

Lucas Pereira Cotrim - 8989092

Marcos Menon José - 8989112

CONTROLE DE ROBÔ MANIPULADOR COM APRENDIZADO POR REFORÇO

São Paulo

2019

Lucas Pereira Cotrim - 8989092
Marcos Menon José - 8989112

CONTROLE DE ROBÔ MANIPULADOR COM APRENDIZADO POR REFORÇO

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Graduação em Engenharia.

Universidade de São Paulo – USP
Escola Politécnica
Engenharia Mecatrônica

Orientador: Eduardo Lobo Lustosa Cabral

São Paulo
2019

Agradecimentos

Gostaríamos de agradecer ao Prof. Dr. Eduardo Lobo Lustosa Cabral pela valiosa orientação e auxílio ao longo deste trabalho e ao Departamento de Mecatrônica da Escola Politécnica da USP pela disponibilização do robô KUKA-KR16 nas etapas iniciais do projeto. Agradecemos à família e aos amigos, pelo apoio incondicional sem o qual esse trabalho não seria possível.

Resumo

Desde o estabelecimento da robótica em aplicações industriais, a programação de robôs manipuladores envolve o processo repetitivo e demorado de especificação manual de uma trajetória fixa, o que resulta em tempo ocioso de produção e na necessidade de reprogramação completa para diferentes tarefas a serem executadas pelo robô. A tendência de aumento das aplicações da robótica em ambientes não estruturados requer controladores inteligentes e reativos, devido respectivamente à imprevisibilidade do ambiente e a medidas de segurança. Este trabalho apresenta uma arquitetura de controle de manipulador robótico para posicionamento do efetuador em ambientes com obstáculos, sem conhecimento prévio de suas posições ou da dinâmica do braço. O controlador é treinado por métodos da Aprendizado por Reforço, apropriados para aplicações da robótica devido à sua formulação geral e ao aprendizado de dinâmicas complexas diretamente por interação com o ambiente. O sensoriamento é realizado por meio de uma câmera que captura a cada instante o estado do sistema. Por fim, o controlador é validado em ambiente de simulação gráfica do robô industrial KUKA-KR16 e os resultados são comparados a métodos tradicionais.

Palavras-chaves: Robótica; Inteligência Artificial; Aprendizado por Reforço; Redes Neurais Profundas Convolucionais; Visão Computacional

Abstract

Since the establishment of robotics in industrial applications, industrial robot programming involves the repetitive and time-consuming process of manually specifying a fixed trajectory, which results in machine idle time in terms of production and the necessity of completely reprogramming the robot when changing the task to be executed. The increasing number of robotics applications in unstructured environments requires not only intelligent but also reactive controllers, due to the unpredictability of the environment and safety measures respectively. This paper presents a robotic manipulator control architecture for positioning the end effector in obstacle-filled environments, with no previous knowledge of the obstacles' positions or of the robot arm dynamics. The controller is trained by methods of Reinforcement Learning, appropriate for applications in robotics thanks to their general formulation and capability of learning complex dynamics directly from environmental interaction. The sensory system consists of a camera that captures real-time images of the system's current state. The controller's performance is evaluated through simulations of the KUKA-KR16 industrial robot and the results are compared to traditional methods.

Key-words: Robotics, Artificial Intelligence, Reinforcement Learning, Deep Convolutional Neural Networks, Computer Vision.

Listas de ilustrações

Figura 1 – Robô humanoide NEXTTAGE em tarefa de dobrar camiseta (Tsurumine et al. 2018).	24
Figura 2 – Dois Braços Robóticos aprendendo a abrir uma porta com o algoritmo NAF assíncrono (Gu, S et al, 2016).	25
Figura 3 – 6 Graus de Liberdade do Robô KUKA-KR16 (KUKA ROBOTICS, 2003)	27
Figura 4 – Notação Denavit-Hartenberg (Cabral, 2019)	28
Figura 5 – Volume de trabalho do Robô KUKA-KR16 (Fonte Manual do Fabricante).	28
Figura 6 – Sistemas de coordenadas dos ligamentos segundo D-H (PIOTROWSKI N; BARYLSKI A, 2014).	29
Figura 7 – Representação gráfica da cadeia cinemática do robô (Fonte: Autores). .	30
Figura 8 – Representação gráfica do robô com visualização dos ligamentos (Fonte: Autores).	30
Figura 9 – Visualização do Método do Campo Potencial (Fonte: SIEGWART, R, 2004).	31
Figura 10 – Visualização do efeito do campo potencial para robô KUKA-KR16 (Fonte: Autores).	32
Figura 11 – Algoritmo <i>Q-Learning</i> (SUTTON, 2017).	36
Figura 12 – Algoritmo REINFORCE (SUTTON, 2017).	38
Figura 13 – Diagrama simplificado de arquitetura de treino do controlador por aprendizado por reforço (fonte: Autores).	42
Figura 14 – Diagrama de posicionamento de câmeras para aquisição de imagens durante simulação (fonte: autores).	43
Figura 15 – Estado inicial do sistema capturado pelo sistema de câmeras após pré processamento (fonte: autores).	44
Figura 16 – 9 das 64 ativações da primeira camada convolucional da VGG16 para a configuração inicial do robô, convertidas para imagens em escala cinza (fonte: autores).	44
Figura 17 – Diagrama de posicionamento de câmeras superior e lateral (fonte: autores). .	45
Figura 18 – Estado inicial do sistema (24x24x3x2) capturado por câmeras superior e lateral após pré-processamento (fonte: autores).	46
Figura 19 – Visualização de estado terminal devido ao alcance de fim de curso da articulação rotativa A3 do robô (fonte: autores).	51
Figura 20 – Diagrama explicativo da função recompensa escolhida (fonte: autores). .	55
Figura 21 – Possíveis arquiteturas de rede DQN para aproximação de função $Q(\mathbf{s}, \mathbf{a})$ (fonte: autores)	59

Figura 22 – Cinco jogos de Atari 2600 disponíveis no ALE: (Da esquerda para a direita) Pong, Breakout, Space Invaders, Seaquest e Beam Rider (MNIH, 2013)	62
Figura 23 – Simulador MORSE em software ROS de robô móvel para planejamento de trajetória (YAN, 2017)	63
Figura 24 – Visualização de ambiente de pêndulo invertido sobre carro em plataforma <i>Open AI Gym</i> (fonte: autores)	63
Figura 25 – Robô <i>Turtlebot</i> com sensor LIDAR em ambientes de teste implementados (ZAMORA et al, 2013)	64
Figura 26 – Tabela de projetos simplificados desenvolvidos para teste, separados por simplificação adotada e algoritmo implementado (fonte: autores) . .	67
Figura 27 – Diagrama de classes da estrutura de armazenamento de transições (fonte: autores).	68
Figura 28 – Diagrama de classes da estrutura de armazenamento de épocas do treino (fonte: autores).	69
Figura 29 – Diagrama de classes da estrutura de armazenamento de redes DQN (fonte: autores).	69
Figura 30 – Diagrama de atividades representando fluxo de execução do arquivo DQNKuka_image.m, com seções de inicialização (azul), criação de estruturas de armazenamento (laranja), execução de algoritmo DQN (cinza) e visualização de desempenho (verde) (fonte: autores).	70
Figura 31 – Trecho de arquivo kr16_2.urdf responsável pela determinação das propriedades das três primeiras articulações rotativas do robô (fonte: autores). .	71
Figura 32 – Arquitetura parão da rede convolucional VGG16 (fonte: Ferguson (2017)).	72
Figura 33 – Estrutura de camadas de redes da família ConvNet (VGG16 corresponde à coluna D) fonte: Simonyan e Zisserman (2014)	73
Figura 34 – Ambiente de simulação do projeto PGRobotArmR em estado de episódio 38 do treino. Posição desejada (verde), obstáculo (vermelho), braço robótico (azul), ação executada no instante de tempo (ponto preto) (fonte: autores).	75
Figura 35 – Diagrama esquemático da estrutura da rede neural densa que implementa política de ações $\pi_\theta(a s)$ em projeto PGRobotArmR (fonte: autores).	76
Figura 36 – Valores médios normalizados de recompensas obtidas pelo agente em função da época do treino para as funções recompensa r_1 e r_2 (fonte: autores).	78

Figura 37 – Ambiente de simulação do projeto QLearningRobotArmR em estado de episódio 20 do treino. Posição desejada (verde), obstáculo (vermelho), braço robótico (azul), ação executada no instante de tempo (ponto preto) (fonte: autores).	78
Figura 38 – Valores médios normalizados de recompensas obtidas pelo agente em função da época do treino para as funções recompensa r_1 e r_2 (fonte: autores).	80
Figura 39 – Agente após treino em configuração com posição de destino fora de alcance (fonte: autores).	81
Figura 40 – Ambiente de simulação do projeto PGRobotArmRR em estado de episódio 8 do treino. Posição desejada (verde), obstáculo (vermelho), braço robótico (azul), ação executada no instante de tempo (ponto preto) (fonte: autores).	81
Figura 41 – Diagrama esquemático da estrutura da rede neural densa que implementa política de ações $\pi_\theta(\mathbf{a} \mathbf{s})$ em projeto PGRobotArmRR (fonte: autores).	82
Figura 42 – Valores médios normalizados de recompensas obtidas pelo agente em função da época do treino para as funções recompensa r_2 (vermelho) e r_3 (azul) (fonte: autores).	84
Figura 43 – Ambiente de simulação do projeto QLearningRobotArmRR em estado de episódio 2 do treino. Posição desejada (verde), obstáculo (vermelho), braço robótico (azul), ação executada no instante de tempo (pontos pretos) (fonte: autores).	84
Figura 44 – Valores médios normalizados de recompensas obtidas pelo agente em função da época do treino para as funções recompensa r_2 e r_3 ao longo de três treinos. Vales correspondem a colisões com obstáculo (fonte: autores).	86
Figura 45 – Trajetórias encontradas pelo agente ao longo do treino. Algoritmo covergiu para matriz Q que executa trajetória superior, uma vez que verifica-se apenas colisão com efetuador do robô (fonte: autores).	87
Figura 46 – Ambiente de simulação gráfica do projeto PGKuka. Comparação entre trajetória gerada por política de ações inicial (a) e trajetória após treino do agente (b) (fonte: autores)	87
Figura 47 – Diagrama esquemático da estrutura da rede neural densa que implementa política de ações $\pi_\theta(\mathbf{a} \mathbf{s})$ em projeto PGKuka (fonte: autores). .	88
Figura 48 – Trajetórias obtidas pelo mesmo agente após treino inicial (a) e após treino de mesma política de ações sob nova configuração de posição de destino e de obstáculo (fonte: autores)	90

Figura 49 – Curvas de recompensas médias normalizadas em função de época para treino inicial (vermelho) e para treino após substituição de posições do <i>setpoint</i> e do <i>obstáculo</i> (azul) (fonte: autores).	90
Figura 50 – Distribuição de probabilidades $\pi_\theta(\mathbf{a} \mathbf{s}_0)$ sobre espaço de ações em estado inicial ao longo do treino em épocas 1, 4, 8 e 12 (fonte: autores)	91
Figura 51 – Trajetórias <i>greedy</i> , ou seja, sem escolha de ações aleatórias para exploração, geradas por agente no início do treino e em época 19.(fonte: autores)	92
Figura 52 – Diagrama esquemático da estrutura da rede neural densa que implementa função valor de estados e ações $Q_\theta(\mathbf{s}, \mathbf{a})$ em projeto DQNKuka (fonte: autores).	92
Figura 53 – Trajetória gerada por agente DQN após treino (a) e gráfico de recompensas médias em função de épocas (b) para projeto DQNKuka (fonte: autores)	94
Figura 54 – Valores de $Q_\theta(\mathbf{s}_0, \mathbf{a})$ no estado inicial \mathbf{s}_0 após treino do agente. Ação de maior valor é $\mathbf{a}_{190} = (-1, 0, 1, 1, 1)$ (fonte: autores).	95
Figura 55 – Trajetória gerada por agente DQN após treino (a) e gráfico de recompensas médias em função de épocas (b) para projeto DQNKuka com obstáculo desconhecido entre posição inicial e de destino (fonte: autores) .	96
Figura 56 – Trajetórias geradas por agente DQN após treino (a),(b) e gráfico de recompensas médias (c) para projeto DQNKuka_RandomPositions.(fonte: autores)	97
Figura 57 – Trajetória percorrida por agente durante o treino (a), trajetória após treino em configuração aleatória (b) e gráfico de recompensas médias em função de época do treino (c) (fonte: autores)	99
Figura 58 – Diagrama esquemático de possíveis trajetórias percorridas por agentes após treino com diferentes fatores de desconto γ (fonte: autores).	103
Figura 59 – Captura de tela de jogo <i>Pong</i> em simulação do OpenAI Gym (fonte: autores).	104
Figura 60 – Desempenho de dois agentes do projeto QLearningRobotArmRR ao longo do treino com fatores de desconto $\gamma = 0.01$ (azul) e $\gamma = 0.9$ (amarelo) (fonte: autores).	105
Figura 61 – Trajetórias simuladas na primeira (a) e última (b) épocas do treino no projeto DQNKukaImage (fonte: autores)	109
Figura 62 – Trajetória obtida após convergência e gráfico de desempenho do agente ao longo do treino (fonte: autores)	109
Figura 63 – Diagrama esquemático do fluxo de algoritmo Ator-Crítico (SUTTON, 1998).	111

Figura 64 – Diagrama esquemático de arquitetura proposta para treino com múltiplos agentes (fonte: autores) 113

Lista de tabelas

Tabela 1 – Parâmetros D-H do robô KUKA-KR16.	29
Tabela 2 – Limites de Fim de Curso de Articulações do Robô KUKA-KR16 ajustados para posição inicial escolhida em $[0, 0, 0, 0, 0, 0]^T$.	50
Tabela 3 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto PGRobotArmR	77
Tabela 4 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto QLearningRobotArmR	79
Tabela 5 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto PGRobotArmRR	83
Tabela 6 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto QLearningRobotArmRR	85
Tabela 7 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto PGKuka	89
Tabela 8 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto DQNkuka	93
Tabela 9 – Tabela de tempos médios aproximados para treino do agente em função do algoritmo implementado e das características do projeto	101
Tabela 10 – Tabela valores da constante de exploração ϵ em função da época de treino para $\text{epsilonDecayPerEpoch} = 0.995$ e $\text{epsilonDecay} = 0.999$	106
Tabela 11 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto DQNkukaImage	108

Lista de abreviaturas e siglas

RL	<i>Reinforcement Learning</i>
DRL	<i>Deep Reinforcement Learning</i>
CNN	<i>Convolutional Neural Network</i>
DNN	<i>Deep Neural Network</i>
MDP	<i>Markov Decision Process</i>
DQN	<i>Deep Q Network</i>
DDPG	<i>Deep Deterministic Policy Gradient</i>
IA	Inteligência Artificial

Listas de símbolos

π_θ	Política de ações parametrizada por θ
s_t	Estado do sistema em instante t conforme formulação de Processos de Decisão de Markov
a_t	Ação tomada por agente em instante t conforme formulação de Processos de Decisão de Markov
$p(s' s, a)$	Probabilidade de o sistema transicionar do estado s para o estado s' sob a ação a
$r(s_t, a_t)$	Recompensa imediata obtida pelo agente ao tomar ação a_t no estado s_t
G_t	Valor esperado de recompensas descontadas a partir de instante t , também denominado retorno associado ao instante t
$J(\pi_\theta)$	Performance de política de ações π_θ
γ	Fator de desconto
α	Taxa de aprendizado
ϵ	Constante de exploração de política ϵ -Greedy
$v(s_t)$	Valor do estado s_t
$Q(s_t, a_t)$	Valor do par estado ação (s_t, a_t)

Sumário

I	INTRODUÇÃO	17
1	INTRODUÇÃO	18
1.1	Tema e Motivação	18
1.2	Objetivos Gerais	19
1.3	Interesse Científico, Econômico e Social	19
II	PESQUISA E TEORIA	20
2	ESTADO DA ARTE	21
2.1	Robótica	21
2.2	Reinforcement Learning	21
2.3	Aplicações de RL na Robótica	23
3	FUNDAMENTOS TEÓRICOS	26
3.1	Cinemática de Robôs Manipuladores	26
3.1.1	Parâmetros de Denavit-Hartenberg	27
3.1.2	Modelo Cinemático do Robô KUKA-KR16	28
3.2	Campo Potencial	30
3.3	Reinforcement Learning	32
3.3.1	Processo de Decisão de Markov (MDP)	32
3.3.2	Política de Ações π	33
3.3.3	Função Valor das Ações q e Função Valor dos Estados v	34
3.3.4	<i>Value Iteration</i> e <i>Policy Iteration</i>	35
3.3.5	<i>Q-Learning</i>	35
3.3.6	Métodos de <i>Policy Gradient</i>	36
III	DETALHAMENTO DO PROJETO	39
4	REQUISITOS DO PROJETO	40
4.1	Requisitos	40
5	SOLUÇÃO PROPOSTA	42
5.1	Arquitetura de controle	42
5.2	Espaço de Estados	43

5.2.1	Primeiro Espaço de Estados Considerado: Configuração para Percepção de Profundidade	43
5.2.2	Segundo Espaço de Estados Considerado: Câmeras Superior e Lateral	45
5.3	Espaço de Ações	46
5.4	Dinâmica do Sistema	47
5.4.1	Critérios de Parada	48
5.4.1.1	Número Máximo T de <i>Timesteps</i>	49
5.4.1.2	Posição Desejada Alcançada	49
5.4.1.3	Colisões com Mesa e Obstáculo	49
5.4.1.4	Limite de Curso de Articulação	50
5.5	Função Recompensa	51
5.5.1	Primeira Função Recompensa Considerada: Distâncias Absolutas	51
5.5.2	Segunda Função Recompensa Considerada: Discreta sob Aproximação ou Distanciamento	53
5.5.3	Terceira Função Recompensa Considerada: Projeção de Vetor Deslocamento	54
5.6	Algoritmos	55
5.6.1	Primeiro Algoritmo Implementado: REINFORCE Episódico	56
5.6.2	Segundo Algoritmo Implementado: DQN	57
IV	FERRAMENTAS	61
6	SOFWARES	62
Softwares		62
6.1	Matlab	64
6.1.1	Arquitetura de Projetos Desenvolvidos	65
6.1.2	Estruturas de Armazenamento	68
6.1.2.1	Armazenamento de Transições	68
6.1.2.2	Armazenamento de Trajetórias	68
6.1.2.3	Armazenamento de Redes DQN	69
6.1.3	Fluxo de Execução do Programa	69
6.1.4	Arquivo URDF	71
6.2	VGG16	71
V	RESULTADOS	74
7	PROJETOS DE TESTE	75
7.1	Projeto 1: PGRobotArmR	75
7.1.1	Especificação do Sistema	75

7.1.2	Resultados	77
7.2	Projeto 2: QLearningRobotArmR	78
7.2.1	Especificação do Sistema	79
7.2.2	Resultados	80
7.3	Projeto 3: PGRobotArmRR	81
7.3.1	Especificação do Sistema	81
7.3.2	Resultados	83
7.4	Projeto 4: QLearningRobotArmRR	84
7.4.1	Especificação do Sistema	84
7.4.2	Resultados	85
7.5	Projeto 5: PGKuka	87
7.5.1	Especificação do Sistema	87
7.5.2	Resultados	89
7.6	Projeto 6: DQNKuka	92
7.6.1	Especificação do Sistema	92
7.6.2	Resultados	93
7.7	Projeto 7: PGKukaRandomPositions	96
7.7.1	Especificação do Sistema	96
7.7.2	Resultados	97
7.8	Projeto 8: DQNKukaRandomPositions	97
7.8.1	Especificação do Sistema	98
7.8.2	Resultados	98
8	ANÁLISE DOS RESULTADOS	100
8.1	Escolha do algoritmo de treino do agente	100
8.1.1	Estabilidade e convergência	100
8.1.2	Tempo de treino	101
8.2	Escolha da função de recompensa $r(s, a)$	101
8.3	Determinação de hiper-parâmetros	103
8.3.1	Fator de desconto γ	103
8.3.2	Taxa de decaimento da constante de exploração ϵ	105
8.3.3	Taxa de aprendizado α	106
9	RESULTADOS DO PROJETO FINAL	107
9.1	Especificação do Sistema	107
9.2	Resultados	108
VI	CONCLUSÃO	110
10	CONCLUSÃO	111

Conclusão	111
10.1 Aprendizado por Reforço na Robótica	111
10.2 Arquitetura de Treino	112
VII REFERÊNCIAS BIBLIOGRÁFICAS	114
APÊNDICES	117
APÊNDICE A – REPOSITÓRIO GITHUB	118
APÊNDICE B – DOCUMENTAÇÃO DE PRINCIPAIS FUNÇÕES	119
B.1 fillReplayBufferNTrajs (transition_buffer, dqn, epoch_index, eps_init)	119
B.2 transitionBuffer.sampleMiniBatch (obj,n)	120
B.3 gradientDescent(dqn,transitions_batch)	120
B.4 action(dqn,s,prob)	120
B.5 dynamics(s,a)	121
B.6 reward(s,s_new)	121
ANEXOS	122
ANEXO A – CÓDIGO MATLAB	123
A.1 Script Principal: DQNKuka_image.m	123
A.2 action.m	128
A.3 dynamics.m	129
A.4 reward_3.m	131
ANEXO B – ARQUIVO URDF	134
B.1 kr16_2.urdf	134

Parte I

Introdução

1 Introdução

1.1 Tema e Motivação

A diversidade de aplicações da robótica no contexto industrial possibilita o surgimento de robôs com diferentes níveis de autonomia, apropriados para a execução de tarefas como soldagem, usinagem, montagem e movimentação de cargas. O surgimento de sensores mais sofisticados, o aumento da capacidade computacional de controladores e avanços nas áreas de visão computacional e inteligência artificial promoveram uma mudança no controle de manipuladores robóticos: As rotinas repetitivas, fixas e pré-programadas deram espaço a controles mais flexíveis e reativos, capazes de identificar dinamicamente a orientação de objetos de trabalho ou ainda aprender rotinas a partir de dados (ROSEN, 1999).

Esse trabalho consiste no desenvolvimento de uma arquitetura de controle de braços robóticos com capacidade de posicionamento do efetuador em ambientes com obstáculos sem conhecimento prévio da posição dos mesmos ou da dinâmica do braço, de modo que o controlador realiza o processo de interação com o ambiente e aprendizagem da ação de controle de modo autônomo. O controlador é treinado por meio da Aprendizagem por Reforço, método consolidado em aplicações da robótica devido à sua formulação geral e aprendizado de dinâmicas complexas por interação com o ambiente. O sensoriamento é realizado por meio de uma câmera que captura uma imagem que contém a cada instante o estado do sistema: posição e orientação do robô, posição de destino e posições dos obstáculos.

Na programação tradicional de robôs industriais posiciona-se o robô em diversos pontos fixos, as coordenadas destes pontos são armazenadas e utilizadas em comandos que interpolam uma trajetória suave entre eles. Essa abordagem apresenta alta precisão na execução da tarefa desejada, mas pouca generalidade com relação à ambientes e trajetórias distintas. Evoluções do controle tradicional de manipuladores industriais incorporam sensores em um algoritmo reativo de Geração de Trajetórias Online (OTG), que realiza deformações locais na trajetória para desvio de obstáculos de modo a proporcionar segurança em aplicações colaborativas com humanos (ZHAO, 2015).

A utilização do sensoriamento e de algoritmos inteligentes para aprendizado da própria trajetória do robô, escopo deste trabalho, ainda não é amplamente aplicada em manipuladores industriais devido à necessidade de comprovação de eficiência e garantia de segurança. Por meio do Aprendizado por Reforço, pode-se substituir a programação manual ponto a ponto pela programação de alto nível da tarefa a ser desejada por meio de uma função recompensa, que representa o comportamento desejado do robô. Assim, a tarefa de obtenção da trajetória passa a ser do próprio robô, que aprende comportamentos

complexos de modo dinâmico a partir de sucessivas interações com o ambiente.

1.2 Objetivos Gerais

O objetivo do projeto consiste em desenvolver, como alternativa a métodos tradicionais de programação de robôs industriais, uma arquitetura que implementa a programação autônoma do robô manipulador industrial KUKA KR16 de seis graus de liberdade diretamente a partir da interação com o ambiente. O posicionamento do efetuador em diferentes pontos deve ser realizado em um ambiente sujeito a obstáculos em posições genéricas a partir de métodos de aprendizado por reforço integrados a um sistema de aquisição de imagens em tempo real.

Deseja-se avaliar o desempenho do controlador obtido em comparação à técnica tradicional de programação e a outros métodos de inteligência artificial utilizados na robótica moderna. Para fins comparativos, fatores como tempo de treino, escalabilidade e segurança são considerados.

1.3 Interesse Científico, Econômico e Social

O trabalho apresenta uma abordagem utilizada recentemente para o controle de braços robóticos, com potencial para aprendizado de ações de controle complexas cuja programação manual é inviável. A arquitetura projetada pode ser adaptada para incluir sensores adicionais e aprender a executar diferentes tarefas, tal como paletização, sem a necessidade de reprogramação, apenas com maior tempo de treinamento. Além disso, a capacidade de lidar com ambientes desconhecidos e não estruturados constitui uma habilidade essencial no desenvolvimento de robôs colaborativos domésticos, que representam um novo setor com potencial comercial crescente.

Em aplicações industriais, a implementação do controle inteligente de robôs apresenta uma camada adicional de segurança, permitindo que o robô evite colisões com peças em posições desconhecidas ou até mesmo indivíduos em distância não segura do espaço de trabalho. Há ainda a otimização do tempo que seria gasto na programação manual de trajetórias.

Parte II

Pesquisa e Teoria

2 Estado da Arte

2.1 Robótica

Desde o desenvolvimento do primeiro robô industrial programável por George Devol e Joe Engleberger em 1954 até os dias atuais os métodos de sensoriamento e controle de robôs constituem um tema amplamente estudado. Zamalloa et al. (2017) classificam o desenvolvimento da robótica em 4 gerações: Geração 1: Primeiros Manipuladores (1950-1967), marcada pelos primeiros robôs industriais capazes de realizar tarefas simples e repetitivas; Geração 2: Robôs Sensorizados (1968-1977), caracterizada pela integração de sensores e de Controladores Lógico Programáveis (CLPs); Geração 3: Robôs Industriais (1978-1999), na qual houve investimento considerável em automação de linhas de produção, o surgimento de controladores dedicados e novas linguagens de programação de robôs. Finalmente, a Geração 4: Robôs Inteligentes (2000-Atualmente) faz uso de capacidade computacional elevada e sensores sofisticados, como câmeras de profundidade, para integrar inteligência artificial e aprendizado por dados em sistemas de controle de robôs. O uso de Aprendizagem por Reforço (Reinforcement Learning ou RL) em Robótica foi estabelecido tanto por meio de simulações quanto por implementações em robôs reais na Geração 4: Robôs Inteligentes.

2.2 Reinforcement Learning

A Aprendizagem por Reforço pode ser vista como o terceiro paradigma do Aprendizado de Máquina, sendo os dois primeiros definidos como Aprendizado Supervisionado e Não Supervisionado respectivamente. Sua formulação básica consiste em permitir que um agente aprenda um comportamento que maximiza uma determinada recompensa. Isso é feito sem que o agente tenha conhecimentos *a priori* da dinâmica do sistema, ele deve descobrir, por interações sucessivas com o ambiente, quais ações levam à maior recompensa em cada estado (SUTTON, 2017).

Problemas resolvidos por meio de RL são tipicamente formulados como um Processo de Decisão de Markov (*Markov Decision Process* ou MDP), definido como um conjunto (S, A, P, R) , em que S é um conjunto finito de estados que representam o sistema, A é um conjunto finito de ações, $P(s_{t+1} = s'|s_t = s, a_t = a)$ é a probabilidade de que uma ação a levará o sistema do estado $s \in S$ no instante de tempo t para o estado $s' \in S$ no instante de tempo $t + 1$ e R é o conjunto de recompensas $r(s, a)$ imediatas recebidas após tomar a ação a no estado s . O problema central em um MDP é encontrar uma função política de ações $\pi(s) : S \rightarrow A$ que especifica a ação a ser tomada pelo agente no estado s .

de modo a maximizar um retorno esperado de recompensas, exemplificado no capítulo 3 deste trabalho.

Os algoritmos para resolução de problemas de RL, ou seja, a obtenção da política de ações π , podem ser divididos em duas classes de métodos: Iteração sobre política de ações e iteração sobre função valor de estados e ações. Na primeira classe de métodos procura-se diretamente por funções π parametrizadas que maximizam uma Função Valor do par Estado-Ação, $Q_\pi(s, a) : (S \times A) \rightarrow R$, que determina o valor de determinada ação, dado o estado atual do sistema. Esta função é definida como a recompensa imediata recebida somada ao valor descontado do novo estado do sistema $Q_\pi(s, a) = r(s, a) + \gamma V_\pi(s')$. De acordo com a equação de Bellman e o princípio da otimalidade pode-se escrever $Q_\pi(s, a) = r(s, a) + \gamma \max_{a'} Q_\pi(s', a')$, ou seja, o valor de um estado $V_\pi(s')$ é o maior valor da função $Q_\pi(s', a')$ para todas as ações a' que podem ser tomadas naquele estado (Bellman, 1967, 1971).

Já na segunda classe de métodos, a política de ações é ignorada e maximiza-se a Função Valor dos Estados $V_\pi(s) : S \rightarrow R$. Posteriormente obtém-se a política de ações ótima π^* a partir da função V^* , simplesmente tomando as ações que levam aos estados de maior valor. O método de Iteração sobre a Política de Ações pode ser visto como a formulação dual de Lagrange do problema de otimização do método de Iteração sobre a Função Valor (KÖBER, 2013).

Um algoritmo clássico da classe de métodos de Iteração sobre a Função Valor, denominado *Q-Learning*, consiste em escrever a função $Q_\pi(s, a)$ como uma matriz $Q \in R^{n_s \times n_a}$, inicializada arbitrariamente ou de modo a capturar conhecimento prévio do sistema e atualizada de modo iterativo a partir dos valores atuais da matriz e das recompensas obtidas a partir de interações do agente com o ambiente (WATKINS, 1989). Limitações desse algoritmo consistem na necessidade de discretização do espaço de estados S e do espaço das ações A , o que inviabiliza aplicações práticas em robótica devido à alta dimensionalidade dos estados.

Com o objetivo de endereçar essas limitações, uma evolução deste algoritmo, denominada DQN foi desenvolvida (MNIH et al, 2013). Ela consiste na representação da função Q não por uma matriz, mas por uma Rede Neural Profunda, de modo que estados s pertencentes a um espaço contínuo S e ações $a \in A$ discretas são mapeadas em valores $Q_\pi(s, a)$. Os pesos da rede são atualizados conforme o agente interage com o ambiente e a política ótima de ações é extraída de modo análogo: Dado que o sistema se encontra em um estado s a ação a a ser tomada é aquela que maximiza a função valor $Q_\pi(s, a)$. Os autores propuseram ainda uma técnica de aumento de convergência denominada *Experience Replay*, que armazena transições (s_t, a_t, r_t, s_{t+1}) em uma memória e amostra aleatoriamente grupos de transições desta memória durante o treino, em oposição a treinar com transições subsequentes. Isso impede correlação entre os dados, impedindo

convergência a mínimos locais sub-ótimos e promovendo maior eficiência de amostras, que são utilizadas múltiplas vezes durante ao longo do treino.

2.3 Aplicações de RL na Robótica

Na robótica, múltiplos graus de liberdade e estados contínuos tornam a alta dimensionalidade dos espaços de estados e ações o principal problema na aplicação de métodos tradicionais baseados na discretização dos espaços de estados e ações. Os primeiros sucessos em aplicações de RL na robótica foram obtidos com métodos de aproximação da função Valor dos Estados e das Ações, no entanto, avanços consideráveis foram obtidos posteriormente com métodos de Iteração sobre a Política de Ações. Isso se deve à menor dimensão do espaço de políticas de ações em comparação ao espaço de pares Estado-Ação (KORMUSHEV, 2013).

James e Johns (2016) obtiveram sucesso parcial na transferência de um modelo DQN treinado em simulação para aplicação real de um robô manipulador de sete graus de liberdade na ação de localizar e levantar um cubo a partir de imagens. O ambiente de trabalho foi estruturado de modo a maximizar a semelhança com o ambiente de simulação e possibilitar a transferência do modelo. O controlador implementado no robô real foi capaz de localizar e se aproximar do cubo, mas não apresentou sucesso na tarefa de levantá-lo em aplicação real, somente na simulação.

Tsurumine et al. (2018) propuseram dois algoritmos de DRL eficientes em termos de amostras que combinam a atualização suave da política de ação com a capacidade de extração automática de features por redes neurais: *Deep P-Network* (DPN) e *Dueling Deep P-Network* (DDPN). Os algoritmos propostos foram comparados a métodos existentes de DRL em ambiente de simulação para braços robóticos de diferentes números de graus de liberdade e testados em duas aplicações reais por robôs manipuladores de tecido: Virar um guardanapo e dobrar uma camiseta (figura 1) a partir de um número reduzido de amostras.



Figura 1 – Robô humanoide NEXTAGE em tarefa de dobrar camiseta (Tsurumine et al. 2018).

Há ainda exemplos de métodos de DRL baseado em visão computacional aplicados a robôs manipuladores para resolução da tarefa de segurar objetos com taxas de sucesso de até 96%, como no desenvolvimento e aplicação do algoritmo QT-Opt baseado na generalização de Q-learning por meio do treinamento de uma DQN de modo *off-policy*, ou seja, os valores de $Q_\pi(s, a)$ são aproximados sem a necessidade de se manter registro da política de ações atual. Os atores utilizam ainda uma função recompensa binária ($r=1$ se o objeto foi levantado e $r=0$ caso contrário) a fazem uso de um conjunto de 580.000 tentativas de levantamento para treinar a DQN (KALASHNIKOV et al, 2018).

Um dos grandes desafios associados à implementação de Aprendizado por Reforço na robótica é a baixa eficiência amostral, de modo que um grande volume de dados é necessário durante o treino e a geração deste volume torna-se frequentemente inviável em aplicações práticas. Para contornar este problema, utiliza-se demonstrações prévias e inicializações específicas que capturam o comportamento desejado. No entanto, essa abordagem é conflitante com a principal vantagem de RL: O aprendizado autônomo de diversos comportamentos com mínima intervenção humana.

Shixiang Gu et. al (2016) apresentaram uma arquitetura inovadora dos algoritmos DDPG (*Deep Deterministic Policy Gradient*) e NAF (*Normalized Advantage Function*), em que múltiplos robôs interagem com o ambiente, adquirem experiência de acordo com suas políticas de ações atuais e enviam dados de modo assíncrono a um servidor, que amostra transições e treina a DQN (figura 2). Essa arquitetura permite que o robô continue interagindo com o ambiente e coletando transições de estados enquanto os parâmetros da DQN são atualizados, promovendo escalabilidade para inclusão de novos robôs. Os autores validaram a arquitetura proposta no aprendizado da tarefa de abertura de uma porta por robôs manipuladores de 7 graus de liberdade e a política de ações foi obtida sem demonstrações prévias.

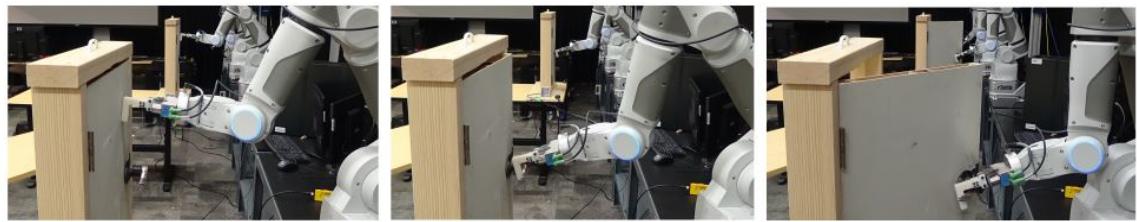


Figura 2 – Dois Braços Robóticos aprendendo a abrir uma porta com o algoritmo NAF assíncrono (Gu, S et al, 2016).

3 FUNDAMENTOS TEÓRICOS

Esta seção trata dos conceitos teóricos fundamentais utilizados no decorrer do projeto, fornecendo os principais resultados e algoritmos utilizados. Primeiramente serão derivadas as equações cinemáticas de robôs manipuladores, especificamente do robô KUKA, utilizadas no ambiente de visualização e simulação dos algoritmos de Aprendizado por Reforço no Matlab. Depois será apresentada a formulação do campo potencial para geração de trajetórias de robôs móveis, método simples de controle reativo utilizado para comparação com as trajetórias geradas pelo controlador após treinamento por Aprendizagem por Reforço. Por fim serão abordados a teoria necessária de *Reinforcement Learning* e *Deep Reinforcement Learning*, assim como os algoritmos implementados em simulação e no robô KUKA.

3.1 Cinemática de Robôs Manipuladores

A cinemática de robôs manipuladores consiste no estudo das posições e velocidades de seu efetuador e ligamentos. Um modelo cinemático de um robô permite o equacionamento que relaciona as posições e velocidades de suas articulações com a posição e velocidade de seu efetuador, o qual deseja-se controlar. Na Cinemática Direta deseja-se obter a posição e velocidade do efetuador em função das posições e velocidades das articulações do robô, controladas pelos atuadores presentes nas articulações ativas. Em aplicações práticas, no entanto, temos a trajetória a ser seguida pelo efetuador e desejamos obter as posições e velocidades das articulações, essa análise denomina-se Cinemática Inversa.

A figura 3 apresenta um diagrama esquemático do robô KUKA-KR16 e seus seis graus de liberdade. As articulações A1-A3 são utilizadas para efetuar o posicionamento do efetuador e as articulações A4-A6 (punho esférico) são responsáveis pela sua orientação.

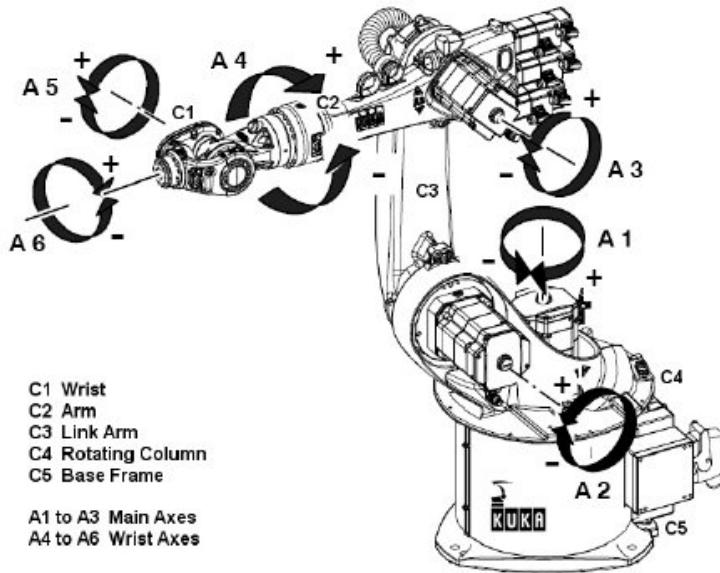


Figura 3 – 6 Graus de Liberdade do Robô KUKA-KR16 (KUKA ROBOTICS, 2003)

3.1.1 Parâmetros de Denavit-Hartenberg

A análise de uma cadeia cinemática é facilitada pelo uso da notação de Denavit-Hartenberg, que descreve os parâmetros necessários para determinar a matriz de transformação homogênea entre os sistemas de coordenadas de ligamentos consecutivos. Se as seguintes convenções forem obedecidas:

- Eixo z_i do sistema de coordenadas $i + 1$ sobre o eixo da articulação.
- Eixo x_i paralelo à normal comum: $x_i = z_i \times z_{i-1}$.
- Eixo y_i de acordo com a regra da mão direita a partir de x_i e z_i .

temos que a matriz de transformação homogênea entre os sistemas de coordenadas $i-1$ e i é dada por:

$${}_{i-1}\mathbf{A} = \text{Rot}_z(\theta_i)\text{Trans}_z(d_i)\text{Trans}_x(a_i)\text{Rot}_x(\alpha_i) \quad (3.1)$$

A notação de Denavit-Hartenberg consiste em uma representação dos quatro parâmetros da equação acima para cada ligamento, relacionando a posição e orientação de seu sistema de coordenadas com o sistema de coordenadas do ligamento anterior. Os parâmetros de Denavit-Hartenberg d_i , θ_i , r_i e α_i são definidos de modo genérico conforme a figura 4.

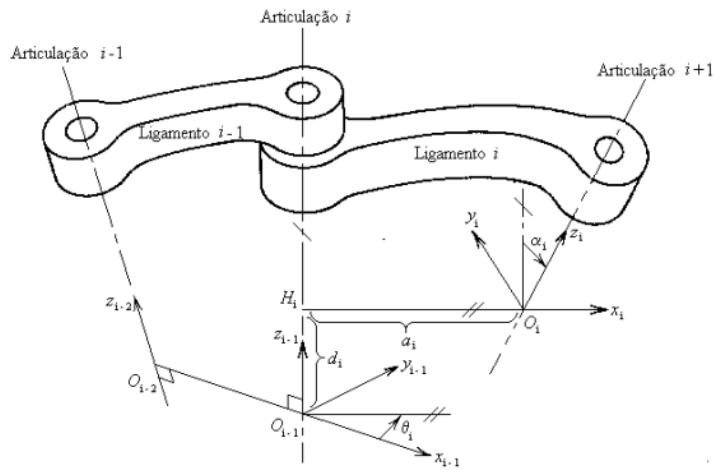


Figura 4 – Notação Denavit-Hartenberg (Cabral, 2019)

3.1.2 Modelo Cinemático do Robô KUKA-KR16

De acordo com as dimensões do robô KUKA-KR16 fornecidas no manual pelo fabricante, conforme figura 5, determina-se os eixos de cada sistema de coordenadas (figura 6) e a tabela com os respectivos parâmetros de Denavit-Hartenberg (D-H):

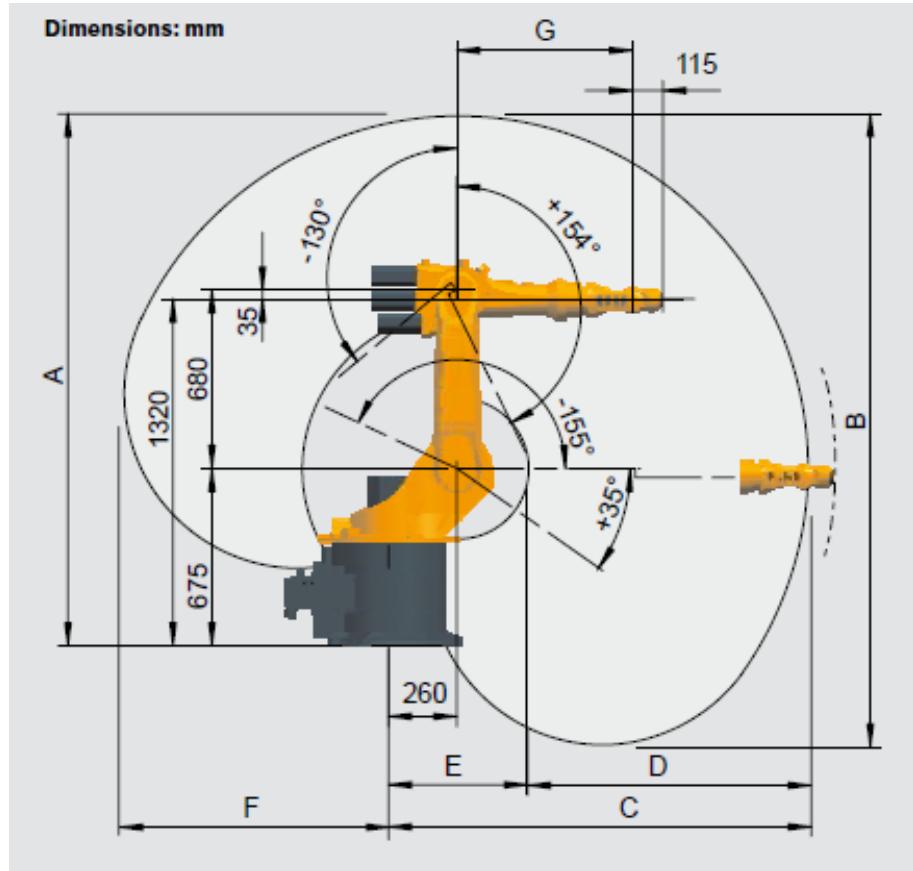


Figura 5 – Volume de trabalho do Robô KUKA-KR16 (Fonte Manual do Fabricante).

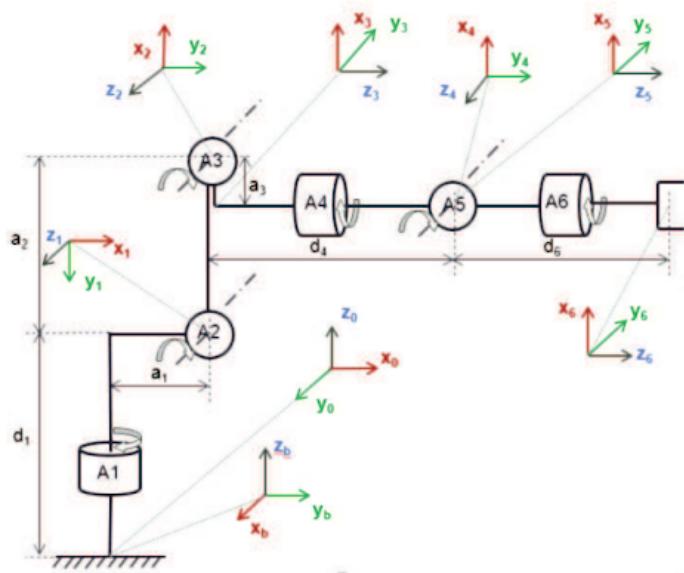


Figura 6 – Sistemas de coordenadas dos ligamentos segundo D-H (PIOTROWSKI N; BARYLSKI A, 2014).

i	θ°	d[mm]	a[mm]	α°	θ_{min}°	θ_{max}°
1	q_1	$d_1 = 675$	$a_1 = 260$	-90	-185	185
2	$q_2 - 90$	0	$a_2 = 680$	0	-155	35
3	q_3	0	$-a_3 = -35$	-90	-130	154
4	q_4	$d_4 = 670$	0	90	-350	350
5	q_5	0	0	-90	-130	130
6	q_6	$d_1 = 115$	0	0	-350	350

Tabela 1 – Parâmetros D-H do robô KUKA-KR16.

Assim, pode-se efetuar a cinemática direta e determinar a matriz de transformação homogênea entre o sistema de coordenadas $O_0X_0Y_0Z_0$ fixo na base do robô e o sistema de coordenadas $O_0X_0Y_0Z_0$ no efetuador:

$${}^6_0\mathbf{A} = {}^1_0\mathbf{A}_1^2 {}^2_3\mathbf{A}_3^4 {}^4_5\mathbf{A}_5^6 {}^6_0\mathbf{A} \quad (3.2)$$

A partir do modelo cinemático do robô foi criada uma interface gráfica em Matlab para visualização do robô em diferentes configurações. Foi utilizada a biblioteca de robótica do Matlab para criar um objeto da classe RigidBodyTree que representa a cadeia cinemática do robô KUKA a partir de um arquivo urdf (Unified Robot Description Format) que descreve as posições e orientações dos sistemas de coordenadas de cada ligamento. O arquivo urdf contém ainda os nomes de arquivos stl com representações visuais das malhas

de cada ligamento. As figuras 7 e 8 mostram a cadeia cinemática do robô KUKA KR16 em sua configuração inicial e a respectiva visualização do robô na mesma configuração.

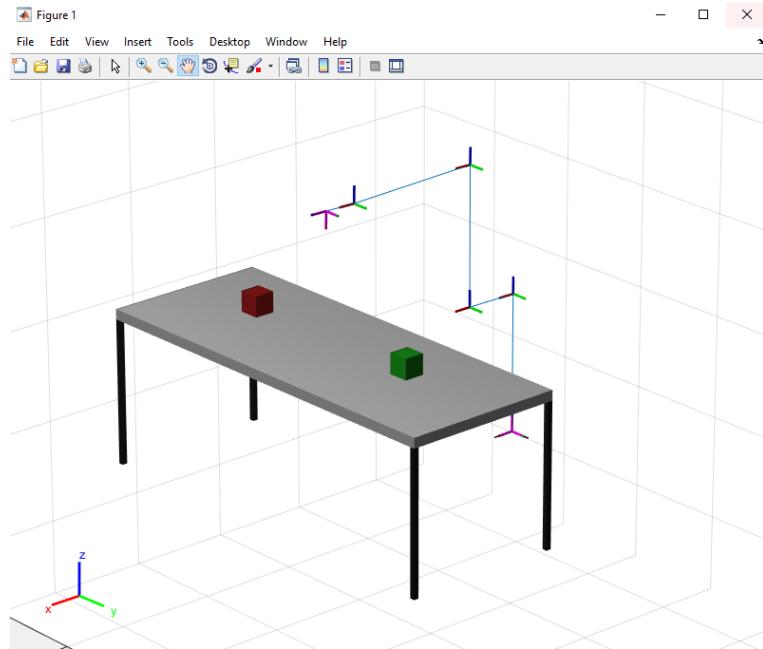


Figura 7 – Representação gráfica da cadeia cinemática do robô (Fonte: Autores).

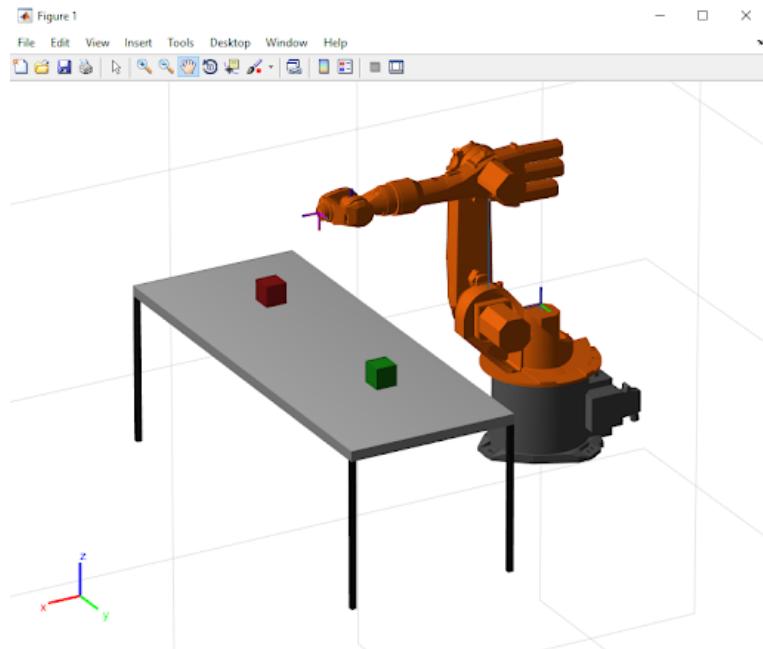


Figura 8 – Representação gráfica do robô com visualização dos ligamentos (Fonte: Autores).

3.2 Campo Potencial

Um método consolidado em aplicações da robótica móvel para solucionar o problema de geração de trajetória em ambiente sujeito a obstáculos é denominado Método do Campo

Potencial. Com inspiração nos efeitos físicos de campos eletromagnéticos e gravitacionais, o método consiste em fazer com que o robô seja atraído pelo ponto final desejado e repelido por obstáculos. Assim, para cada posição do robô, a partir de sua distância em relação à posição desejada e aos obstáculos, calcula-se o campo potencial devido a cada obstáculo e à posição de destino, determina-se as forças (de atração ou repulsão) geradas pelos campos individuais e suas contribuições são somadas para obtenção da força resultante, que determina a direção de movimento do robô (figura 9). Deseja-se que o campo gerado possua apenas um mínimo global na posição final desejada e nenhum outro mínimo local, de modo a evitar configurações em que o robô fique preso e garantir convergência para posição final desejada.

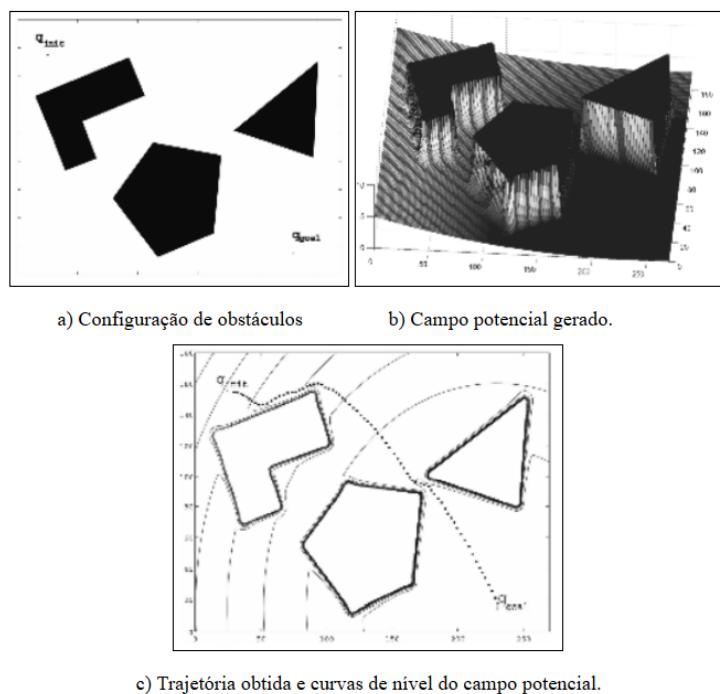


Figura 9 – Visualização do Método do Campo Potencial (Fonte: SIEGWART, R, 2004).

Na interface gráfica para visualização e simulação do robô KUKA-KR16, detalhada no capítulo 6 deste trabalho, foi implementado o método do campo potencial para configurações com apenas um obstáculo, em que as forças de repulsão, atração e resultante sobre o efetuador são representadas pelos vetores vermelho, verde e azul respectivamente (Figura 10):

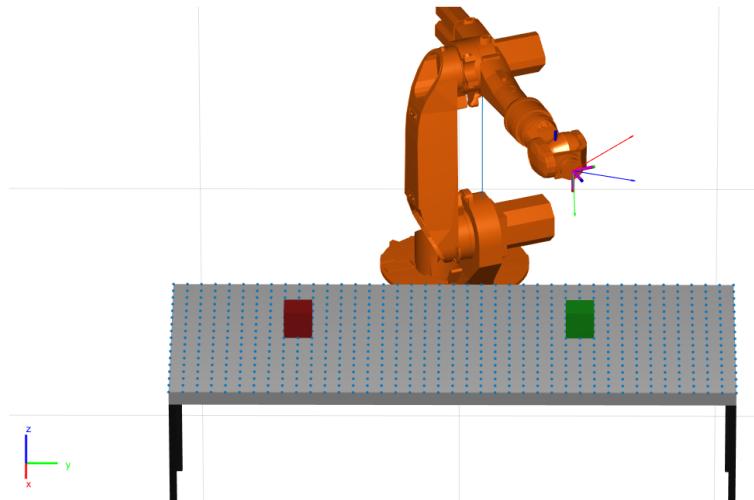


Figura 10 – Visualização do efeito do campo potencial para robô KUKA-KR16 (Fonte: Autores).

3.3 Reinforcement Learning

Quando pensamos na natureza do aprendizado em humanos e outros seres vivos percebemos que a interação com o ambiente é um dos principais conceitos envolvidos. Na maior parte do tempo não há presença direta de um professor, o indivíduo simplesmente interage com o mundo ao seu redor. Sob uma noção abstrata de objetivo a ser alcançado e resultado obtido, avalia suas ações de modo a incentivar aquelas que apresentaram melhor resultado na situação em que se encontrava. Assim, ao encontrar novamente aquele cenário, o indivíduo tende a se comportar de modo semelhante. Esse tipo de aprendizado pode ser facilmente observado no treinamento de animais de estimação, que recebem um incentivo positivo, por exemplo comida, ao realizar determinada ação, como buscar um objeto.

O Aprendizado por Reforço é a área de Aprendizado de Máquina responsável pela modelagem e estudo de métodos de treinamento de um agente para realizar ações em determinado ambiente de modo a maximizar certa recompensa cumulativa. Diferentemente de aprendizado supervisionado, não é preciso apresentar dados e suas respectivas categorias corretas. Em vez disso, o agente aprende diretamente por meio de interação com o ambiente, observando a recompensa obtida por tomar determinada ação em determinado estado.

3.3.1 Processo de Decisão de Markov (MDP)

Em problemas de *Reinforcement Learning* o ambiente pode ser modelado como um Processo de Decisão de Markov (MDP), uma estrutura matemática que modela processos estocásticos em tempo discreto para estudo e otimização da tomada de decisões. Um MDP é formulado como um vetor $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, onde:

- \mathcal{S} é um conjunto finito de estados.

- \mathcal{A} é um conjunto finito de ações.
- \mathcal{P} é um conjunto de probabilidades $p(s'|s, a) = p(S_{t+1} = s' | S_t = s, A_t = a)$ de transicionar ao estado s' dado que o agente se encontrava no estado s e tomou a ação a .
- \mathcal{R} é um conjunto de recompensas imediatas $r(s, a)$ obtidas por tomar a ação a no estado s .

Uma das vantagens de se modelar o ambiente como um Processo de Decisão de Markov é a utilização da Propriedade de Markov: Toda informação necessária ao agente está contida no estado atual, ou seja, o estado s_{t+1} depende somente do estado s_t e da ação a_t .

3.3.2 Política de Ações π

O problema central em um MDP é encontrar uma função *policy*, ou política de ação, $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$ que especifica a ação a ser tomada pelo agente no estado s de modo a maximizar um retorno esperado de recompensas que pode ser definido das seguintes formas:

$$J(\pi) = \mathbf{E}\left[\sum_{t=0}^T r(s_t, a_t)\right] \quad (3.3)$$

$$J(\pi) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \quad (3.4)$$

$$J(\pi) = \mathbf{E}\left[\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^T r(s_t, a_t)\right] \quad (3.5)$$

Os modelos de retorno cumulativo representados pelas equações (3.1) - (3.3), respectivamente denominados *finite horizon model*, *infinite horizon discounted model* e *average reward model*, representam alguns critérios segundo os quais uma função policy π pode ser otimizada. No primeiro critério o agente leva em consideração apenas recompensas até certo instante de tempo T . No segundo, onde $\gamma \in [0, 1]$ é um fator de desconto que determina a importância atribuída a recompensas futuras, a escolha de γ pode alterar drasticamente a solução ótima obtida π^* que maximiza $J(\pi)$: se o fator de desconto for muito próximo de 0 o agente considera apenas recompensas imediatas e pode apresentar comportamento inadequado (KAELBLING, 1996). Por fim, o terceiro critério consiste em

maximizar o limite do valor médio das recompensas para tempo infinito, neste caso não é preciso determinar um fator de desconto γ ou um horizonte máximo T , de modo que o agente leva em consideração o comportamento assintótico da recompensa média. Essa abordagem não é capaz de distinguir funções *policy* que apresentam altas recompensas iniciais em regime transitório de outras que não apresentam. No entanto, devido à maior importância da estabilidade assintótica em relação ao comportamento transitório, o critério da recompensa média é preferível em aplicações reais (KOBER, 2013).

Em sistemas não determinísticos podemos definir a política de ações como $\pi : \mathcal{S} \rightarrow \mathcal{A}$, onde $\pi(\mathbf{a}, \mathbf{s})$ representa a probabilidade de o agente escolher a ação \mathbf{a} dado que o sistema se encontra no estado \mathbf{s} .

3.3.3 Função Valor das Ações q e Função Valor dos Estados v

Dado que o agente segue determinada política de ações π deseja-se expressar o valor associado a tomar uma ação \mathbf{a} no estado \mathbf{s} . Esse valor pode ser expresso em função das recompensas imediatas da seguinte forma:

$$Q_\pi(\mathbf{s}, \mathbf{a}) = \mathbf{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a} \right] \quad (3.6)$$

Utiliza-se o valor esperado da soma descontada das recompensas imediatas recebidas em cada instante de tempo, isso é feito porque o ambiente e a política de ações podem ser não determinísticos. Outra função de interesse é o valor associado simplesmente a estar no estado \mathbf{s} , definido analogamente como:

$$v_\pi(s) = \mathbf{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right] \quad (3.7)$$

Nota-se que a otimização da política de ações π segundo o modelo *infinite horizon discounted model* conforme a equação (3.4) consiste em maximizar o valor do estado inicial \mathbf{s}_0 sob a política π : $v_\pi(s_0)$. Isso faz sentido, uma vez que os estados seguintes dependem unicamente do estado inicial e da política de ações.

Por fim, com o objetivo de facilitar a notação, podemos definir o Retorno cumulativo descontado a partir do instante t como:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (3.8)$$

De modo que o valor de um estado s pode ser escrito como:

$$\begin{aligned}
 v_\pi(s) &= \mathbf{E}_\pi[G_t | s_t = s] \\
 &= \mathbf{E}_\pi[R_{t+1} + \gamma G_{t+1} | s_t = s] \\
 &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s, a) [r(s, a) + \gamma \mathbf{E}_\pi[G_{t+1} | s_{t+1} = s']] \\
 &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s, a) [r(s, a) + \gamma v_\pi(s')]
 \end{aligned} \tag{3.9}$$

A equação (3.9) é denominada Equação de Bellman e estabelece a relação entre o valor de um estado e os valores de estados seguintes, ponderados pela probabilidade das ações (determinada pela política π) e pela probabilidade de transição de estados do sistema. A Equação de Bellman pode ser escrita de forma equivalente para a Função Valor das Ações como:

$$Q_\pi(s, a) = \sum_{s' \in \mathcal{S}} p(s'|s, a) [r(s, a) + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s', a')] \tag{3.10}$$

3.3.4 Value Iteration e Policy Iteration

Os principais algoritmos de RL podem ser divididos em duas categorias:

- *Value Iteration*: Métodos baseados em obter a política ótima de ações π^* indiretamente por meio da obtenção da Função Valor das Ações $Q^*(s, a)$, definindo:

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a) = \{a | a \in \mathcal{A} \wedge Q^*(s, a) \leq Q^*(s, a'), \forall a' \in \mathcal{A}\} \tag{3.11}$$

Nesta categoria, inicializa-se a função $Q(s, a)$ ou a função $v(s)$ arbitrariamente e por métodos iterativos, é obtida a Função Valor das Ações ou Função Valor dos Estados sob uma política ótima. Por fim é extraída a política ótima selecionando sempre as ações que correspondem aos maiores valores em cada estado.

- *Policy Iteration*: Métodos baseados em obter diretamente a política ótima de ações π^* . A política de ações é parametrizada e obtida diretamente por meio de métodos iterativos, sem a necessidade de aproximar a Função Valor das Ações em primeiro lugar.

3.3.5 Q-Learning

Desenvolvido em 1989 por Watkins, *Q-Learning* é um algoritmo da categoria *Value Iteration* e consiste em atualizar iterativamente a Função Valor das Ações $Q(s, a)$ de modo a convergir à Função Valor das Ações sob política ótima $Q^*(s, a)$. Isso é feito sem

a necessidade de um modelo do ambiente (*model-free*) e requer apenas a interação com o mesmo e a observação das recompensas obtidas a cada instante de tempo. Além disso, não é preciso manter registro da política de ações, que é seguida de acordo com a equação (3.11).

O algoritmo consiste em inicializar a função $Q(\mathbf{s}, \mathbf{a})$ arbitrariamente para todas as combinações de estados e ações. Conforme o agente interage com o ambiente, os valores $Q(\mathbf{s}, \mathbf{a})$ são atualizados de acordo com a Equação de Bellman para os valores das ações aplicando diferenças temporais:

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow (1 - \alpha)Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha[R_{t+1} + \gamma \max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}_{t+1}, \mathbf{a}_t)] \quad (3.12)$$

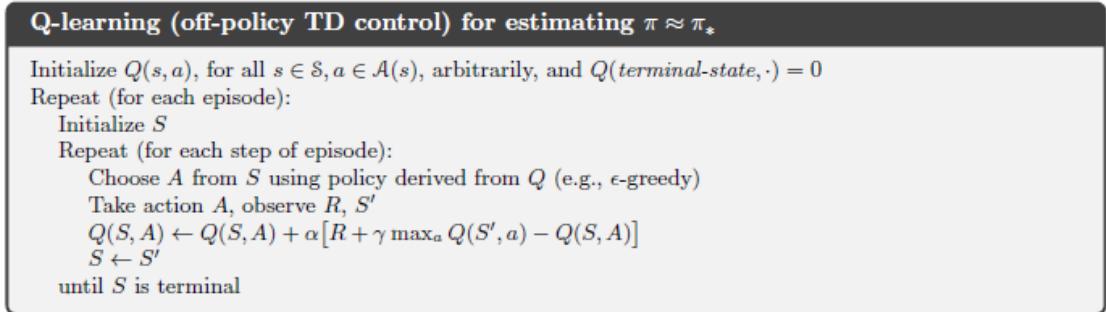


Figura 11 – Algoritmo *Q-Learning* (SUTTON, 2017).

Q-Learning é um algoritmo eficiente e com convergência garantida para valores suficientemente pequenos da *learning rate* α . No entanto, em sua versão mais simples, é restrito à Espaços de Estados S finitos e apresenta dificuldade de escalabilidade com o aumento das dimensões de \mathcal{S} e \mathcal{A} . Isso impossibilita sua aplicação direta em problemas complexos de robótica, nos quais a necessidade de discretizar o espaço de estados e de ações levaria a problemas de precisão (no caso de baixa discretização) ou inviabilidade computacional (para alta discretização). A figura 11 ilustra o fluxo do algoritmo *Q-Learning* tradicional.

3.3.6 Métodos de *Policy Gradient*

Diferentemente de *Q-Learning*, Métodos de *Policy Gradient* pertencem à categoria de algoritmos de RL que otimizam diretamente a política de ações π . Isso é feito por meio da parametrização da política de ações π com um vetor θ que pode representar os coeficientes de um polinômio, os pesos de uma rede neural ou quaisquer parâmetros de uma função aproximadora para a política ótima de ações.

Seja $J : \Pi \rightarrow \mathfrak{R}$ uma função performance, de modo que $J(\pi)$ mede a qualidade da política π , onde Π é o espaço de todas políticas possíveis. Seja $\pi_\theta \in \Pi_\theta \subset \Pi$ uma função

política de ações parametrizada pelo vetor θ , podemos treinar esta política iterativamente da seguinte forma:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \quad (3.13)$$

Seja $\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\}$ uma trajetória gerada por uma política de ações π_θ qualquer. Podemos definir a Performance $J(\pi_\theta)$ simplificada por $J(\theta)$ como:

$$J(\theta) = \mathbf{E}_{\tau \sim \pi(\tau)}[r(\tau)] \quad (3.14)$$

onde $r(\tau) = \sum_{t=1}^T \gamma^{t-1} r_t = V_\pi(s_0)$ é o valor esperado das recompensas descontadas ao longo da trajetória τ , que equivale ao valor do estado inicial segundo a política de ações π . Aplicando a definição de Valor Esperado temos:

$$J(\theta) = \int_{\tau} \pi_\theta(\tau) r(\tau) d\tau \quad (3.15)$$

onde $\pi_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t)$ denota a probabilidade da trajetória τ sob a política de ações π_θ . Calculando o gradiente da função performance J :

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int_{\tau} \pi_\theta(\tau) r(\tau) d\tau \\ &= \int_{\tau} \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau \\ &= \int_{\tau} \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau \end{aligned} \quad (3.16)$$

Aplicando novamente a definição de Valor Esperado e expandindo o termo $\nabla_\theta \log \pi_\theta(\tau) r(\tau)$ temos:

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau \sim \pi(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \quad (3.17)$$

$$\begin{aligned} \nabla_\theta \log \pi_\theta(\tau) r(\tau) &= \nabla_\theta \left[\log \left(p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t) \right) \right] r(\tau) \\ &= \nabla_\theta \left[\log(p_0) + \sum_{t=0}^{T-1} \left(\log(\pi_\theta(a_t | s_t)) + \log(p(s_{t+1} | s_t, a_t)) \right) \right] r(\tau) \\ &= \sum_{t=0}^{T-1} \nabla \log \pi_\theta(a_t | s_t) r(\tau) \end{aligned} \quad (3.18)$$

Substituindo a (3.18) na (3.17) obtemos uma expressão através da qual podemos estimar o gradiente da função performance por meio da observação de trajetórias $\tau^{(i)}$, $i = 1, \dots, N$:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left[\left(\sum_{t=0}^{T-1} \nabla \log \pi_{\theta}(a_t | s_t) \right) \left(\sum_{t=1}^T \gamma^{t-1} r_t \right) \right] \quad (3.19)$$

No entanto, percebemos que o termo $\sum_{t=1}^T \gamma^{t-1} r_t$ considera recompensas desde o primeiro estado s_0 , no entanto, a velocidade de convergência é maior se o agente considerar apenas recompensas futuras ao estado s_t . Logo:

$$\begin{aligned} \nabla_{\theta} J(\theta) &\approx \frac{1}{N} \sum_{i=1}^N \left[\sum_{t=0}^{T-1} \left(\nabla \log \pi_{\theta}(a_t | s_t) \sum_{t'=t}^T \gamma^{t'-t} r_{t'} \right) \right] \\ \nabla_{\theta} J(\theta) &\approx \frac{1}{N} \sum_{i=1}^N \left[\sum_{t=0}^{T-1} G_t \frac{\nabla \pi_{\theta}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} \right] \end{aligned} \quad (3.20)$$

Williams (1992) propôs um algoritmo clássico baseado na parametrização e busca direta de funções política, denominado REINFORCE (figura 12):

```
REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization  $\pi(a|s, \theta)$ 
Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$ 
Repeat forever:
  Generate an episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , following  $\pi(\cdot|s, \theta)$ 
  For each step of the episode  $t = 0, \dots, T - 1$ :
     $G \leftarrow$  return from step  $t$ 
     $\theta \leftarrow \theta + \alpha \gamma^t G \nabla_{\theta} \ln \pi(A_t | S_t, \theta)$ 
```

Figura 12 – Algoritmo REINFORCE (SUTTON, 2017).

Parte III

Detalhamento do projeto

4 Requisitos do Projeto

4.1 Requisitos

Este trabalho tem como objetivo o desenvolvimento de um algoritmo de controle inteligente de robôs manipuladores industriais para planejamento de rota e posicionamento em ambiente sujeito a obstáculos em posições desconhecidas. Para avaliar o desempenho do controlador, sua capacidade de aprendizado e a viabilidade de implementação em robôs reais, foram especificados os seguintes requisitos de projeto:

1. Treinamento do controlador por simulação

- Automatizado e deve ser realizado sem necessidade de intervenção humana.
- Posições de destino e de obstáculos devem ser aleatórias dentro da área de trabalho determinada para assegurar generalidade do aprendizado.
- Tempo de processamento médio de cada imagem da simulação inferior a 0.05s/-frame.

2. Validação por simulação

- Taxa de sucesso mínima de 90% para posicionamento em configurações genéricas de destino e de obstáculo.
- Sucesso em casos nos quais o obstáculo bloqueia o caminho direto à posição de destino.

3. Escalabilidade da simulação à robôs manipuladores reais

- Viabilidade do algoritmo em aplicações industriais sob condição de re-treinamento.
- Condições de interrupção por colisão com obstáculos e limitação de curso de atuadores do robô devem ser simuladas e verificadas.

Como visto, os requisitos adotados são divididos em três categorias: Treinamento, Validação e Escalabilidade para robôs reais. A primeira diz respeito à critérios que devem ser satisfeitos durante a etapa de aprendizado do agente, nesta categoria encontram-se requisitos associados à automatização do treino para geração de volume de dados e ao tempo de processamento para garantir tempo total adequado de treino. A segunda contém critérios de desempenho do agente após treino e a terceira compreende requisitos necessários para extensão do trabalho a robôs manipuladores em aplicações reais.

Uma vez que deseja-se estudar a viabilidade do algoritmo implementado em robôs manipuladores industriais, o ambiente de simulação utilizado para o treino deve ser suficientemente próximo do ambiente real no que diz respeito à capacidade de interação do agente com o mesmo por meio do acionamento de cada atuador do robô e da observação do estado do sistema a partir de câmeras. Assim, o modelo do robô KUKA KR16 utilizado na simulação deve ser dimensionalmente e funcionalmente compatível com o robô real. Analogamente, deve ser possível determinar o posicionamento e direcionamento das câmeras no ambiente de simulação.

5 Solução Proposta

Essa seção tem como objetivo detalhar a solução escolhida por meio da especificação da arquitetura do controlador, dos algoritmos de aprendizado e do fluxo operacional adotado ao longo dos testes para otimização de hiperparâmetros.

5.1 Arquitetura de controle

A arquitetura de controle implementada pode ser dividida em três módulos: Ambiente, Interpretador e Agente (figura 13). O ambiente é composto pelo robô, seu sistema de acionamento e o espaço de trabalho. Ao receber uma ação a_t do agente, o ambiente evolui do estado s_t para o estado s_{t+1} segundo sua dinâmica. O Interpretador recebe a cada instante uma imagem representativa do estado atual do sistema e a processa, retornando o estado s_t e a recompensa imediata r_t associada à última ação tomada pelo agente. O agente utiliza a recompensa para o treino e, a partir do estado atual, determina a ação que deve ser tomada de modo a maximizar as recompensas acumuladas.

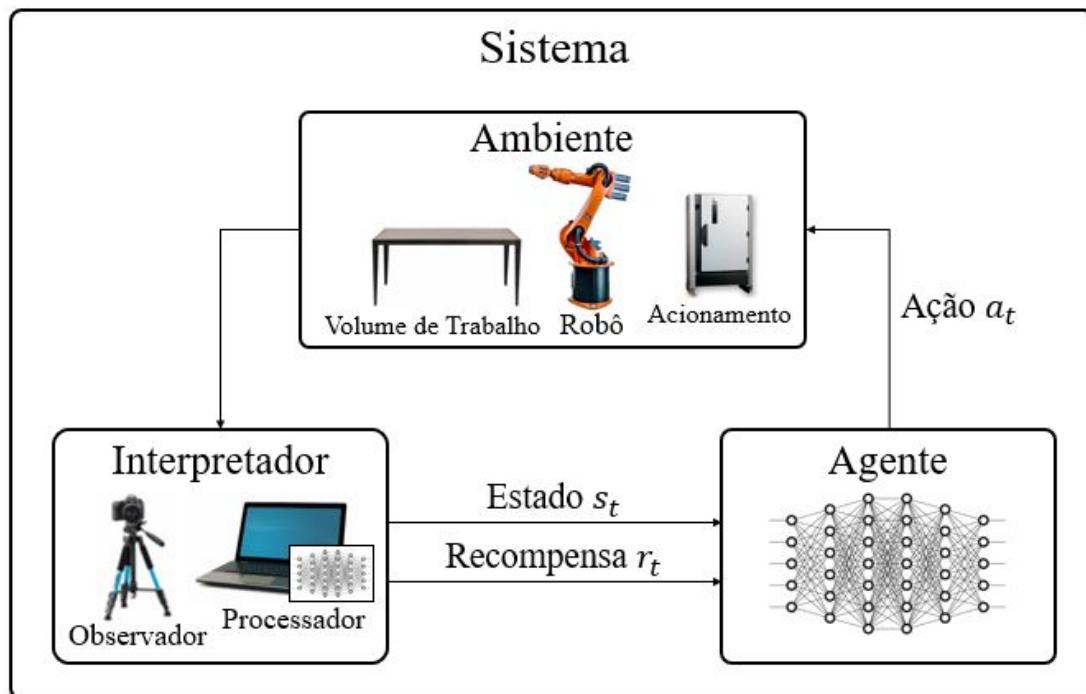


Figura 13 – Diagrama simplificado de arquitetura de treino do controlador por aprendizado por reforço (fonte: Autores).

5.2 Espaço de Estados

Para determinação de um modelo adequado para representação do espaço de estados na solução final foram testados dois modelos diferentes: O primeiro consiste em duas imagens obtidas por câmeras distintas em configuração para percepção de profundidade, enquanto o segundo implementa uma câmera lateral e uma superior.

5.2.1 Primeiro Espaço de Estados Considerado: Configuração para Percepção de Profundidade

O primeiro espaço de estados considerado é dado por $\mathcal{S} \subset K^2$, onde $K = \{p \in \mathbb{Z} | 0 \leq p \leq 255\}^{224 \times 224 \times 3}$ representa o conjunto de imagens RGB de tamanho 224 x 224 pixels. Na simulação foram utilizadas imagens provenientes de duas câmeras em posições distintas e com foco no mesmo ponto, de modo a permitir que o agente possua noção de profundidade (figura 14). As imagens são então pré-processadas para garantir contraste entre o robô e o fundo (figura 15) e posteriormente processadas pela rede convolucional pré-treinada Vgg16 (figura 16). Os *features* extraídos tornam-se entradas da rede densa a ser treinada, que representa a função valor das ações $q(s, a)$ no caso do algoritmo DQN ou a política de ações $\pi_\theta(a_t | s_t)$ no caso do algoritmo REINFORCE.

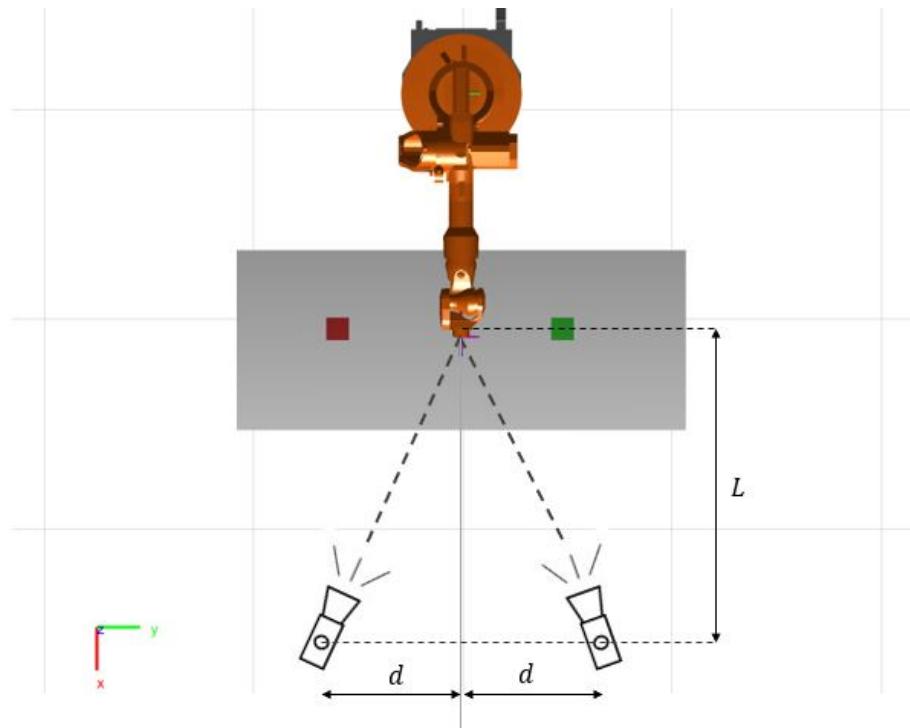


Figura 14 – Diagrama de posicionamento de câmeras para aquisição de imagens durante simulação (fonte: autores).

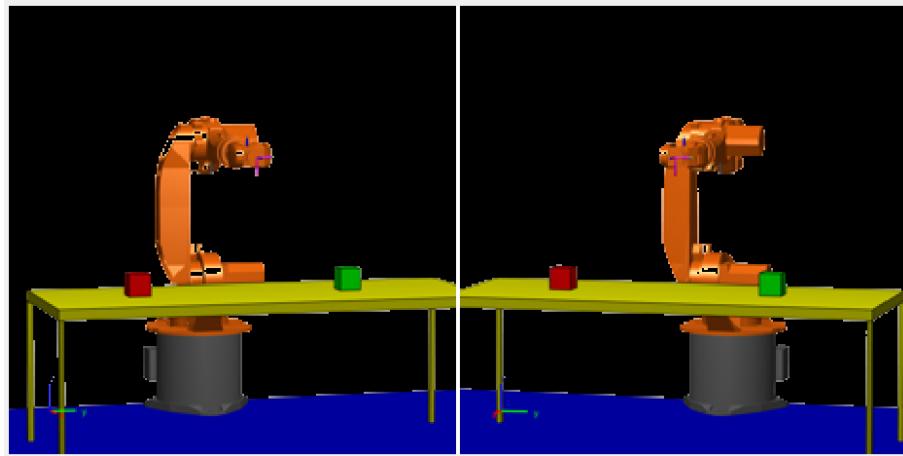


Figura 15 – Estado inicial do sistema capturado pelo sistema de câmeras após pré processamento (fonte: autores).

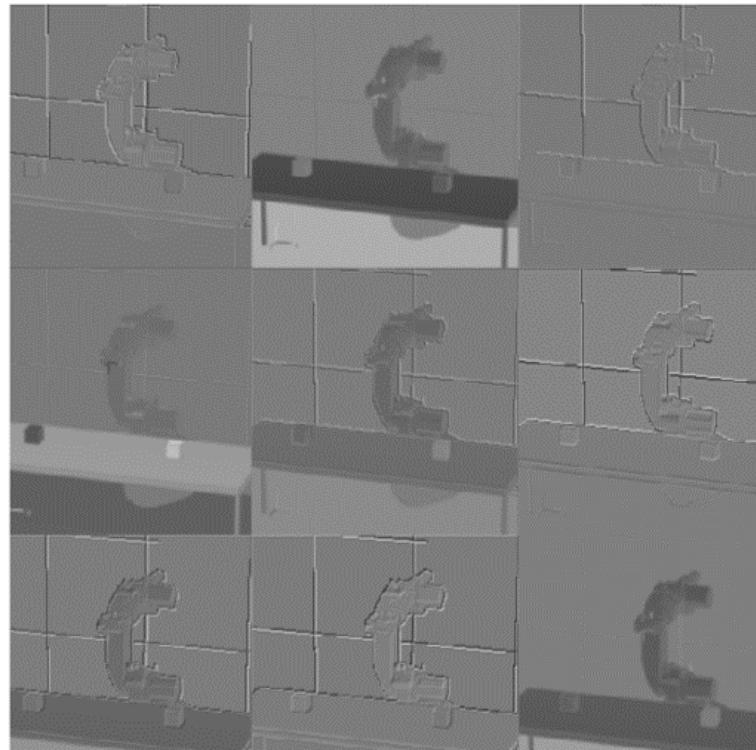


Figura 16 – 9 das 64 ativações da primeira camada convolucional da VGG16 para a configuração inicial do robô, convertidas para imagens em escala cinza (fonte: autores).

Os principais problemas associados à primeira representação do espaço de estados são sua alta dimensionalidade e a necessidade de dimensionamento correto da distância $2d$ entre as câmeras para permitir abstração da noção de profundidade pela rede convolucional. Uma separação pequena entre as câmeras resulta em pouco ganho de informação em relação a uma única câmera e consequente desperdício de capacidade computacional, enquanto

uma separação excessiva impossibilita a percepção de profundidade e impossibilita o controle do robô na direção x .

5.2.2 Segundo Espaço de Estados Considerado: Câmeras Superior e Lateral

Com o objetivo de corrigir os problemas associados ao último espaço de estados, foi considerado um espaço de estados alternativo a partir do reposicionamento das câmeras para maximização da quantidade de informação contida em cada imagem. A utilização da rede convolucional VGG16 e subsequente aplicação de uma rede densa sobre os *features* extraídos foi substituída por uma rede densa aplicada diretamente sobre os *pixels* das imagens que constituem um estado. Apesar de sua aplicação em problemas de processamento e classificação de imagens, redes convolucionais não são absolutamente necessárias neste problema devido à simplicidade das imagens extraídas da simulação. Outro motivo para utilização de uma rede densa diretamente é a possibilidade de redução do número de parâmetros treináveis necessários, de aproximadamente 20 milhões para 1,5 milhão, obtida pela redução da dimensão das imagens capturadas de 224x224x3 para apenas 24x24x3.

Assim, o espaço de estados implementado é dado por $\mathcal{S} \subset K^2$, onde $K = \{p \in \mathbb{Z} | 0 \leq p \leq 255\}^{24x24x3}$ representa o conjunto de imagens RGB de tamanho 24 x 24 pixels. Na simulação foram utilizadas imagens provenientes de duas câmeras distintas, uma superior para identificação de movimentos nas direções x e y, e uma lateral para identificação de movimentos nas direções x e z, conforme figura 17.

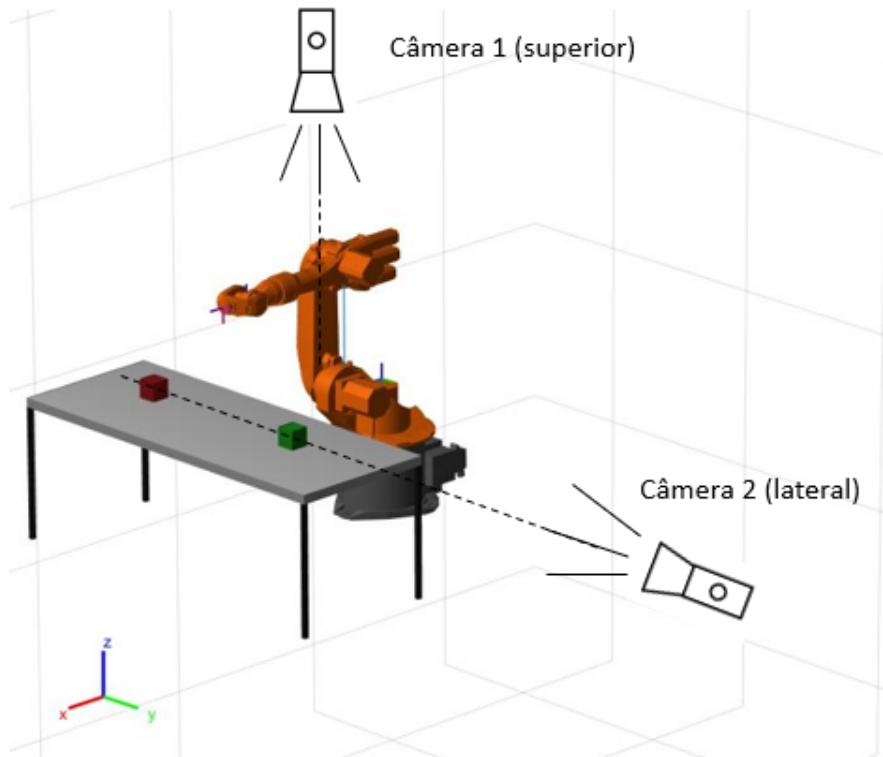


Figura 17 – Diagrama de posicionamento de câmeras superior e lateral (fonte: autores).

A imagem captada pelas câmeras virtuais da simulação devolvem duas imagens que são redimensionadas para o tamanho desejado de 24 por 24 pixels e tem suas partes brancas do fundo trocadas por preto (é aplicado um threshold que substitui valores muito próximos de 255 para 0) como mostra a figura 18. Essa mudança na cor serve para os valores do fundo para não ativarem camadas da rede neural convolucional e interferir no treinamento.

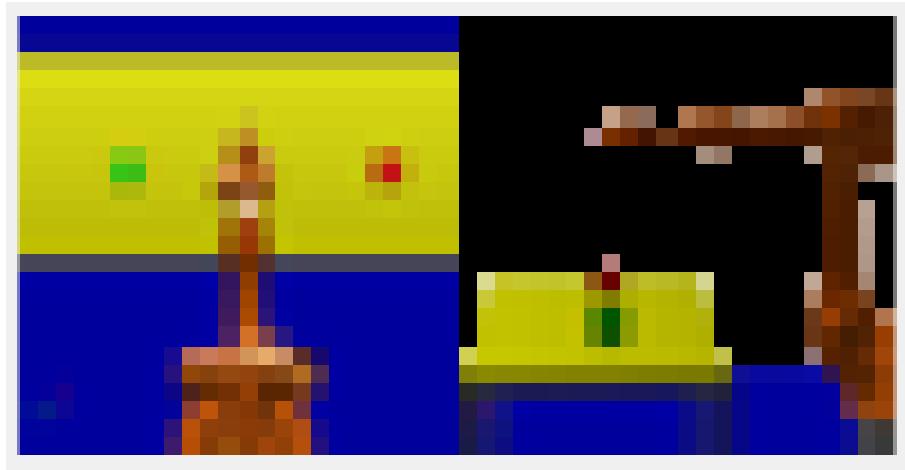


Figura 18 – Estado inicial do sistema ($24 \times 24 \times 3 \times 2$) capturado por câmeras superior e lateral após pré-processamento (fonte: autores).

5.3 Espaço de Ações

O robô KUKA-KR16 apresenta seis graus de liberdade rotativos, os três primeiros são utilizados principalmente para posicionamento e os três últimos para orientação do efetuador por meio de uma configuração de punho esférico, na qual os eixos de rotação das três últimas articulações se interceptam em um único ponto. Para tarefas de posicionamento, no entanto, o último grau de liberdade do robô é desnecessário, uma vez que corresponde apenas a uma rotação em torno da direção de ataque da ferramenta acoplada. Assim, neste trabalho, o último grau de liberdade do robô foi mantido fixo em sua posição inicial.

O primeiro passo para determinação do espaço de ações do problema foi a discretização do acionamento dos motores associados a cada articulação, de modo que cada motor permita apenas três possibilidades:

- Deslocamento angular positivo de um ângulo $\Delta\theta$ ($a_i = 1$).
- Permanência na posição angular atual ($a_i = 0$).
- Deslocamento angular negativo de um ângulo $\Delta\theta$ ($a_i = -1$).

A discretização das ações foi realizada com o objetivo de simplificar a modelagem do sistema e permitir a comparação de diferentes classes de algoritmos, uma vez que

algoritmos de aproximação da função valor, como o DQN, requerem número finito de ações possíveis para extrair a política ótima por meio da busca da ação com maior valor.

O passo seguinte consiste em enumerar as combinações possíveis de ações para o conjunto de cinco articulações angulares controláveis e armazená-las em uma estrutura adequada para indexação lógica e acesso durante a execução do algoritmo. Como há cinco articulações controláveis e cada uma admite três ações distintas, temos um total de 3^5 combinações. Ou seja, o robô possui 243 ações possíveis e o espaço de ações é dado por $\mathcal{A} = \{-1, 0, 1\}^5$. A matriz \mathbf{A} contém em suas linhas todos elementos de \mathcal{A} .

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & 0 \\ 1 & 1 & 1 & -1 & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \quad (5.1)$$

Por fim, a convenção de enumeração das ações exemplificada na equação (5.1) foi utilizada por todas as funções do projeto associadas à tomada da ação por parte do agente ou à determinação do novo estado do sistema dada a ação escolhida.

5.4 Dinâmica do Sistema

A determinação do modelo de transição de estados do sistema em função das ações tomadas pelo agente consiste na função $\mathbf{f} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ que determina o estado seguinte \mathbf{s}' em função do estado atual \mathbf{s} e da ação \mathbf{a} , de modo que $\mathbf{s}' = \mathbf{f}(\mathbf{s}, \mathbf{a})$. A função de transição de estados em Problemas de Decisão de Markov (MDPs) é frequentemente probabilística, conforme descrito na seção 3.3.1 deste trabalho, de modo que, dado um estado e uma ação temos a distribuição de probabilidade do estado seguinte. No controle de robôs manipuladores, a transição probabilística entre estados pode ser utilizada para acrescentar ao modelo ruídos externos, representar dinâmicas não modeladas ou simular escorregamentos em torno da posição esperada.

Neste trabalho, no entanto, optou-se por utilizar uma função de transição de estados determinística com o objetivo de melhor investigar a convergência do aprendizado, uma vez que esta representa suficientemente bem o problema de posicionamento do robô e a modularidade da arquitetura do programa desenvolvido permite a modificação da função de transição de estados de modo rápido, sem alterações no restante do projeto.

Seja $\mathbf{s}_q = (q_1, q_2, q_3, q_4, q_5, x_{sp}, y_{sp}, z_{sp}, x_{obs}, y_{obs}, z_{obs})^T$ o estado interno do sistema, dado por um vetor composto pelas posições angulares das cinco primeiras articulações do robô e as posições cartesianas do *setpoint* e do obstáculo.

Seja $\mathbf{g} : \mathcal{S}_q \rightarrow \mathcal{S}$ uma função de observação, que recebe o estado interno do sistema e o transforma no estado visto pelo agente, de modo que $\mathbf{s} = \mathbf{g}(\mathbf{s}_q)$ é o par de imagens capturado pelo conjunto de câmeras, conforme seção 5.2, dado que o sistema se encontra no estado interno \mathbf{s}_q . Por fim, vamos considerar a ação tomada $\mathbf{a} \in \mathcal{A}$ como $\mathbf{a} = (a_1, a_2, a_3, a_4, a_5)$, onde $a_i \in \{-1, 0, 1\}, \forall i$.

Desta forma, temos a seguinte função de transição de estados:

$$\mathbf{f}(\mathbf{s}, \mathbf{a}) = \mathbf{g}(\mathbf{s}'_q) = \mathbf{g} \left(\mathbf{s}_q + \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right)$$

onde $\mathbf{s}_q = \mathbf{g}^{-1}(\mathbf{s})$ é o estado interno do sistema correspondente ao estado \mathbf{s} observado.

5.4.1 Critérios de Parada

Cada trajetória, ou tentativa de posicionamento do manipulador robótico, necessita de um critério de parada, uma vez que o algoritmo REINFORCE episódico e algumas variações do DQN são classificados como Métodos de Monte Carlo e requerem a simulação completa de um ou mais episódios antes de atualizar os parâmetros da função aproximadora que se deseja treinar. Esta seção tem como objetivo documentar os diversos critérios de parada implementados nos testes e na solução final para determinação do fim de um episódio.

5.4.1.1 Número Máximo T de *Timesteps*

Um dos critérios de parada mais simples é o alcance do número limite T de transições de estado na simulação de uma trajetória. Denominado Número Máximo de *Timesteps*, T deve ser adequadamente dimensionado para o problema em questão, sendo grande o suficiente para que o sistema seja capaz de alcançar o estado desejado a partir do estado inicial, preferencialmente por diversas trajetórias além da ótima. A escolha de um limite T excessivo, no entanto, leva a maior tempo de treino, uma vez que a simulação de cada episódio demanda mais tempo.

É possível ainda não especificar um número limite de transições de estados e deixar que a trajetória percorrida até que o sistema alcance um estado terminal, conforme os próximos critérios de parada discutidos. No entanto, esta solução apresenta risco de trajetórias circulares infinitas, ou extremamente longas. Além disso, as recompensas obtidas em estados finais, ao serem propagadas por toda trajetória com o fator de desconto γ , não serão relevantes em estados próximos ao inicial.

Ao longo dos testes foi determinado um número máximo de transições adequado em $T = 70$, considerando o volume de trabalho adotado e a mínima variação angular escolhida como $\Delta\theta = 1^\circ$.

5.4.1.2 Posição Desejada Alcançada

Outro critério de parada adotado foi o posicionamento correto do efetuador do robô dentro de uma esfera de raio determinado $r_{setpoint}$ centrada na posição desejada. Ao atingir este estado, a simulação da trajetória atual é interrompida e o agente recebe uma recompensa pontual positiva, ou bônus, constante B_{goal} .

Conforme será visto na seção 5.5, esta recompensa pontual pode ser vista como um termo condicional $B_{goal}(\mathbf{s}, \mathbf{a})$ presente na função recompensa, que pode ser nulo caso esse critério de parada não tenha sido atingido ou igual ao bônus caso contrário:

$$B_{goal}(\mathbf{s}, \mathbf{a}) = \begin{cases} B_{goal}, & \text{se } \|\mathbf{p}_{setpoint} - \mathbf{p}'_{ef}\| < r_{setpoint} \\ 0, & \text{caso contrário} \end{cases} \quad (5.2)$$

onde \mathbf{p}'_{ef} é a posição do efetuador no estado $\mathbf{s}' = \mathbf{f}(\mathbf{s}, \mathbf{a})$ seguinte ao estado \mathbf{s} ao tomar a ação \mathbf{a} .

5.4.1.3 Colisões com Mesa e Obstáculo

O critério de parada seguinte é composto tanto por colisões com obstáculos quanto por colisões com a própria mesa. De forma análoga ao critério anterior, determina-se a posição do efetuador e verifica-se a proximidade da mesma com o obstáculo dentro de um raio $r_{obstacle}$ e com a mesa dentro de um raio r_{table} . Ao ser atingido um estado que satisfaça

este critério, a simulação da trajetória é interrompida e o agente recebe uma recompensa pontual negativa, ou penalidade, fixa $P_{collision}$.

Analogamente, esta recompensa pontual pode ser vista como um termo condicional $P_{collision}(\mathbf{s}, \mathbf{a})$ presente na função recompensa:

$$P_{collision}(\mathbf{s}, \mathbf{a}) = P_{table\ collision}(\mathbf{s}, \mathbf{a}) + P_{obstacle\ collision}(\mathbf{s}, \mathbf{a}) \quad (5.3)$$

onde

$$P_{table\ collision}(\mathbf{s}, \mathbf{a}) = \begin{cases} P_{table\ collision}, & se \|\mathbf{p}_{table} - \mathbf{p}'_{ef}\| < r_{table} \\ 0, & caso contrário \end{cases} \quad (5.4)$$

com \mathbf{p}_{table} sendo um ponto pertencente à mesa diretamente abaixo do ponto \mathbf{p}'_{ef} , de modo que considera-se apenas a colisão com a parte superior da mesa.

$$P_{obstacle\ collision}(\mathbf{s}, \mathbf{a}) = \begin{cases} P_{obstacle\ collision}, & se \|\mathbf{p}_{obstacle} - \mathbf{p}'_{ef}\| < r_{obstacle} \\ 0, & caso contrário \end{cases} \quad (5.5)$$

5.4.1.4 Limite de Curso de Articulação

Por fim, foram considerados como critério de parada os casos em que um estado ultrapassa os limites de fim de curso de posição angular de uma ou mais articulações do robô, conforme tabela 2:

Articulação	Limites Angulares
A1	$[-185^\circ, 185^\circ]$
A2	$[-65^\circ, 125^\circ]$
A3	$[-220^\circ, 64^\circ]$
A4	$[-350^\circ, 350^\circ]$
A5	$[-130^\circ, 130^\circ]$
A6	$[-350^\circ, 350^\circ]$

Tabela 2 – Limites de Fim de Curso de Articulações do Robô KUKA-KR16 ajustados para posição inicial escolhida em $[0, 0, 0, 0, 0, 0]^T$.

Caso, no estado atual \mathbf{s} seja tomada uma ação \mathbf{a} que leve o sistema para um estado \mathbf{s}' que viola os limites de fim de curso de pelo menos uma das articulações do robô a simulação da trajetória é interrompida e o agente recebe uma recompensa pontual negativa, ou penalidade, $P_{joint\ boundary}$.

Como nos casos anteriores, pode-se tratar esta recompensa pontual negativa como um termo condicional presente sempre na função recompensa $P_{joint\ boundary}(\mathbf{s}, \mathbf{a})$:

$$P_{joint\ boundary}(\mathbf{s}, \mathbf{a}) = \begin{cases} P_{joint\ boundary}, & \text{se } \mathbf{s}'_{\mathbf{q}} = \mathbf{g}^{-1}(\mathbf{s}') \text{ tal que } \mathbf{s}'_{\mathbf{q}_i} \notin A_{i\ lim}, \forall i = 1, \dots, 5 \\ 0, & \text{caso contrário} \end{cases} \quad (5.6)$$

onde $\mathbf{s}' = \mathbf{f}(\mathbf{s}, \mathbf{a})$. A figura 19 exemplifica um estado terminal em que o limite de fim de curso da articulação rotativa A3 foi alcançado:

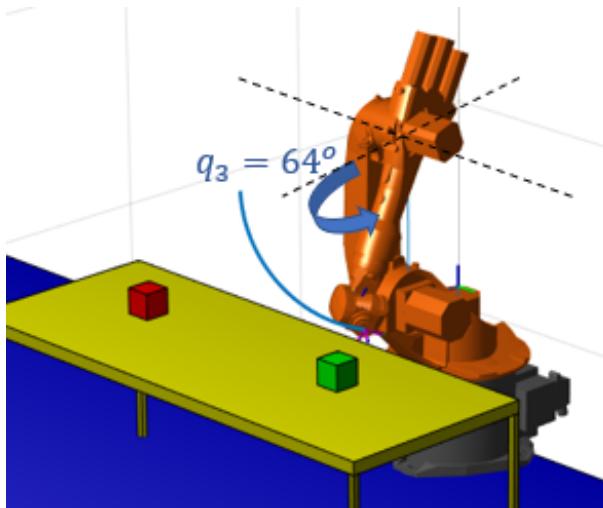


Figura 19 – Visualização de estado terminal devido ao alcance de fim de curso da articulação rotativa A3 do robô (fonte: autores).

5.5 Função Recompensa

A escolha da função recompensa é fundamental em qualquer aplicação de aprendizado por reforço. A função recompensa representa a qualidade de ações tomadas pelo agente e determina quais políticas o mesmo é capaz de aprender.

Com o objetivo de determinar uma função recompensa adequada para o problema, foram realizados testes em variações simplificadas do problema, como robôs com número reduzido de graus de liberdade ou mantendo a posição de destino e do obstáculo fixas durante todo o treino.

5.5.1 Primeira Função Recompensa Considerada: Distâncias Absolutas

As primeiras funções recompensa consideradas foram inspiradas no método do campo potencial, utilizado para planejamento de trajetória de robôs móveis. Em oposição a determinar o vetor de direção do movimento devido ao campo potencial gerado pelas

posições de destino e do obstáculo, as distâncias absolutas entre essas posições e a do efetuador foram utilizadas para cálculo do escalar que representa a recompensa obtida.

$$r_1(\mathbf{s}, \mathbf{a}) = k(r_{setpoint}(\mathbf{s}, \mathbf{a}) + r_{obstacle}(\mathbf{s}, \mathbf{a}) + c) + B_{goal}(\mathbf{s}, \mathbf{a}) + P_{joint\ boundary}(\mathbf{s}, \mathbf{a}) + P_{collision}(\mathbf{s}, \mathbf{a}) \quad (5.7)$$

onde k e c são constantes, B_{goal} é um bônus recebido se, ao tomar a ação \mathbf{a} no estado \mathbf{s} o sistema alcança um estado \mathbf{s}' suficientemente próximo da posição desejada. Analogamente, $P_{joint\ boundary}$ e $P_{collision}$ são penalidades, ou recompensas negativas, obtidas se o estado \mathbf{s}' é um estado em que os limites de curso de alguma articulação do robô são ultrapassados ou se é um estado de colisão com a mesa, respectivamente. Por fim, os termos $r_{setpoint}$ e $r_{obstacle}$ são termos dependentes das distâncias entre o efetuador do robô no estado \mathbf{s}' e a posição de destino e de obstáculos, respectivamente:

$$\begin{cases} r_{setpoint} = -k_1 \|\mathbf{p}_{setpoint} - \mathbf{p}'_{ef}\| \\ r_{obstacle} = k_2 \|\mathbf{p}_{obstacle} - \mathbf{p}'_{ef}\| \end{cases} \quad (5.8)$$

onde \mathbf{p}_{ef} é a posição cartesiana do efetuador do robô no estado \mathbf{s}' no referencial global fixo de sua base, obtida a partir da matriz homogênea ${}^6\mathbf{A}'$ relativa à transformação do sistema de coordenadas fixo da base do robô ao sistema de coordenadas do efetuador:

$$\mathbf{p}'_{ef} = \begin{bmatrix} p'_{ef_x} \\ p'_{ef_y} \\ p'_{ef_z} \end{bmatrix} = \begin{bmatrix} {}^6A'_{1,4} \\ {}^6A'_{2,4} \\ {}^6A'_{3,4} \end{bmatrix} \quad (5.9)$$

A implementação desta função de recompensa nos algoritmos considerados apresentou velocidade de convergência consideravelmente baixa e problemas de instabilidade. Isso se deve principalmente a dois fatores:

- Ações que aumentavam a distância entre o efetuador e a posição desejada não necessariamente recebiam recompensas negativas, muitas vezes as recompensas obtidas eram positivas, apenas menores do que a recompensa associada à ação que minimizava a distância. Esse fator, combinado com exploração limitada de ações em cada estado devido à dimensionalidade, faz com que o algoritmo apresente comportamento convergente para políticas sub-ótimas.
- Outro problema associado à primeira função recompensa implementada é a amplitude de dimensão de seus valores, que fazia com que alterações sucessivas dos pesos da rede política de ações (ou rede DQN) apresentassem magnitude excessiva, o que levava a instabilidade durante o treino. O problema era amenizado com a redução da taxa de aprendizado α , que proporcionava maior estabilidade, mas fazia com

que recompensas de magnitude pequena (particularmente as obtidas em torno do estado inicial), apresentassem mínima influência sobre atualizações da política de ações. Uma solução para este problema é a uniformização da faixa de valores que as recompensas podem assumir.

Com o objetivo de resolver os problemas de instabilidade e tempo de convergência observados, uma segunda função recompensa foi implementada.

5.5.2 Segunda Função Recompensa Considerada: Discreta sob Aproximação ou Distanciamento

A segunda função recompensa utilizada corrige os problemas apresentados pela primeira a partir da limitação da faixa de valores assumidos e translação para média nula, de modo que, dado um estado \mathbf{s} , ações \mathbf{a} consideradas "melhores" do que a média produzem recompensas $r(\mathbf{s}, \mathbf{a})$ positivas e ações "piores" do que a média produzem recompensas $r(\mathbf{s}, \mathbf{a})$ negativas. Esta técnica é semelhante a otimizações do algoritmo REINFORCE para algoritmos Ator-Crítico, que substituem o termo de retorno G_t por uma função vantagem $A_t = G_t - E_{\pi_\theta}[G_t] = Q(\mathbf{s}, \mathbf{a}) - V(\mathbf{s})$, de modo que atualizações da política de ações dependem do valor do retorno obtido em relação ao retorno médio esperado naquele estado.

Na segunda função recompensa implementada, se o efetuador se aproximou da posição desejada devido à ação tomada, um termo positivo e constante $r_{setpoint}$ é atribuído. De modo análogo, se o efetuador se distanciou do obstáculo, dentro de uma específica esfera de influência do obstáculo, um termo constante de mesma magnitude $r_{obstacle}$ é atribuído. Os bônus e penalidades atribuídos a pares Estado-Ação \mathbf{s}, \mathbf{a} que levam a estados terminais de fim de curso $P_{joint boundary}(\mathbf{s}, \mathbf{a})$, colisão $P_{collision}(\mathbf{s}, \mathbf{a})$ e objetivo alcançado $B_{goal}(\mathbf{s}, \mathbf{a})$, são mantidos:

$$r_2(\mathbf{s}, \mathbf{a}) = (k_s r_{setpoint}(\mathbf{s}, \mathbf{a}) + k_o r_{obstacle}(\mathbf{s}, \mathbf{a})) + B_{goal}(\mathbf{s}, \mathbf{a}) + P_{joint boundary}(\mathbf{s}, \mathbf{a}) + P_{collision}(\mathbf{s}, \mathbf{a}) \quad (5.10)$$

em que k_s e k_o são constantes para ajuste de magnitude, $r_{setpoint}(\mathbf{s}, \mathbf{a})$ é uma contribuição discreta devida à aproximação ou distanciamento da posição do efetuador em relação à posição desejada, $r_{obstacle}(\mathbf{s}, \mathbf{a})$ é uma contribuição discreta devida à aproximação ou distanciamento da posição do efetuador em relação à posição do obstáculo, condicional à presença do estado \mathbf{s} dentro de uma esfera de influência de raio r_{infl} centrada no obstáculo.

Assim:

$$r_{setpoint} = \begin{cases} -1, & \text{se } \|\mathbf{p}_{setpoint} - \mathbf{p}'_{ef}\| > \|\mathbf{p}_{setpoint} - \mathbf{p}_{ef}\| \\ 0, & \text{se } \|\mathbf{p}_{setpoint} - \mathbf{p}'_{ef}\| = \|\mathbf{p}_{setpoint} - \mathbf{p}_{ef}\| \\ 1, & \text{se } \|\mathbf{p}_{setpoint} - \mathbf{p}'_{ef}\| < \|\mathbf{p}_{setpoint} - \mathbf{p}_{ef}\| \end{cases} \quad (5.11)$$

$$r_{setpoint} = \begin{cases} 0, & \text{se } \|\mathbf{p}_{obstacle} - \mathbf{p}'_{ef}\| > r_{infl} \\ \begin{cases} -1, & \text{se } \|\mathbf{p}_{obstacle} - \mathbf{p}'_{ef}\| < \|\mathbf{p}_{obstacle} - \mathbf{p}_{ef}\| \\ 0, & \text{se } \|\mathbf{p}_{obstacle} - \mathbf{p}'_{ef}\| = \|\mathbf{p}_{obstacle} - \mathbf{p}_{ef}\|, \text{ caso contrário} \\ 1, & \text{se } \|\mathbf{p}_{obstacle} - \mathbf{p}'_{ef}\| > \|\mathbf{p}_{obstacle} - \mathbf{p}_{ef}\| \end{cases} & \end{cases} \quad (5.12)$$

onde \mathbf{p}_{ef} e \mathbf{p}'_{ef} são as posições cartesianas do efetuador do robô nos estados \mathbf{s} e \mathbf{s}' respectivamente, obtidas a partir das matrizes de transformação homogênea ${}^0_0\mathbf{A}$ e ${}^0_0\mathbf{A}'$, resultantes do cálculo da cinemática direta do robô a partir de seus parâmetros de Denavit-Hartenberg.

O desempenho obtido com a implementação desta função recompensa foi consideravelmente melhor do que com a anterior. Apesar de incentivar a aproximação à posição desejada, o modo de aproximação não era controlado, o que levava a trajetórias sub-ótimas nas quais o robô frequentemente tomava um caminho desnecessariamente mais longo. Com o objetivo de corrigir este problema foi desenvolvida uma terceira função recompensa.

5.5.3 Terceira Função Recompensa Considerada: Projeção de Vetor Deslocamento

Na terceira e última função recompensa implementada, os problemas de instabilidade da primeira e de convergência a trajetórias sub-ótimas da segunda são corrigidos. Dado o estado atual \mathbf{s} e a ação tomada \mathbf{a} , a recompensa obtida é novamente positiva sob aproximação da posição do efetuador à posição desejada, no entanto, em vez de utilizar um termo discreto de contribuição, a contribuição agora depende da projeção do vetor deslocamento sobre o vetor que aponta para a posição desejada (ou posição do obstáculo), calculada por meio do produto escalar dos vetores normalizados:

$$r_3(\mathbf{s}, \mathbf{a}) = (k_s r_{setpoint}(\mathbf{s}, \mathbf{a}) + k_o r_{obstacle}(\mathbf{s}, \mathbf{a})) + B_{goal}(\mathbf{s}, \mathbf{a}) + P_{joint boundary}(\mathbf{s}, \mathbf{a}) + P_{collision}(\mathbf{s}, \mathbf{a}) \quad (5.13)$$

onde:

$$\begin{cases} r_{setpoint}(\mathbf{s}, \mathbf{a}) = (\mathbf{n}_{ef->ef'} \bullet \mathbf{n}_{ef->setpoint}) \\ r_{obstacle}(\mathbf{s}, \mathbf{a}) = \begin{cases} 0, & \text{se } \|\mathbf{p}_{obstacle} - \mathbf{p}'_{ef}\| > r_{infl} \\ (\mathbf{n}_{ef->ef'} \bullet \mathbf{n}_{ef->obstacle}), & \text{caso contrário} \end{cases} \end{cases} \quad (5.14)$$

com $\mathbf{n}_{ef \rightarrow ef'} = \frac{\mathbf{p}_{ef'} - \mathbf{p}_{ef}}{\|\mathbf{p}_{ef'} - \mathbf{p}_{ef}\|}$, $\mathbf{n}_{ef \rightarrow setpoint} = \frac{\mathbf{p}_{setpoint} - \mathbf{p}_{ef}}{\|\mathbf{p}_{setpoint} - \mathbf{p}_{ef}\|}$ e $\mathbf{n}_{ef \rightarrow obstacle} = \frac{\mathbf{p}_{obstacle} - \mathbf{p}_{ef}}{\|\mathbf{p}_{obstacle} - \mathbf{p}_{ef}\|}$.

A figura 20 exemplifica a função de recompensa escolhida para o estado inicial $s = s_0$ e sob duas ações distintas $\mathbf{a}_A = [1, 0, 0, 0, 0]^T$ e $\mathbf{a}_B = [-1, 0, 0, 0, 0]^T$ tais que $r(s, \mathbf{a}_A) < 0$ e $r(s, \mathbf{a}_B) > 0$. Na figura os vetores deslocamento $\mathbf{p}_{ef \rightarrow ef'}$ são representados por setas cheias e os vetores $\mathbf{p}_{ef \rightarrow setpoint}$ e $\mathbf{p}_{ef \rightarrow obstacle}$ são setas tracejadas.

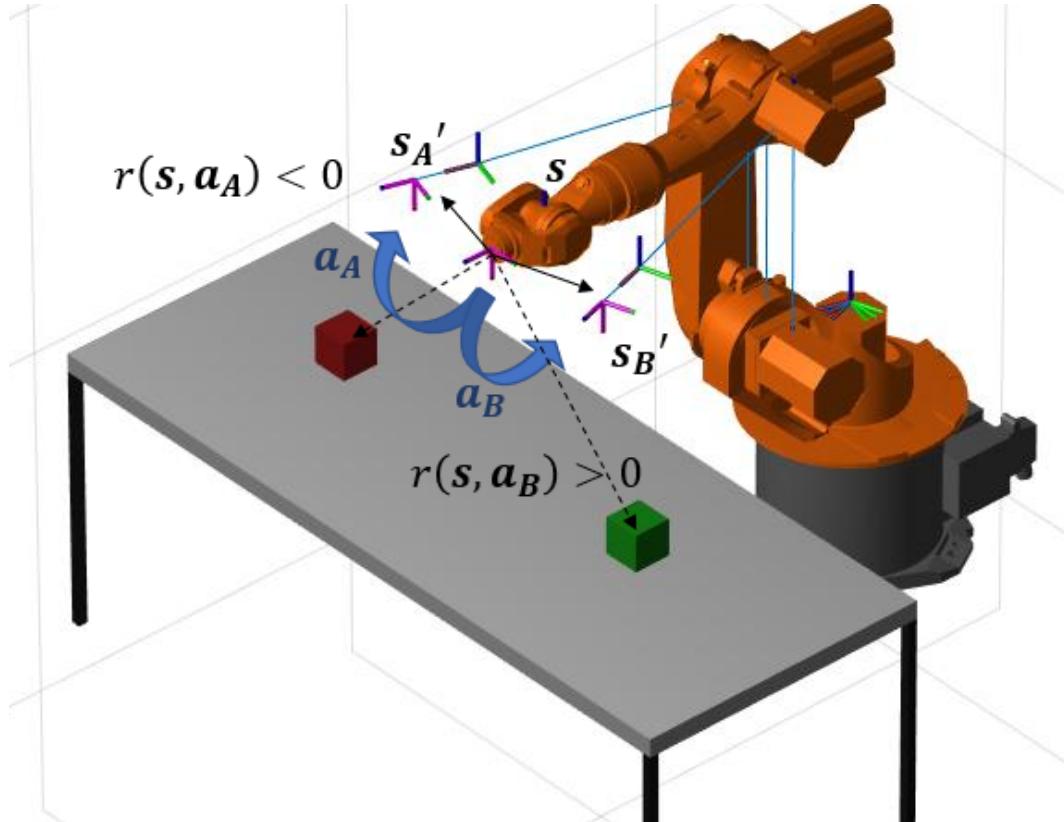


Figura 20 – Diagrama explicativo da função recompensa escolhida (fonte: autores).

5.6 Algoritmos

Ao longo dos testes foram implementados algoritmos das duas classes apresentadas na seção 3.3.4, algoritmos de iteração sobre a política de ações e algoritmos de iteração sobre a função valor dos estados. Da primeira categoria foi implementado o REINFORCE episódico, em que a política de ações $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ parametrizada por uma rede densa de pesos θ é atualizada entre épocas de N trajetórias com o objetivo de maximizar uma função performance $J(\theta)$. Da segunda categoria implementou-se o algoritmo DQN, que aproxima a função valor do par Estado-Ação $Q(s, a)$ e deriva a política de ações ótima a partir desta função, com algumas técnicas de aumento de convergência e estabilização, como a utilização de um *Replay Buffer* para armazenamento de transições de estados com amostragem aleatória priorizada para treino estável da rede DQN, sem correlação temporal entre transições subsequentes.

5.6.1 Primeiro Algoritmo Implementado: REINFORCE Episódico

O primeiro algoritmo implementado foi o algoritmo clássico de iteração sobre a função política de ações, o REINFORCE episódico proposto por Williams (1992), adaptado para o problema de posicionamento de robô manipulador conforme pseudo-código abaixo. O REINFORCE consiste em parametrizar a função política de ações $\pi(\mathbf{s})$ como $\pi_\theta(\mathbf{s})$, e treiná-la por meio de atualizações sucessivas dos parâmetros θ de modo a maximizar uma função Performance $J(\pi_\theta)$ que representa a qualidade da política π_θ .

O algoritmo inicializa a política parametrizada aleatoriamente e a utiliza para gerar N trajetórias, a partir das quais o gradiente da função performance $\nabla J(\theta) = E_\pi \left[G_t \frac{\nabla \pi_\theta(\mathbf{A}_t | \mathbf{S}_t)}{\pi_\theta(\mathbf{A}_t | \mathbf{S}_t)} \right]$ é estimado e utilizado para atualizar os parâmetros θ de acordo com o gradiente ascendente: $\theta \leftarrow \theta + \alpha \nabla J(\theta)$. O vetor $\frac{\nabla \pi_\theta(\mathbf{A}_t | \mathbf{S}_t)}{\pi_\theta(\mathbf{A}_t | \mathbf{S}_t)}$ representa a direção no espaço de parâmetros θ que maximiza a probabilidade de a ação A_t ser tomada no estado S_t . Assim, retornos positivos G_t fazem com que a probabilidade de o agente tomar as ações que levaram a aqueles retornos seja aumentada em futuras visitas ao mesmo estado, sendo que o aumento da probabilidade é diretamente proporcional ao retorno G_t e retornos negativos fazem com que as respectivas ações sejam desencorajadas no futuro.

Algoritmo 1: REINFORCE Episódico

- Inicializa Robô, *setpoint*, *obstáculo*, estado inicial \mathbf{s}_0 e espaço de ações \mathcal{A} ;
- Inicializa Hiperparâmetros (bônus e penalidades, tamanho da rede, número de *timesteps*, trajetórias e épocas, fator de desconto γ e taxa de aprendizado α);
- Inicializa estruturas de armazenamento de épocas e políticas de ações;
- Inicializa política de ações parametrizada π_{θ_0} aleatória e armazena em PolicyBuffer(1);

Gera N trajetórias $\{\tau_n\}_{n=1}^N$ a partir de política de ações π_{θ_0} , onde

$\tau_n = \mathbf{S}_0, \mathbf{A}_0, R_0, \dots, \mathbf{S}_{T-1}, \mathbf{A}_{T-1}, R_T$;

Calcula retornos $\{G_t\}_{t=0}^{T-1}$ e armazena em cada trajetória;

Armazena $\{\tau\}_{n=1}^N$ em EpochBuffer(1);

for $ep \leftarrow 2$ **to** $MaxEpoch$ **do**

Aplica Gradiente Ascendente sobre $\pi_{\theta_{ep-1}}$ para obter $\pi_{\theta_{ep}}$:

$$\theta_{ep} \leftarrow \theta_{ep-1} + \alpha \frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{T-1} G_t \frac{\nabla \pi_\theta(\mathbf{s}_t, \mathbf{a}_t)}{\pi_\theta(\mathbf{s}_t, \mathbf{a}_t)};$$

Armazena $\pi_{\theta_{ep}}$ em PolicyBuffer(ep);

Gera N trajetórias $\{\tau_n\}_{n=1}^N$ a partir de política de ações $\pi_{\theta_{ep}}$, onde

$\tau_n = \mathbf{S}_0, \mathbf{A}_0, R_0, \dots, \mathbf{S}_{T-1}, \mathbf{A}_{T-1}, R_T$;

Calcula retornos $\{G_t\}_{t=0}^{T-1}$ e armazena em cada trajetória;

Armazena $\{\tau\}_{n=1}^N$ em EpochBuffer(ep);

Mostra performance média da época atual;

Mostra melhor trajetória da época atual;

end

Algoritmos da classe de iteração sobre a função política de ações são tipicamente utilizados em problemas nos quais o espaço de ações é contínuo, onde algoritmos da classe oposta são inviáveis, uma vez que, dado o estado atual \mathbf{s} , necessitam buscar todas as ações possíveis e tomar aquela com maior valor $Q(\mathbf{s}, \mathbf{a})$. Neste problema, implementamos o REINFORCE de modo que a saída $\mathbf{y} = \pi_\theta(\mathbf{s})$ contém as probabilidades individuais de cada ação $\{\mathbf{a}_i\}_{i=1}^{243}$ ser tomada no estado \mathbf{s} e a ação é escolhida de acordo com a distribuição gerada por meio do método da roleta. No entanto, pode-se reformular o REINFORCE para ações contínuas modelando a distribuição de probabilidades sobre o espaço de ações dado um determinado estado como uma Gaussiana $\mathcal{N}(\mu, \Sigma)$, de modo que a política de ações retorna sua média e variância (ou matriz de covariância no caso de ações multidimensionais): $\pi_\theta(\mathbf{s}) = [\mu, \Sigma]^T$.

Em aplicações práticas a entrada de controle é o conjunto de tensões elétricas aplicadas em cada motor e o espaço de ações \mathcal{A} resultante é contínuo, de modo que algoritmos de iteração sobre a função política de ações seriam mais apropriados. No entanto, no problema analisado, algoritmos de iteração sobre a função valor, como o DQN, demonstraram melhor desempenho em testes iniciais. Assim, o REINFORCE episódico foi implementado apenas em testes simplificados, nos quais o espaço de estados é dado pelas posições angulares das articulações do robô em conjunto com as posições cartesianas de destino e de obstáculo, em oposição ao problema original em que são utilizadas câmeras para captura do estado. A seção seguinte explicita o algoritmo implementado para resolver o problema original.

5.6.2 Segundo Algoritmo Implementado: DQN

O segundo algoritmo utilizado, denominado DQN, consiste na parametrização da função valor do par estado-ação $Q_\theta(\mathbf{s}, \mathbf{a})$ por uma rede neural com pesos θ , treinada de modo a satisfazer a Equação de Bellman (Equação 3.10), ou seja:

$$Q_\theta(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q_\theta(\mathbf{s}_{t+1}, \mathbf{a}') \quad (5.15)$$

O DQN pode ser visto como um algoritmo de Aprendizado Supervisionado em que o valor alvo, ou *target*, é não estacionário, uma vez que depende da própria função Q . Essa é uma das grandes dificuldades associadas a este método e faz com que sua convergência dependa da exploração suficiente de diferentes ações sobre todo o espaço de estados. Outra dificuldade é a instabilidade durante o treino devido à atualizações sucessivas em estados vizinhos: A longo do treino, especialmente se o espaço de estados for dado por imagens, a diferença entre estados vizinhos é mínima e atualizações sucessivas dos parâmetros θ da rede DQN vão focar em uma única região do espaço de estados, isso por sua vez, comprometerá as estimativas dos valores Q de estados distintos e impedirá a exploração de uma nova região do espaço de estados \mathcal{S} . Os problemas de instabilidade e exploração

adequada são resolvidos por meio das seguintes técnicas de melhoria de convergência do DQN:

- **Buffer de Experiência:** A implementação de um *Buffer* de Experiência que contém um número fixo de objetos $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}, \text{bool}_{term})$, denominados transições, soluciona o problema da correlação entre amostras e proximidade de estados durante o treino. Cada transição contém:
 - \mathbf{s}_t : O estado atual do sistema no instante de captura da transição;
 - \mathbf{a}_t : A ação tomada pelo agente no instante de captura da transação;
 - r_t : Recompensa obtida pelo agente ao tomar a ação \mathbf{a}_t no estado \mathbf{s}_t ;
 - \mathbf{s}_{t+1} : Estado seguinte após transição do sistema devido à execução da ação \mathbf{a}_t no estado \mathbf{s}_t . Como o problema é modelado como um Processo de Decisão de Markov determinístico, temos que $\mathbf{s}_{t+1} = \mathbf{f}(\mathbf{s}_t, \mathbf{a}_t)$;
 - bool_{term} : Variável booleana que determina se a transição levou a um estado \mathbf{s}_t terminal, conforme critérios de parada definidos na seção 5.4.1;

Em vez de treinar a rede DQN a partir de transições sequenciais, faz-se uso da mesma para preencher um *buffer* com diversas transições e realiza-se uma amostragem aleatória deste *buffer* sobre a qual o treino será feito. Isso estabiliza o treinamento do agente assumindo que o conjunto de transições armazenadas é suficientemente representativo e diverso, o que é solucionado através de técnicas de exploração de estados-ações.

- **Política ϵ -Greedy:** A rede DQN inicializada aleatoriamente frequentemente apresentará valores $Q(\mathbf{s}, \mathbf{a})$ maiores para estados $\mathbf{s} \in \mathcal{W} \subset \mathcal{S}$ em uma região do espaço de estados \mathcal{S} em torno do estado inicial \mathbf{s}_0 associados a uma única ação. Nesses casos, a rede DQN é considerada tendenciosa e a política de ações derivada da mesma escolhendo-se sempre a ação de maior valor Q apresentará pouca exploração do ambiente. Ao mesmo tempo, conforme o treino se aproxima do fim, o agente possui conhecimento mais próximo da função valor real Q^* e a exploração de estados distintos é menos importante do que o refinamento da política quase ótima que o agente já possui. A esse conceito é dado o nome de *Exploration x Exploitation Trade-Off*. Um método simples de garantir que a exploração seja incentivada no começo do treino e gradualmente substituída pelo aproveitamento é a implementação de uma política " ϵ -Greedy" para escolha da ação a ser tomada, conforme equação

$$\mathbf{a}_t = \begin{cases} \text{argmax}_{\mathbf{a}' \in \mathcal{A}} Q(\mathbf{s}_t, \mathbf{a}'), & \text{se } \text{rand}(0, 1) > \epsilon \\ \text{ação } \mathbf{a} \in \mathcal{A} \text{ aleatória,} & \text{caso contrário} \end{cases} \quad (5.16)$$

onde $\text{rand}(0,1)$ é um número aleatório amostrado de uma distribuição uniforme no intervalo $(0, 1)$ e ϵ é um parâmetro que decresce ao longo do treino. Adotando $\epsilon = 1$ para a primeira época e multiplicando-o por um fator de decaimento, por exemplo 0,99, a cada época, o agente irá gradativamente transicionar entre uma política de exploração pura para uma política de aproveitamento. O valor de ϵ , assim como seu fator de decaimento, são hiperparâmetros do algoritmo e devem ser dimensionados adequadamente para o problema em questão.

- **Amostragem Priorizada:** O treinamento do agente é realizado a partir de transições amostradas do *Buffer* de Experiência. No entanto, a amostragem aleatória de transições para treino, apesar de apresentar de apresentar resultados melhores em comparação ao treino com transições sequenciais, ainda promove tempo de convergência sub-ótimo do aprendizado. Intuitivamente, deseja-se utilizar as transições de maior importância para o treino, de modo a maximizar a extração de informações novas por parte do agente. Desta forma, a prioridade com a qual transições são amostradas é diretamente proporcional ao erro de Bellman associado: $|r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q_\theta(\mathbf{s}_{t+1}, \mathbf{a}') - Q(\mathbf{s}_t, \mathbf{a}_t)|$.

Apesar de $Q_\theta(\mathbf{s}, \mathbf{a})$ ser uma função do espaço dos pares estado-ação $\mathcal{S} \times \mathcal{A}$ no espaço de estados \mathcal{S} , a estrutura da rede DQN utilizada para aproxima-la não precisa refletir as mesmas dimensões de entrada e saída. A implementação de uma rede DQN com entradas em $\mathcal{S} \times \mathcal{A}$ (figura 21 (a)) apresenta custo computacional elevado para determinação da ação de maior valor $\mathbf{a} = \text{argmax}_{\mathbf{a}' \in \mathcal{A}} Q_\theta(\mathbf{s}, \mathbf{a}')$, uma vez que deve-se calcular a saída da rede para cada ação individual $\mathbf{a}_i \in \mathcal{A}$. Uma alternativa é a aproximação da função $Q(\mathbf{s}, \mathbf{a})$ por uma rede DQN com entradas em \mathcal{S} e saídas em \mathcal{A} , de modo que o problema se reduz a procurar o maior valor entre as saídas $Q(\mathbf{s}, a_1), \dots, Q(\mathbf{s}, a_n)$ (figura 21 (b)).

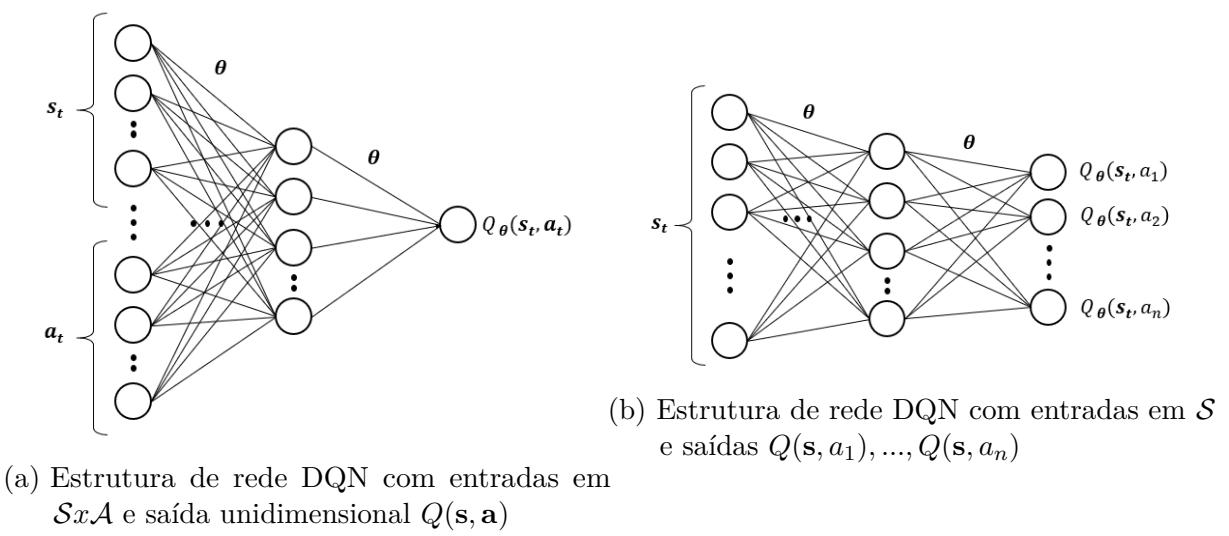


Figura 21 – Possíveis arquiteturas de rede DQN para aproximação de função $Q(\mathbf{s}, \mathbf{a})$ (fonte: autores)

O algoritmo DQN, originalmente proposto por MNIH et al, 2013, foi adaptado para o problema de posicionamento de robô manipulador especificado no capítulo 5 deste trabalho, conforme pseudo-código abaixo:

Algoritmo 2: DQN

- Inicializa Robô, *setpoint*, *obstáculo*, estado inicial s_0 e espaço de ações \mathcal{A} ;
- Inicializa Hiperparâmetros (bônus e penalidades, tamanho da rede e das imagens, número de *timesteps*, épocas e transições no *Buffer*, fator de desconto γ , taxa de aprendizado α) e ϵ ;
- Inicializa estruturas de armazenamento de épocas e redes DQN;
- Inicializa rede DQN parametrizada Q_{θ_0} aleatória e armazena em DQNBuffer(1);

for $ep \leftarrow 1$ **to** $MaxEpoch$ **do**

- inicializa estado: $s \leftarrow s_0$;
- Preenche *Buffer* de Experiência com N transições segundo política ϵ -*Greedy* e rede atual $Q_{\theta_{ep}}$;
- Amostra *mini-batch* aleatório de tamanho N_{batch} ;
- for** $i \leftarrow 1$ **to** N_{batch} **do**

 - Ler i-ésima transição: $(s, a, r, s', bool_{term})$;
 - if** $bool_{term} == true$ **then**

 - $y = r$;

 - else**

 - $y = r + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{ep}}(s', a')$;

 - end**
 - Armazenar saída prevista $q = Q(s, a)$ e alvo y ;

- end**
- Aplica Gradiente Descendente para minimizar função custo dada por

$$\mathcal{L}(\theta) = \frac{1}{2}(Q_\theta(s, a) - y)^2$$
, ou seja:

$$\theta_{ep+1} \leftarrow \theta_{ep} - \alpha \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \nabla_\theta \frac{1}{2}(Q_\theta(s, a) - y)^2$$
;
- Armazena rede $Q_{\theta_{ep+1}}$ em DQNBuffer(ep+1);
- Limpa *Buffer* de transições;
- Mostra trajetória *greedy* atual;
- Mostra valor médio de recompensas imediatas;

end

Parte IV

Ferramentas

6 Softwares

Aplicações atuais de Aprendizado por Reforço na Robótica ainda são mais comuns em ambientes virtuais de simulação do que em ambientes físicos devido principalmente a três motivos: O primeiro consiste no desenvolvimento relativamente recente de uma teoria formalizada de RL, de modo que é natural que a mesma seja primeiramente testada antes de ser implementada em aplicações práticas. O segundo está relacionado ao grande volume de dados necessário para o treino do agente em algoritmos de RL, uma vez que os mesmos são frequentemente inefficientes em termos de amostras. Implementações em ambientes virtuais de simulação corrigem este problema, pois possibilitam a geração de volume de dados de forma rápida e automática. Por fim, a própria natureza de Aprendizado por Reforço requer que o agente explore ações positivas e negativas, de modo que o treino direto em ambientes físicos pode proporcionar risco à segurança de indivíduos e à integridade de equipamentos. Assim, novas arquiteturas de controle são frequentemente testadas em ambientes de simulação.

Softwares típicos para teste e desenvolvimento de algoritmos de Aprendizado por Reforço refletem um setor propenso para a aplicação dos mesmos: Jogos eletrônicos. Jogos são facilmente modelados como Processos de Decisão de Markov, possuindo espaços de estados e ações inherentemente bem definidos. Entre as principais plataformas para desenvolvimento e análise de algoritmos de RL e DRL está o *Arcade Learning Environment* (BELLEMARE, 2013), uma plataforma integrada a um emulador do console Atari 2600 com centenas de jogos e uma interface específica para testes de algoritmos de IA. A figura 22 mostra cinco capturas de tela de jogos disponíveis no ALE.



Figura 22 – Cinco jogos de Atari 2600 disponíveis no ALE: (Da esquerda para a direita) Pong, Breakout, Space Invaders, Seaquest e Beam Rider (MNIH, 2013)

No que diz respeito a robótica, o software ROS (*Robot Operating System*) é a principal referência na simulação de manipuladores e de robôs móveis, possuindo bibliotecas e ferramentas de abstração da camada de hardware, de visualização gráfica, de simulação de controladores e de sensores. A figura 23 ilustra a interface gráfica de visualização do software ROS na simulação de robô móvel na tarefa de planejamento de rotas.

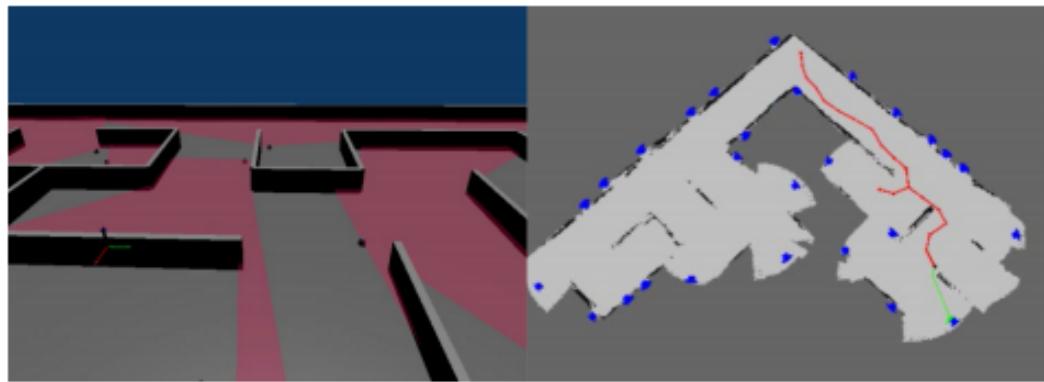


Figura 23 – Simulador MORSE em software ROS de robô móvel para planejamento de trajetória (YAN, 2017)

Por fim, a linguagem de programação Python possui acesso à conhecida plataforma de desenvolvimento e comparação de algoritmos de RL, denominada *OpenAI Gym*, além de bibliotecas que facilitam a implementação e o treino de redes neurais, como *Tensorflow* e *Keras*. Similar ao ALE, o *OpenAI Gym* possui ambientes de teste já implementados com o objetivo de facilitar o treino de agentes. Entre os ambientes disponíveis no *OpenAI Gym* estão *Pong*, *Space Invaders* e o problema de equilíbrio de um pêndulo invertido sobre carro (figura 24).

A existência de trabalhos semelhantes desenvolvidos em linguagem Python sobre a plataforma *OpenAI Gym* para fins comparativos em conjunto com o amplo suporte de redes neurais fazem do software uma escolha adequada para o problema. No entanto, a ausência de suporte a robôs manipuladores industriais como o KUKA-KR16 em conjunto com a dificuldade de implementação de interface gráfica para visualização da simulação inviabilizam a escolha da linguagem Python para resolução do problema proposto neste projeto.

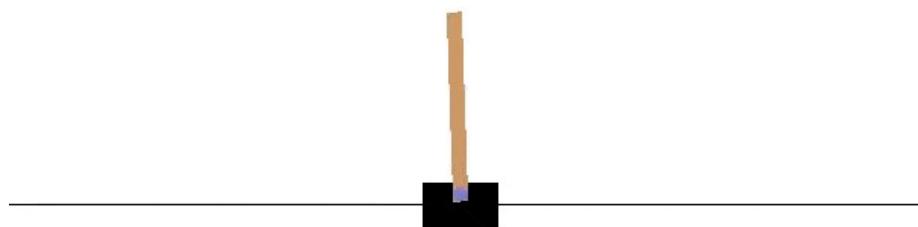


Figura 24 – Visualização de ambiente de pêndulo invertido sobre carro em plataforma *Open AI Gym* (fonte: autores)

Zamora *et al.* (2013) desenvolveram uma extensão da plataforma *Open AI Gym*

integrada aos softwares de simulação robótica ROS e Gazebo e demonstraram seu funcionamento no problema de navegação em labirinto e ambientes sujeitos a obstáculos por parte dos robôs *Turtlebot*, *Erle-Rover* e *Erle-Copter*, conforme figura 25. No entanto, ainda não há suporte para robôs manipuladores industriais em plataformas como a *Open AI Gym*. Em estudos futuros, a simulação do robô KUKA-KR 16 em software ROS com suporte de ambientes semelhante ao fornecido pela plataforma *Open AI Gym* é a arquitetura ideal para desenvolvimento e implementação de algoritmos de RL para solução do problema proposto.

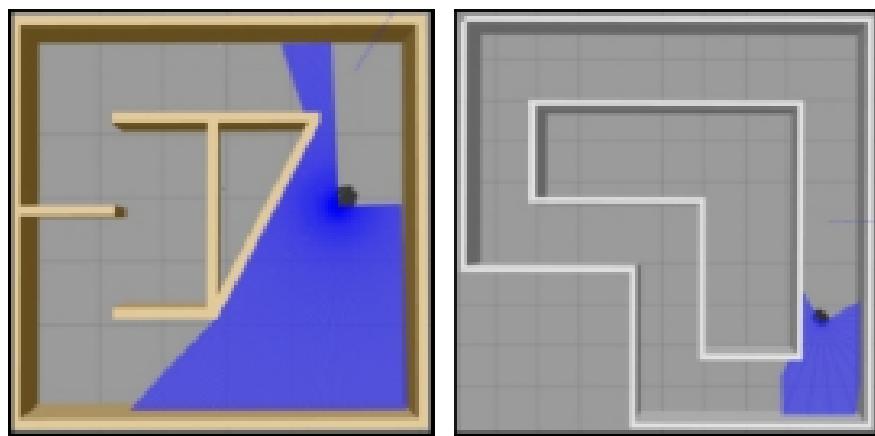


Figura 25 – Robô *Turtlebot* com sensor LIDAR em ambientes de teste implementados (ZAMORA et al, 2013)

Por fim, outro software disponível para implementação de um modelo do ambiente e do robô, treinamento do agente e visualização e análise dos resultados é o *Matlab*. A existência de *Toolboxes* específicas de Robótica, Redes Neurais e Aprendizado por Reforço permite, respectivamente, a importação de um modelo do robô baseado em uma árvore de corpos rígidos para cálculo de cinemática direta e visualização de pose, importação ou criação de redes densas e implementação e treino de agentes de algoritmos de RL. Assim, o *Matlab* foi escolhido como software para realização dos testes iniciais e implementação da solução final.

6.1 Matlab

Esta seção tem como objetivo introduzir o software utilizado para simulação do ambiente e do robô, explicitar os motivos que levaram a esta decisão, detalhar a arquitetura de projetos desenvolvidos, assim como o fluxo de execução do código e a documentação das funções implementadas.

6.1.1 Arquitetura de Projetos Desenvolvidos

Anteriormente à implementação da solução final, foram realizados diversos testes em versões simplificadas do problema ou dos algoritmos, de modo a focar exclusivamente em seus fundamentos, eliminando fontes de instabilidade e não convergência. Entre as simplificações realizadas estão: Diminuição do número de graus de liberdade do robô, redução da dimensionalidade dos espaços de estados \mathcal{S} e de ações \mathcal{A} , variações simplificadas dos algoritmos implementados. Para cada simplificação adotada foi desenvolvido um projeto específico para análise daquele caso particular, sendo que diferentes projetos compartilham funções similares que foram desenvolvidas de forma modular para permitir compatibilidade e escalabilidade do estudo. Conforme resultados positivos de convergência eram obtidos, as simplificações adotadas foram gradativamente reduzidas e a complexidade dos projetos foi aumentada.

Os nomes dos projetos desenvolvidos, as simplificações adotadas e o objetivo proposto por cada um são descritos a seguir:

- **PGRobotArmR:** O primeiro projeto desenvolvido foi a implementação do algoritmo REINFORCE para treinamento na tarefa de mover um braço robótico plano de um grau de liberdade rotativo (R) para alcançar um objeto pontual e desviar de um obstáculo, ambos conhecidos pelo agente. O estado s considerado contém a posição angular do braço robótico e as posições x e y do *setpoint* e obstáculo, possuindo dimensão 5. O agente possui apenas três ações possíveis, movimento horário de $\Delta\theta$, movimento anti-horário de $\Delta\theta$ e permanecer parado. O principal objetivo deste projeto é a verificação de convergência do treino da rede política de ações $\pi_\theta(\mathbf{a}|s)$ no ambiente mais simples possível.
- **QLearningRobotArmR:** Desenvolvido a partir de finalidade comparativa com o primeiro projeto, o segundo implementa exatamente o mesmo ambiente, mas substitui o algoritmo pelo Q-Learning clássico baseado em tabela. O agente é treinado para atualizar sucessivamente uma tabela que associa cada estado (em suas linhas) e cada ação (em suas colunas), com um valor $Q(s, a)$. Os espaços de estados e ações são iguais ao projeto anterior, mas as ações são escolhidas agora a partir da verificação da ação com maior valor Q na tabela. A figura 25 ilustra o ambiente de simulação deste projeto no primeiro episódio do treino.
- **PGRobotArmRR:** O terceiro projeto suaviza as simplificações sobre o número de graus de liberdade do robô, adotando agora um robô plano de dois graus de liberdade rotativos (RR). Neste projeto, a dimensão de cada estado s cresceu em relação aos anteriores de 5 para 6, com a necessidade de armazenar a posição angular da segunda articulações rotativa. Enquanto o tamanho de cada ação a cresceu de uma para duas dimensões. A principal diferença, no entanto, foi sobre as dimensões

dos espaços de estados \mathcal{S} e de ações \mathcal{A} , que cresceram exponencialmente. O principal objetivo deste projeto foi a comparação com o primeiro, de modo a manter fixo o algoritmo REINFORCE utilizado e analisar a influência do número de graus de liberdade.

- **QLearningRobotArmRR:** Este projeto possui relação com o anterior análoga à relação entre o primeiro par de projetos. Nele foi mantido idêntico o ambiente e as dimensionalidades do espaço de estados e de ações, mas foi alterado o agente para implementar o algoritmo Q-Learning clássico baseado em tabela. Os principais objetivos foram as comparações no que diz respeito à dimensionalidade com o projeto QLearningRobotArmR e no que diz respeito ao algoritmo com o projeto PGRobotArmRR. A figura 26 ilustra o ambiente de simulação deste projeto no segundo episódio do treino.
- **PGKuka:** Após os testes de ambos algoritmos em robôs com estruturas simplificadas optou-se pela implementação do algoritmo REINFORCE episódico em ambiente tridimensional com robô KUKA-KR16, conforme especificado para solução final no capítulo 5 deste trabalho. As simplificações adotadas neste projeto foram a utilização de espaço de estados interno composto por posições angulares das articulações do robô e posições cartesianas do *setpoint* e obstáculo, em oposição à utilização de imagens. Outra simplificação é a configuração fixa de posições de destino e de obstáculo, de modo que o agente não precisa ser capaz de efetuar o posicionamento para configurações genéricas.
- **DQNKuka:** Este projeto implementa o mesmo ambiente que o projeto PGKuka, em conjunto com as mesmas simplificações de utilização de posições angulares das articulações do robô e de fixação de configuração de *setpoint* e obstáculo. A diferença é a implementação do agente, que agora é treinado segundo o algoritmo DQN. Dada dimensão do espaço de estados (aproximadamente $2,54 \cdot 10^{15}$ para $\Delta\theta = 1^\circ$) e de ações (3^5), é inviável implementar o algoritmo Q-Learning por tabela e torna-se necessária a utilização de uma função aproximadora, como uma rede neural, para representar a função $Q(\mathbf{s}, \mathbf{a})$.
- **PGKukaRandomPositions:** Este projeto é equivalente ao projeto PGKuka com a suavização da hipótese de configuração fixa de posição de destino e posição de obstáculo. É implementado o REINFORCE episódico com configurações geradas aleatoriamente sobre a mesa entre cada época do treino. O principal objetivo deste projeto foi a verificação de capacidade de generalização do agente para configurações nunca observadas. Ou seja, verificar se uma única política de ações ao longo do treino é capaz de abstrair o comportamento necessário para posicionar o efetuador em posições completamente diferentes.

- **DQNKukaRandomPositions:** Este projeto implementa o algoritmo DQN em ambiente idêntico ao anterior e sob a mesma hipótese simplificadora de que o agente tem conhecimento das posições angulares de cada articulação do robô e as usa como estado em vez de imagens capturadas do sistema. O principal objetivo deste projeto é a verificação do desempenho e do tempo de treino de um agente DQN sujeito a tarefa de posicionamento do efetuador em posições genéricas em comparação a um agente treinado pelo algoritmo REINFORCE em mesmo ambiente.
- **DQNKukaImage:** A última simplificação a ser removida foi a utilização de posições das articulações como estados em vez de imagens capturadas. Este projeto implementa o algoritmo DQN diretamente sobre os *pixels* de imagens da simulação, conforme seção 5.2 deste trabalho.

A figura 26 ilustra uma tabela dos projetos de teste desenvolvidos agrupados de acordo com seus algoritmos e simplificações adotadas.

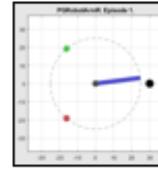
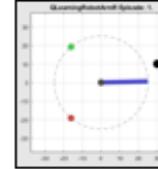
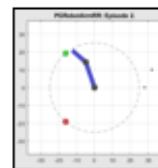
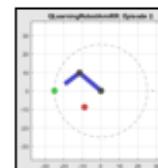
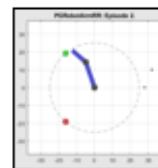
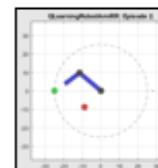
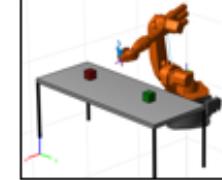
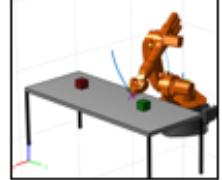
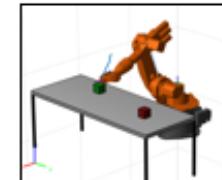
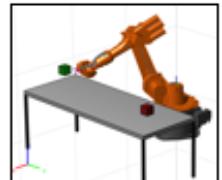
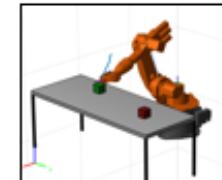
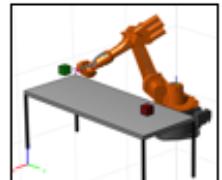
Projetos de Teste	REINFORCE	Q-Learning
1 Grau de Liberdade (R)		
		
2 Graus de Liberdade (RR)		
		
6 Graus de Liberdade (6R), configuração fixa		
		
6 Graus de Liberdade (6R), configurações aleatórias		

Figura 26 – Tabela de projetos simplificados desenvolvidos para teste, separados por simplificação adotada e algoritmo implementado (fonte: autores)

6.1.2 Estruturas de Armazenamento

A implementação do *Buffer* de Experiências no algoritmo DQN requer o armazenamento de um conjunto de transições que constituem a memória do agente, ou seja, os dados a partir dos quais o mesmo é treinado a cada época. Com o objetivo de analisar o desempenho do agente em diferentes épocas do treino e verificar a convergência do aprendizado foram implementadas adicionalmente estruturas de armazenamento de trajetórias e da rede DQN. Esta seção tem como objetivo apresentar as estruturas de armazenamento, as variáveis e os métodos contidos em cada uma.

6.1.2.1 Armazenamento de Transições

Foi implementada a classe *transitionBuffer* que contém um vetor de objetos da classe *transition* e um inteiro que representa o número de transições armazenadas na memória do agente. O diagrama de classes da figura 27 exemplifica a relação entre as classes de armazenamento de transições.

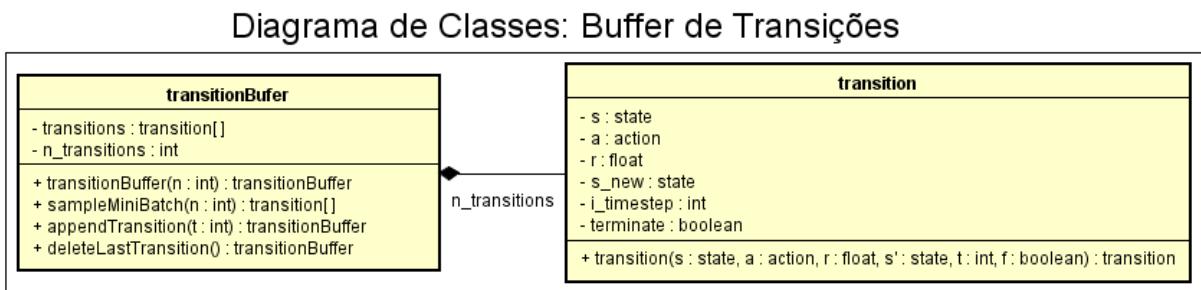


Figura 27 – Diagrama de classes da estrutura de armazenamento de transições (fonte: autores).

6.1.2.2 Armazenamento de Trajetórias

O armazenamento de trajetórias foi realizado por meio de estrutura de três classes: Cada trajetória $\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\}$ é armazenada em um objeto da classe *tau*, a classe *trajectories* contém um vetor de objetos da classe *tau* e a classe *epochBuffer* contém um vetor de épocas, onde cada época do treino é caracterizada por um conjunto de trajetórias a partir do qual as transições utilizadas para treino foram amostradas. A figura 28 mostra a relação entre as classes utilizadas para armazenamento de trajetórias.

Diagrama de Classes: Armazenamento de Trajetórias

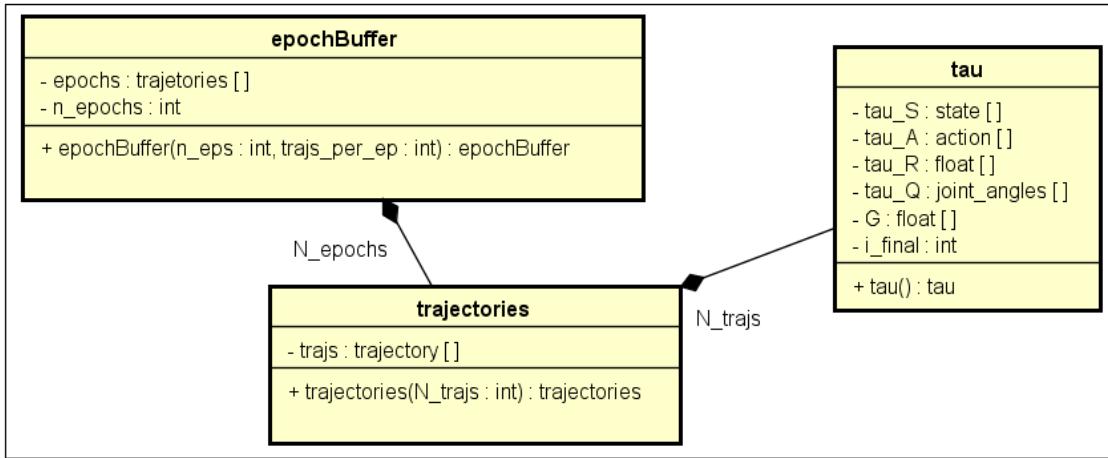


Figura 28 – Diagrama de classes da estrutura de armazenamento de épocas do treino (fonte: autores).

6.1.2.3 Armazenamento de Redes DQN

Por fim, a rede DQN que aproxima a função $Q(\mathbf{s}, \mathbf{a})$ foi armazenada ao longo do treino, por exemplo a cada 10 épocas, com o objetivo de comparar os valores a convergência dos valores Q de pares estado-ação. A figura 29 exemplifica a relação entre as classes *dqn_net* e *policyBuffer* que armazenam respectivamente um objeto nativo do Matlab com os parâmetros de uma rede densa e um vetor de objetos *dqn_net*.

Diagrama de Classes: Armazenamento de Redes DQN

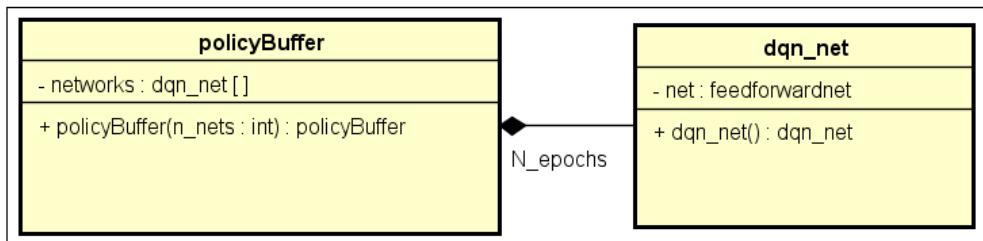


Figura 29 – Diagrama de classes da estrutura de armazenamento de redes DQN (fonte: autores).

6.1.3 Fluxo de Execução do Programa

Esta seção tem como objetivo detalhar o fluxo de execução do programa principal do projeto DQNKukaImage, especificando os pontos de chamada das sub-rotinas e funções auxiliares que implementam o algoritmo da seção 5.6.2. Sua execução pode ser sub-dividida em quatro seções principais:

- Inicialização de variáveis de ambiente do sistema, como posições do ponto de destino,

do obstáculo e dimensões da mesa, e de hiper-parâmetros do algoritmo, como dimensões da rede DQN, taxa de desconto γ , taxa de aprendizado α e tamanho dos conjuntos de transições amostrados utilizados para treino. No final desta seção é inicializada a rede DQN com pesos aleatórios conforme método de inicialização de Xavier.

- Criação de estruturas de armazenamento de transições, épocas e redes ao longo do treino, conforme seção 6.1.2 deste trabalho.
- Execução do Algoritmo DQN com chamadas alternadas e sucessivas de sub-rotinas de simulação, treino e visualização.
- Visualização final de resultados do treino por métricas de desempenho.

O diagrama de atividades da figura 30 descreve o fluxo de execução do programa principal, onde as regiões azul, laranja, cinza e verde representam respectivamente as seções de inicialização de variáveis, criação de estruturas de armazenamento, execução do algoritmo e visualização de resultados. As principais funções implementadas são documentadas no apêndice A deste trabalho.

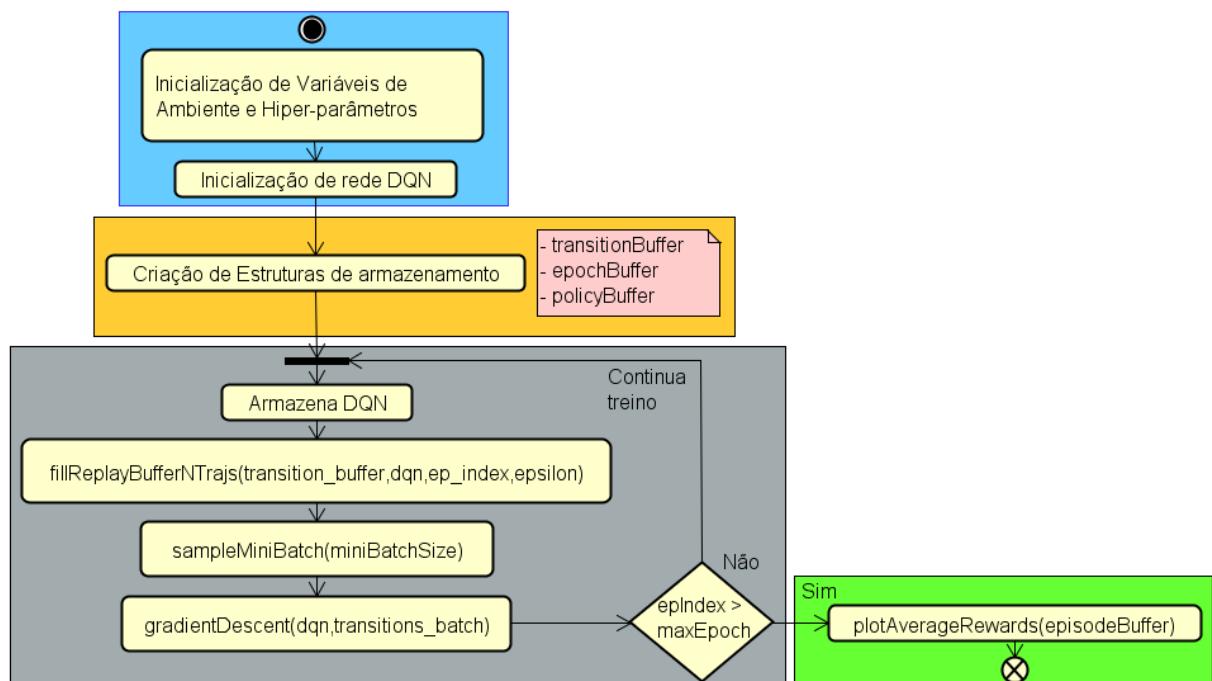


Figura 30 – Diagrama de atividades representando fluxo de execução do arquivo DQN-Kuka_image.m, com seções de inicialização (azul), criação de estruturas de armazenamento (laranja), execução de algoritmo DQN (cinza) e visualização de desempenho (verde) (fonte: autores).

6.1.4 Arquivo URDF

Softwares como ROS e Matlab importam os modelos matemáticos de robôs em termos de elos, articulações e suas interconexões a partir de arquivos independentes do código ou aplicação desenvolvida. A principal formatação utilizada atualmente para estes arquivos é denominada URDF, ou *Unified Robotics Description Format*, em que definem-se as posições e orientações dos sistemas de coordenadas solidários a cada elo do robô, assim como as massas, matrizes de inércia e malhas gráficas de cada elo. Além disso, neste arquivo especifica-se o tipo de movimento permitido por cada articulação, os elos de seu par cinemático, sua velocidade máxima e seus limites de curso. O arquivo kr16_2.urdf utilizado para importação do modelo cinemático do robô KUKA_KR16 encontra-se no Anexo B deste trabalho. O trecho do arquivo urdf responsável pela especificação das três primeiras articulações rotativas do robô é apresentado na figura 31.

```
<!-- JOINTS -->
<!-- joint 1 (A1) -->
<joint name="joint_a1" type="revolute">
  <origin rpy="0 0 0" xyz="0 0 0.675"/>
  <parent link="base_link"/>
  <child link="link_1"/>
  <axis xyz="0 0 -1"/>
  <limit effort="0" lower="-3.22885911619" upper="3.22885911619" velocity="2.72271363311"/>
</joint>
<!-- joint 2 (A2) -->
<joint name="joint_a2" type="revolute">
  <origin rpy="0 0 0" xyz="0.26 0 0"/>
  <parent link="link_1"/>
  <child link="link_2"/>
  <axis xyz="0 1 0"/>
  <limit effort="0" lower="-1.134464013796314" upper="2.181661564992912" velocity="2.72271363311"/>
</joint>
<!-- joint 3 (A3) -->
<joint name="joint_a3" type="revolute">
  <origin rpy="0 0 0" xyz="0 0 0.68"/>
  <parent link="link_2"/>
  <child link="link_3"/>
  <axis xyz="0 1 0"/>
  <limit effort="0" lower="-3.839724354387525" upper="1.117010721276371" velocity="2.72271363311"/>
</joint>
```

Figura 31 – Trecho de arquivo kr16_2.urdf responsável pela determinação das propriedades das três primeiras articulações rotativas do robô (fonte: autores).

6.2 VGG16

A rede convolucional pré-treinada VGG 16, proposta originalmente por Simonyan e Zisserman (2014), possui 138 milhões de parâmetros e apresenta arquitetura de camadas representada pelas figuras 32 e 33, com cinco conjuntos de camadas convolucionais separadas por camadas de *maxpooling* para redução de dimensionalidade, seguidas por três camadas densas com função de ativação *softmax* aplicada à última para classificação.

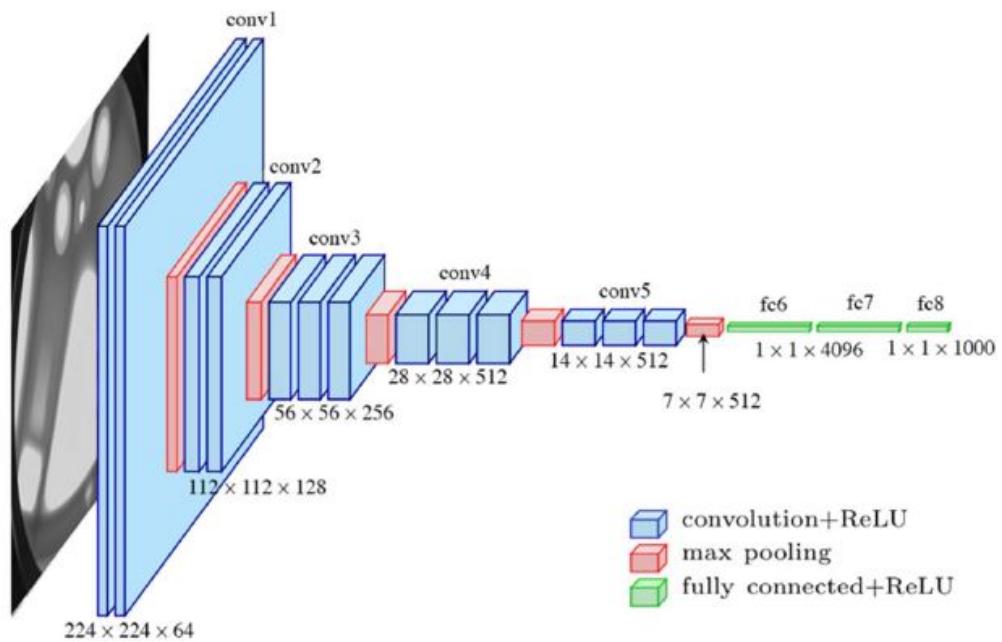


Figura 32 – Arquitetura parão da rede convolucional VGG16 (fonte: Ferguson (2017)).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 33 – Estrutura de camadas de redes da família ConvNet (VGG16 corresponde à coluna D) fonte: Simonyan e Zisserman (2014)

Parte V

Resultados

7 Projetos de Teste

Primeiramente, com o objetivo de verificar a convergência de diferentes algoritmos de Aprendizado por Reforço e analisar como esta evolui com o aumento da complexidade do sistema, foram realizadas simulações de variações simplificadas do problema proposto originalmente. O objetivo desta seção é detalhar os resultados parciais obtidos nos projetos de teste e explicitar o modo como eles contribuíram para a identificação de problemas e refinamento de parâmetros do projeto final.

7.1 Projeto 1: PGRobotArmR

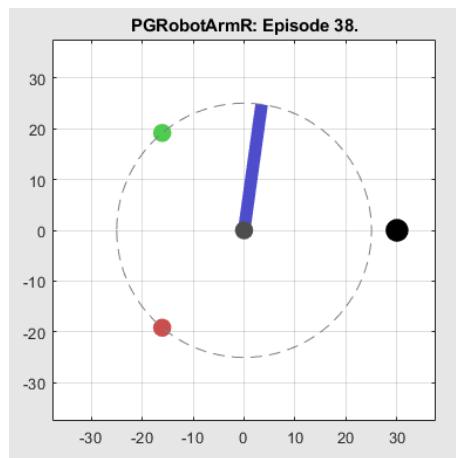


Figura 34 – Ambiente de simulação do projeto PGRobotArmR em estado de episódio 38 do treino. Posição desejada (verde), obstáculo (vermelho), braço robótico (azul), ação executada no instante de tempo (ponto preto) (fonte: autores).

7.1.1 Especificação do Sistema

O primeiro projeto de teste consiste na aplicação do algoritmo REINFORCE episódico para solução do problema de posicionamento de braço robótico plano rotativo em ambiente com obstáculo em posição conhecida. A figura 34 ilustra o ambiente de simulação implementado. Para permitir a visualização das ações tomadas pelo agente foi incluído um ponto preto no lado direito que se encontra sobre eixo horizontal para ação $a_t = 0$, acima dele para ação $a_t = 1$ e abaixo dele para ação $a_t = -1$.

O espaço de estados \mathcal{S} do problema foi definido como a seguinte discretização das posições angulares da articulação rotativa combinada com as possíveis posições cartesianas do *setpoint* e do obstáculo no plano:

$$\mathcal{S} = \left\{ -180 + i\Delta\theta \mid i = 1, \dots, \frac{360}{\Delta\theta} \right\} \times \mathcal{R}^2 \times \mathcal{R}^2 \quad (7.1)$$

O espaço de ações \mathcal{A} é dado pelo conjunto $\{-1, 0, 1\}$ correspondente a ações de mover o braço $\Delta\theta$ no sentido horário, mantê-lo parado e movê-lo $\Delta\theta$ no sentido anti-horário.

A política de ações $\pi_\theta(\mathbf{a}|\mathbf{s})$ foi representada por uma rede neural densa de apenas uma camada oculta de dimensão 10 e função de ativação ReLu, com dimensão da camada de entrada igual à dimensão dos estados (5) e dimensão da camada de saída igual à dimensão das ações (3), com um total de apenas 93 parâmetros treináveis, conforme figura 35.

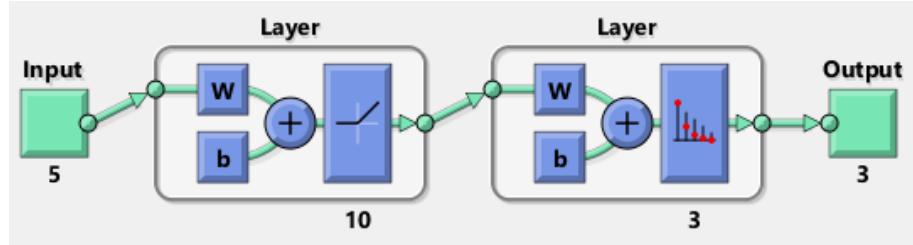


Figura 35 – Diagrama esquemático da estrutura da rede neural densa que implementa política de ações $\pi_\theta(\mathbf{a}|\mathbf{s})$ em projeto PGRobotArmR (fonte: autores).

Neste projeto foram implementadas as funções recompensa das seções 5.5.1 e 5.5.2, de distâncias absolutas e discreta sob aproximação ou distanciamento respectivamente r_1 e r_2 .

$$r_1(s, a) = k \left(r_{setpoint}(s, a) + r_{obstacle}(s, a) + c \right) + B_{goal}(s, a) + P_{joint boundary}(s, a) + P_{collision}(s, a) \quad (7.2)$$

$$\begin{cases} r_{setpoint} = -k_1 \left\| \mathbf{p}_{setpoint} - \mathbf{p}'_{ef} \right\| \\ r_{obstacle} = k_2 \left\| \mathbf{p}_{obstacle} - \mathbf{p}'_{ef} \right\| \end{cases}$$

$$r_2(s, a) = \left(k_s r_{setpoint}(s, a) + k_o r_{obstacle}(s, a) \right) + B_{goal}(s, a) + P_{joint boundary}(s, a) + P_{collision}(s, a) \quad (7.3)$$

$$r_{setpoint} = \begin{cases} -1, & \text{se } \left\| \mathbf{p}_{setpoint} - \mathbf{p}'_{ef} \right\| > \left\| \mathbf{p}_{setpoint} - \mathbf{p}_{ef} \right\| \\ 0, & \text{se } \left\| \mathbf{p}_{setpoint} - \mathbf{p}'_{ef} \right\| = \left\| \mathbf{p}_{setpoint} - \mathbf{p}_{ef} \right\| \\ 1, & \text{se } \left\| \mathbf{p}_{setpoint} - \mathbf{p}'_{ef} \right\| < \left\| \mathbf{p}_{setpoint} - \mathbf{p}_{ef} \right\| \end{cases}$$

$$r_{setpoint} = \begin{cases} 0, & \left\| \mathbf{p}_{obstacle} - \mathbf{p}'_{ef} \right\| > r_{infl} \\ \begin{cases} -1, & \text{se } \left\| \mathbf{p}_{obstacle} - \mathbf{p}'_{ef} \right\| < \left\| \mathbf{p}_{obstacle} - \mathbf{p}_{ef} \right\| \\ 0, & \text{se } \left\| \mathbf{p}_{obstacle} - \mathbf{p}'_{ef} \right\| = \left\| \mathbf{p}_{obstacle} - \mathbf{p}_{ef} \right\|, \text{ caso contrário} \\ 1, & \text{se } \left\| \mathbf{p}_{obstacle} - \mathbf{p}'_{ef} \right\| > \left\| \mathbf{p}_{obstacle} - \mathbf{p}_{ef} \right\| \end{cases} & \end{cases}$$

A tabela 3 especifica os valores das variáveis de ambiente e hiperparâmetros utilizados.

Parâmetro	Descrição	Valor
L	Comprimento do braço robótico	25
$\mathbf{p}_{\text{setpoint}}$	Posição de destino	$(L \cos 130^\circ, L \sin 130^\circ)$
$\mathbf{p}_{\text{obstacle}}$	Posição de obstáculo	$(L \cos 230^\circ, L \sin 230^\circ)$
$\Delta\theta$	Mínima variação angular	0.1°
α	Taxa de Aprendizado	0.01
B_{goal}	Bônus de destino	100
$P_{\text{collision}}$	Penalidade de colisão	-100
r_{infl}	Raio de influência do obstáculo	∞
$MaxEpoch$	Número máximo de épocas de treino	50
T	Número máximo de ações por trajetória	5000
γ	Fator de desconto	0.3
$dim(s)$	Dimensão do estado s	5
$dim(a)$	Dimensão da ação a	1
$(n_{\text{in}}, n_{h1}, n_{\text{out}})$	Número de neurônios da rede $\pi_\theta(\mathbf{a} \mathbf{s})$	(5, 10, 3)
k	Fator multiplicativo de r_1	100
c	Constante de função de recompensa r_1	0
k_s	Fator multiplicativo de r_2	100
k_o	Fator multiplicativo de r_2	100

Tabela 3 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto PGRobotArmR

7.1.2 Resultados

O agente foi treinado segundo o algoritmo REINFORCE ao longo de 50 episódios três vezes para cada função recompensa considerada, os gráficos de recompensas médias obtidas em função da época do treino, normalizados sob seus respectivos máximos para critério comparativo, são ilustrados na figura 36.

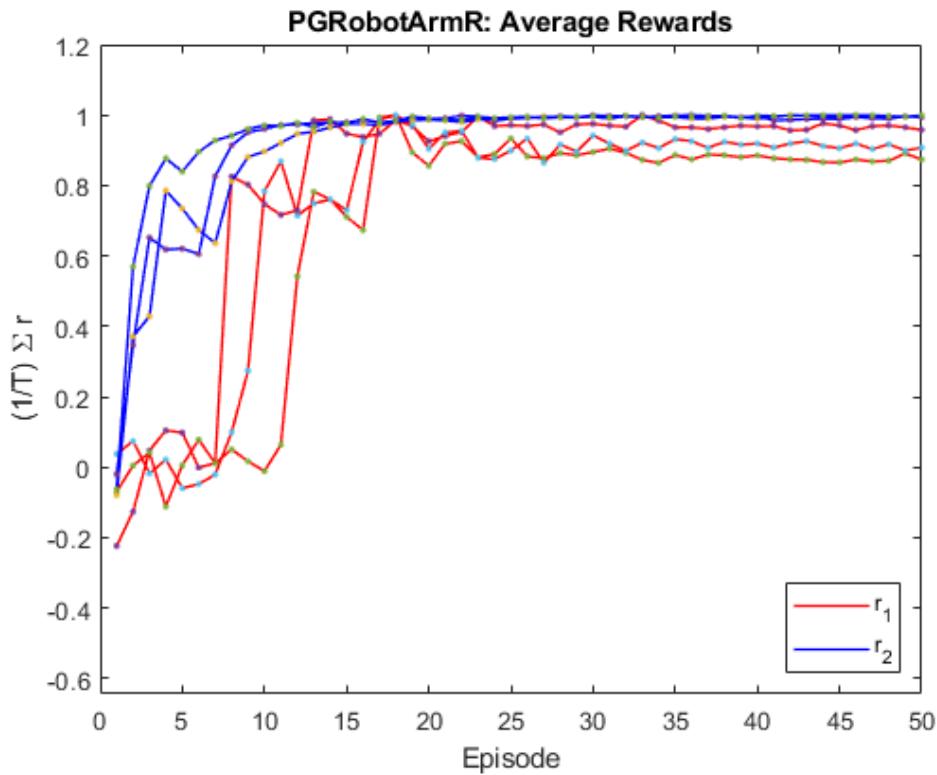


Figura 36 – Valores médios normalizados de recompensas obtidas pelo agente em função da época do treino para as funções recompensa r_1 e r_2 (fonte: autores).

7.2 Projeto 2: QLearningRobotArmR

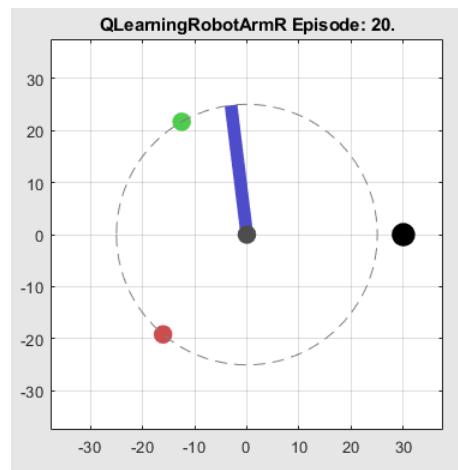


Figura 37 – Ambiente de simulação do projeto QLearningRobotArmR em estado de episódio 20 do treino. Posição desejada (verde), obstáculo (vermelho), braço robótico (azul), ação executada no instante de tempo (ponto preto) (fonte: autores).

7.2.1 Especificação do Sistema

O segundo projeto de teste consiste na aplicação do algoritmo Q-Learning clássico para solução do problema de posicionamento de braço robótico plano rotativo em ambiente com obstáculo em posição conhecida. A figura 37 ilustra o ambiente de simulação implementado, igual ao ambiente do projeto anterior.

Os espaços de estados \mathcal{S} e de ações \mathcal{A} foram definidos de modo idêntico ao projeto 1, de modo a se isolar as diferenças entre ambos e possibilitar comparação apenas dos algoritmos implementados.

A função valor do par Estado-Ação $Q_\theta(\mathbf{s}, \mathbf{a})$ foi representada por uma matriz com número de linhas igual ao número total de estados do sistema e número de colunas igual ao número de ações, $Q_\theta(\mathbf{s}, \mathbf{a}) \in \mathcal{M}^{\dim(\mathcal{S}) \times \dim(\mathcal{A})}$, conforme algoritmo de Q-Learning tradicional (WATKINS, 1989).

As funções recompensa implementadas foram as mesmas do projeto 1 (equações 7.2 e 7.3) e os hiperparâmetros utilizados são descritos na tabela 4.

Parâmetro	Descrição	Valor
L	Comprimento do braço robótico	25
psetpoint	Posição de destino	$(L \cos 130^\circ, L \sin 130^\circ)$
pobstacle	Posição de obstáculo	$(L \cos 230^\circ, L \sin 230^\circ)$
$\Delta\theta$	Mínima variação angular	0.1°
α	Taxa de Aprendizado (de 0 a 1)	0.99
B_{goal}	Bônus de destino	8000
$P_{collision}$	Penalidade de colisão	-8000
r_{infl}	Raio de influência do obstáculo	∞
MaxEpoch	Número máximo de épocas de treino	50
T	Número máximo de ações por trajetória	5000
γ	Fator de desconto	0.3
$dim(s)$	Dimensão da cada estado s	5
$dim(a)$	Dimensão de cada ação a	1
$size(\mathcal{S})$	Tamanho de Espaço de Estados \mathcal{S}	3600
$size(\mathcal{A})$	Tamanho de Espaço de Ações \mathcal{A}	3
$dim(Q)$	Dimensões de matriz $Q(s, a)$	3600 x 3
k	Fator multiplicativo de r_1	20
c	Constante de função de recompensa r_1	0
k_s	Fator multiplicativo de r_2	1
k_o	Fator multiplicativo de r_2	0.7

Tabela 4 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto QLearnin-gRobotArmR

7.2.2 Resultados

O agente foi treinado por meio de atualizações sucessivas da matriz Q ao longo de 50 episódios de até 5000 transições de estado cada, a evolução do valor médio das recompensas obtidas a cada época é ilustrada na figura 38 para as funções de recompensa r_1 e r_2 .

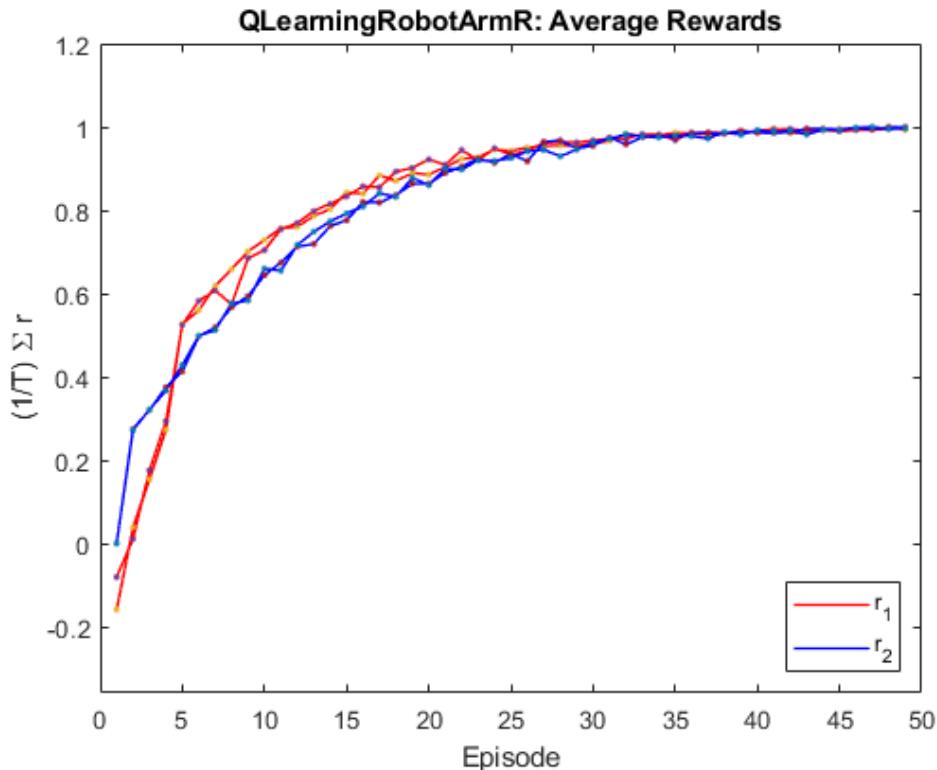


Figura 38 – Valores médios normalizados de recompensas obtidas pelo agente em função da época do treino para as funções recompensa r_1 e r_2 (fonte: autores).

O algoritmo implementado apresentou robustez sob treino em configurações inesperadas de posição desejada e de obstáculo, como em casos de *setpoint* exterior ao espaço de trabalho do robô e fora de alcance, conforme figura 39:

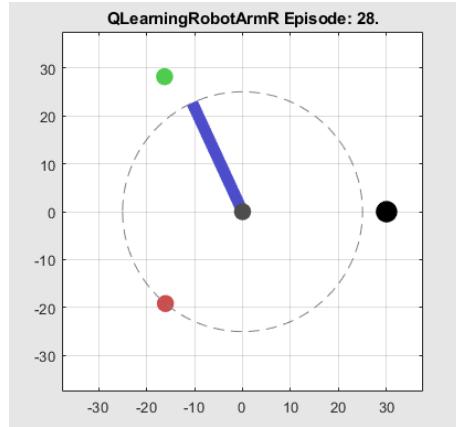


Figura 39 – Agente após treino em configuração com posição de destino fora de alcance (fonte: autores).

7.3 Projeto 3: PGRobotArmRR

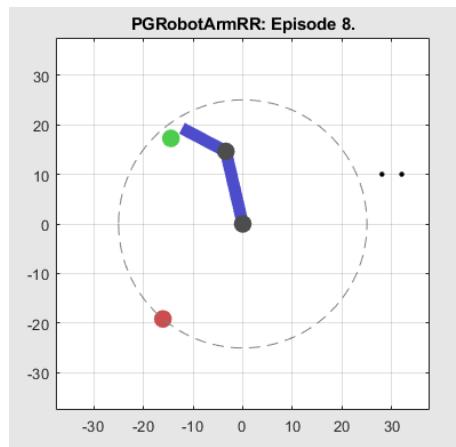


Figura 40 – Ambiente de simulação do projeto PGRobotArmRR em estado de episódio 8 do treino. Posição desejada (verde), obstáculo (vermelho), braço robótico (azul), ação executada no instante de tempo (ponto preto) (fonte: autores).

7.3.1 Especificação do Sistema

Neste projeto foi aumentado o número de graus de liberdade do robô para verificar o tempo de treino e desempenho obtido pelo REINFORCE episódico em sistemas com espaços de estados \mathcal{S} e de ações \mathcal{A} de maiores dimensões.

O novo espaço de estados do sistema é dado pelo produto cartesiano entre os conjuntos de ângulos admissíveis para cada articulação, conforme equação:

$$\mathcal{S} = \left\{ -180 + i\Delta\theta \mid i = 1, \dots, \frac{360}{\Delta\theta} \right\} \times \left\{ -180 + j\Delta\theta \mid j = 1, \dots, \frac{360}{\Delta\theta} \right\} \times \mathcal{R}^2 \times \mathcal{R}^2 \quad (7.4)$$

Analogamente, o espaço de ações \mathcal{A} é dado pelas combinações de movimento

rotativo em sentido positivo, negativo e ausência de rotação para ambos motores:

$$\mathcal{A} = \{(1, 1), (1, 0), (1, -1), (0, 1), (0, 0), (0, -1), (-1, 1), (-1, 0), (-1, -1)\} \quad (7.5)$$

A política de ações $\pi_\theta(\mathbf{a}|\mathbf{s})$ teve sua estrutura ajustada para adequação à maior dimensionalidade de estados e de ações. Um diagrama esquemático da arquitetura da rede implementada é ilustrado na figura 41.

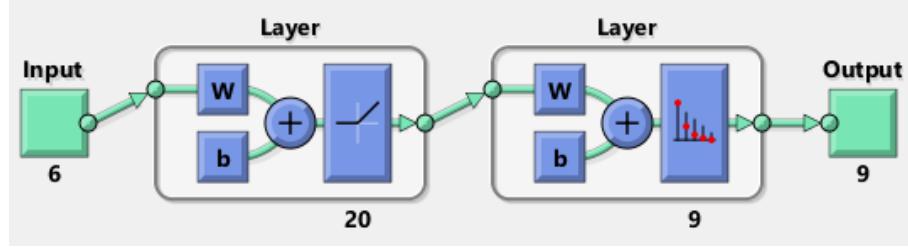


Figura 41 – Diagrama esquemático da estrutura da rede neural densa que implementa política de ações $\pi_\theta(\mathbf{a}|\mathbf{s})$ em projeto PGRobotArmRR (fonte: autores).

A partir dos resultados dos projetos anteriores optou-se pela utilização das funções de recompensa $r_2(\mathbf{s}, \mathbf{a})$, sensível apenas à aproximação ou distanciamento da posição de destino, e $r_3(\mathbf{s}, \mathbf{a})$, dependente do ângulo de aproximação, conforme equação 7.6. A tabela 5 ilustra os valores adotados para as variáveis de ambiente e hiper-parâmetros de treino.

$$r_3(\mathbf{s}, \mathbf{a}) = (k_s r_{setpoint}(\mathbf{s}, \mathbf{a}) + k_o r_{obstacle}(\mathbf{s}, \mathbf{a})) + B_{goal}(\mathbf{s}, \mathbf{a}) + P_{joint\ boundary}(\mathbf{s}, \mathbf{a}) + P_{collision}(\mathbf{s}, \mathbf{a}) \quad (7.6)$$

onde:

$$\begin{cases} r_{setpoint}(\mathbf{s}, \mathbf{a}) = k_1 (\mathbf{n}_{ef->ef'} \bullet \mathbf{n}_{ef->setpoint}) \\ r_{obstacle}(\mathbf{s}, \mathbf{a}) = \begin{cases} 0, & se \|\mathbf{p}_{obstacle} - \mathbf{p}'_{ef}\| > r_{infl} \\ k_2 (\mathbf{n}_{ef->ef'} \bullet \mathbf{n}_{ef->obstacle}), & caso contrário \end{cases} \end{cases}$$

Parâmetro	Descrição	Valor
L_1	Comprimento do primeiro elo	15
L_2	Comprimento do segundo elo	10
$\mathbf{P}_{\text{setpoint}}$	Posição de destino	(-14.46, 17.24)
$\mathbf{P}_{\text{obstacle}}$	Posição de obstáculo	(-16.07, -19.15)
$\Delta\theta_1$	Mínima variação angular	0.1°
$\Delta\theta_2$	Mínima variação angular	0.1°
α	Taxa de Aprendizado	0.001
B_{goal}	Bônus de destino	600
$P_{\text{collision}}$	Penalidade de colisão	-600
r_{infl}	Raio de influência do obstáculo	∞
$MaxEpoch$	Número máximo de épocas de treino	50
T	Número máximo de ações por trajetória	5000
γ	Fator de desconto	0.3
$dim(s)$	Dimensão do estado \mathbf{s}	6
$dim(a)$	Dimensão da ação \mathbf{a}	2
$size(\mathcal{A})$	Tamanho do espaço de ações \mathcal{A}	9
$(n_{in}, n_{h1}, n_{out})$	Números de neurônios da rede $\pi_\theta(\mathbf{a} \mathbf{s})$	(6, 20, 9)
k	Fator multiplicativo de r_1	100
c	Constante de função de recompensa r_1	0
k_s	Fator multiplicativo de r_2	100
k_o	Fator multiplicativo de r_2	70

Tabela 5 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto PGRobotArmRR

7.3.2 Resultados

Foram treinados agentes sob as funções de recompensa r_2 e r_3 , descritas nas seções 5.5.2 e 5.5.3 deste trabalho. A figura 42 ilustra os valores médios das recompensas obtidas pelos agentes em função da época de treino, normalizados para análise comparativa. Apesar de ambos agentes convergirem para uma trajetória bem sucedida de posicionamento, o treino sob função de recompensa r_2 apresentou instabilidade, com queda de desempenho temporária, enquanto o treino sob a terceira função recompensa apresentou maior estabilidade e menor tempo de convergência.

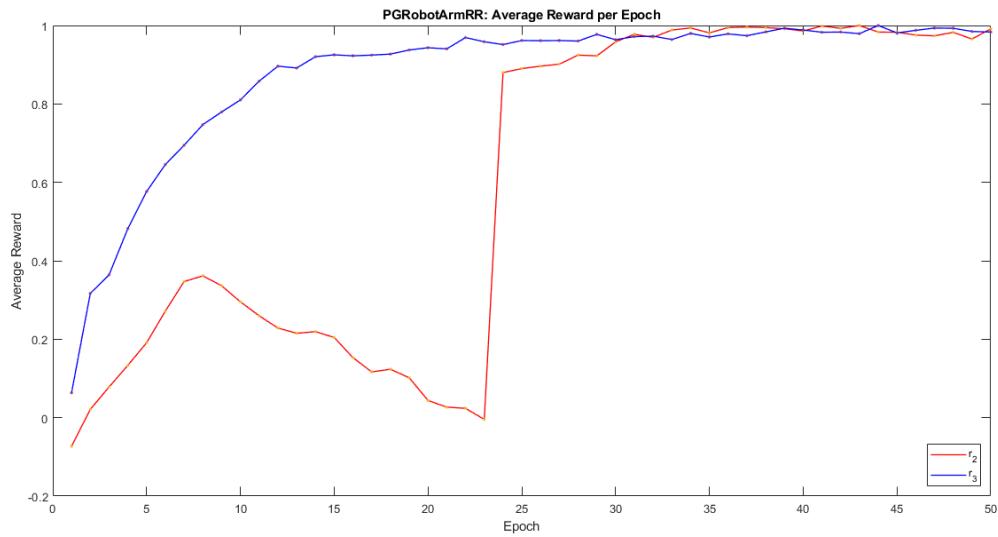


Figura 42 – Valores médios normalizados de recompensas obtidas pelo agente em função da época do treino para as funções recompensa r_2 (vermelho) e r_3 (azul) (fonte: autores).

7.4 Projeto 4: QLearningRobotArmRR

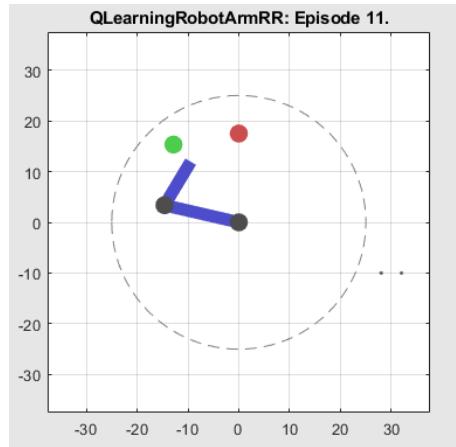


Figura 43 – Ambiente de simulação do projeto QLearningRobotArmRR em estado de episódio 2 do treino. Posição desejada (verde), obstáculo (vermelho), braço robótico (azul), ação executada no instante de tempo (pontos pretos) (fonte: autores).

7.4.1 Especificação do Sistema

Por fim, o projeto QLearningRobotArmRR fez uso do mesmo ambiente que o projeto 3, com exceção da discretização do espaço de estados, que teve que ser alterada para $\Delta\theta_1 = \Delta\theta_2 = 1^\circ$, uma vez que a discretização anterior implicaria em um espaço de estados de dimensão $\dim(\mathcal{S}) = 12960000$. As funções recompensa implementadas foram novamente r_2 e r_3 , a tabela 6 resume os parâmetros utilizados no treino.

Parâmetro	Descrição	Valor
L_1	Comprimento do primeiro elo	15
L_2	Comprimento do segundo elo	10
Psetpoint	Posição de destino	(-12.86, 15.32)
Pobstacle	Posição de obstáculo	(0.00, 17.50)
$\Delta\theta_1$	Mínima variação angular	1°
$\Delta\theta_2$	Mínima variação angular	1°
α	Taxa de Aprendizado (0-1)	0.99
B_{goal}	Bônus de destino	8000
$P_{collision}$	Penalidade de colisão	-8000
r_{infl}	Raio de influência do obstáculo	∞
$MaxEpoch$	Número máximo de épocas de treino	100
T	Número máximo de ações por trajetória	5000
γ	Fator de desconto	0.3
$dim(s)$	Dimensão de cada estado s	6
$dim(a)$	Dimensão de cada ação a	2
$size(\mathcal{S})$	Tamanho do espaço de estados \mathcal{S}	129600
$size(\mathcal{A})$	Tamanho do espaço de ações \mathcal{A}	9
$dim(Q)$	Dimensões da matriz Q	129600 x 9
k_s	Fator multiplicativo de r_2	100
k_o	Fator multiplicativo de r_2	70

Tabela 6 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto QLearnin-gRobotArmRR

7.4.2 Resultados

Foram realizados três treinos completos do agente para as duas funções recompensa, os valores médios das recompensas obtidas ao longo das 50 épocas de treino foram armazenados e normalizados segundo seus respectivos máximos após convergência. A figura 44 ilustra a evolução de desempenho dos agentes sob as funções de recompensa r_2 e r_3 .

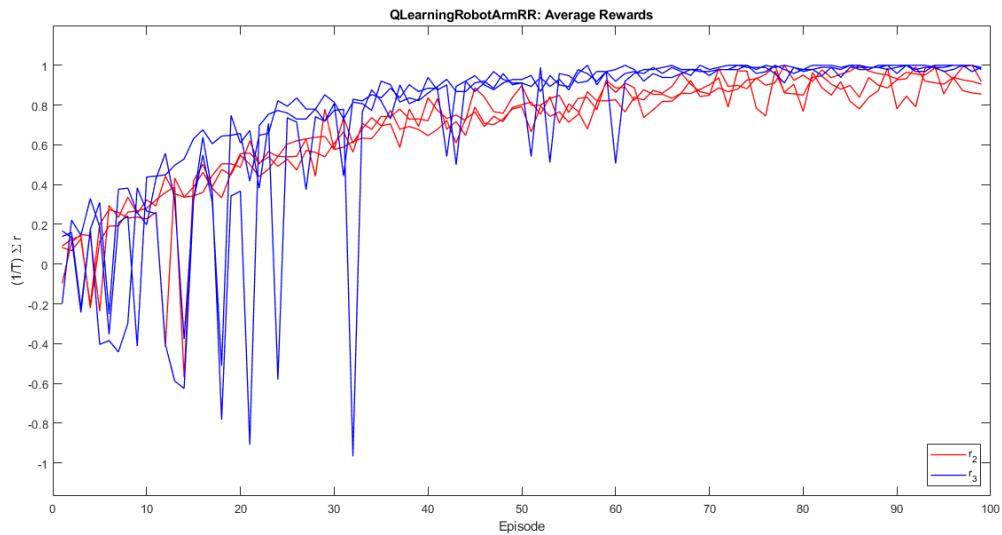


Figura 44 – Valores médios normalizados de recompensas obtidas pelo agente em função da época do treino para as funções recompensa r_2 e r_3 ao longo de três treinos. Vales correspondem a colisões com obstáculo (fonte: autores).

Observou-se que o agente treinado pela função de recompensa r_2 , discreta sob aproximação ou distanciamento, apresentou desempenho inferior àquele treinado pela r_3 , dependente da projeção do vetor deslocamento sobre vetor que aponta para posição desejada. No entanto, em configurações nas quais há obstáculos diretamente no caminho entre posição inicial do efetuador e posição desejada, a função de recompensa r_3 apresenta maior número de trajetórias com estados terminais de colisão com obstáculo, enquanto a maior aleatoriedade da exploração promovida pela função recompensa r_2 faz com que o agente leve maior tempo para treino, mas apresente menos colisões com obstáculo.

Devido à não implementação de colisão com a extensão do braço robótico neste projeto, ao longo do treino foi possível observar preferência do agente para duas trajetórias distintas, ilustradas na figura 45.

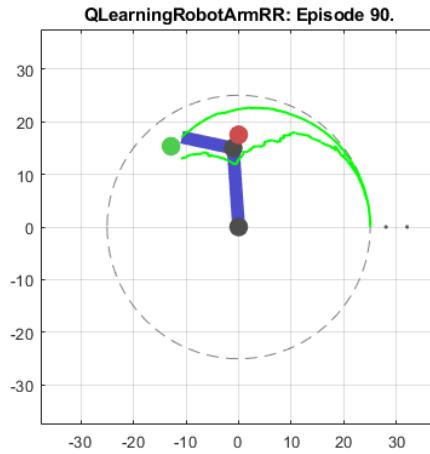


Figura 45 – Trajetórias encontradas pelo agente ao longo do treino. Algoritmo covergiu para matriz Q que executa trajetória superior, uma vez que verifica-se apenas colisão com efetuador do robô (fonte: autores).

7.5 Projeto 5: PGKuka

Após os testes em robôs planos com número de graus de liberdade reduzido e com base nas análises comparativas de desempenho de treino sob diferentes funções recompensas, foi implementado o algoritmo REINFORCE episódico de iteração sobre política de ações no controle de posicionamento do robô KUKA-KR16. A figura 46 ilustra o ambiente de simulação do projeto e a trajetória obtida pelo agente após treino.

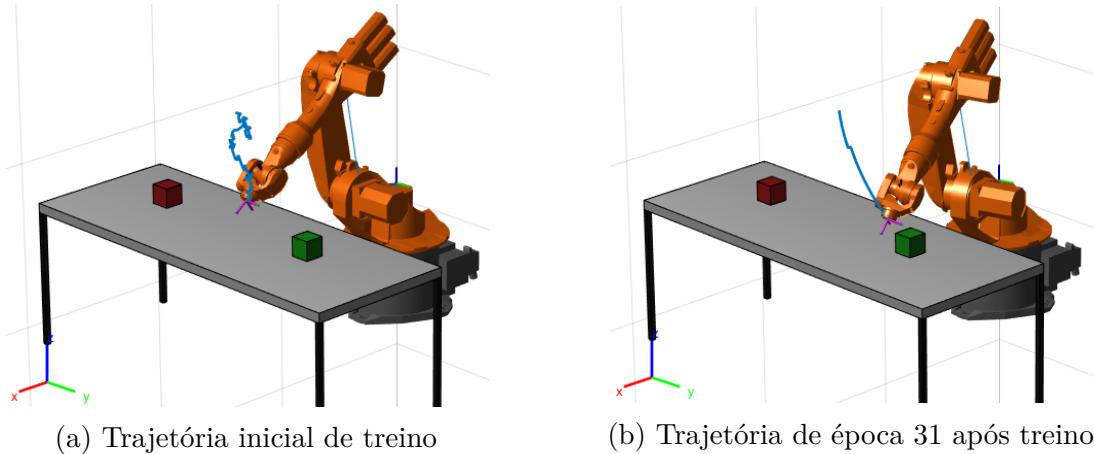


Figura 46 – Ambiente de simulação gráfica do projeto PGKuka. Comparação entre trajetória gerada por política de ações inicial (a) e trajetória após treino do agente (b) (fonte: autores)

7.5.1 Especificação do Sistema

Neste projeto foi implementado o modelo cinemático do robô KUKA-KR16, em oposição aos robôs de um e dois graus de liberdade dos projetos anteriores. Assim, os

espaços de estados \mathcal{S} e de ações \mathcal{A} foram completamente modificados. O espaço de estados agora é função dos limites angulares de cada articulação do robô, dados pela tabela 2 na seção 5.4.1.4, e da mínima variação angular adotada ($\Delta\theta = 1^\circ$). Assim, temos:

$$\mathcal{S} = \left(\prod_{i=1}^5 \mathcal{S}_i \right) \times \mathcal{R}^3 \times \mathcal{R}^3, \text{ onde } \mathcal{S}_i = \{\theta_{i_{inf}} + i\Delta\theta | i = 1, \dots, \frac{\theta_{i_{sup}}}{\Delta\theta}\} \quad (7.7)$$

onde cada \mathcal{S}_i denota o conjunto de ângulos que a i -ésima articulação do robô pode assumir e as posições do *setpoint* e obstáculo pertencem a subconjuntos de \mathcal{R}^3 .

Analogamente, o espaço de ações deve levar em consideração a generalização para maior número de graus de liberdade da cadeia cinemática do robô, representando agora o conjunto de todas combinações possíveis de movimento em sentido horário, anti-horário e permanência na mesma posição para cada uma das cinco articulações controláveis do robô:

$$\mathcal{A} = \prod_{i=1}^5 \mathcal{A}_i, \text{ onde } \mathcal{A}_i = \{-1, 0, 1\} \quad (7.8)$$

Por fim, a implementação do algoritmo REINFORCE episódico sobre este sistema exigiu compatibilização da estrutura da rede neural densa da política de ações, que foi aumentada tanto para adequação de dimensões das camadas de entrada e saída quanto para abstração de comportamentos mais complexos. A estrutura da rede $\pi(\mathbf{a}|\mathbf{s})$ é ilustrada na figura 47.

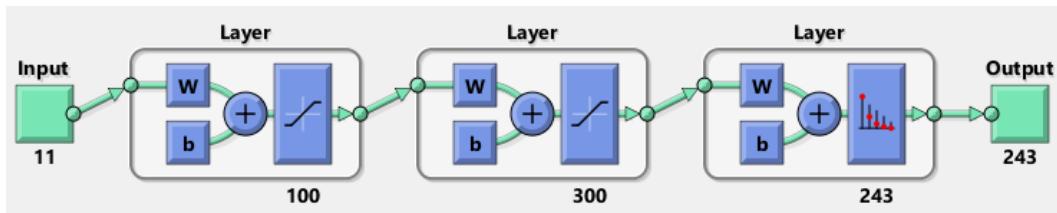


Figura 47 – Diagrama esquemático da estrutura da rede neural densa que implementa política de ações $\pi_\theta(\mathbf{a}|\mathbf{s})$ em projeto PGKuka (fonte: autores).

A tabela 7 resume os valores das principais variáveis de ambiente e hiper-parâmetros adotados no treino.

Parâmetro	Descrição	Valor
$p_{setpoint}$	Posição de destino (m)	(1.05, 0.45, 0.75)
$p_{obstacle}$	Posição de obstáculo (m)	(1.05, -0.55, 0.75)
$table_{length}$	Comprimento da mesa (m)	2
$table_{width}$	Largura da mesa (m)	0.8
$table_{height}$	Altura da mesa (m)	0.7
$\Delta\theta$	Mínima variação angular	1°
α	Taxa de Aprendizado	0.0005
B_{goal}	Bônus de destino	400
$P_{collision}$	Penalidade de colisão	-400
$P_{joint boundary}$	Penalidade de fim de curso	-400
r_{infl}	Raio de influência do obstáculo (m)	0.50
$MaxEpoch$	Número máximo de épocas de treino	50
N	Número de trajetórias por época	30
T	Número máximo de ações por trajetória	70
γ	Fator de desconto	0.3
$dim(s)$	Dimensão do vetor de estados s	11
$dim(a)$	Dimensão do vetor de ações a	5
$size(\mathcal{A})$	Tamanho do espaço de ações \mathcal{A}	243
$(n_{in}, n_{h1}, n_{h2}, n_{out})$	Número de neurônios da rede $\pi_\theta(a s)$	(11, 100, 300, 243)
k_s	Fator multiplicativo de r_3	100
k_o	Fator multiplicativo de r_3	70

Tabela 7 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto PGKuka

7.5.2 Resultados

A partir dos resultados dos projetos anteriores, verificou-se que agentes treinados pela função de recompensa r_3 apresentavam maior desempenho em mesmo número de épocas de treino. Assim, optou-se pela implementação apenas da função recompensa r_3 neste projeto.

Em preparação para os projetos seguintes, nos quais a hipótese de configuração fixa para posição de destino e do obstáculo é eliminada, optou-se pelo seguinte fluxo de treino:

- Em primeiro lugar a política de ações $\pi_\theta(a|s)$ foi inicializada aleatoriamente e o agente foi treinado de acordo com os parâmetros da tabela 7.
- Após verificação de convergência do aprendizado, permutou-se as posições de destino e do obstáculo e o treino foi repetido fazendo uso da política pós-treino anterior como política inicial.

A figura 48 ilustra trajetórias obtidas pelo agente após treino inicial e após treino com posições de destino e de obstáculo permutadas.

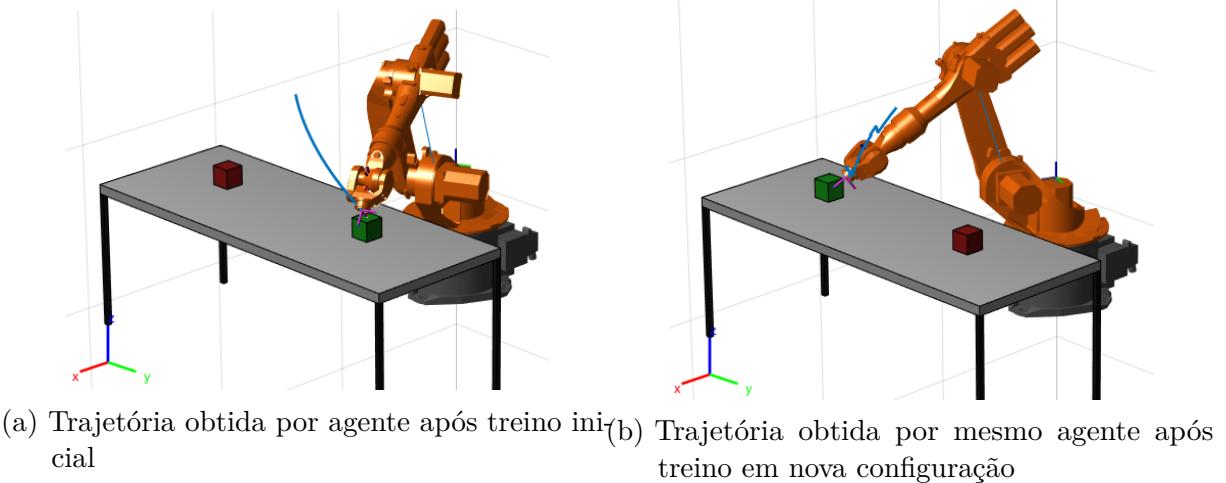


Figura 48 – Trajetórias obtidas pelo mesmo agente após treino inicial (a) e após treino de mesma política de ações sob nova configuração de posição de destino e de obstáculo (fonte: autores)

Os principais objetivos deste teste foram analisar a convergência do algoritmo a partir de política inicial de baixa performance dada a nova configuração e verificar a capacidade de abstração de comportamento adequado para ambas trajetórias simultaneamente após segundo treino.

A figura 49 mostra as curvas das recompensas médias obtidas por época ao longo do treino inicial (vermelho) e do segundo treino após permutação da posição de destino com a posição do obstáculo (azul).

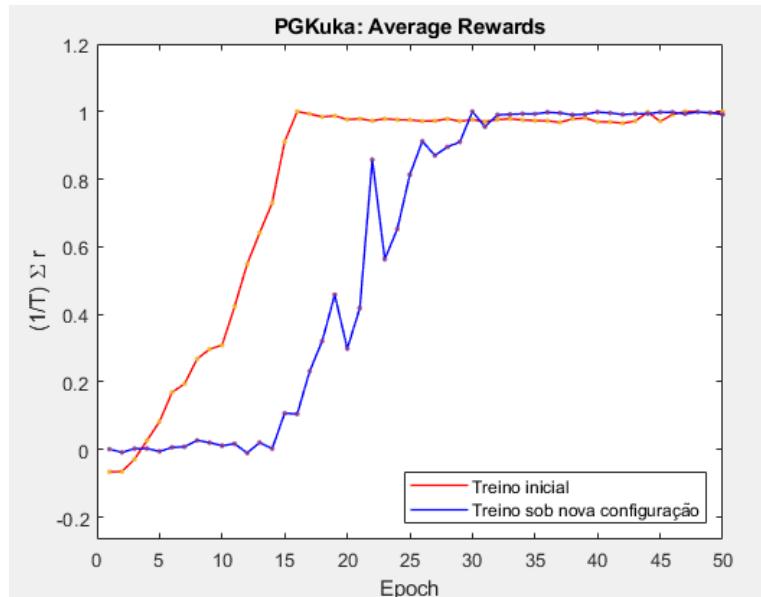


Figura 49 – Curvas de recompensas médias normalizadas em função de época para treino inicial (vermelho) e para treino após substituição de posições do *setpoint* e do *obstáculo* (azul) (fonte: autores).

Conforme esperado, a convergência do segundo treino foi mais lenta devido à inicialização com política abaixo da média em oposição à inicialização com política aleatória. O treino inicial foi mais estável, com recompensas médias estritamente crescentes ao longo das primeiras 15 épocas de treino. O algoritmo REINFORCE episódico implementado faz uso de uma política de ações estocástica, de modo que, dado um estado s a função $\pi_\theta(a|s)$ fornece a distribuição de probabilidades do agente tomar cada ação de $a \in \mathcal{A}$. A figura 50 ilustra a evolução das probabilidades sobre \mathcal{A} no estado inicial conforme a política de ações $\pi_\theta(a|s)$ é treinada.

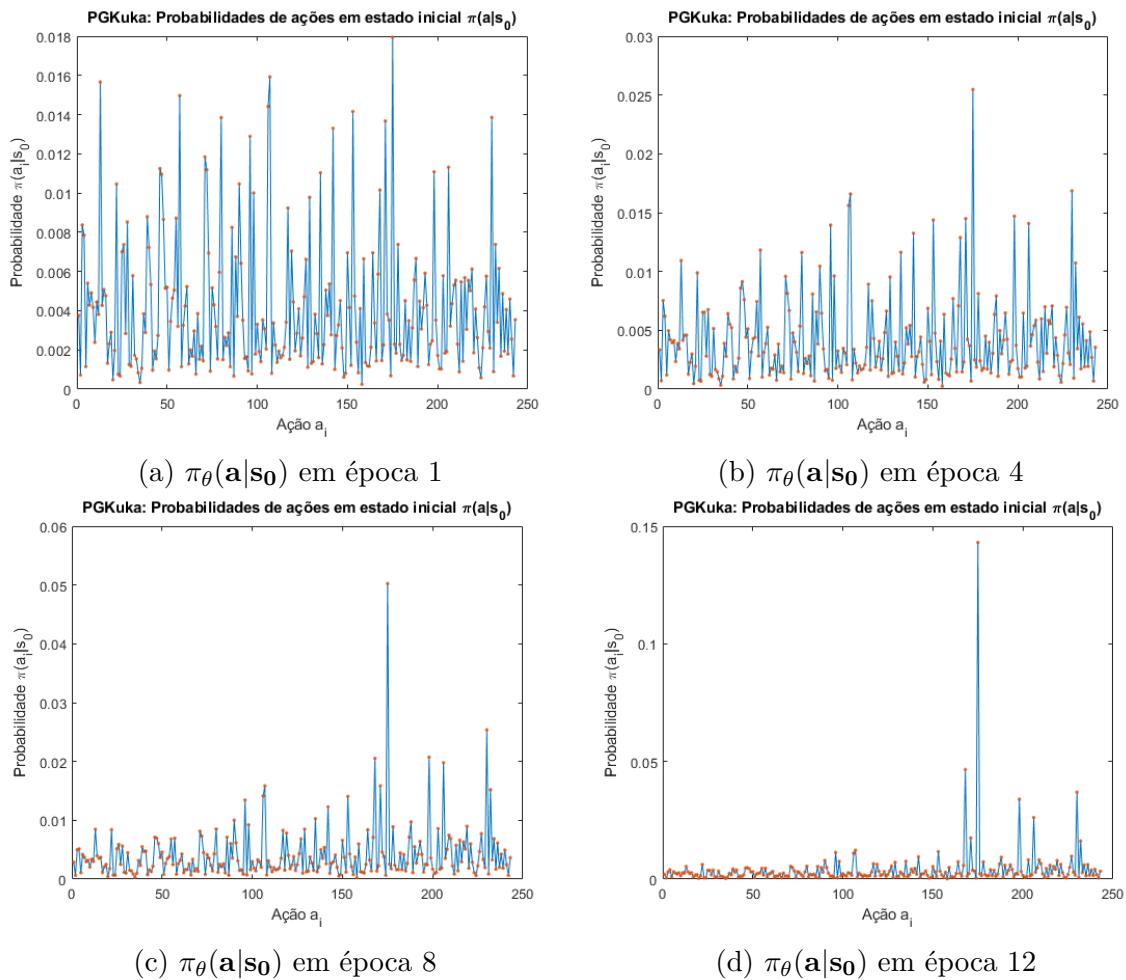


Figura 50 – Distribuição de probabilidades $\pi_\theta(a|s_0)$ sobre espaço de ações em estado inicial ao longo do treino em épocas 1, 4, 8 e 12 (fonte: autores)

Verificou-se que o algoritmo REINFORCE apresentou performance convergente após treino sob inicialização com política sub-ótima para nova configuração. No entanto, ao testar o agente na configuração de *setpoint* e obstáculo do primeiro treino, observou-se colisão com a mesa e perda do aprendizado inicial. Concluiu-se que o treino para configurações genéricas deve ser realizado sob re-arranjo aleatório e frequente de posição de destino e de obstáculo para evitar o mesmo problema.

7.6 Projeto 6: DQNKuka

Para este projeto foi mantido o modelo do robô, do espaço de estados e do espaço de ações, substituindo apenas o agente por um treinado segundo o algoritmo DQN, conforme seção 5.6.2. A figura 51 ilustra as trajetórias consideradas ótimas pelo agente na primeira e na décima nona épocas, ou seja, as trajetórias obtidas seguindo uma política determinística a partir de $Q_\theta(\mathbf{s}, \mathbf{a})$ em oposição a política ϵ -greedy.

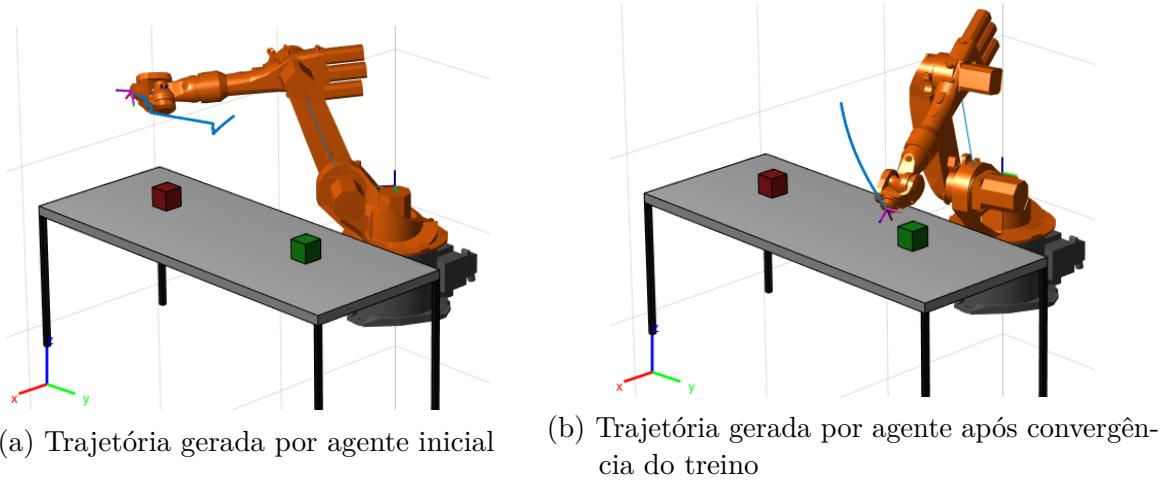


Figura 51 – Trajetórias *greedy*, ou seja, sem escolha de ações aleatórias para exploração, geradas por agente no início do treino e em época 19.(fonte: autores)

7.6.1 Especificação do Sistema

A função recompensa implementada neste projeto manteve-se a função $r_3(\mathbf{s}, \mathbf{a})$ de modo a permitir comparação exclusiva entre os algoritmos implementados. A rede neural densa que aproxima a função valor do para Estado-Ação $Q(\mathbf{s}, \mathbf{a})$ teve sua estrutura mantida, com exceção da função de ativação da última camada, que foi substituída de função *softmax*, necessária para que saída represente distribuição de probabilidades sobre espaço de ações, para função linear (figura 52).

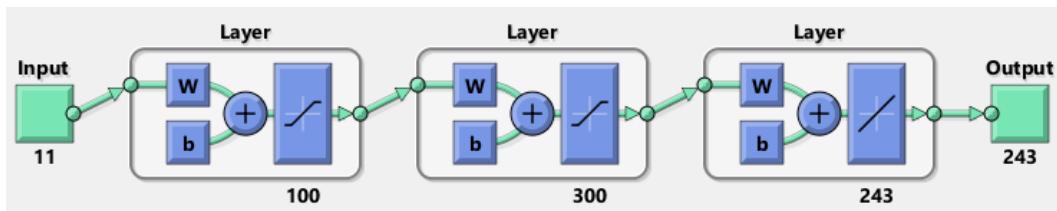


Figura 52 – Diagrama esquemático da estrutura da rede neural densa que implementa função valor de estados e ações $Q_\theta(\mathbf{s}, \mathbf{a})$ em projeto DQNKuka (fonte: autores).

A tabela 8 apresenta os valores das variáveis de ambiente e hiper-parâmetros utilizados no projeto PGKuka.

Parâmetro	Descrição	Valor
$\mathbf{p}_{\text{setpoint}}$	Posição de destino (m)	(1.05, 0.45, 0.75)
$\mathbf{p}_{\text{obstacle}}$	Posição de obstáculo (m)	(1.05, -0.55, 0.75)
$table_{length}$	Comprimento da mesa (m)	2
$table_{width}$	Largura da mesa (m)	0.8
$table_{height}$	Altura da mesa (m)	0.7
$\Delta\theta$	Mínima variação angular	1°
α	Taxa de Aprendizado	0.002
B_{goal}	Bônus de destino	20
$P_{\text{collision}}$	Penalidade de colisão	-20
$P_{\text{joint boundary}}$	Penalidade de fim de curso	-10
r_{infl}	Raio de influência do obstáculo (m)	0.40
$MaxEpoch$	Número máximo de épocas de treino	250
$Ntrajs$	Número de trajetórias por época	10
$MiniBatchSize$	Número de transições amostradas para treino	200
T	Número máximo de transições por trajetória	70
γ	Fator de desconto	0.3
$dim(\mathbf{s})$	Dimensão do vetor de estados \mathbf{s}	11
$dim(\mathbf{a})$	Dimensão do vetor de ações \mathbf{a}	5
$size(\mathcal{A})$	Tamanho do espaço de ações	243
$(n_{in}, n_{h1}, n_{h2}, n_{out})$	Número de neurônio das camadas da rede $\pi_\theta(\mathbf{a} \mathbf{s})$	(11, 100, 300, 243)
k_s	Fator multiplicativo de r_3	1
k_o	Fator multiplicativo de r_3	0.9

Tabela 8 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto DQNKuka

7.6.2 Resultados

O agente foi submetido a 250 épocas de treino para atualização da rede DQN, a figura 53 ilustra a trajetória obtida após convergência e a curva recompensas médias obtidas ao longo do treino.

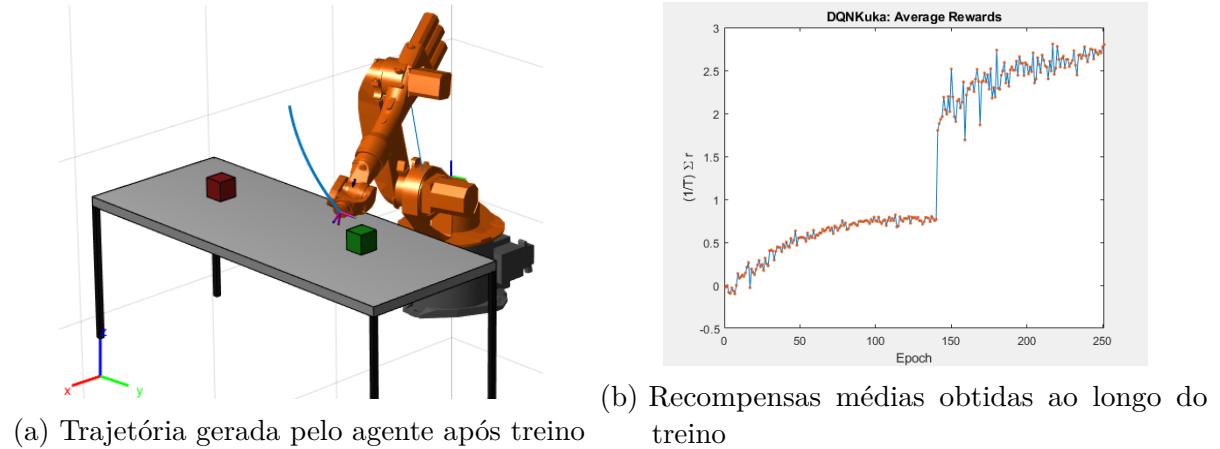


Figura 53 – Trajetória gerada por agente DQN apóas treino (a) e gráfico de recompensas médias em função de épocas (b) para projeto DQNKuka (fonte: autores)

De modo análogo ao procedimento realizado para visualização da distribuição de probabilidades sobre o espaço de ações no estado inicial $\pi(\mathbf{a}|\mathbf{s}_0)$, foram obtidos os valores $Q_\theta(\mathbf{s}_0, \mathbf{a})$ para todas as ações no estado inicial (figura 54). Dada a ordenação adotada para a enumeração de ações no espaço \mathcal{A} , descrita na seção 5.3 deste trabalho, percebe-se valores maiores de $Q_\theta(\mathbf{s}_0, \mathbf{a}_i)$ para ações \mathbf{a}_i correspondentes ao último terço do eixo das abcissas (reta tracejada verde), cujo acionamento do motor da articulação rotativa A1 do robô é no sentido de aproximação do efetuador à posição desejada. Analogamente, em cada terço do espaço de ações, observa-se que aquelas que correspondem ao acionamento da articulação rotativa A2 no sentido de descida do efetuador apresentaram valores maiores do que as correspondentes à manutenção de posição angular ou movimentação no sentido de subida (retas tracejadas laranjas).

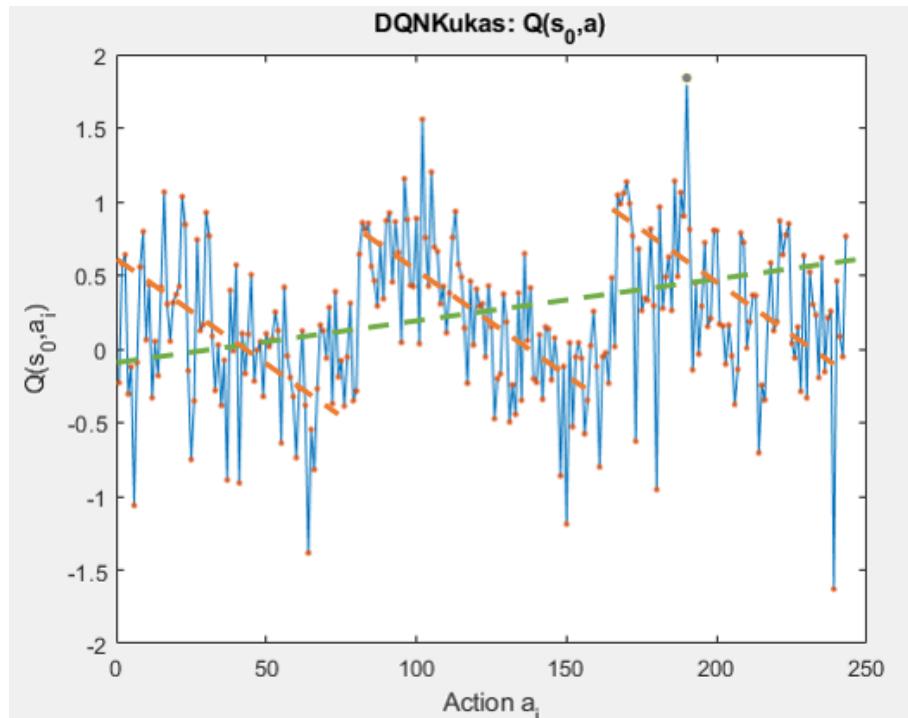


Figura 54 – Valores de $Q_\theta(\mathbf{s}_0, \mathbf{a})$ no estado inicial \mathbf{s}_0 após treino do agente. Ação de maior valor é $\mathbf{a}_{190} = (-1, 0, 1, 1, 1)$ (fonte: autores).

Além do teste tradicional de convergência do aprendizado do agente para posicionamento dada configuração fixa do *setpoint* e obstáculo, foram realizados testes para verificar o desempenho em situações nas quais o obstáculo se encontra diretamente entre as posições do efetuador e de destino. Tais configurações são tipicamente problemáticas em aplicações do método do campo potencial devido à possível existência de mínimos locais da função potencial. No aprendizado por reforço, o fator de desconto γ deve ser corretamente dimensionado para que recompensas negativas de colisão com o obstáculo sejam propagadas em uma vizinhança grande o suficiente para que o agente opte por um desvio temporário a volte a se aproximar da posição de destino após ultrapassar o obstáculo.

A figura 55 (a) ilustra o ambiente implementado para simulação do algoritmo DQN em configuração com obstáculo diretamente no caminho entre o efetuador do robô na posição inicial e a posição desejada, para

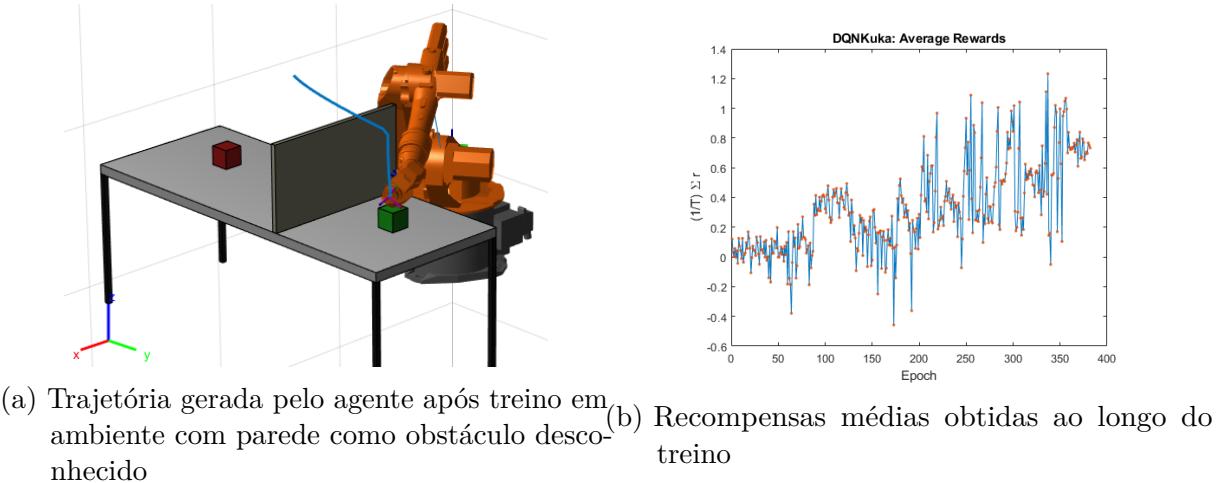


Figura 55 – Trajetória gerada por agente DQN após treino (a) e gráfico de recompensas médias em função de épocas (b) para projeto DQNKuka com obstáculo desconhecido entre posição inicial e de destino (fonte: autores)

Foi introduzida uma parede de posição e dimensões desconhecidas, de modo que colisões são a única forma do agente adquirir conhecimento de sua existência. A função de recompensa utilizada foi a função r_3 , que incentiva ações que deslocam o efetuador diretamente para a posição de destino. A figura 55 (b) ilustra a métrica de desempenho do agente dada pelos valores médios das recompensas ao longo de toda trajetória em função da época de treino.

7.7 Projeto 7: PGKukaRandomPositions

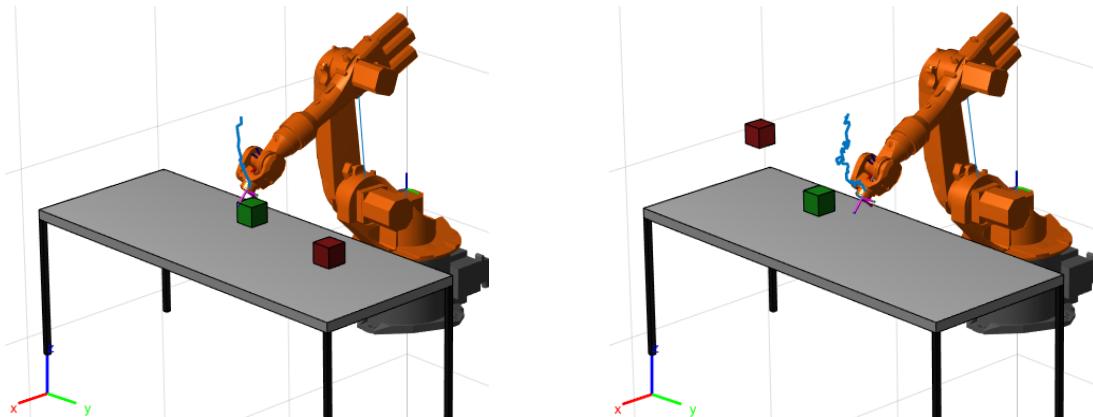
Os últimos testes realizados foram de permutação aleatória da configuração de posição desejada e de obstáculo ao longo do treino com o objetivo de verificar a capacidade do agente de efetuar o posicionamento do robô em posições genéricas a partir da mesma política de ações $\pi_\theta(\mathbf{a}|\mathbf{s})$ no caso do projeto 7, ou rede DQN $Q_\theta(\mathbf{s}, \mathbf{a})$ no caso do projeto 8.

7.7.1 Especificação do Sistema

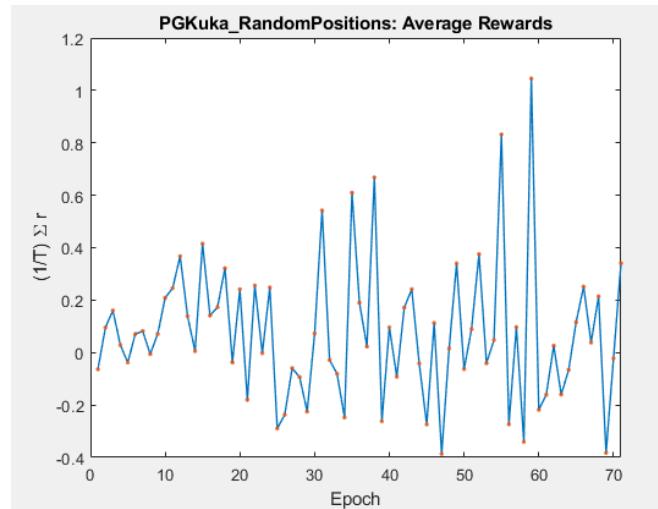
O sistema implementado é igual ao dos projetos anteriores no que diz respeito ao modelo do robô, dos espaços de estados \mathcal{S} e de ações \mathcal{A} . A diferença está na variação das posições de destino e do obstáculo entre épocas de treino dentro de $\mathcal{W} \subset \mathbb{R}^3$, onde $\mathcal{W} = \{(x, y, z) \in \mathbb{R}^3 | 0.80 < x < 1.4, -0.90 < y < 0.90, 0.80 < z < 1.00\}$ é um volume sobre a mesa. Os valores dos hiper-parâmetros utilizados são iguais aos do projeto 5, dados pela tabela 7.

7.7.2 Resultados

As posições de destino e do obstáculo foram modificadas aleatoriamente a cada trajetória simulada e o agente foi treinado por 71 épocas, mas não apresentou comportamento convergente. Em função da configuração gerada aleatoriamente a trajetória produzida pelo agente posiciona o efetuador corretamente. No entanto, configurações em que observa-se proximidade entre a posição de destino e a do obstáculo produzem trajetórias inadequadas de colisão com a mesa. As figuras 56 (a) e (b) ilustram exemplos das trajetórias mencionadas e a figura 56 (c) mostra o desempenho do agente na forma de recompensas médias por época ao longo do treino.



(a) Trajetória de posicionamento correto gerada pelo agente após treino
(b) Trajetória de colisão com a mesa gerada pelo agente após treino



(c) Recompensas médias obtidas ao longo do treino

Figura 56 – Trajetórias geradas por agente DQN após treino (a),(b) e gráfico de recompensas médias (c) para projeto DQNKuka_RandomPositions.(fonte: autores)

7.8 Projeto 8: DQNKukaRandomPositions

Com o objetivo de efetuar uma análise comparativa entre os desempenhos de agentes treinados pelos algoritmos REINFORCE episódico e DQN para generalização das

configurações de posição de destino e de obstáculo, o modelo do sistema foi mantido e o algoritmo implementado para treino foi alterado. Deseja-se verificar se, sob variação das configurações, a aproximação da função valor dos pares estado-ação $Q(\mathbf{s}, \mathbf{a})$ é mais simples do que a da função política de ações $\pi(\mathbf{a}|\mathbf{s})$.

7.8.1 Especificação do Sistema

O sistema permaneceu inalterado com relação ao último projeto: O espaço de estados \mathcal{S} consiste nas posições angulares das articulações do robô em conjunto com as posições cartesianas de destino e do obstáculo conhecido, conforme simplificação adotada. O espaço de ações \mathcal{A} é dado por todas as combinações de acionamento positivo, negativo e neutro de variação angular fixa sobre cada articulação do robô. A função de recompensa implementada é a função $r_3(\mathbf{s}, \mathbf{a})$, cujos termos de contribuição correspondentes às posições de destino e do obstáculo são proporcionais à projeção do vetor deslocamento do efetuador do robô sobre os vetores que apontam para tais posições. Por fim, a estrutura da rede neural densa a ser treinada como função aproximadora da função $Q(\mathbf{s}, \mathbf{a})$ é idêntica à do projeto 6.

7.8.2 Resultados

O agente foi treinado por 150 épocas e demonstrou posicionamento correto do efetuador com maior frequência do que o agente treinado pelo algoritmo REINFORCE episódico. A figura 57 ilustra as trajetórias obtidas para duas configurações distintas de convergência para posição desejada, assim como a evolução das recompensas médias obtidas ao longo do treino.

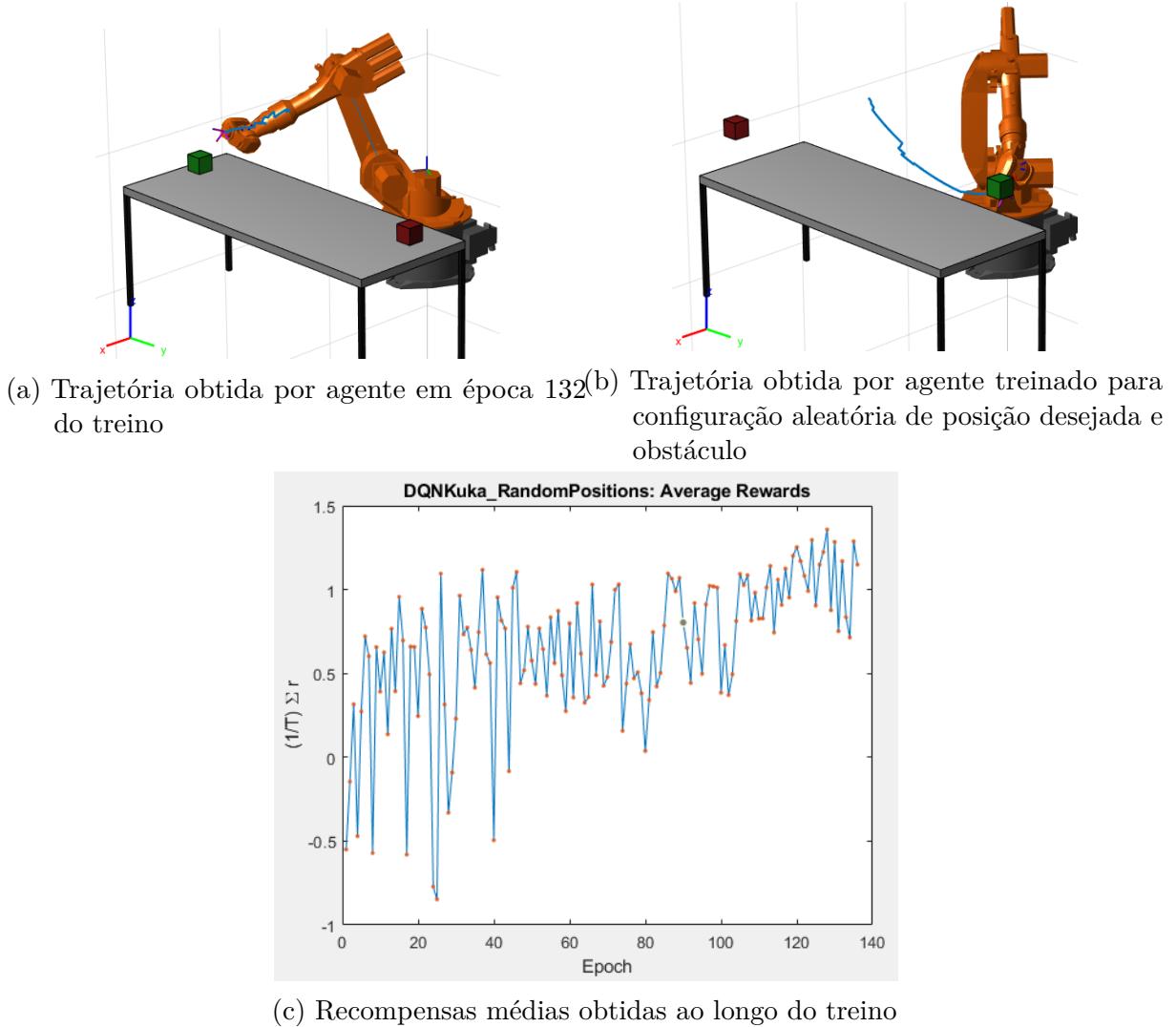


Figura 57 – Trajetória percorrida por agente durante o treino (a), trajetória após treino em configuração aleatória (b) e gráfico de recompensas médias em função de época do treino (c) (fonte: autores)

8 Análise dos Resultados

A forte dependência da convergência do algoritmo e do desempenho do agente treinado sobre a escolha da função recompensa e sobre a determinação de hiper-parâmetros adequados motivou o desenvolvimento dos projetos de teste cujos resultados são documentados no capítulo anterior. O objetivo deste capítulo é analisar os resultados obtidos e apresentar a influência dos mesmos sobre a especificação da solução final.

Será realizada a comparação do desempenho do agente sob duas perspectivas: A da função recompensa escolhida e a do algoritmo implementado. A determinação adequada de parâmetros como o fator de desconto γ , a taxa de decaimento da constante de exploração ϵ e a taxa de aprendizado α também será abordada, assim como sua influência na convergência e tempo de execução do algoritmo.

8.1 Escolha do algoritmo de treino do agente

Esta seção fornece uma análise comparativa dos algoritmos implementados sob os critérios de estabilidade, convergência e tempo de treino. A partir dos resultados obtidos nos projetos de teste e com base nos requisitos de projeto definidos no capítulo 4, optou-se pela implementação do algoritmo DQN no projeto final. Os motivos desta escolha são: Maior adequação do algoritmo para problemas com espaço de ações \mathcal{A} discretos, generalização para configurações genéricas de posição desejada do efetuador e de posição do obstáculo, tempo reduzido de treino.

8.1.1 Estabilidade e convergência

Ao longo dos projetos de teste, observou-se que, em aplicações do REINFORCE episódico, a convergência da política de ações $\pi_\theta(\mathbf{a}|\mathbf{s})$ para a política ótima $\pi^*(\mathbf{a}|\mathbf{s})$ no caso do primeiro projeto (ou quase ótima no caso dos demais) apresentou mais suavidade do que a convergência sob treino da função $Q(\mathbf{s}, \mathbf{a})$ em aplicações de *Q-Learning*. No primeiro algoritmo, trajetórias geradas em épocas subsequentes apresentaram pequena variação e a estabilidade do treino é observada no comportamento quase estritamente crescente do gráfico de recompensas médias (figura 42). Enquanto no segundo, houve maior oscilação e diferenças consideráveis entre trajetórias geradas em épocas subsequentes (figura 44).

Essa diferença se deve fundamentalmente à diferença entre funções aproximadas por ambos algoritmos, ou seja, à classe à qual pertencem: Algoritmos de iteração sobre a função política de ações aproximam diretamente a própria função $\pi(\mathbf{a}|\mathbf{s})$ utilizada para determinação da ação a ser tomada e consequente determinação da trajetória seguida.

Enquanto algoritmos de iteração sobre a função valor aproximam a função $Q(\mathbf{s}, \mathbf{a})$ utilizada indiretamente para extração da política de ações a partir da escolha da decisão de maior valor. Entre épocas do treino, a ação \mathbf{a} de maior valor em um mesmo estado \mathbf{s} é alterada e promove diferenças consideráveis nas trajetórias percorridas.

No entanto, sob o critério de convergência, o algoritmo REINFORCE episódico apresentou desempenho inferior na generalização do posicionamento para configurações aleatórias de posição desejada e de obstáculo em comparação ao DQN, conforme pode ser observado nas curvas de desempenho dos projetos 7 e 8 (figuras 56 (c) e 57 (c)).

8.1.2 Tempo de treino

Em aplicações do algoritmo REINFORCE episódico, o caráter probabilístico da função política de ações exige não somente que a política $\pi_\theta(\mathbf{a}|\mathbf{s})$ seja inicializada de forma a gerar uma distribuição de probabilidades suficientemente uniforme sobre o espaço \mathcal{A} , mas também que o número N de trajetórias simuladas por época de treino seja suficientemente alto. Isso promove melhor aproximação do gradiente da performance $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left[\sum_{t=0}^{T-1} G_t \frac{\nabla \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \right]$ e permite convergência. Em oposição, a aproximação da função $Q(\mathbf{s}, \mathbf{a})$ pelo algoritmo DQN é realizada a partir de transições $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ e não requer simulação de diversas trajetórias completas entre atualizações da rede. A tabela 9 ilustra os tempos médios de treino do agente para cada projeto de teste, em função do algoritmo implementado e da simplificação adotada.

Projeto	REINFORCE	DQN
1 Grau de Liberdade (R)	6,8min	6,9 min
2 Graus de Liberdade (RR)	11.7min	7.4min
6 Graus de Liberdade (6R), configuração fixa	30h	16h
6 Graus de Liberdade (6R), configurações aleatórias	—	25h

Tabela 9 – Tabela de tempos médios aproximados para treino do agente em função do algoritmo implementado e das características do projeto

8.2 Escolha da função de recompensa $r(\mathbf{s}, \mathbf{a})$

No aprendizado por reforço, a função de recompensa representa uma noção de qualidade das ações escolhidas e consiste na principal forma de introduzir conhecimento acerca da tarefa para a qual deseja-se treinar o agente. Em aplicações nas quais o agente não possui um modelo do ambiente, as recompensas obtidas através de interações sucessivas com o mesmo constituem a única forma de treiná-lo.

Ao longo dos projetos de teste, observou-se que agentes treinados com a função de recompensa $r_1(\mathbf{s}, \mathbf{a})$, baseada nas distâncias absolutas entre efetuador e posição de destino e obstáculo, apresentaram desempenho reduzido e necessitaram de maior número de épocas de treino para convergência, principalmente em aplicações do algoritmo REINFORCE (figura 36). Nos projetos 3 e 4 de treino do agente para controle de robô RR, agentes treinados pela função de recompensa r_1 não apresentaram comportamento convergente para a posição desejada. Assim, a utilização da mesma no projeto final foi descartada e foram desenvolvidas as funções r_2 e r_3 , descritas nas seções 5.5.2 e 5.5.3 deste trabalho.

Conforme esperado, o treinamento do agente com base nas funções de recompensa $r_2(\mathbf{s}, \mathbf{a})$ e $r_3(\mathbf{s}, \mathbf{a})$ foi mais rápido do que o baseado na primeira. Isso se deve principalmente à propriedade de média nula que ambas apresentam: Dado determinado estado, ações que diminuem a distância do efetuador à posição de destino recebem recompensas positivas e ações que o distanciam da mesma, negativas. De forma que as primeiras são encorajadas em futuras visitas ao estado e as últimas são desincentivadas.

No projeto 3, que implementa o algoritmo REINFORCE para o robô de dois graus de liberdade, percebe-se que o treino do agente a partir da função de recompensa r_3 apresentou curva de desempenho, medido pelos valores médios das recompensas obtidas a cada época, adequada, estável e crescente. Enquanto o agente treinado pela função de recompensa r_2 apresentou trecho de desempenho decrescente ao longo do treino seguido por convergência (figura 42).

No projeto 4, por outro lado, a configuração de obstáculo diretamente no caminho entre posição inicial do efetuador e posição de destino ocasionou maior estabilidade na curva de desempenho do treino que utiliza a função de recompensa r_2 do que no treino equivalente com função r_3 (figura 44). Pode-se perceber vales com desempenho reduzido em épocas nas quais houve colisão com o obstáculo. Isso se deve à priorização por parte da função r_3 de ações que levam a trajetórias nas quais o efetuador se aproxima da posição desejada sem desvios. Enquanto a função de recompensa r_2 pode ser considerada discreta sob aproximação ou distanciamento, sem preferência pela forma de aproximação.

No entanto, a especificação da forma de aproximação da posição desejada é preferível no que diz respeito à convergência em número reduzido de épocas de treino e melhor aproximação a trajetórias ótimas de posicionamento. Por esses motivos, optou-se pela utilização no projeto final da função de recompensa $r_3(\mathbf{s}, \mathbf{a}) = (k_s r_{setpoint}(\mathbf{s}, \mathbf{a}) + k_o r_{obstacle}(\mathbf{s}, \mathbf{a})) + B_{goal}(\mathbf{s}, \mathbf{a}) + P_{joint boundary}(\mathbf{s}, \mathbf{a}) + P_{collision}(\mathbf{s}, \mathbf{a})$, onde $r_{setpoint}(\mathbf{s}, \mathbf{a})$ é um termo proporcional à projeção do vetor deslocamento do efetuador sobre o vetor que aponta para a posição desejada e $r_{obstacle}(\mathbf{s}, \mathbf{a})$ é proporcional a projeção do vetor deslocamento sobre o vetor que aponta na direção oposta do obstáculo.

8.3 Determinação de hiper-parâmetros

8.3.1 Fator de desconto γ

O fator de desconto γ representa a importância de recompensas futuras na escolha de qual ação deve ser tomada no presente. Agentes treinados com fatores de desconto γ próximos de 0 priorizam recompensas imediatas próximas ao estado atual e tendem a evitar ações que produzem recompensas baixas a princípio, mesmo que as mesmas levem a estados de maior valor no futuro. Em oposição, agentes treinados com fatores de desconto γ próximos de 1 tendem a atribuir a mesma importância para recompensas obtidas em estados futuros distantes e para recompensas obtidas imediatamente. Assim, podem percorrer caminhos localmente sub-ótimos se os mesmos forem compensados no futuro por recompensas maiores do que as que seriam obtidas caso contrário.

A figura 58 ilustra o comportamento descrito e exemplifica a influência do fator de desconto γ para um caso simples no qual o obstáculo (vermelho) é desconhecido pelo agente, que recebe uma recompensa negativa em colisões, e a recompensa é maior quanto mais alinhado o vetor de deslocamento com a posição desejada (verde). O agente treinado com fator de desconto γ_1 "identifica" a presença do obstáculo, tomando uma ação sub-ótima no que diz respeito ao alinhamento com a posição desejada, somente em estados próximos do obstáculo. Já o agente treinado com fator de desconto γ_3 efetua o caminho de desvio do obstáculo com maior antecedência devido à propagação para estados anteriores das recompensas negativas recebidas em colisões com o mesmo ao longo do treino.

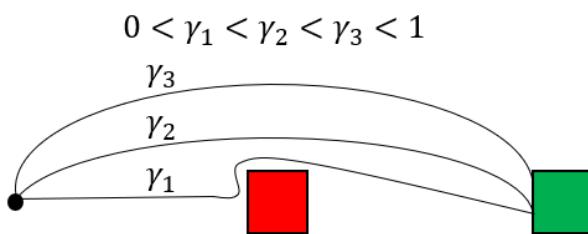


Figura 58 – Diagrama esquemático de possíveis trajetórias percorridas por agentes após treino com diferentes fatores de desconto γ (fonte: autores).

Aplicações de Aprendizado por Reforço apresentam um desafio denominado: Problema da Atribuição de Recompensas. Suponha que deseja-se treinar um agente A para receber o estado de um jogo, por exemplo o jogo de Atari *Pong*, e determinar a ação a ser tomada para marcar pontos sobre o oponente fazendo a bola atravessar sua linha de gol enquanto protege sua própria (figura 59). Vamos supor ainda que no início do treino o agente teve sorte e conseguiu marcar um ponto. O mesmo recebe uma recompensa associada à pontuação, mas como saber qual movimento dentre as dezenas de movimentos executados pelo agente deve ser encorajado e quais movimentos devem ser evitados em

visitas futuras ao mesmo estado? A escolha adequada do fator de desconto γ é uma das formas de abordar este problema.

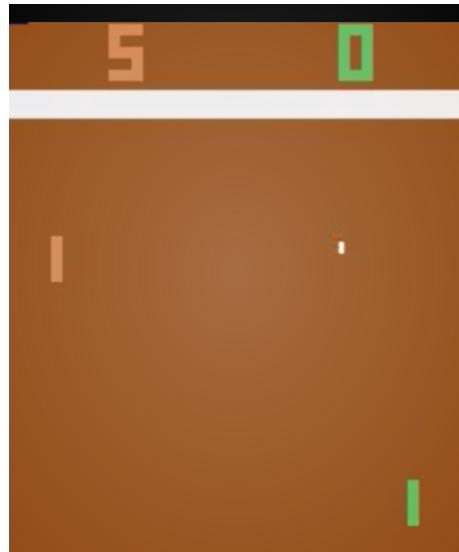


Figura 59 – Captura de tela de jogo *Pong* em simulação do OpenAI Gym (fonte: autores).

Em sistemas nos quais o sucesso do agente está diretamente associado a ações tomadas em estados consideravelmente distantes, o fator de desconto γ deve ser alto o suficiente para que a recompensa obtida seja "propagada" para estados anteriores e incentive as respectivas ações responsáveis, como no caso do agente que joga *Pong*. Nestes casos é comum utilizar fatores de desconto acima de 0.99. No entanto, sistemas em que recompensas podem ser atribuídas a ações mais recentes tipicamente implementam fatores de desconto menores, como é o caso do agente treinado para posicionamento de braço robótico neste projeto.

Para determinação do fator de desconto γ a ser utilizado foram realizados testes nos primeiros 4 projetos devido ao tempo de treino reduzido. Observou-se que a convergência do aprendizado era mais rápida para valores menores de γ (figura 60) em casos nos quais havia um caminho livre de obstáculos entre a posição inicial e a posição desejada do efetuador. Considerando configurações nas quais o caminho direto para a posição de destino se encontra obstruído por um obstáculo, assim como a penalidade obtida por colisão com o mesmo, optou-se pela utilização de fator de desconto $\gamma = 0.3$ no projeto final.

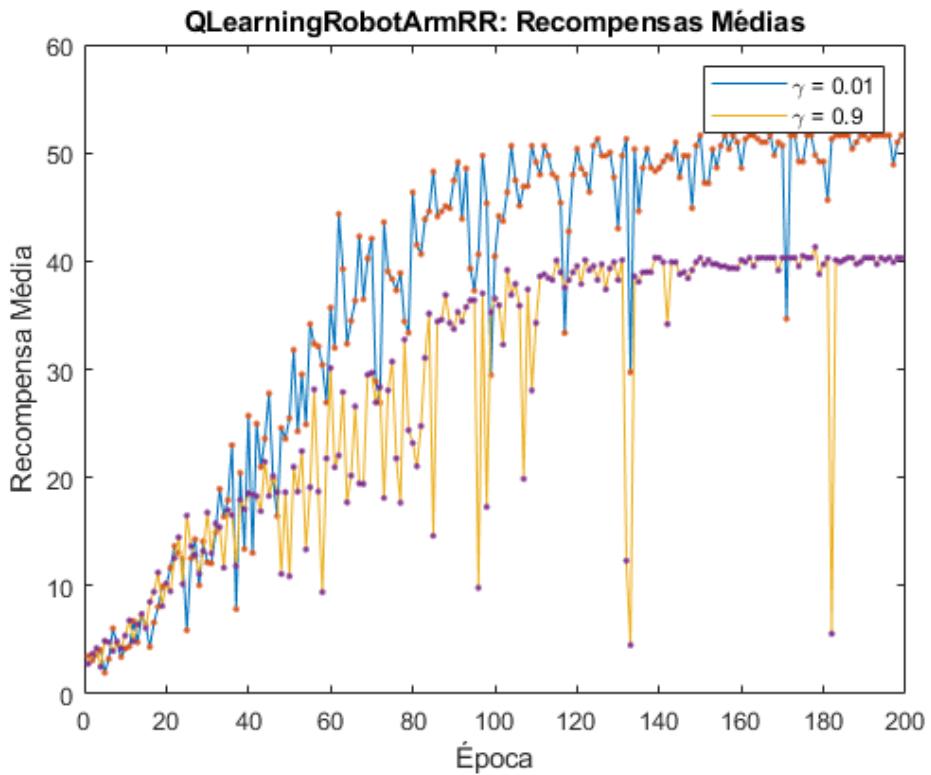


Figura 60 – Desempenho de dois agentes do projeto QLearningRobotArmRR ao longo do treino com fatores de desconto $\gamma = 0.01$ (azul) e $\gamma = 0.9$ (amarelo) (fonte: autores).

8.3.2 Taxa de decaimento da constante de exploração ϵ

A taxa de decaimento da constante de exploração ϵ representa a velocidade com a qual o agente transiciona da política inicial aleatória de exploração para a política final determinística de aproveitamento. A escolha desse parâmetro deve ser compatível com o número de transições de estados e com o número esperado de épocas de treino. Taxas de decaimento muito baixas podem levar o agente a nunca explorar o ambiente o suficiente para encontrar a melhor ação a ser tomada em determinado estado, o que pode promover convergência a trajetórias sub-ótimas.

Ao longo dos projetos de teste percebeu-se que o não conhecimento *a priori* do número de transições de estados em determinada trajetória, devido aos critérios de parada descritos na seção 5.4.1, dificulta a determinação adequada da taxa de decaimento de ϵ . Assim, foram implementadas taxas distintas para o decaimento da constante de exploração entre épocas *epsilonDecayPerEpoch* e entre transições dentro de determinada época *epsilonDecay*. Essa solução permite maior controle sobre a constante de exploração entre épocas do treino, conforme tabela 10.

Época	Faixa de valores de ϵ ($t = 0 \rightarrow T$)
1	1 → 0.932
50	0.778 → 0.726
100	0.606 → 0.565
150	0.471 → 0.440
200	0.367 → 0.342
250	0.286 → 0.266
300	0.222 → 0.207

Tabela 10 – Tabela valores da constante de exploração ϵ em função da época de treino para $epsilonDecayPerEpoch = 0.995$ e $epsilonDecay = 0.999$

8.3.3 Taxa de aprendizado α

A taxa de aprendizado α foi escolhida de modo compatível com os valores médios do gradiente da função custo $\nabla_{\theta} \frac{1}{2} (Q_{\theta}(\mathbf{s}, \mathbf{a}) - y)^2$ para proporcionar atualizações suficientemente pequenas nos pesos da rede neural $Q_{\theta}(\mathbf{s}, \mathbf{a})$ e proporcionar estabilidade. No entanto, observou-se que o tempo total de treino, a partir de projeções feitas com base na tabela 9, seria adequado para a escolha da taxa de aprendizado, uma vez que taxas de aprendizado muito baixas garantem estabilidade de treino em troca de tempo de execução. A partir dos testes realizados optou-se por $\alpha = 0.005$.

9 Resultados do Projeto Final

Este capítulo tem como objetivo apresentar os resultados obtidos no projeto DQN^{KukaImage}, descrito na seção 6.1.1 deste trabalho, assim como breve análise tendo em consideração os resultados obtidos nos projetos anteriores.

9.1 Especificação do Sistema

O projeto final consiste no treinamento de um agente por meio do algoritmo DQN para controle de posicionamento, em ambiente de simulação gráfica, de um modelo do robô manipulador KUKA-KR16 em ambiente sujeito a obstáculos. Cada estado s do sistema (seção 5.2.2) é modelado como um conjunto de duas imagens RGB de tamanho 24x24x3 obtidas por câmeras posicionadas superior e lateralmente. O espaço de ações \mathcal{A} consiste em todas as combinações de movimentos em sentido horário, anti-horário ou permanência em posição angular para cada uma das cinco articulações controláveis do robô (seção 5.3). Optou-se pela implementação da função de recompensa r_3 , proporcional à projeção do vetor de deslocamento do efetuador do robô sobre o vetor que aponta para a posição desejada, com termos discretos que representam bônus e penalidades (seção 5.5.3). A tabela 11 ilustra os parâmetros adotados.

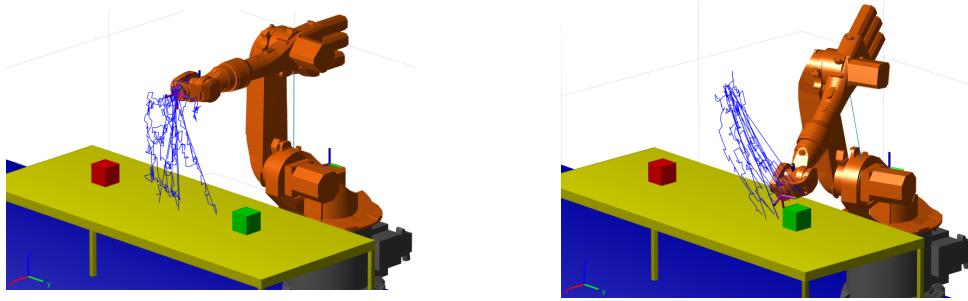
Parâmetro	Descrição	Valor
$p_{setpoint}$	Posição de destino (m)	(1.05, 0.45, 0.75)
$p_{obstacle}$	Posição de obstáculo (m)	(1.05, -0.55, 0.75)
$table_{length}$	Comprimento da mesa (m)	2
$table_{width}$	Largura da mesa (m)	0.8
$table_{height}$	Altura da mesa (m)	0.7
$\Delta\theta$	Mínima variação angular	1°
α	Taxa de Aprendizado	0.005
B_{goal}	Bônus de destino	20
$P_{collision}$	Penalidade de colisão	-20
$P_{joint boundary}$	Penalidade de fim de curso	-10
r_{infl}	Raio de influência do obstáculo (m)	0.50
$MaxEpoch$	Número máximo de épocas de treino	300
$Ntrajs$	Número de trajetórias por época	10
$MiniBatchSize$	Número de transições amostradas para treino	200
T	Número máximo de transições por trajetória	70
γ	Fator de desconto	0.3
$dim(s)$	Dimensão de cada estado s	3456
$dim(a)$	Dimensão de cada ação a	5
$size(\mathcal{A})$	Tamanho do espaço de ações	243
$(n_{in}, n_{h1}, n_{h2}, n_{out})$	Dimensões de camadas da rede $\pi_\theta(a s)$	(3456, 400, 300, 243)
k_s	Fator multiplicativo de r_3	2
k_o	Fator multiplicativo de r_3	1

Tabela 11 – Tabela para inicialização de variáveis e hiper-parâmetros do projeto DQNKu-kaImage

9.2 Resultados

O agente foi treinado e foi observada convergência no posicionamento em apenas 30 épocas. No entanto, o tempo demandado em cada época para simulação de conjunto de 10 trajetórias, armazenamento de transições em *buffer* de experiência, amostragem aleatória de 200 transições e atualização de parâmetros θ da rede $Q_\theta(s, a)$ a partir de conjunto de transições foi consideravelmente superior ao tempo equivalente nos projetos de teste 6 e 8.

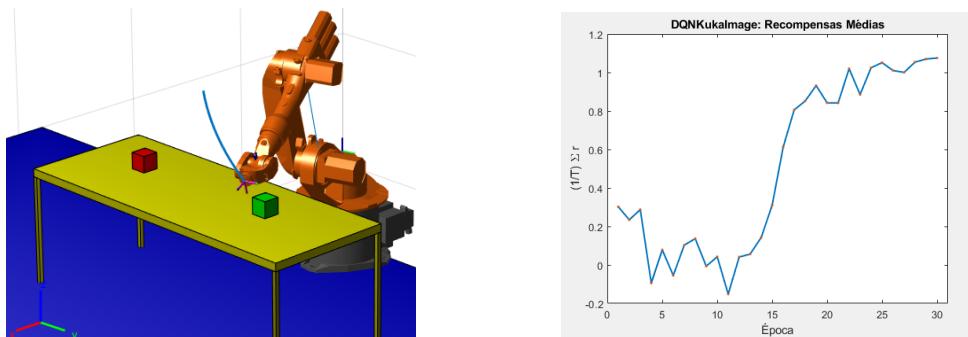
A figura 61 ilustra as trajetórias percorridas pelo agente para armazenamento de transições para treino na primeira (a) e última (b) épocas. Percebe-se que a rede inicial $Q_{\theta_0}(s, a)$ prioriza, para estados em torno do estado inicial s_0 , ações que levam o efetuador a colisões com a mesa sem preferência aparente por aproximação da posição desejada ou do obstáculo. No final do treino, as trajetórias geradas são direcionadas à posição de destino e a variabilidade entre elas se deve apenas à constante de exploração ϵ para busca por caminho ótimo.



(a) Conjunto de trajetórias simuladas na primeira época de treino. (b) Conjunto de trajetórias simuladas na época 30 de treino.

Figura 61 – Trajetórias simuladas na primeira (a) e última (b) épocas do treino no projeto DQNKukaImage (fonte: autores)

Após a primeira época com 10 tentativas bem sucedidas de posicionamento do efetuador o treinamento do agente foi interrompido e a trajetória ótima foi determinada (figura 62 (a)), ou seja, obtida a partir da escolha a cada estado s_t da ação $a_t = \text{argmax}_{a' \in \mathcal{A}} Q_\theta(s_t, a')$. Percebe-se que a trajetória resultante é suavizada ao longo do treino e a eliminação da exploração aleatória após o mesmo faz com que o caminho percorrido apresente desvios mínimos do caminho ótimo.



(a) Trajetória *greedy* ($\epsilon = 0$) obtida por (b) Recompensas médias obtidas em função agente após treino. de época do treino.

Figura 62 – Trajetória obtida após convergência e gráfico de desempenho do agente ao longo do treino (fonte: autores)

Parte VI

Conclusão

10 Conclusão

Este capítulo tem como objetivo a documentação das conclusões obtidas a partir dos resultados apresentados nos capítulos anteriores. Serão abordadas as consequências da análise comparativa de algoritmos de Aprendizado por Reforço em aplicações de robôs manipuladores, a viabilidade dos mesmos em comparação com métodos tradicionais consolidados, técnicas de aumento de convergência e melhorias na arquitetura do sistema de treino.

10.1 Aprendizado por Reforço na Robótica

A possibilidade de modelar o espaço de ações do sistema como um espaço discreto $\mathcal{A} = \{-1, 0, 1\}$ ⁵ neste trabalho permitiu a implementação do algoritmo DQN, que apresentou desempenho superior nos projetos de teste. No entanto, aplicações nas quais as variáveis de entrada controladas pelo agente são contínuas, como o controle de um robô manipulador a partir das tensões elétricas em seus motores, são mais apropriadas para treino sob algoritmos de iteração sobre a função política de ações $\pi(s, a)$, como REINFORCE, DDPG ou algoritmos Ator-Crítico (figura 63), nos quais tanto a política π_θ quanto a função valor $Q_\omega(s, a)$ são aproximadas. Assim, o controle de baixo nível do robô KUKA-KR16, não limitado ao planejamento de trajetória, exige adaptação do algoritmo de aprendizado para permitir geração de probabilidades sobre espaço contínuo de ações.

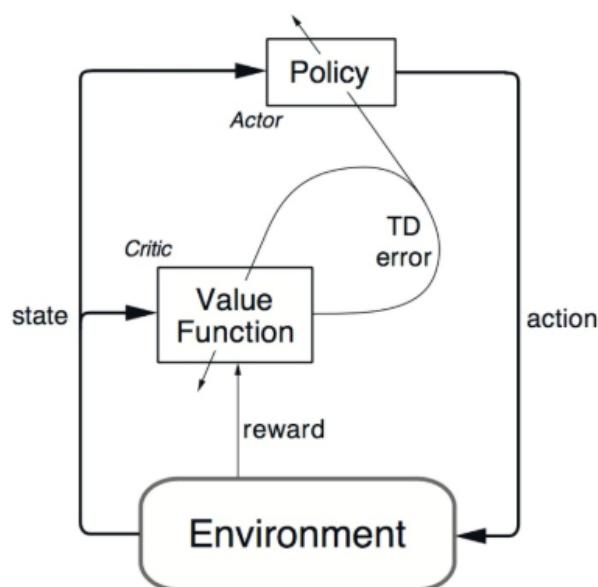


Figura 63 – Diagrama esquemático do fluxo de algoritmo Ator-Crítico (SUTTON, 1998).

Outra limitação associada à aplicações de aprendizado por reforço para controle de robôs manipuladores industriais é a necessidade de geração de grande volume de dados no robô físico devido à dificuldade de transferência do aprendizado de ambientes de simulação para robôs reais (James, 2016). O tempo de treinamento, conforme discutido na seção seguinte, pode ser reduzido sob adaptação da arquitetura de treino para implementação de múltiplos agentes.

10.2 Arquitetura de Treino

Ao longo dos testes foi identificada grande dependência da exploração suficiente de estados e ações para aproximação adequada da função $Q(\mathbf{s}, \mathbf{a})$. A inicialização da rede DQN de modo a atribuir valores muito maiores a um conjunto restrito de ações em relação às demais em conjunto com o rápido decaimento da constante de exploração ϵ pode levar à exploração não representativa dos espaços de estados e ações, o que pode levar o agente a convergir para políticas sub-ótimas. Esse comportamento foi observado no projeto DQNKukaImage em um treino de 36 épocas no qual o pulso esférico do robô alinhava a orientação do efetuador com a posição de destino, mas não a alcançava.

Com o objetivo de resolver esse problema e promover a redução do tempo total de treino em aplicações com robôs reais é proposta uma arquitetura multi-agente (figura 64) na qual múltiplos agentes, inicializados com redes DQN arbitrárias $Q_{\theta_1}(\mathbf{s}, \mathbf{a}), \dots, Q_{\theta_n}(\mathbf{s}, \mathbf{a})$ executam trajetórias de exploração do ambiente, armazenam transições em um *buffer* único de experiência e de tempos em tempos atualizam os parâmetros de suas respectivas redes DQN $Q_{\theta_i}(\mathbf{s}, \mathbf{a})$ a partir de um servidor central que executa o passo de minimização do custo $\frac{1}{2}(Q_{\theta}(\mathbf{s}, \mathbf{a}) - y)^2$ a partir da experiência adquirida por todos agentes. É esperado que o treino realizado nesta arquitetura apresente maior estabilidade em comparação com o treino baseado nas transições adquiridas por um único agente (GU, S et al, 2016).

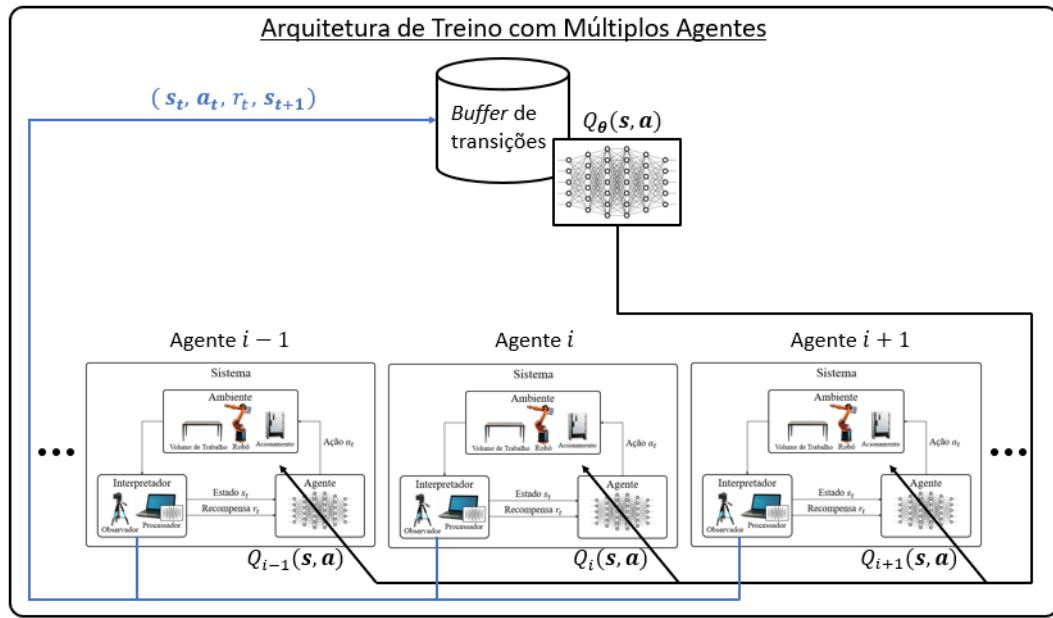


Figura 64 – Diagrama esquemático de arquitetura proposta para treino com múltiplos agentes (fonte: autores).

Parte VII

Referências Bibliográficas

Referências Bibliográficas

- BELLEMARE, M. G et al. **The Arcade Learning Environment: An Evaluation Platform for General Agents**, Canada, 2013.
- BELLMAN, R. E. **Introduction to the Mathematical Theory of Control Processes**, volume 40-I. Academic Press, New York, NY, 1967.
- BELLMAN, R. E. **Introduction to the Mathematical Theory of Control Processes**, volume 40-II. Academic Press, New York, NY, 1971.
- CABRAL, E. **Análise de Robôs: Capítulo 5 - Cinemática Direta de Robôs Manipuladores**. São Paulo, 2019.
- FERGUSON, M et al. **Automatic Localization of Casting Defects with Convolutional Neural Networks**, IEEE International Conference on Big Data, 2017.
- GU, S et al. **Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Update**. Google Brain, 2016.
- HOWARD, A. G.; ZHU, M; CHEN, B.; KALENICHENKO, D.; WANG, E.; WEYAND, T.; ANDREETTO, M.; HARTIWIG, A. **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**, 2017
- JAMES, S.; JOHNS, E. **3D Simulation for Robot Arm Control with Deep Q-Learning**, Imperial College London, UK, 2016.
- KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. **Reinforcement Learning: A Survey**, Computer Science Department, Brown University, 1996.
- KALASHNIKOV, D et al. **QT-Opt: Scalable Deep Reinforcement Learning for Vision Based Robotic Manipulation**, 2nd Conference on Robot Learning, Zurich, Switzerland, 2018.
- KOBER, J.; BAGNELL, A. J.; PETER J. **Reinforcement Learning in Robotics: A Survey**, 2013.
- KORMUSHEV, P.; CALINON, S.; CALDWELL, D.G. **Reinforcement Learning in Robotics: Applications and Real-World Challenges**. Department of Advanced Robotics, Istituto Italiano di Tecnologia, 2013.
- KUKA ROBOTICS. **Kuka KR6, KR16, KR16 L6, KR 16S specification**, 2003.
- LILLICRAP, T et al. **Continuous Control With Deep Reinforcement Learning**, Google DeepMind, London. ICLR, 2016.

- MNIH, V et al. **Human-Level Control Through Deep Reinforcement Learning**, Google DeepMind, London, 2015.
- MNIH, V et al. **Playing Atari With Deep Reinforcement Learning**, Google DeepMind, 2013.
- PIOTROWSKI, N; BARYLSKI, A. **Modelling a 6-DOF Manipulator Using Matlab Software**. Archives of Mechanical Technology and Automation. Vol. 34, n. 3. Gdansk University of Technology, 2014.
- ROSEN, C.A. **Handbook of Industrial Robotics: Chapter 3 - Robots and Machine Intelligence**. 2nd Edition. New York: John Wiley & Sons, 1999, p19-30.
- SIEGWART, R; NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robots**, MIT Press, 2004.
- SIMONYAN, K; ZISSERMAN, A. **Very deep convolutional networks for large-scale image recognition**, 2014.
- SUTTON, R. S.; BARTO A. G. **Reinforcement Learning: An Introduction**. 2nd Edition. The MIT Press, 2017.
- TSURUMINE, Y. et al. **Deep Reinforcement Learning with Smooth Policy Update: Application to Robotic Cloth Manipulation**, Nara Institute of Science and Technology, Division of Information Science. Japan, 2018.
- WATKINS, C. **Learning from Delayed Rewards**, King's College, 1989.
- WILLIAMS, R. J. **Simple statistical gradient-following algorithms for connectionist reinforcement learning**, 1992.
- YAN, Z et al. **Building a ROS-Based Testbed for Realistic Multi-Robot Simulation: Taking the Exploration as an Example**, MDPI Robotics Journal, 2017.
- ZAMALLOA, I. et al. **Dissecting Robotics - historical overview and future perspectives**, 2007.
- ZAMORA, I et al. **Extending the OpenAI Gym for Robotics: A Toolkit for Reinforcement Learning Using ROS and Gazebo**
- ZHAO, R. **Trajectory Planning and Control for Robot Manipulators**, Université Paul Sabatier - Toulouse III, 2015

Apêndices

APÊNDICE A – Repositório github

O seguinte link contém o repositório de códigos de todos projetos implementados:

<https://github.com/MMenonJ/Controle_Robo_Manipulador>

APÊNDICE B – Documentação de Principais Funções

Esta seção tem como objetivo documentar as funções implementadas no projeto final para criação do ambiente de simulação, implementação do agente e algoritmo de aprendizado, armazenamento e visualização dos resultados.

B.1 fillReplayBufferNTrajs (transition_buffer, dqn, epoch_index, eps_init)

- Descrição:

Função que recebe o *buffer* de experiência vazio do agente, a rede DQN a ser utilizada para simulação de trajetórias, o índice da época do treino e o valor inicial de ϵ para realização de exploração conforme política ϵ - *Greedy* e executa a simulação de $Ntrajs$ trajetórias, armazenando no *buffer* as transições de estado do sistema e retornando o *buffer* preenchido. O número de transições individuais armazenadas é variável, uma vez que depende do instante de tempo em que um estado terminal foi atingido em cada trajetória.

- *Inputs*:

- **transition_buffer**: Objeto da classe transitionBuffer que contém vetor de transições ($s_t, a_t, r_t, s_{t+1}, bool_{term}$) e número de transições armazenadas no *buffer*.
- **dqn**: Rede DQN a ser utilizada para simulação de trajetórias para preenchimento do *buffer* de experiências.
- **epoch_index**: Índice da época de treino utilizado para armazenamento de conjunto de trajetórias em estrutura global *epochBuffer* para análise posterior.
- **eps_init**: Valor inicial da variável $\epsilon \in [0, 1]$ utilizada em política ϵ - *Greedy* para determinação de ação a_t a ser tomada.

- *Output*:

transition_buffer preenchido com transições ($s_t, a_t, r_t, s_{t+1}, bool_{term}$) de $Ntrajs$ trajetórias diferentes, onde $Ntrajs$ é um hiper-parâmetro que determina quantas trajetórias são armazenadas na memória do agente a cada época do treino.

B.2 transitionBuffer.sampleMiniBatch (obj,n)

- Descrição:

Método da classe *transitionBuffer* que recebe *buffer* obj preenchido com transições e amostra n transições para composição de *batch* para treino.

- *Inputs*:

- **obj**: Objeto da classe *transitionBuffer* que contém vetor de transições ($s_t, a_t, r_t, s_t, bool_{term}$) e número de transições armazenadas no *buffer*.
- **n**: Número inteiro n de transições a serem amostradas para composição de *batch* para treino.

- *Output*:

transitions_batch: Vetor $\{transitions(i)\}_{i=1}^n$ de objetos da classe *transition*, onde $transitions(i) = (s_t, a_t, r_t, s_t, bool_{term})$.

B.3 gradientDescent(dqn,transitions_batch)

- Descrição:

Função que recebe rede neural DQN e *batch* de transições *transitions_batch*, efetua um passo do gradiente descendente sobre os pesos da rede para minimizar $\mathcal{L}(\theta) = \frac{1}{2}(Q_\theta(s, a) - y)^2$, conforme algoritmo da seção 5.6.2 deste trabalho.

- *Inputs*:

- **dqn**: Rede neural DQN a ser treinada para aproximar os valores de $Q(s, a)$.
- **transitions_batch**: Vetor de objetos da classe *transition* que representa *Batch* de transições amostradas para treino.

- *Output*:

dqn: Rede DQN após atualização de pesos conforme gradiente descendente para minimização da função custo $\mathcal{L}(\theta) = \frac{1}{2}(Q_\theta(s, a) - y)^2$

B.4 action(dqn,s,prob)

- Descrição:

Função que recebe rede neural DQN, estado **s** e probabilidade de tomar uma ação aleatória e retorna a ação a ser tomada pelo agente, conforme política ϵ - *Greedy*.

- *Inputs*:

- **dqn**: Rede neural DQN a ser utilizada para determinação de ação **a** a ser tomada pelo agente.

- **s:** Estado atual s do sistema.
- prob: Probabilidade de o agente tomar uma ação aleatória $a \in \mathcal{A}$ segundo política ϵ - *Greedy*.
- *Output:*
- **a:** Ação a a tomada pelo agente em estado s .

B.5 dynamics(s, a)

- *Descrição:*
Função que implementa dinâmica do sistema conforme seção 5.4. Recebe estado atual s do sistema, ação a a ser tomada e retorna estado seguinte $s' = f(s, a)$.
- *Inputs:*
 - **s:** Estado s atual do sistema.
 - **a:** Ação a a ser executada.
- *Output:*
s_new: Estado seguinte s' do sistema.

B.6 reward(s, s_{new})

- *Descrição:*
Função que recebe estado atual s e estado seguinte s' e retorna recompensa $r(s, a)$ recebida pelo agente, onde a é a ação que leva o sistema do estado s ao estado s' . Optou-se pelo par de estados vizinhos como parâmetros em oposição ao tradicional par estado-ação com o objetivo de evitar custo computacional associado à avaliação da função de dinâmica do sistema $f(s, a)$.
- *Inputs:*
 - **s:** Estado s atual do sistema.
 - **s_new:** Estado seguinte s' do sistema.
- *Output:*
r: Recompensa imediata $r(s, a)$ recebida pelo agente pela execução da ação a no estado s .

Anexos

ANEXO A – Código Matlab

A.1 Script Principal: DQNKuka_image.m

```
1 %% Deep Q Network Kuka Image (DQNkukaImage) – Deep Q-learning
  Kuka Robot Arm 5R (5DoF) %%
2 %
```

```
3 clear all;
4 close all;
5 clc;
6
7
8 %% SETTINGS
9
10 %Set Point
11 global x_sp;
12 global y_sp;
13 global z_sp;
14 global setpoint;
15 x_sp = 1.05;
16 y_sp = 0.45;
17 z_sp = 0.75;
18 setpoint = [x_sp; y_sp; z_sp];
19 global setpoint_width;
20 setpoint_width = 0.1;
21
22 %Obstacle
23 global x_obs;
24 global y_obs;
25 global z_obs;
26 global obstacle;
27 x_obs = 1.05;
28 y_obs = -0.55;
29 z_obs = 0.75;
30 obstacle = [x_obs; y_obs; z_obs];
```

```
31 global obstacle_width;
32 obstacle_width = 0.1;
33
34 % Table
35 global table_length;
36 global table_width;
37 global table_height;
38 global table_thickness;
39 table_length = 2;
40 table_width = 0.8;
41 table_height = 0.7;
42 table_thickness = 0.05;
43
44 global xtable_orig;
45 global ytable_orig;
46 global ztable_orig;
47 xtable_orig = 0.7;
48 ytable_orig = -0.5*table_length;
49 ztable_orig = table_height - table_thickness;
50
51 % Robot (RigidBodyTree)
52 global robot;
53 robot = importrobot('kr16_2.urdf');
54 robot.DataFormat = 'column';
55 showdetails(robot)
56
57 % Initial and Current Robot Configuration (degrees)
58 global Q_0;
59 global Q;
60 Q_0 = rad2deg(robot.homeConfiguration);
61 Q = Q_0;
62
63 % Initial State Vector:
64 % (q1_1,q2_0,q3_0,q4_0,q5_0,x_sp,y_sp,z_sp,x_obs,y_obs,z_obs)
65 global S_0;
66 S_0 = cat(1,Q_0(1:5),setpoint,obstacle);
67 tform_0 = getTransform(robot,Q_0,'tool0','base');
68 euler_angles_0 = rad2deg(tform2eul(tform_0,'ZYX'));
```

```
70
71 % Variação Angular Delta Theta (em graus)
72 global delta_theta;
73 delta_theta = 1;
74
75 % Bonus for reaching set-point
76 global winBonus;
77 winBonus = 20; % Option to give a very large bonus when the
    system reaches the desired state).
78 global obstaclePenalty;
79 obstaclePenalty = -20;
80 % Flag to determine end of trajectory (setpoint or obstacle has
    been reached)
81 global terminate;
82 terminate = false;
83 global BoundaryError;
84 BoundaryError = false;
85 % Penalty for reaching boundary of robot's work volume.
86 global JointBoundaryPenalty;
87 JointBoundaryPenalty = -10;
88 % Penalty for table collision.
89 global TableCollisionPenalty;
90 TableCollisionPenalty = -20;
91
92 % Actions
93 global A;
94 A = createActionSpace();
95
96 % Number of Episodes
97 global maxEpoch;
98 maxEpoch = 300;
99 % Number of Trajectories in Replay Buffer
100 global Ntrajs;
101 Ntrajs = 10;
102 % Number of Timesteps in one trajectory
103 global T;
104 T = 70; % Maximum number of (st,at,rt+1), t=1...T-1, tuples in a
    trajectory (size of trajectory)
105
```

```
106 % Discount Factor for return Gt of state st .
107 global gamma;
108 gamma = 0.3;
109 % Learning Rate alpha
110 global alpha;
111 alpha = 0.005;
112 %% Exploration vs. exploitation (Epsilon-greedy)
113 % Probability of picking random action vs estimated best action
114 global epsilon_0;
115 global epsilonDecay;
116 epsilon_0 = 1; % Initial value
117 epsilonDecay = 0.999; % Decay factor per iteration.
118
119 % Replay Buffer Max Size and mini batch size
120 global replayBufferSize;
121 global miniBatchSize;
122 replayBufferSize = Ntrajs*T;
123 miniBatchSize = 200;
124
125 % Number of pixels in frames used as states
126 global imgNumPixels;
127 imgNumPixels = 24;
128
129 % DQN Network Parameters
130 global input_dim;
131 global n_layers;
132 global output_dim;
133
134 input_dim = imgNumPixels*imgNumPixels*3*2;
135 n_layers = 3;
136 output_dim = 243;
137
138 % Convolutional Neural Network (VGG16)
139 % net_conv = vgg16;
140
141 %% Initial Q Neural Network
142 dqn = createDQN(input_dim , n_layers , output_dim );
143 view(dqn);
144 dqn_0 = dqn;
```

```
145
146 %% Create Transition Buffer , Epoch Buffer and Policy Buffer
147 transition_buffer = transitionBuffer( replayBufferSize );
148
149 policies = policies( maxEpoch );
150
151 global epochBuffer ;
152 epochBuffer = episodes( maxEpoch , Ntrajs );
153
154 InitStateQValues = zeros( output_dim , 1 , maxEpoch );
155
156 %% DQN Algorith
157 dqn_cont = 1;
158 for ep_index = 1:maxEpoch
159     tic ;
160     fprintf( '-----Epoch %d... -----\\n' , ep_index );
161     epochBuffer . n_eps = ep_index ;
162     Q = Q_0;
163     s = S_0;
164     epsilon = (0.995^(ep_index-1))*epsilon_0 ;
165     % epsilon = epsilon_0 ;
166
167     % Store current network
168     if( rem(ep_index , 50) == 0 || ep_index == 1)
169         policies . networks(dqn_cont) . policy = dqn ;
170         dqn_cont = dqn_cont + 1;
171     end
172
173     % Fill Replay Buffer with NTrajs trajectories
174     transition_buffer = fillReplayBufferNTrajs( transition_buffer
175 , dqn , ep_index , epsilon );
176
177     % Sample mini batch and train
178     transitions_batch = transition_buffer . sampleMiniBatch (
179         miniBatchSize );
180     dqn = gradientDescent( dqn , transitions_batch );
181
182     % Clear Replay Buffer
183     transition_buffer = transition_buffer . clearBuffer () ;
```

```

182
183 % Plot Greedy Trajectory
184 InitStateQValues (: , 1 , ep_index ) = plotStateQValues (dqn , S_0) ;
185 plotGreedyTrajectory (dqn , S_0) ;
186
187 % Average Rewaard In Epoch
188 avg_reward = averageRewardInEpoch ( epochBuffer . eps ( ep_index ) )
189 ;
190 fprintf ( 'Average Reward in Epoch = %d \n' , avg_reward ) ;
191 toc ;
192 end

```

A.2 action.m

```

1 function f = action (dqn , s , prob)
2 % Fun o que retorna a a o a ser tomada dada a DQN, o
3 % estado s e a
4 % probabilidade epsilon-greedy de se tomar uma a o aleat ria
5 .
6 global A;
7 global output_dim;
8 global input_dim;
9
10 if (rand > prob)
11     % Get frames from state s
12     [img_s1 , img_s2] = getRobotPerspective (s) ;
13     % Normalize Images and reshape (flatten)
14     img_s_norm = normalizeImage ([img_s1 img_s2]) ;
15     s_img = reshape (img_s_norm , input_dim , 1) ;
16     % Apply Fully Connected DQN Network
17     y = net_output (dqn , s_img) ;
18     % Determine best action given state s
19     [~, a_index] = max (y) ;
20 else
21     a_index = randi ([1 output_dim]) ;
22 end
23 f = A (a_index , : ) ;
24 end

```

A.3 dynamics.m

```
1 function f = dynamics(s,a)
2 % Função que recebe o estado atual e a ação tomada e retorna
3 % o próximo
4
5 % Input: s - Estado atual.
6 % Input: a - Ação a ser executada.
7 % Output: f - Estado novo após execução da ação.
8
9 global robot;
10 global Q;
11 global delta_theta;
12 global x_sp;
13 global y_sp;
14 global z_sp;
15 global x_obs;
16 global y_obs;
17 global z_obs;
18 global BoundaryError;
19 BoundaryError = false;
20
21 % Current State
22 q1 = s(1);
23 q2 = s(2);
24 q3 = s(3);
25 q4 = s(4);
26 q5 = s(5);
27
28 % Action to be taken (1, 0, -1)
29 a1 = a(1);
30 a2 = a(2);
31 a3 = a(3);
32 a4 = a(4);
33 a5 = a(5);
34
35 % New State
36 q1_new = q1 + a1*delta_theta;
37 q2_new = q2 + a2*delta_theta;
```

```
38 q3_new = q3 + a3*delta_theta;
39 q4_new = q4 + a4*delta_theta;
40 q5_new = q5 + a5*delta_theta;
41
42 q_new = [q1_new; q2_new; q3_new; q4_new; q5_new; 0];
43 Q = q_new;
44
45 s_new = [q1_new; q2_new; q3_new; q4_new; q5_new; x_sp; y_sp; z_sp; x_obs
46 ; y_obs; z_obs];
47
48 if (q1_new < rad2deg(robot.getBody('link_1').Joint.
49 PositionLimits(1)) || q1_new > rad2deg(robot.getBody('link_1'.
50 ) .Joint.PositionLimits(2)))
51     BoundaryError = true;
52 elseif (q2_new < rad2deg(robot.getBody('link_2').Joint.
53 PositionLimits(1)) || q2_new > rad2deg(robot.getBody('link_2'.
54 ) .Joint.PositionLimits(2)))
55     BoundaryError = true;
56 elseif (q3_new < rad2deg(robot.getBody('link_3').Joint.
57 PositionLimits(1)) || q3_new > rad2deg(robot.getBody('link_3'.
58 ) .Joint.PositionLimits(2)))
59     BoundaryError = true;
60 elseif (q4_new < rad2deg(robot.getBody('link_4').Joint.
61 PositionLimits(1)) || q4_new > rad2deg(robot.getBody('link_4'.
62 ) .Joint.PositionLimits(2)))
63     BoundaryError = true;
64 elseif (q5_new < rad2deg(robot.getBody('link_5').Joint.
65 PositionLimits(1)) || q5_new > rad2deg(robot.getBody('link_5'.
66 ) .Joint.PositionLimits(2)))
67     BoundaryError = true;
68
69 if (BoundaryError == true)
70     disp('Boundary Error!')
71     disp(s_new)
72 end
73
74 f = s_new;
75 end
```

A.4 reward_3.m

```
1 function f = reward_3(s, s_new)
2 % Função que recebe o estado novo do sistema e retorna a
3 % recompensa associada, ou seja, a recompensa por ter tomado a ação no
4 % estado s_antigo de modo que s_novo = dynamics(s_antigo,a).
5
6 % Input: s - Estado em que o sistema se encontra.
7 % Input: s_new - Estado seguinte.
8 % Output: f - Recompensa associada à transição entre estado
9 % atual e
10 % seguinte
11 global setpoint;
12 global obstacle;
13 global winBonus;
14 global obstaclePenalty;
15 global JointBoundaryPenalty;
16 global terminate;
17 global BoundaryError;
18 global robot;
19
20 % Old State
21 q = deg2rad([s(1:5); 0]);
22 tform = getTransform(robot,q,'tool0','base');
23
24 x = tform(1,4);
25 y = tform(2,4);
26 z = tform(3,4);
27 p = [x;y;z];
28
29 % New State
30 q_new = deg2rad([s_new(1:5); 0]);
31 tform_new = getTransform(robot,q_new,'tool0','base');
32
33 x_new = tform_new(1,4);
34 y_new = tform_new(2,4);
35 z_new = tform_new(3,4);
```

```
36 p_new = [x_new;y_new;z_new];  
37  
38 %d_p_obs = distance(p,obstacle);  
39 %d_p_setp = distance(p,setpoint);  
40  
41 d_p_obs_new = distance(p_new,obstacle);  
42 d_p_setp_new = distance(p_new,setpoint);  
43  
44  
45 setpoint_bonus = 0;  
46 collisionPenalty = 0;  
47 obstacle_penalty = 0;  
48  
49 % Verify whether action violates joint boundaries.  
50 % _____  
51 if(BoundaryError == false) % Action ok: No Penalty  
52     boundaryPenalty = 0;  
53 else % Action violates boundaries: JointBoundaryPenalty  
54     terminate = true;  
55     boundaryPenalty = JointBoundaryPenalty;  
56 end  
57  
58 % Verify Table Collision  
59 % _____  
60 if (tableCollision(p_new) == true)  
61     collisionPenalty = collisionPenalty + TableCollisionPenalty;  
62     disp('Hit Table!')  
63     terminate = true;  
64 end  
65  
66 % Verify Setpoint Reached  
67 % _____  
68 if (d_p_setp_new < 0.08)  
69     setpoint_bonus = winBonus;  
70     disp('Reached Setpoint!')  
71     terminate = true;  
72 end  
73  
74 % Verify Obstacle Hit
```

```
75 % _____
76 if (d_p_obs_new < 0.14)
77     obstacle_penalty = obstaclePenalty;
78     disp('Hit Obstacle!')
79     terminate = true;
80 end
81
82 % Dot Product
83 if (p_new(1) == p(1) && p_new(2) == p(2) && p_new(3) == p(3))
84     r_setp = 0;
85     r_obs = 0;
86 else
87     vetor_p_p_new = p_new - p;
88     vetor_p_p_new = vetor_p_p_new/norm(vetor_p_p_new);
89
90     vetor_p_setp = setpoint - p;
91     vetor_p_setp = vetor_p_setp/norm(vetor_p_setp);
92
93     vetor_p_obs = obstacle - p;
94     vetor_p_obs = vetor_p_obs/norm(vetor_p_obs);
95
96     dot_product_setp = dot(vetor_p_p_new, vetor_p_setp);
97     r_setp = 2*dot_product_setp;
98
99     dot_product_obs = dot(vetor_p_p_new, vetor_p_obs);
100    if (d_p_obs_new < 0.40)
101        r_obs = -dot_product_obs;
102    else
103        r_obs = 0;
104    end
105 end
106
107 % Reward
108 f = (r_setp + 0.9*r_obs) + setpoint_bonus + obstacle_penalty +
109     boundaryPenalty + collisionPenalty;
110 end
```

ANEXO B – Arquivo URDF

B.1 kr16_2.urdf

```

1 <?xml version="1.0" ?>
2 <!--
=====
-->
3 <!-- | This document was autogenerated by xacro from kr16_2 .
      xacro | -->
4 <!-- | EDITING THIS FILE BY HAND IS NOT RECOMMENDED
      | -->
5 <!--
=====
-->
6 <!--Generates a urdf from the macro in kr16_2_macro.xacro -->
7 <robot name="kuka_kr16_2" xmlns:xacro="http://wiki.ros.org/xacro
      ">
8   <material name="Grey">
9     <color rgba="0.4 0.4 0.4 1.0"/>
10    </material>
11   <material name="Orange">
12     <color rgba="1.0 0.423529411765 0.0392156862745 1.0"/>
13    </material>
14   <!-- LINKS -->
15   <!-- base link -->
16   <link name="base_link">
17     <inertial>
18       <origin rpy="0 0 0" xyz="0 0 0"/>
19       <mass value="2"/>
20       <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyx="0" iyz="0" izz
          ="0.01"/>
21     </inertial>
22     <visual>
23       <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
24       <geometry>
25         <mesh filename="package://kuka_kr16_support/meshes/

```

```
        kr16_2/visual/base_link.dae"/>
26    </geometry>
27    <material name="Grey"/>
28    </visual>
29    <collision>
30        <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
31        <geometry>
32            <mesh filename="package://kuka_kr16_support/meshes/
33                kr16_2/collision/base_link.stl"/>
34        </geometry>
35    </collision>
36    </link>
37    <!-- link 1 (A1) -->
38    <link name="link_1">
39        <inertial>
40            <origin rpy="0 0 0" xyz="0 0 0"/>
41            <mass value="2"/>
42            <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyz="0" izz
43                ="0.01"/>
44        </inertial>
45        <visual>
46            <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
47            <geometry>
48                <mesh filename="package://kuka_kr16_support/meshes/
49                    kr16_2/visual/link_1.dae"/>
50            </geometry>
51            <material name="Orange"/>
52        </visual>
53        <collision>
54            <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
55            <geometry>
56                <mesh filename="package://kuka_kr16_support/meshes/
57                    kr16_2/collision/link_1.stl"/>
58            </geometry>
59        </collision>
60    </link>
61    <!-- link 2 -->
62    <link name="link_2">
63        <inertial>
```

```
60      <origin rpy="0 0 0" xyz="0 0 0"/>
61      <mass value="2"/>
62      <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyz="0" izz
63          ="0.01"/>
64    </inertial>
65    <visual>
66      <origin rpy="1.57079632679 -1.57079632679 0" xyz="0 0 0"/>
67      <geometry>
68        <mesh filename="package://kuka_kr16_support/meshes/
69            kr16_2/visual/link_2.dae"/>
70      </geometry>
71      <material name="Orange"/>
72    </visual>
73    <collision>
74      <origin rpy="1.57079632679 -1.57079632679 0" xyz="0 0 0"/>
75      <geometry>
76        <mesh filename="package://kuka_kr16_support/meshes/
77            kr16_2/collision/link_2.stl"/>
78      </geometry>
79    </collision>
80  </link>
81  <!-- link 3 -->
82  <link name="link_3">
83    <inertial>
84      <origin rpy="0 0 0" xyz="0 0 0"/>
85      <mass value="2"/>
86      <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyz="0" izz
87          ="0.01"/>
88    </inertial>
89    <visual>
90      <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
91      <geometry>
92        <mesh filename="package://kuka_kr16_support/meshes/
93            kr16_2/visual/link_3.dae"/>
```

```
94      <geometry>
95          <mesh filename="package://kuka_kr16_support/meshes/
96              kr16_2/collision/link_3.stl"/>
97      </geometry>
98  </link>
99  <!-- link 4 -->
100 <link name="link_4">
101     <inertial>
102         <origin rpy="0 0 0" xyz="0 0 0"/>
103         <mass value="2"/>
104         <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyz="0" izz
105             ="0.01"/>
106     </inertial>
107     <visual>
108         <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
109         <geometry>
110             <mesh filename="package://kuka_kr16_support/meshes/
111                 kr16_2/visual/link_4.dae"/>
112         </geometry>
113         <material name="Orange"/>
114     </visual>
115     <collision>
116         <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
117         <geometry>
118             <mesh filename="package://kuka_kr16_support/meshes/
119                 kr16_2/collision/link_4.stl"/>
120         </geometry>
121     </collision>
122  </link>
123  <!-- link 5 -->
124 <link name="link_5">
125     <inertial>
126         <origin rpy="0 0 0" xyz="0 0 0"/>
127         <mass value="2"/>
128         <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyz="0" izz
129             ="0.01"/>
130     </inertial>
131     <visual>
```

```
128      <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
129      <geometry>
130          <mesh filename="package://kuka_kr16_support/meshes/
131              kr16_2/visual/link_5.dae"/>
132      </geometry>
133      <material name="Orange"/>
134  </visual>
135  <collision>
136      <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
137      <geometry>
138          <mesh filename="package://kuka_kr16_support/meshes/
139              kr16_2/collision/link_5.stl"/>
140      </geometry>
141  </collision>
142  </link>
143  <!-- link 6 -->
144  <link name="link_6">
145      <inertial>
146          <origin rpy="0 0 0" xyz="0 0 0"/>
147          <mass value="2"/>
148          <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.01" iyz="0" izz
149              ="0.01"/>
150      </inertial>
151      <visual>
152          <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
153          <geometry>
154              <mesh filename="package://kuka_kr16_support/meshes/
155                  kr16_2/visual/link_6.dae"/>
156          </geometry>
157          <material name="Orange"/>
158  </visual>
159  <collision>
160      <origin rpy="1.57079632679 0 0" xyz="0 0 0"/>
161      <geometry>
162          <mesh filename="package://kuka_kr16_support/meshes/
163              kr16_2/collision/link_6.stl"/>
164      </geometry>
165  </collision>
166  </link>
```

```
162    <!-- tool 0 -->
163    <!-- This frame corresponds to the $FLANGE coordinate system
        in KUKA KRC controllers. -->
164    <link name="tool0"/>
165    <!-- END LINKS -->
166    <!-- JOINTS -->
167    <!-- joint 1 (A1) -->
168    <joint name="joint_a1" type="revolute">
169        <origin rpy="0 0 0" xyz="0 0 0.675"/>
170        <parent link="base_link"/>
171        <child link="link_1"/>
172        <axis xyz="0 0 -1"/>
173        <limit effort="0" lower="-3.22885911619" upper
174            ="3.22885911619" velocity="2.72271363311"/>
175    </joint>
176    <!-- joint 2 (A2) -->
177    <joint name="joint_a2" type="revolute">
178        <origin rpy="0 0 0" xyz="0.26 0 0"/>
179        <parent link="link_1"/>
180        <child link="link_2"/>
181        <axis xyz="0 1 0"/>
182        <limit effort="0" lower="-1.134464013796314" upper
183            ="2.181661564992912" velocity="2.72271363311"/>
184    </joint>
185    <!-- joint 3 (A3) -->
186    <joint name="joint_a3" type="revolute">
187        <origin rpy="0 0 0" xyz="0 0 0.68"/>
188        <parent link="link_2"/>
189        <child link="link_3"/>
190        <axis xyz="0 1 0"/>
191        <limit effort="0" lower="-3.839724354387525" upper
192            ="1.117010721276371" velocity="2.72271363311"/>
193    </joint>
194    <!-- joint 4 (A4) -->
195    <joint name="joint_a4" type="revolute">
196        <origin rpy="0 0 0" xyz="0.67 0 -0.035"/>
```

```
197      <limit effort="0" lower="-6.10865238198" upper  
198          ="6.10865238198" velocity="5.75958653158"/>  
199      </joint>  
200      <!-- joint 5 (A5) -->  
201      <joint name="joint_a5" type="revolute">  
202          <origin rpy="0 0 0" xyz="0 0 0"/>  
203          <parent link="link_4"/>  
204          <child link="link_5"/>  
205          <axis xyz="0 1 0"/>  
206          <limit effort="0" lower="-2.26892802759" upper  
207              ="2.26892802759" velocity="5.75958653158"/>  
208      </joint>  
209      <!-- joint 6 (A6) -->  
210      <joint name="joint_a6" type="revolute">  
211          <origin rpy="0 0 0" xyz="0 0 0"/>  
212          <parent link="link_5"/>  
213          <child link="link_6"/>  
214          <axis xyz="-1 0 0"/>  
215          <limit effort="0" lower="-6.10865238198" upper  
216              ="6.10865238198" velocity="10.7337748998"/>  
217      </joint>  
218      <!-- tool frame - fixed frame -->  
219      <joint name="joint_a6-tool0" type="fixed">  
220          <parent link="link_6"/>  
221          <child link="tool0"/>  
222          <origin rpy="0 1.57079632679 0" xyz="0.158 0 0"/>  
223      </joint>  
224      <!-- END JOINTS -->  
225      <!-- ROS base_link to KUKA $ROBROOT coordinate system  
226          transform -->  
227      <link name="base"/>  
228      <joint name="base_link-base" type="fixed">  
229          <origin rpy="0 0 0" xyz="0 0 0"/>  
230          <parent link="base_link"/>  
231          <child link="base"/>  
232      </joint>  
233  </robot>
```