 2019).

A.3 **RRT***

O algoritmo RRT* é implementado na linguagem *Python*. Ao invés de representar a árvore de exploração por um conjunto de vértices e arestas, este trabalho implementa apenas o conjunto de configurações e representa as arestas através da referência à outros vértices, o que é possível já que cada configuração têm apenas uma antecessora. Os custos relacionados ao vértice inicial e ao antecessor são armazenados no próprio objeto que representa uma configuração.

O limiar que indica quando uma configuração está próxima, e portanto faz parte do conjunto $\mathcal{V}_{prox}$, é 1m, para o método analítico, que considera distâncias de RS. Para o controlador de aprendizado por reforço, isto é equivalente a, aproximadamente, 240 passos de simulação.

Na fase de amostragem de uma configuração aleatória, há 5% de chance do vértice aleatório ser a configuração final, $\vec{g_f}$. Este procedimento é comum em diversas implementações (MATLAB, 2022).

APÊNDICE B – Pseudocódigos

Esta seção apresenta algumas funções ou sub-rotinas, sob a forma de pseudocódigos, cujas implementações foram omitidas no texto principal.

A notação $x \sim U(0,1)$ indica que à variável x é atribuído o valor de uma amostra da distribuição uniforme padrão. A notação $\vec{g} \sim U(\mathcal{G}_{livre})$ indica que $\vec{g}$ é uma configuração amostrada uniformemente do espaço de configurações livres de obstáculos.

B.1 Configuração Aleatória

Esta seção apresenta os algoritmos relacionados à obtenção de configurações aleatórias para o sistema robótico. O Algoritmo 7 é utilizado para o RRT* tradicional com amostragem uniforme. O Algoritmo 8 é empregado quando há um modelo generativo capaz de gerar amostras de configurações para o robô. Assume-se que o espaço latente escolhido é uma distribuição normal multivariada e que o decodificador (p) é determinístico.

Algoritmo 7 Amostrar Configuração

função AMOSTRARCONFIGURACAO($\mathcal{G}_{livre}$)
    $\vec{g}_{aleatoria} \sim U(\mathcal{G}_{livre})$
    retorna $\vec{g}_{aleatoria}$
fim função

Algoritmo 8 Amostrar Configuração Ótima

função AMOSTRARCONFIGURACAOOTIMA($\mathcal{G}_{livre}, \alpha, p, \vec{y}$)
    $r \sim U(0,1)$
    se $r < \alpha$ **então**
        $\vec{z} \sim N(\vec{0}, \vec{1})$
        $\vec{g}_{aleatoria} = p(\vec{z}, \vec{y})$
    caso contrário
        $\vec{g}_{aleatoria} \sim U(\mathcal{G}_{livre})$
    fim se
    retorna $\vec{g}_{aleatoria}$
fim função

B.2 Configurações Próximas

Esta seção contém as sub-rotinas relacionadas à obtenção de uma ou mais configurações próximas, de acordo com uma métrica de distância ou função de custo. O Algoritmo 9 obtém o vértice mais próximo da configuração amostrada. O Algoritmo 10 obtém o conjunto de configurações próximas, de acordo com um limiar ϵ_g. O Algoritmo 11 retorna o vértice intermediário que implica em menor custo para a nova configuração $\vec{g}_{nova}$.

Algoritmo 9 Configuração Mais Próxima

```
1: função ConfigMaisProxima(g⃗_aleatoria, V, ρ)
2:     d_min ← ∞
3:     para todo g⃗_j ∈ V faça
4:         d ← ρ(g⃗_j, g⃗_aleatoria)
5:         se então d < d_min
6:             d_min ← d
7:             g⃗_prox ← g⃗_j
8:         fim se
9:     fim para
10:    retorna g⃗_prox
11: fim função
```

Algoritmo 10 Configurações Próximas

```
1: função ConfigProximas(g⃗_nova, V, ρ, ε_g)
2:     V_prox ← ∅
3:     para todo g⃗_j ∈ V faça
4:         se ρ(g⃗_j, g⃗_nova) < ε_g então
5:             V_prox ← V_prox ∪ {g⃗_j}
6:         fim se
7:     fim para
8:     retorna G_prox
9: fim função
```

B.3 Outros

Esta seção apresenta outros pseudocódigos relacionados às árvores de exploração.

Algoritmo 11 Configuração de Menor Custo

```
1: função ConfigMenorCusto(V_prox, g_nova, C)
2:     c_min ← ∞
3:     para todo g_j ∈ V faça
4:         τ⟨g_j, g_nova⟩ ← Simular(g_prox, g_aleatoria, U, Δt)
5:         c ← C(g_j) + C(τ⟨g_j, g_nova⟩)
6:         se c < c_min e LivreDeColisoes(τ⟨g_j, g_nova⟩) então
7:             c_min ← c
8:             g_min ← g_j
9:         fim se
10:    fim para
11:    retorna g_min
12: fim função
```

Algoritmo 12 Religação

```
1: função Religar(g_nova, E, V_prox)
2:     para todo g_j ∈ V_prox faça
3:         τ⟨g_nova, g_final⟩ ← Simular(g_nova, g_j, U, Δt)
4:         se g_final = g_nova então
5:             c ← C(g_nova) + C(τ⟨g_nova, g_j⟩)
6:             se c < C(g_j) e LivreDeColisoes(τ⟨g_nova, g_j⟩) então
7:                 E ← E \ {τ⟨g_j', g_j⟩}
8:                 E ← E ∪ {τ⟨g_nova, g_j⟩}
9:             fim se
10:        fim se
11:    fim para
12:    retorna E
13: fim função
```

APÊNDICE C – Hiper-parâmetros

Esta Seção contém informações relacionadas à busca pelos hiper-parâmetros ideais. A Seção C.1 apresenta os resultados referentes ao módulo de controle. As configurações geradas pelos diversos modelos treinados na obtenção do módulo de planejamento são incluídas na Seção C.2. A busca por parâmetros do modelo ordinal e da abordagem analítica são abordadas nas Seções C.3 e C.4, respectivamente.

C.1 Módulo de Controle

A Figura 80 apresenta as curvas de recompensa acumulada (média nos episódios de teste) e de entropia da distribuição empírica de objetivos atingidos, para cada conjunto de parâmetros experimentado.

C.2 Módulo de Planejamento

As Figura 81, 82, 83 e 84 mostram as configurações geradas pelos diferentes modelos β-CVAE otimizados durante a busca por hiper-parâmetros.

C.3 Modelo Ordinal

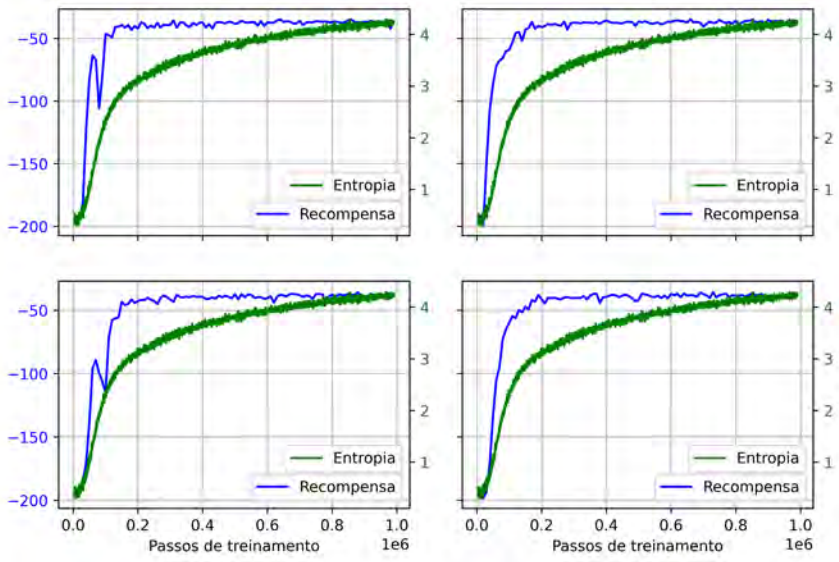
A busca por hiper-parâmetros para o modelo ordinal é realizada de forma inteligente pelo software Optuna, com base nos possíveis valores indicados na Tabela 7. As melhores escolhas para os parâmetros, após aproximadamente 50 iterações de busca, também são apresentadas na Tabela 7.

Tabela 7 - Busca por hiper-parâmetros para o modelo ordinal

Parâmetro	Valores	Valor escolhido
Número de estimadores (*ensemble*)	$\{10; 15; \ldots; 35; 40\}$	10
Número de camadas escondidas	$\{1; 2; 3\}$	3
Tamanho do mini-lote	$\{16; 32; \ldots; 496; 512\}$	128
Número de neurônios artificiais por camada	$\{16; 32; \ldots; 496; 512\}$	256

Fonte: O autor, 2022.

Figura 80 - Curvas de recompensa e entropia



Fonte: O autor, 2022.

Tabela 8 - Comparação dos hiper-parâmetros para o módulo de controle

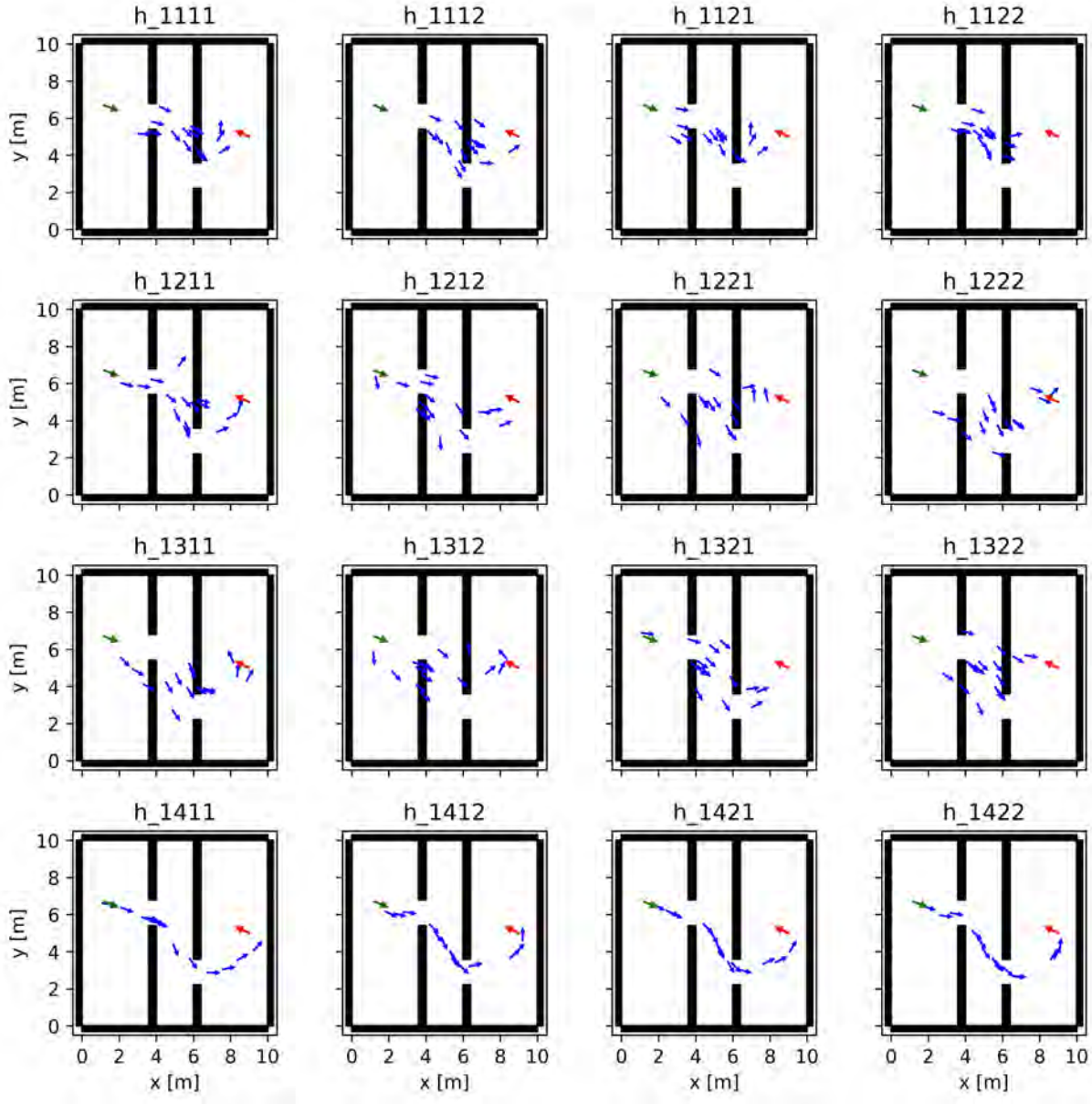
Parâmetro	Valores	Valor escolhido
Diâmetro das curvas de RS	{2,2; 2,3; 2,4; 2,5}	2,3
Peso dos desvios de orientação no custo	{0,01; 0,05*; 0,10}	0,05
Peso da variação do sinal de controle no custo	{0,001; 0,01*}	0,01
Passo de simulação do modelo	{0,01*; 0,1}	0,01
Horizonte de predição	{5, 10, 20, 50, 100*}	100

Legenda: O asterisco indica o melhor valor.
Fonte: O autor, 2022.

C.4 **Solução Analítica**

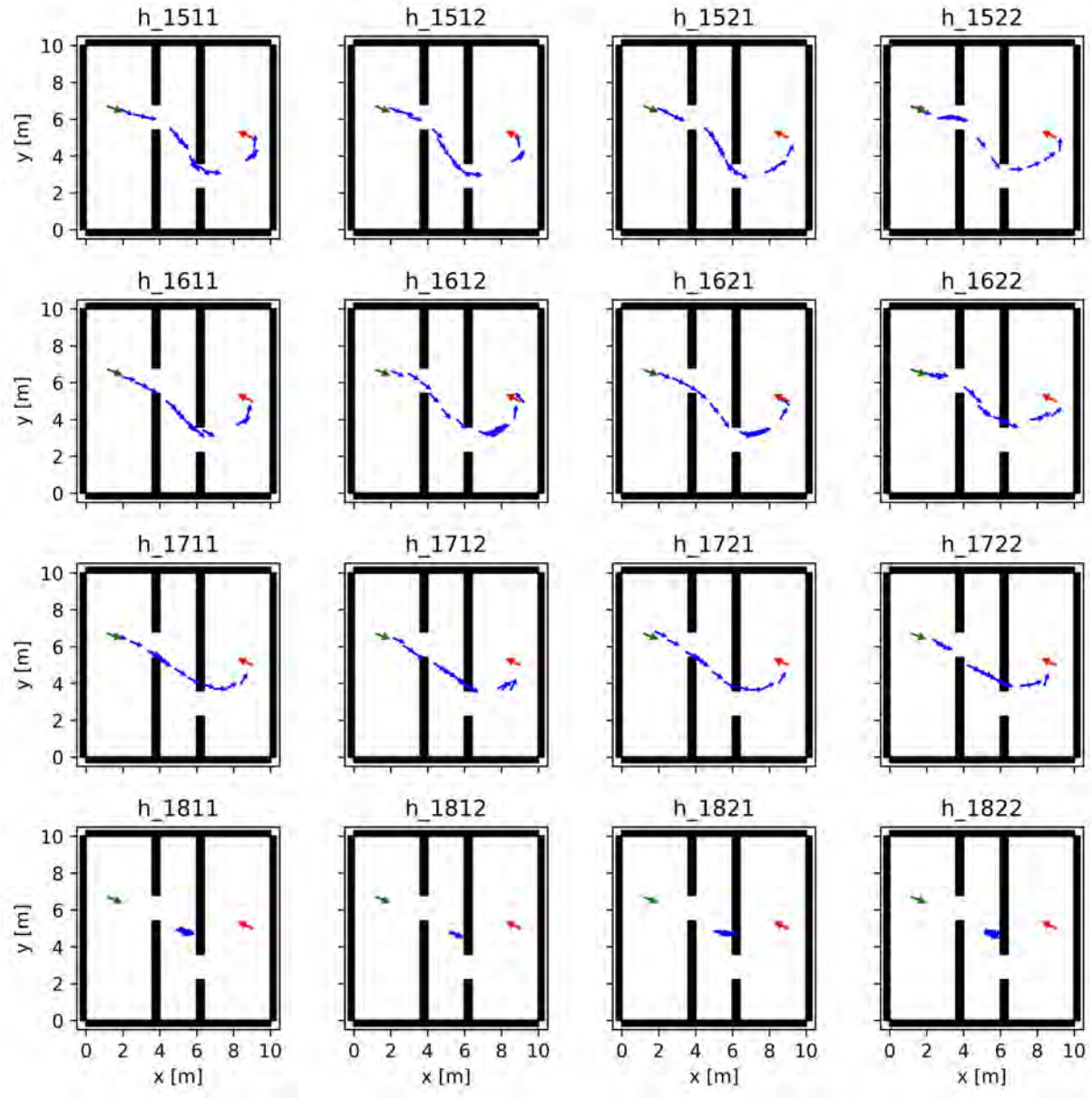
Para a abordagem analítica, que combina curvas de RS com o CPM, realizou-se uma busca com combinações aleatórias dos parâmetros apresentados na Tabela 8.

Figura 81 - Amostras de configurações geradas pelos modelos (parte 1)



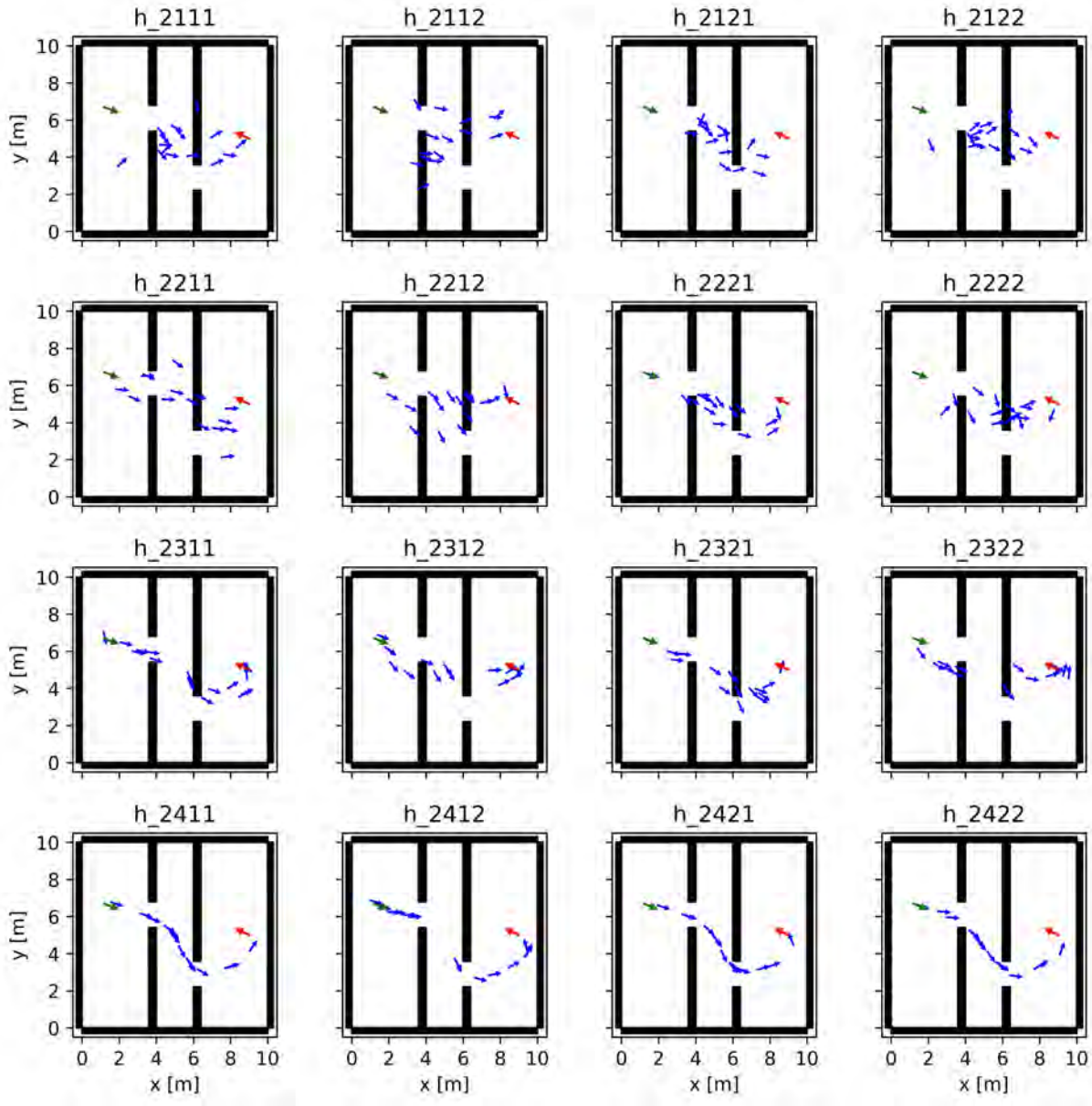
Fonte: O autor, 2022.

Figura 82 - Amostras de configurações geradas pelos modelos (parte 2)



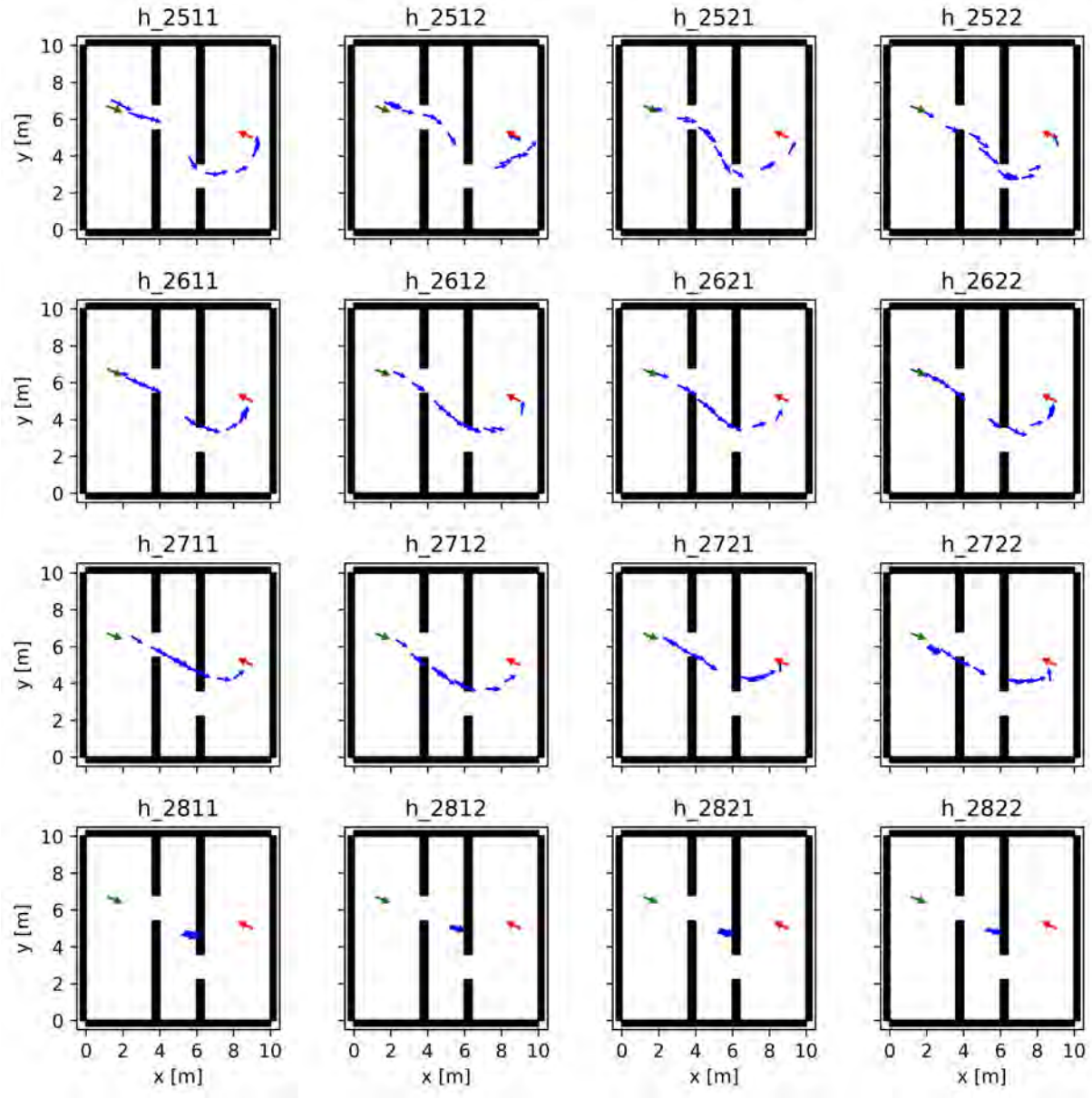
Fonte: O autor, 2022.

Figura 83 - Amostras de configurações geradas pelos modelos (parte 3)



Fonte: O autor, 2022.

Figura 84 - Amostras de configurações geradas pelos modelos (parte 4)

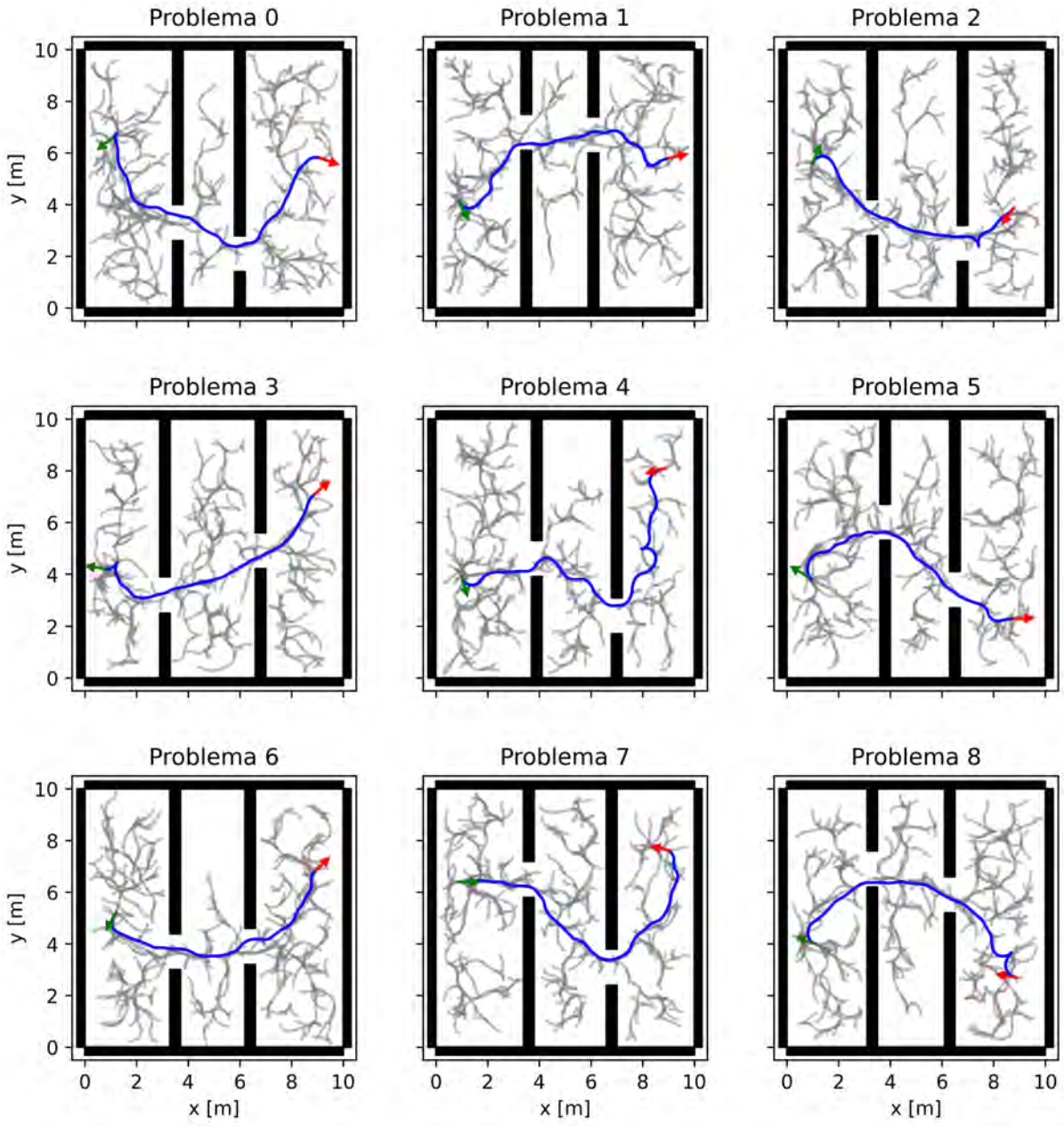


Fonte: O autor, 2022.

APÊNDICE D – Resultados Adicionais

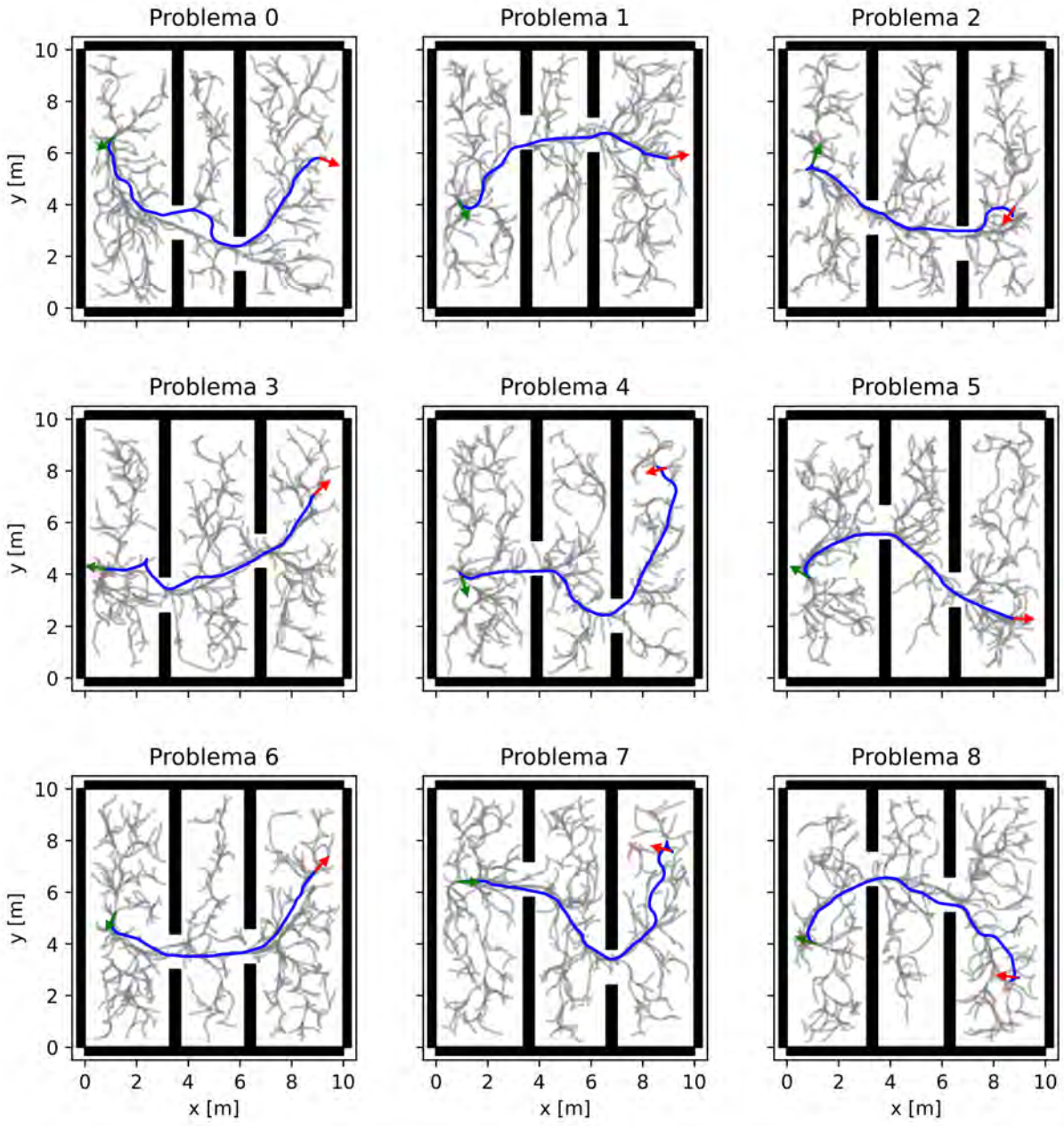
Esta seção apresenta alguns resultados experimentais omitidos no texto principal. As árvores de exploração e as soluções geradas pelas variações do RRT* nos 9 problemas propostos são apresentadas nas Figuras 85 e 86. O seguimento das trajetórias encontradas pode ser visto nas Figuras 87 e 88.

Figura 85 - Árvores de exploração geradas pelo RRT*+ApM



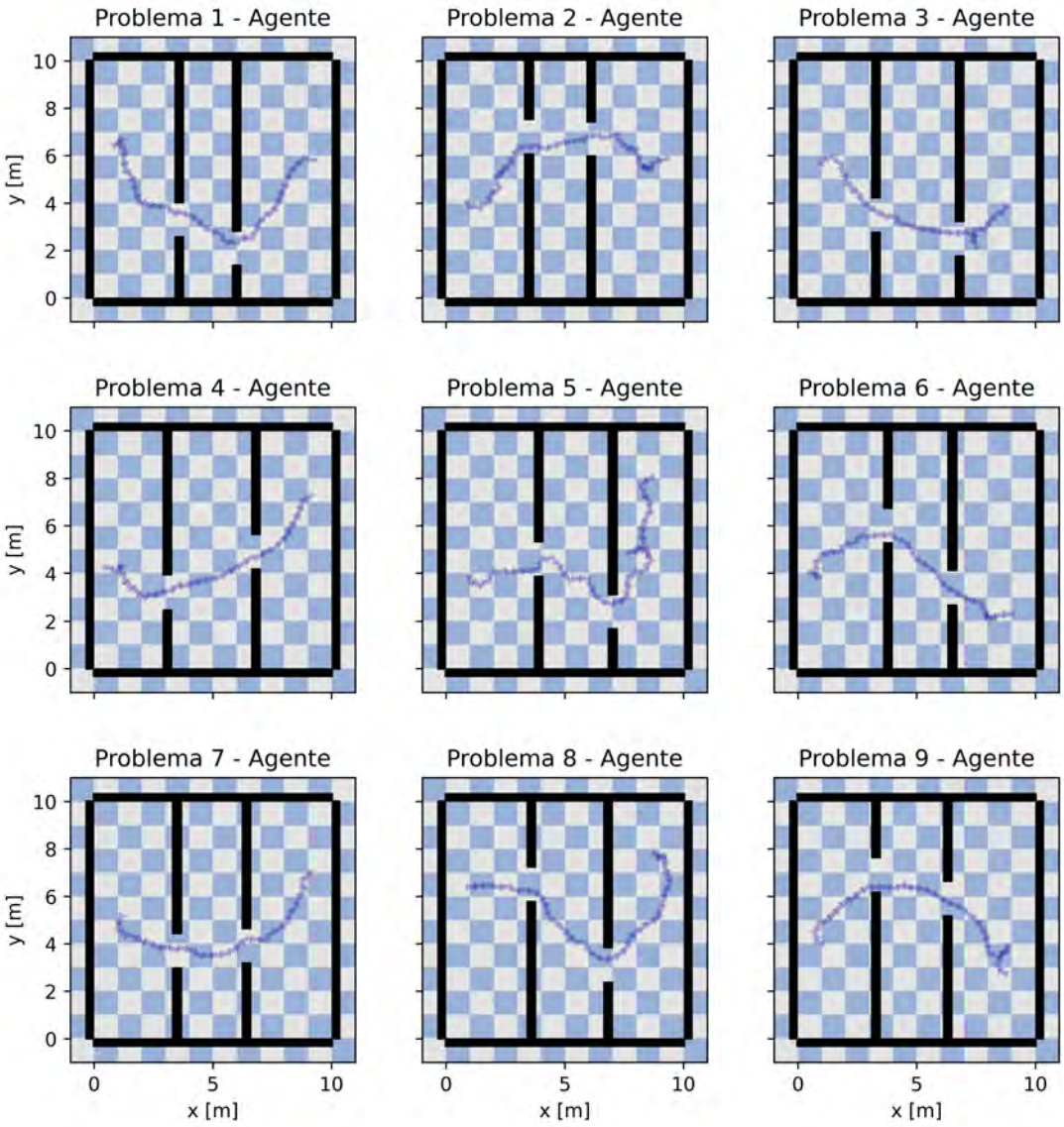
Fonte: O autor, 2022.

Figura 86 - Árvores de exploração geradas pelo RRT*+RS+CPM



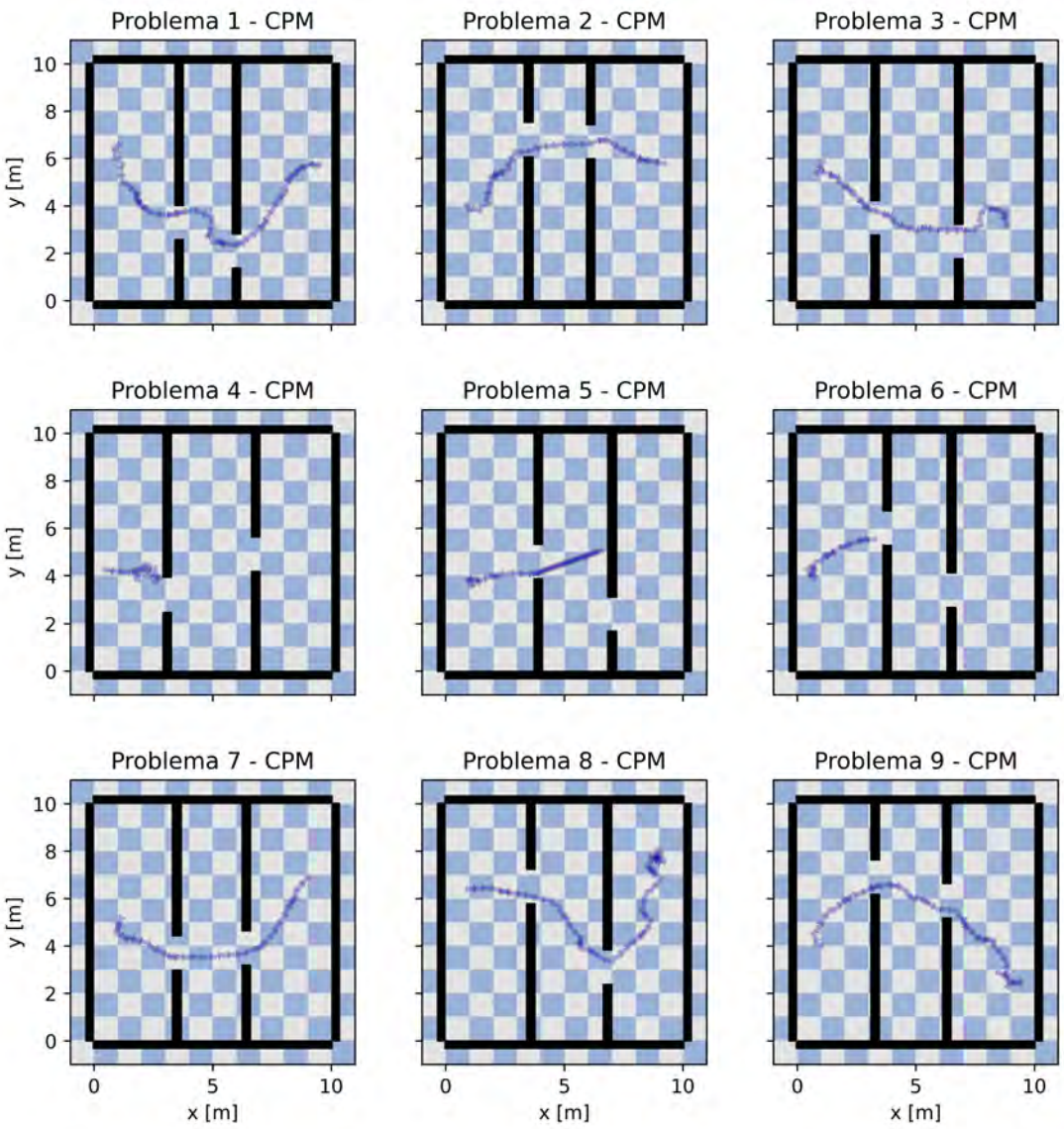
Fonte: O autor, 2022.

Figura 87 - Trajetórias realizadas pelo agente MEGA



Fonte: O autor, 2022.

Figura 88 - Trajetórias realizadas pelo CPM



Fonte: O autor, 2022.