GUILHERME HENRIQUE MARTINS DE OLIVEIRA
LUAN GUSTAVO DE BRITO

CONSTRUÇÃO DE FERRAMENTA PARA O ENSINO PRÁTICO DE APRENDIZADO POR REFORÇO EM ROBÔ MANIPULADOR

São Paulo
2022

GUILHERME HENRIQUE MARTINS DE OLIVEIRA
LUAN GUSTAVO DE BRITO

CONSTRUÇÃO DE FERRAMENTA PARA O ENSINO PRÁTICO DE APRENDIZADO POR REFORÇO EM ROBÔ MANIPULADOR

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

São Paulo
2022

GUILHERME HENRIQUE MARTINS DE OLIVEIRA
LUAN GUSTAVO DE BRITO

CONSTRUÇÃO DE FERRAMENTA PARA O ENSINO PRÁTICO DE APRENDIZADO POR REFORÇO EM ROBÔ MANIPULADOR

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

Orientador:
Larissa Driemeier

Co-orientador:
Thiago de Castro Martins

São Paulo
2022

AGRADECIMENTOS

Agradecemos imensamente a toda a comunidade desenvolvedora de software livre, este trabalho só foi possível graças ao trabalho deles.

RESUMO

A aplicação de Aprendizado por Reforço na robótica representa a evolução natural na programação e controle de robôs manipuladores. O emprego desta e outras técnicas de aprendizado de máquina permitiram que os robôs se tornassem capazes de se adaptar a situações imprevistas encontradas em seus ambientes. Devido a sua importância e ao potencial de desenvolvimento da área, o entendimento e a capacidade de desenvolver soluções no campo de aprendizado por reforço devem fazer parte do currículo do engenheiro moderno que deseja trabalhar com robótica. Este trabalho propõe o desenvolvimento de uma ferramenta didática para facilitar o ensino de técnicas de aprendizado por reforço na robótica. A ferramenta conta com um ambiente de simulação contendo um modelo do robô KUKA KR-16 e um modelo de câmera com visão computacional. Como teste é implementado o algoritmo DQN para tentar fazer com que o robô capture uma bola arremessada dentro de uma cesta acoplada ao seu punho.

Palavras-Chave – Aprendizado por Reforço, DQN, Robô Manipulador, MuJoCo, Simulação.

ABSTRACT

The application of Reinforcement Learning in robotics represents the natural evolution in the programming and control of robots manipulator. The use of this and other machine learning techniques allowed robots to become capable of adapting to unforeseen situations found in their environments. Due to its importance and the development potential of the area, being able to understand and having the ability to develop solutions in the field of reinforcement learning should be part of the curriculum of the modern engineer who wants to work with robotics. This work proposes the development of a didactic tool to facilitate the teaching of reinforcement learning techniques in robotics. The tool has a simulation environment containing a model of the KUKA KR-16 robot and a camera model with computer vision. As a test, the DQN algorithm is implemented to try to make the robot capture a ball thrown into a basket attached to its fist.

Keywords – Reinforcement Learning, DQN, Robot Manipulator , MuJoCo, Simulation.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

PARTE I

INTRODUÇÃO

1 INTRODUÇÃO

1.1 Contextualização e Apresentação do Tema

A robótica é um campo do conhecimento que saiu das páginas da ficção científica e se tornou realidade no ambiente industrial. Os robôs em suas mais diferentes formas estão presentes em indústrias, laboratórios e universidades ao redor do mundo. A inserção da robótica em uma escala cada vez maior com o passar dos anos levou à necessidade de refinamento e melhora das técnicas empregadas neste campo, seja para sensoriamento, atuação ou controle dos robôs.

O avanço da robótica no campo industrial possibilitou o surgimento de novas tecnologias e facilitou a padronização de peças e a produção em massa. A robótica é amplamente empregada desde a produção de alimentos até a produção de veículos. A sua maior inserção substituiu a mão-de-obra em tarefas que podem ser consideradas perigosas, garantindo maior segurança ao trabalhador. Além de sua utilização na produção industrial os robôs podem ser encontrados em ambientes domésticos, educacionais e hospitalares, sendo utilizados para realização de tarefas e até mesmo como forma de entretenimento. Devido a sua grande importância e em conjunto com a evolução tecnológica que permitiu o desenvolvimento de hardware e software mais sofisticados, novas técnicas de controle foram desenvolvidas utilizando elementos de Inteligência Artificial e Aprendizado de Máquina.

O controle baseado em Aprendizado por Reforço (AR) garante ao robô maior flexibilidade e variabilidade de comportamentos, pois a utilização destas técnicas permite ao robô aprender a realizar ações sem a necessidade de que estas sejam programadas anteriormente. Como explica Kormushev et al. [2013] o aprendizado acontece através de iterativas interações com o ambiente, utilizando uma função de recompensa como uma métrica de valor para um estado, de forma a recompensar ou punir o robô com base no resultado da ação e como ela é comparada à ação desejada

A utilização das técnicas de AR, apesar de inovadora, ainda encontra desafios na aplicação em problemas no mundo real, fora dos ambientes de simulação. De modo a

atingir um modelo que seja capaz de ser treinado e validado em um tempo que seja prático e viável, algumas simplificações devem ser implementadas, includindo muitas vezes a necessidade de se fornecer demonstrações de exemplo. A utilização de Aprendizado por Reforço Profundo (ARP) garante maior robustez ao modelo, pois permite que políticas utilizadas no algoritmo de AR sejam aprendidas sem a necessidade de demonstrações fornecidas pelo usuário.

Aprendizado por Reforço Profundo combina as técnicas de Aprendizado Profundo, nomeadamente, redes neurais artificiais profundas para análise e processamento de dados com a utilização de algoritmos de AR para o aprendizado do robô. A utilização das duas técnicas combinadas funciona bem para tarefas de robôs manipuladores, pois o processamento de dados com Aprendizado Profundo permite que o algoritmo de AR seja capaz de utilizar dados não estruturados, sem a necessidade de tratamento anterior ou simplificação do modelo.

1.2 Objetivos

O objetivo deste projeto consiste em desenvolver uma ferramenta didática de modo a facilitar a aprendizagem ativa do tema aprendizado por reforço para alunos de engenharia da escola politécnica.

A fim de criar um desafio lúdico e se aproveitar da estrutura já existente na Escola Politécnica da USP, propõe-se o seguinte projeto: Controlar o robô industrial KUKA KR16-2 instalado no Departamento de Engenharia Mecatrônica com o uso de aprendizado por reforço profundo, fazendo-o capturar uma bola de tênis lançada em sua direção. A detecção dos objetos será feita através da utilização de uma câmera Intel RealSense D435, valendo-se de algoritmos de visão computacional implementados neste projeto. O treinamento será feito em ambiente virtual com a utilização do software MuJoCo, com uma reprodução simplificada da sala onde o Robô manipulador se encontra.

Pretende-se, futuramente, utilizar as ferramentas aqui desenvolvidas no curso de graduação em engenharia mecatrônica da Escola Politécnica da USP para demonstrar aos estudantes o desenvolvimento prático de controle por aprendizado por reforço profundo, de modo que os futuros engenheiros possam estudar as técnicas empregadas no processo e tenham como visualizar e modificar a implementação.

1.3 Importância e Justificativa do Projeto

O emprego de algoritmos de AR para o controle de movimento de robôs manipuladores é vantajoso em relação à aplicação de técnicas de controle baseadas na programação de trajetórias, pois permite que o robô seja capaz de executar tarefas em que pré-programar sua trajetória não é possível ou não é viável, isso se torna evidente em aplicações as quais exigem correções em tempo real. O controle por AR pode ser usado, inclusive, em situações novas, de modo que o robô seja capaz de aprender como realizar uma ação lidando com valores de parâmetros anteriormente desconhecidos Kormushev et al. [2013] .

Estas técnicas permitiram uma grande disrupção na robótica, de modo que robôs puderam ser inseridos em ambientes anteriormente desconhecidos e ainda sendo capazes de explorá-los. Do ponto de vista de robôs manipuladores em ambientes industriais, como o KUKA, as técnicas de AR podem ser usadas para permitir aos robôs maior liberdade na execução de tarefas, pois podem usar dados fornecidos por sensores como câmeras com visão computacional e, com base em algoritmos de AR, aprender a lidar com perturbações no ambiente ou com a presença de elementos não considerados na programação elaborada *a priori*, como pessoas e obstáculos. Essas características garantem maior versatilidade no emprego destes robôs.

Além disso, dada a crescente aplicação de AR à robótica na industria e em produtos finais, é imprescindível que um engenheiro mecatrônico formado atualmente tenha familiaridade com as técnicas mais modernas de controle robótico e planejamento de trajetória. Dessa forma, a importância deste projeto se torna inequívoca ao criar a oportunidade de mais uma forma de aprendizagem ativa na graduação em engenharia mecatrônica da USP, permitindo que as futuras gerações se formem ainda mais capacitadas a lidar com desafios reais.

PARTE II

PESQUISA

2 ESTADO DA ARTE

Os robôs foram revolucionários na construção do mundo moderno. Seu surgimento gerou mudanças profundas em plantas industriais pelo mundo todo, entretanto o ambiente com o qual o robô fosse interagir necessitava de uma preparação para que tudo estivesse no devido lugar, sendo o comportamento do robô programado previamente. Esse tipo de solução tem pouca capacidade de reagir a perturbações que possam surgir durante os ciclos de atuação do robô. Segundo Zhu et al. [2021], com a evolução dos algoritmos e recursos computacionais, fez-se possível a utilização de sistemas de controle com realimentação nas instalações com robôs, obtendo-se a precisão necessária para o uso industrial em larga escala. Segundo Zamalloa et al. [2017], a robótica encontra-se na geração 4 de sua evolução, marcada pelo uso de algoritmos de aprendizado de máquina e inteligência artificial no controle e tomada de decisão dos robôs. Especialmente, o uso de Aprendizado por reforço na otimização e planejamento de rota dos robôs, em tempo real.

2.1 Aprendizado por reforço

Observando os trabalhos de Lillicrap et al. [2015] (apresenta um modelo de AR capaz de resolver 20 problemas físicos, incluindo equilíbrio de pêndulo invertido, caminhada bípede e quadrúpede 2D) e Mankowitz et al. [2019] (apresenta um modelos de AR treinados em ambiente propositalmente diferentes do ambiente final, sendo que o modelo é capaz de lidar com as duas situações) torna-se evidente que técnicas de aprendizado de máquina têm maior robustez, ou seja, capacidade de lidar com perturbações e varições no sistema real, que seus análogos feitos puramente com empenho humano e rotas e pré-definidas. Tendo isso em vista, é natural que tais técnicas sejam as mais utilizadas atualmente para determinação de rota e controle de robôs nas mais diversas aplicações. Vale ressaltar que a abrangência de seu uso é prova da capacidade de generalização e robustez do modelo de aprendizagem por reforço. Sendo alguns deles: aprendizado de caminhada bípede, em Kormushev et al. [2013], ou quadrúpede, em Shen et al. [2012], movimentos finos em mão robótica humanóide em Andrychowicz et al., OpenAI et al. [2019], robô manipulador

rebatendo uma bola com taco de baseball em Peters and Schaal [2008], encaixe de peças em planta industrial de montagem em Luo et al. [2021].

Entretanto, a alta capacidade de generalização do aprendizado por reforço vem com um custo alto, o número de tentativas que robô tem que fazer durante um treinamento é grande, da ordem de pelo menos centenas de rodadas, assim como mostrado em Kormushev et al. [2013]. É importante perceber também que apesar de avanços estarem sendo feitos na direção de treinamento no mundo real, como em Mahmood et al. [2018], existem inúmeros benefícios em utilizar um ambiente virtual de treinamento, sendo a opção mais comum atualmente. Além disso, diversos trabalhos demonstram que é possível fazer a transição do ambiente simulado para o mundo real de forma bastante acurada, como em Hundt et al. [2020], Peng et al. [2017], Schwab et al., Hu et al. [2021], Christiano et al. [2016], Zhu et al. [2021].

2.2 Aprendizado por reforço profundo

Com o acesso a placas gráficas mais potentes e a imensos bancos de dados para treinamento, as redes neurais convolucionais (RNC) revolucionaram o mundo da inteligencia artificial. Com sua capacidade incrível de generalização se tornou a forma mais utilizada de se fazer reconhecimento e classificação de imagens. Da mesma forma, as RNCs criaram um novo marco no mundo do aprendizado por reforço: o surgimento do Aprendizado por Reforço Profundo (ARP). A primeira vez que o ARP apareceu como grande revolução foi em 2013 no trabalho Mnih et al., unindo o reconhecimento de imagens poderoso das RNCs com a interação com o ambiente trazida do aprendizado por reforço, o modelo criado foi capaz de aprender a jogar 7 jogos diferentes de Atari.

Um ponto crucial no desenvolvimento do ARP foi a não utilização de camadas de pooling nas Redes que se combinam com o aprendizado por reforço. Isso é importante pois as camadas de pooling subtraem a informação do posicionamento do elemento das próximas camadas, recurso útil na generalização para reconhecimento de imagens, mas infeliz quando a disposição dos elementos na imagem importa. Dessa forma, faz-se possível treinar o algoritmo de AR sem a necessidade de simulação gráfica, utilizando coordenadas absolutas, que serão alimentadas para o algoritmo posteriormente pela rede neural, como feito nos trabalhos de Andrychowicz et al., OpenAI et al. [2019].

O uso de aprendizado por reforço profundo na robótica têm sido cada vez mais comum, possibilitando aprendizado simultâneo de visão e trajetória, como em Schwab et al.,

Kalashnikov et al. [2018], ou ainda, como apresentado por Vecerik et al. [2020], Jeong et al. [2020] o uso de aprendizado auto-supervisionado. Com isso é possível fazer ainda mais com menos esforço do desenvolvedor, por isso usos de ARP tem sido tão frequentes.

3 FUNDAMENTOS TEÓRICOS

Nesta seção são apresentados os principais fundamentos teóricos por trás das técnicas aplicadas no desenvolvimento deste trabalho. São discutidas a teoria geral por trás do Aprendizado por Reforço e a explicação dos algoritmos utilizados, além de outros aspectos cuja aplicação se mostrou presente.

Em seguida, na seção sobre cinemática de manipuladores é apresentada a teoria básica de cinemática, incluindo a introdução sobre os parâmetros de Denavit-Hartenberg, a derivação das matrizes de transformação de coordenadas do robô e a apresentação do modelo cinemático do KUKA KR-16.

3.1 Aprendizado por reforço

O Aprendizado por reforço é um algoritmo de aprendizado de máquina que busca modelar o processo de aprendizado humano através da interação com o ambiente. Em diversas ocasiões o ser humano aprende sem a orientação de um mentor ou referências externas, o processo ocorre conforme a pessoa interage com o ambiente ao seu redor e reage aos efeitos de suas ações em um processo de tentativa e erro.

As técnicas de Aprendizado por Reforço buscam imitar esse tipo de comportamento utilizando um sistema de recompensa para estimular as ações desejadas e punir os comportamentos indesejados. Os componentes que fazem parte de AR são: O ambiente, sobre o qual serão realizadas as ações;as ações em si; o agente, que é o responsável por realizar as ações e aprender o comportamento desejado; a recompensa, uma função que estimula o comportamento do agente de modo a realizar o aprendizado.

A Figura 1 representa o modelo básico de Aprendizado de Reforço. Na imagem, a ação **a** é realizada pelo agente **B** sobre o ambiente **T**. **s** é a representação do estado do sistema, que é utilizado pelo agente através de duas funções: **i**, que representa a percepção do estado do sistema pelo agente e **r**, que representa a recompensa associada ao estado e

à ação que foi tomada.

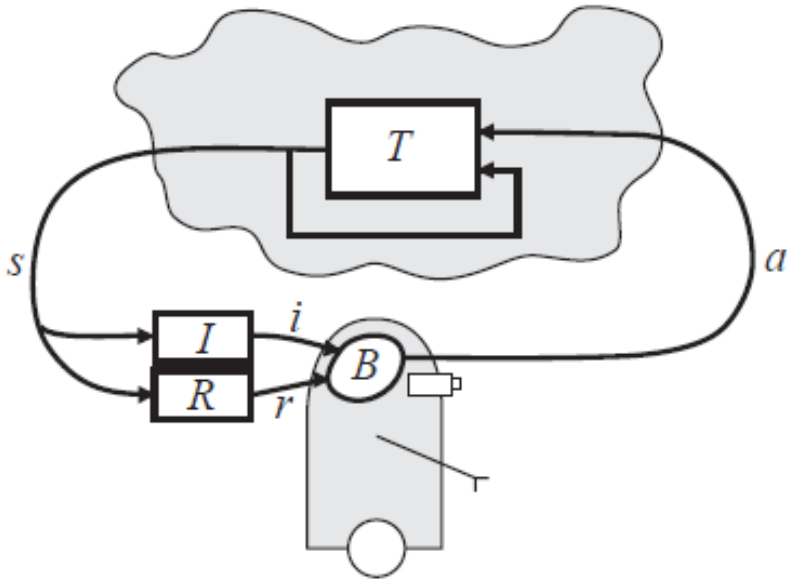


Figura 1: Representação de modelo de Aprendizado por Reforço (KAELBLING, 1996)

3.1.1 Processo de Decisão de Markov

O problema desenvolvido pode ser caracterizado como um modelo de recompensa adiada, Delayed Reward. Problemas deste tipo levam em consideração a otimização a longo prazo, ou seja, a recompensa considera também os estados e ações futuras tomadas pelo agente. Essa classe de problemas pode ser modelada como um processo de decisão de Markov.

Um processo de decisão de Markov consiste de um espaço de estados $\mathcal{S}$, um espaço de ações $\mathcal{A}$, uma função de recompensa $\mathcal{R} : \mathcal{S}X\mathcal{A}\rightarrow\mathcal{R}$ e uma função de transição de estados $\mathcal{T} : \mathcal{S}X\mathcal{A}\rightarrow\Pi(\mathcal{S})$. A função $\mathcal{T}$ denota a probabilidade de se chegar ao estado s' tomando a ação a no estado s. Como o problema desenvolvido é determinístico, ao tomar a ação a no estado s, o mesmo estado s' é alcançado, a função de recompensa torna-se função somente do estado atual e da ação tomada, o que simplifica a convergência do algoritmo.

3.1.2 Política de Ações

Em um modelo de Aprendizado por Reforço, a Política de Ações denota uma função que associa a ação a ser tomada em função do estado do sistema de modo a optimizar o comportamento do agente. Existem diferentes modos de optimizar o comportamento

do agente, levando em consideração as possíveis ações a serem tomadas. Existem três modelos que são normalmente utilizados em trabalhos na área de AR:

$$E(\sum_{t=0}^{\infty} r_t) \tag{3.1}$$

$$E(\sum_{t=0}^{\infty} \gamma^t r_t) \tag{3.2}$$

$$\lim_{h\to\infty} E(\frac{1}{h}\sum_{t=0}^{h} \gamma^t r_t) \tag{3.3}$$

O primeiro modelo é chamado finite horizon model é o mais simples de ser analisado. Neste modelo o valor esperado da soma das recompensas até um instante de tempo T é a função a ser otimizada.

O segundo modelo, infinite horizon model, implementa um fator γ, que considera as futuras ações a serem tomadas. As ações futuras são submetidas a um fator gama cujo valor varia entre zero e um, de modo que sua influência sofre um desconto.

O último modelo de otimização é average-reward model, que considera a média do valor esperado da soma das recompensas, desta forma o agente não precisa levar em conta o comportamento futuro.

Utilizando o modelo infinite-horizon discounted model podemos estabelecer técnicas para encontrar a política de ações ótima. Inicialmente definimos a função valor dos estados V. Esta função associa um valor a um estado a partir do valor esperado da soma das recompensas associadas ao estado atual s e assumindo que o agente segue uma política de ações π.

$$V^*(s) = max_\pi E(\sum_{t=0}^{\infty} \gamma^t r_t) \tag{3.4}$$

A otimização desta função pode ser feita de maneira recursiva utilizando-se a função recompensa e a função de transição dos estados. A patir da função otimizada, podemos definir a política ótima.

$$V^*(s) = max_a(\mathcal{R}(s,a) + \gamma\sum_{s'\in\mathcal{S}} V^*(s')), \forall s\in\mathcal{S} \tag{3.5}$$

$$\pi^*(s) = argmax_a(\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} V^*(s')) \tag{3.6}$$

Uma outra função pode ser definida para encontrar a política ótima, utilizando o estado atual uma política π e a ação tomada pelo agente a. Trata-se da Função Valor das Ações Q.

$$Q_\pi(s,a) = E_\pi(\sum_{t=0}^{\infty} \gamma^t r_t) \tag{3.7}$$

3.1.3 Value Iteration

Da seção anterior pode-se concluir que o problema de encontrar a política ótima pode ser definido como um problema iterativo utilizando o valor da função estado V ou da função valor das ações Q. Em algoritmos desta categoria, os valores de Q ou V são iniciados arbitrariamente e, então, através da iteração sobre os valores de estados e ações são encontradas as funções que maximizam V ou Q e, a partir destas, encontra-se a política ótima π^*

3.1.4 Q-Learning

Um dos exemplos de algoritmos do tipo Value Iteraction é o Q-Learning. O Q-Learning foi desenvolvido em 1989 por Watkins. O funcionamento do algoritmo pode ser descrito da seguinte forma: inicializa-se Q para todos os possíveis valores de estados e ações e define-se o valor de Q para estados terminais como zero; então para cada passo dos episódios é escolhido um valor de A segundo a política de ações e o valor de Q é atualizado através da equação de Bellman. O processo se repete de maneira iterativa até que um estado terminal seja alcançado.

$$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(\mathcal{R}_{t+1} + \gamma max_{a \in \mathcal{A}} Q(s_{t+1},a_t) \tag{3.8}$$

Um aspecto importante do algoritmo Q-Learning é que ele converge a valores ótimos de maneira independente ao comportamento do agente durante a etapa de coleta dos dados. Ou seja, independente de como é feita a exploração.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figura 2: Algoritmo Q-Learning (SUTTON, 2017).

3.1.5 ”Exploration” contra ”Exploitation”

Em primeiro lugar, faz-se necessário desculpar-se com o leitor pois os dois termos acima, amplamente utilizados na literatura especializada, têm a mesma tradução em língua portuguesa. Visto que o objetivo desta seção é colocá-los em uma posição antagônica, faz-se o uso dos termos em língua inglesa.

No aprendizado por reforço, o agente deve explorar o ambiente, de modo que possa aprender com o maior número de ações. No entanto, ao descobrir a ação que melhor otimiza o comportamento do agente, é esperado que o agente tome esta ação. Esta é a ideia por trás da escolha entre *exploration*, ou seja, explorar o ambiente para obter mais referências sobre seu funcionamento e *explotation*, utilizar da melhor estratégia já encontrada para maximizar o resultado final .

O cenário ideal é aquele em que o agente explora as ações possíveis de modo a obter aquela que maximiza a recompensa e passa a utilizá-la de modo a obter o comportamento ótimo. No entanto há alguns obstáculos que dificultam isso, impossibilitando o agente de atingir o melhor resultado, como a limitação do período de treino ou a existência de máximos locais na função de recompensa. Além disso, em casos não determinísticos, não é possível obter a ação que maximiza a recompensa, somente estimar a probabilidade desta.

3.1.6 ϵ-Greedy

Uma das estratégias adotadas para lidar com o problema de *exploration* e *exploitation* é o algoritmo ϵ-greedy. Este se refere a inserir um parâmetro ϵ que define uma probabilidade em que o agente não toma a ação de máximo valor, mas uma ação aleatória. Para replicar o cenário ideal descrito acima, é prática comum definir um decaimento de ϵ até

um determinado ϵ_{min} próximo de 0 e positivo.

O algoritmo determina $\epsilon \mid 0 < \epsilon < 1$, que decai ao longo do treino. Inicia-se o parâmetro $\epsilon = \epsilon_{max}$, sendo ϵ_{max} próximo de 1; utiliza-se uma variável aleatória de distribuição uniforme com valores entre (0,1) de modo que se o valor desta variável for maior que ϵ, é tomada a ação que maximiza o valor da função Q em um determinado estado. Caso o valor da variável aleatória seja menor que ϵ, a ação é escolhida aleatoriamente. Este algoritmo tenta aumentar a etapa de *exploration* no início do algoritmo e, após a aquisição de informações suficientes, prioriza-se aquelas que otimizam o comportamento desejado.

Matematicamente, pode-se representar a função ϵ-greedy da seguinte forma:

$$a_t = \begin{cases} argmax(Q(s_t, a')) \quad | \quad a' \in \mathcal{A}, & \text{se } rand(0,1) > \epsilon \\ a \text{ aleatória} \quad | \quad a \in \mathcal{A}, & \text{caso contrário} \end{cases} \tag{3.9}$$

Note que a técnica de decaimento de ϵ não é descrita pelo algoritmo, entretanto comumente se usa um decaimento exponencial controlado por um parâmetro $k \mid 0 < k < 1$, com k muito próximo de 1, do tipo:

$$\epsilon_i = \begin{cases} \epsilon_{max} \cdot k^i, & \text{se } \epsilon_{max} \cdot k^i > \epsilon_{min} \\ \epsilon_{min}, & \text{caso contrário} \end{cases} \tag{3.10}$$

Esta foi a abordagem escolhida para este trabalho, note porém que o modelo de decaimento em si é mais um hiper-parâmetro do modelo de aprendizado.

3.2 Deep Q-Learning

O algoritmo Deep Q-Learning ou Deep Q-Network (DQN) funciona com uma junção de técnicas de Aprendizado Profundo e Aprendizado por Reforço, ou seja, trata-se de uma técnica de Aprendizado por Reforço Profundo. O DQN funciona de modo similar ao algoritmo Q-Learning, no entanto a função Q é aproximada por uma rede neural artificial ao invés de uma tabela. Sendo mais indicada para casos onde o espaço de estados e/ou o espaço de ações é maior, pois nestes casos o Q-Learning não é viável dada a quantidade de memória alocada necessária para armazenar a tabela Q. Um campo onde o DQN é muito utilizado é nos casos em que a entrada do modelo consiste de imagens, sendo utilizadas redes neurais convolucionais para aproximar a função Q.

No algoritmo DQN, o responsável por decidir as ações a serem tomadas é uma rede neural, que recebe como entrada o estado do ambiente. A função valor das ações, neste caso, é dada em função dos pesos θ da rede neural e é treinada de modo iterativo conforme a equação de Bellman:

$$Q_\theta(s_t, a_t) = \mathcal{R}(s_t, a_t) + \gamma \cdot max_{a' \in \mathcal{A}} Q_\theta(s_{t+1}, a') \tag{3.11}$$

Apesar de sua indicação para casos de maior complexidade, o algoritmo DQN é um algoritmo bastante instável e sua aplicação é comumente associada à técnicas que buscam melhorar a convergência e facilitar o treinamento. Algumas das principais técnicas adotadas são ϵ-greedy para lidar com o *tradeoff* entre *exploration* e *exploitation*, utilização de memória de experiências (*experience buffer*), treinamento em mini-lotes (*minibatch*) e rede-alvo (*target-network*).

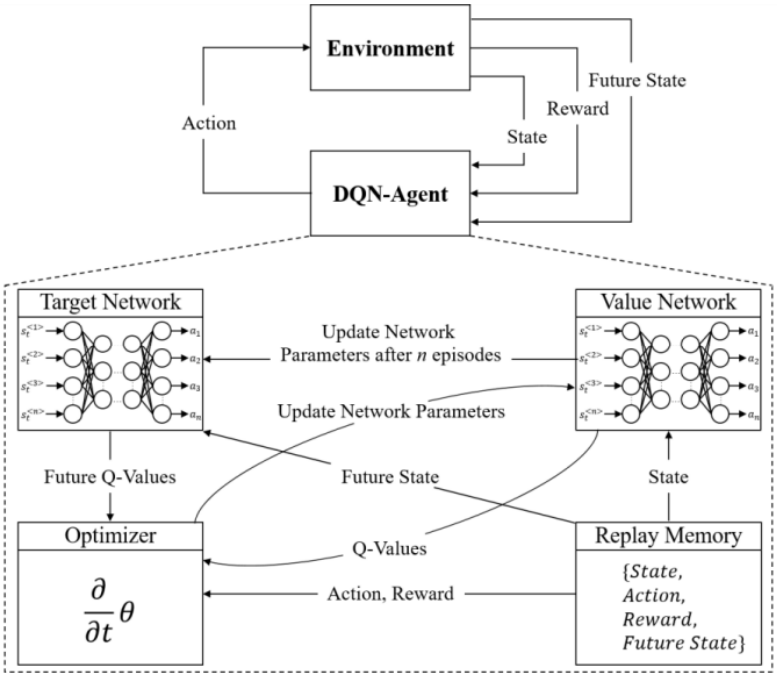


Figura 3: Modelo de utilização do DQN (Lang et al. 2020).

3.2.1 Memória de experiências e treinamento em mini-lotes

A memória de experiencias é uma técnica utilizada para melhorar a convergência em aprendizado por reforço. Esta técnica utiliza um *buffer* de memória onde são armazenados um número fixos de tuplas de transição do tipo $(s_t, a_t, r_t, s_{t+1}, done)$ que contém os seguintes elementos: s_t, denota o estado atual; a_t, denota a ação tomada; r_t, denota a recompensa associada ao par (s_t, a_t); s_{t+1}, denota o próximo estado, sendo uma fun-

ção de (s_t, a_t) por se tratar de um processo de decisão de Markov e done é uma variável booleana que denota se um estado é terminal ou não. A armazenagem dos valores de transição ocorre antes do treino e, na etapa de treinamento, estes valores são atualizados na memória a cada passo. Quando esta técnica é utilizada sozinha, a cada passo a rede é treinada com toda a extensão do *buffer*, entretanto é incomum vê-la sem a associação com treinamento em mini-lotes.

O treinamento em mini-lotes se refere a técnica de utilizar uma amostragem aleatória, de tamanho fixo, do *buffer* de experiências ao treinar a rede neural. Comumente os mini-lotes variam em tamanho de algumas dezenas até poucas centena em numero de elementos. O valor padrão utilizado na literatura é de 32 experiências, entretanto Stooke and Abbeel [2018] encontrou benefícios em utilizar mini-lotes de até 512 elementos.

Os motivos da utilização dos mini-lotes melhorar o modelo ainda são questionados na academia, mas acredita-se que uma amostragem pequena insere mais ruído estatístico no modelo, aumentando sua robustez, com a desvantagem de treinar mais lentamente.

3.2.2 Rede Alvo

A utilização da técnica de rede alvo tem o objetivo de aumentar a estabilização do treinamento da rede neural. Esta técnica consiste em utilizar um cópia da rede neural utilizada para calcular a parcela $Q_\theta(s_{t+1}, a')$ da equação de Bellman. Ou seja, a rede alvo calcula os valores futuros de Q, que são então utilizados pela rede neural principal para treinamento. Uma distinção importante é o fato de que a rede alvo não é treinada, seus parâmetros são atualizados com base na rede principal de maneira periódica, a cada n episódios com base no parâmetro τ que define qual peso na média ponderada os valores da rede de valor terão sobre a rede de alvo. A figura 4 demonstra o funcionamento de um algoritmo DQN utilizando rede alvo.

3.3 Cinemática de robôs manipuladores

A cinemática é o ramo da física que estuda o movimento dos corpos de forma a caracterizá-los sem se preocupar em descrever a sua causa, como o efeito de forças e momentos. No estudo de robôs manipuladores como o KUKA, a cinemática caracteriza o movimento das articulações do robô através de suas velocidades e posições.

A cinemática de robôs manipuladores pode ser dividida em cinemática direta e cinemática inversa. A cinemática direta tem como objetivo obter parâmetros do efetuador

robô como velocidade e posição das articulações em relação aos sistema de coordenadas da base. Já na cinemática inversa a posição do efetuador é conhecida e o objetivo é calcular as posições das articulações de modo a permitir que o efetuador esteja na posição desejada.

O robô KUKA KR-16 conta com seis graus de liberdade, através de suas seis articulações rotativas, representadas por A1-A6 na figura 3. As três primeiras articulações A1-A3 tem a função de posicionar o efetuador, enquanto as três últimas são utilizadas para realizar sua orientação.

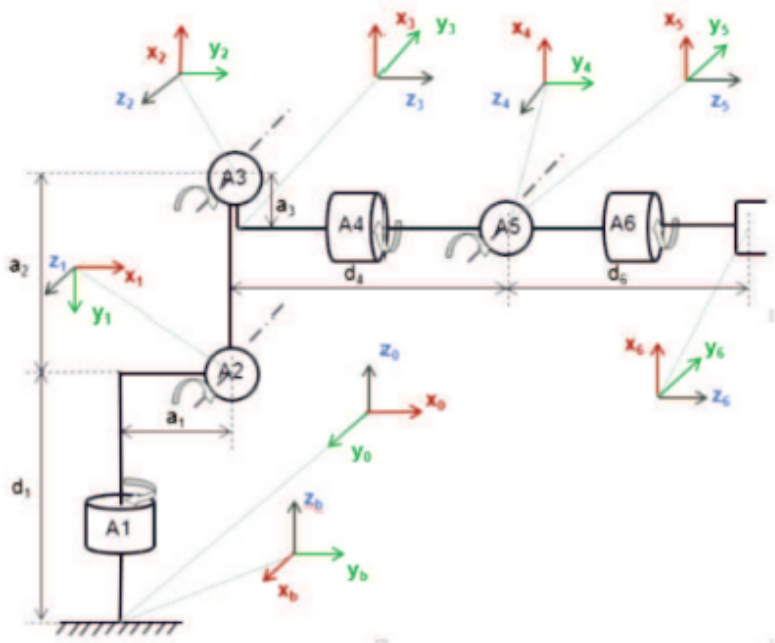


Figura 4: Representação de modelo da cadeia cinemática do KUKA KR-16 segundo D-H (PIOTROWSKI N; BARYLSKI A, 2014)

3.3.1 Parâmetros de Denavit-Hartenberg

Os parâmetros de Denavit-Hartenberg são utilizados para descrever a cadeia cinemática completa de um robô. Estes parâmetros são utilizados de forma a definir um elo, ou seja, "um corpo rígido que define a relação entre eixos de duas juntas vizinhas de um manipulador" (CRAIG, 2012), e a sua relação com elos vizinhos.

O parâmetro (a) representa o comprimento do elo. O parâmetro α representa a torção do elo. O parâmetro d representa o deslocamento do elo, ou distância entre elos. O parâmetro θ representa o ângulo entre juntas. A figura 3 mostra a relação desses parâmetros entre dois eixos i-1 e i.

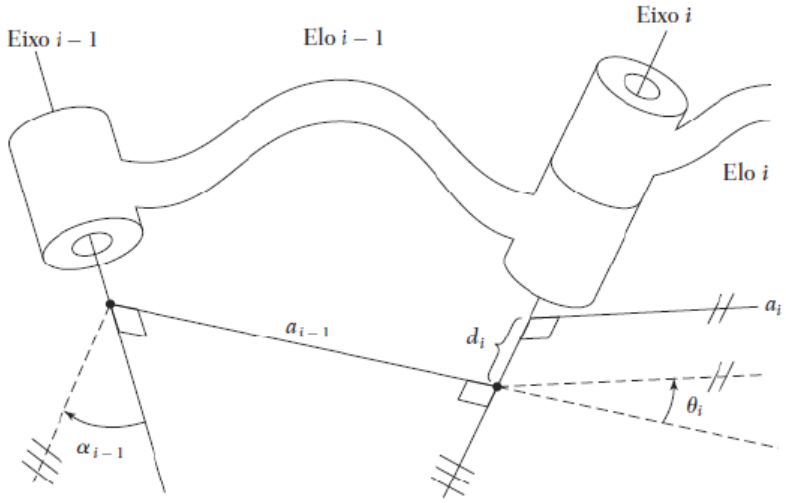


Figura 5: Notação de Denavit-Hartenberg (CRAIG, 2012)

O procedimento para obter a matriz de transformação entre elos utilizando os parâmetros D-H é o seguinte:

Posicionar a origem do sistema de referência i no cruzamento entre a perpendicular ai e o eixo da junta i. O eixo Z é posicionado de modo a coincidir com o eixo da junta i. O eixo X é definido como a direção normal ao plano formado entre Z_i e Z_{i+1}. O eixo Y, por fim, é obtido a partir da aplicação da regra da mão direita utilizando os eixos X e Y.

A matriz obtida utilizando os parâmetros D-H e o procedimento descrito anteriormente é dada a seguir:

$$A_{i-1}^{i} = \begin{bmatrix} \cos\theta_i & -sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -sin\alpha_{i-1} & -sin\alpha_{i-1} d_i \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

3.3.2 Modelo Cinemático do KUKA KR-16

Utilizando as medidas do robô KUKA obtidas do manual do fabricante e os parâmetros D-H calculados na seção anterior e mostrados na tabela 1 é possível calcular a matriz de transformação de coordenadas através da cinemática direta. A matriz A_0^6 representa a transformação de coordenadas entre o sistema de coordenadas da base do robô e o sistema de coordenadas do efetuador.

$$A_0^6 = A_0^1 A_1^2 A_2^3 A_3^4 A_4^5 A_5^6 \tag{3.13}$$

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	$d_6 = 115$	0	0	-350	350

Tabela 1: Parâmetros de Denavit-Hartenberg do robô KUKA KR-16

PARTE III

DETALHAMENTO DE PROJETO

4 REQUISITOS DE PROJETO

4.1 Requisitos

O objetivo deste trabalho é desenvolver uma ferramenta didática para permitir a aprendizagem ativa em técnicas de controle robótico com aprendizado por reforço. Para tanto definiu-se que o escopo deste trabalho se limita ao desenvolvimento de uma estrutura capaz de treinar o robô KUKA em ambiente virtual. Para alcançar este objetivo foram definidos os seguintes requisitos de projeto:

- A implementação do algoritmo em robô KUKA real deve ser viável sem que haja alterações dos resultados obtidos por simulação;
- O robô deve ser capaz de capturar as bolas de tênis arremessadas contra ele com uma taxa de sucesso superior a 95%;
- O usuário deverá poder testar diferentes parâmetros de treinamento sem editar o programa;
- Todo o código deve ser bem documentado e explicado para os futuros usuários da ferramenta;
- O sistema deve funcionar em tempo real.

4.2 Efetuador Final

Com o objetivo de capturar a bolinha, foi necessário desenvolver uma ferramenta a ser fixada na ponta do braço robótico KR16. Dessa forma, projetou-se uma cesta, a ser impressa em PLA por uma impressora 3D de filamento. Nesta seção apresenta-se a documentação de projeto da cesta, que é a única parte física do trabalho.

4.2.1 Fixação

Segundo o manual de especificações do robô KUKA KR16-2, no último elo existem 8 orifícios com rosca métrica para parafuso M6, sendo assim modelou-se uma estrutura capaz de ser fixada com estas condições. Entretanto dada a baixa carga sobre a peça apenas 3 orifícios serão utilizados.

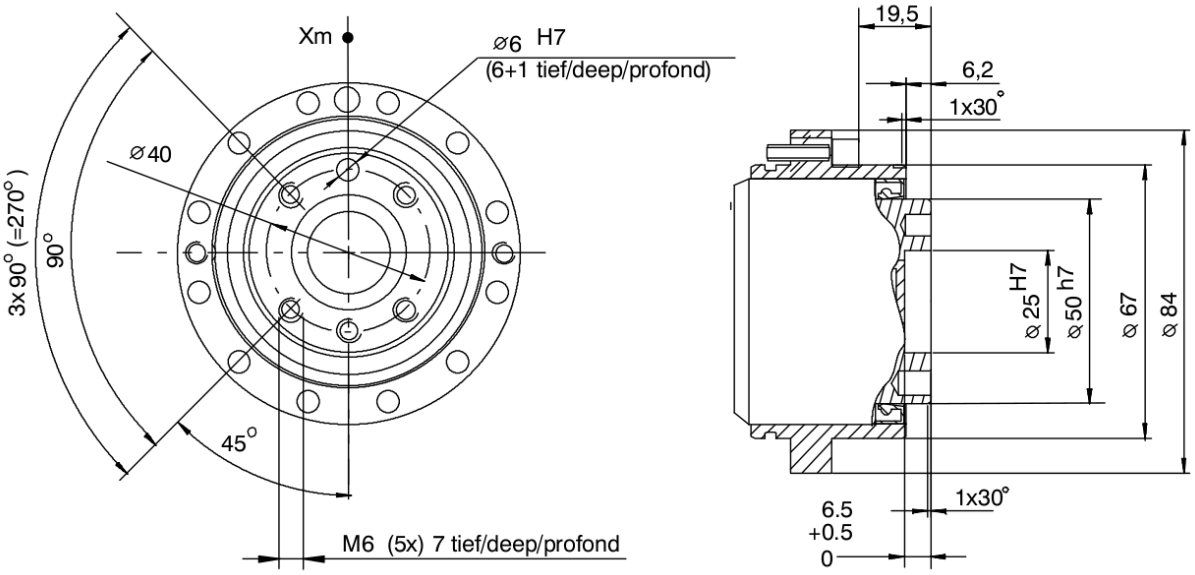


Figura 6: Posicionamento dos orifícios de fixação do robô. (KUKA KR16 Specification)

4.2.2 Geometria

A dimensão mais importante da cesta é o diâmetro da boca, essa dimensão está intimamente ligada com a dificuldade do problema final. Por esse motivo foi feita uma cesta com dimensões generosas, com 140mm de boca. Além disso, precisava-se de uma altura que não deixasse a bola escapar quicando.

Optou-se por fazer a cesta com orifícios para facilitar a visualização da bola dentro dela. Pode ser considerada uma opção estética

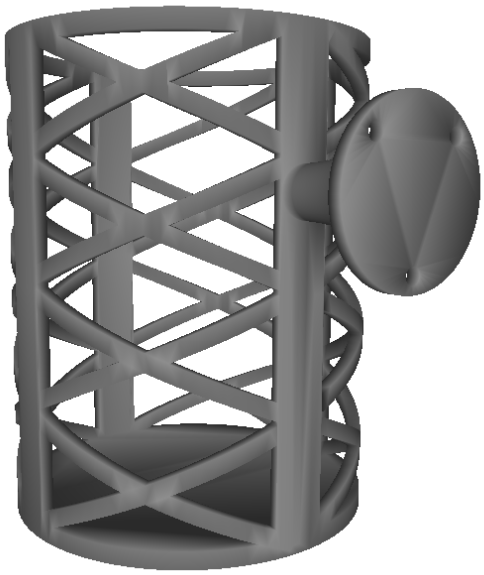
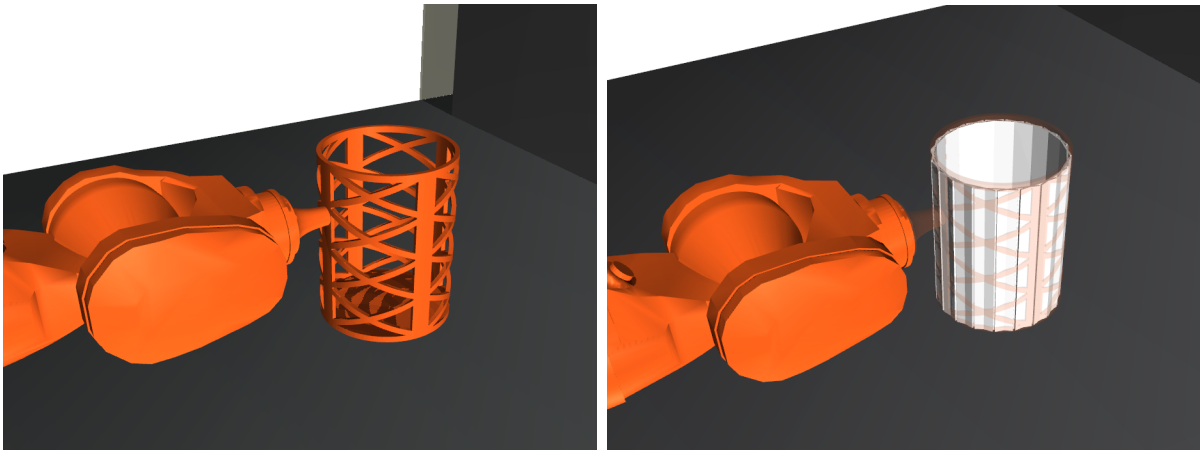


Figura 7: Cesta modelada em 3D. (Fonte: Autores)

Para mais detalhes geométricos vide apêndice, lá está anexado em desenho técnico detalhado da cesta coletora.

4.2.3 Representação no ambiente virtual

A cesta no ambiente virtual é uma entidade apenas renderizada, não sendo levada em conta nas verificações de colisão. Visto que o ambiente aproxima os corpos para uma versão convexa deles para cálculo de colisão e uma cesta é por definição côncava não pudemos utilizar o modelo .stl da cesta diretamente como objeto físico. Para solucionar este problema contornamos a cesta com 20 caixas de espessura desprezível que aproximam bem o cilindro principal da cesta e ainda um cilindro de espessura desprezível utilizado como tampa inferior.



(a) Modelo visual da cesta. (b) Modelo de colisão da cesta

Figura 8: Representação da cesta no ambiente virtual.(Fonte: Autores)

5 AMBIENTE DE SIMULAÇÃO

5.1 KUKA KR16-2

A representação do robô no ambiente virtual se apresentou como um grande desafio em si, em especial quando à dinâmica do sistema, visto que a KUKA não disponibiliza modelos de distribuição de massa e tampouco curvas de torque e reduções em seus motores. Por esses motivos o escopo do projeto foi limitado à etapa de planejamento de trajetória e não à etapa de controle dinâmico do sistema.

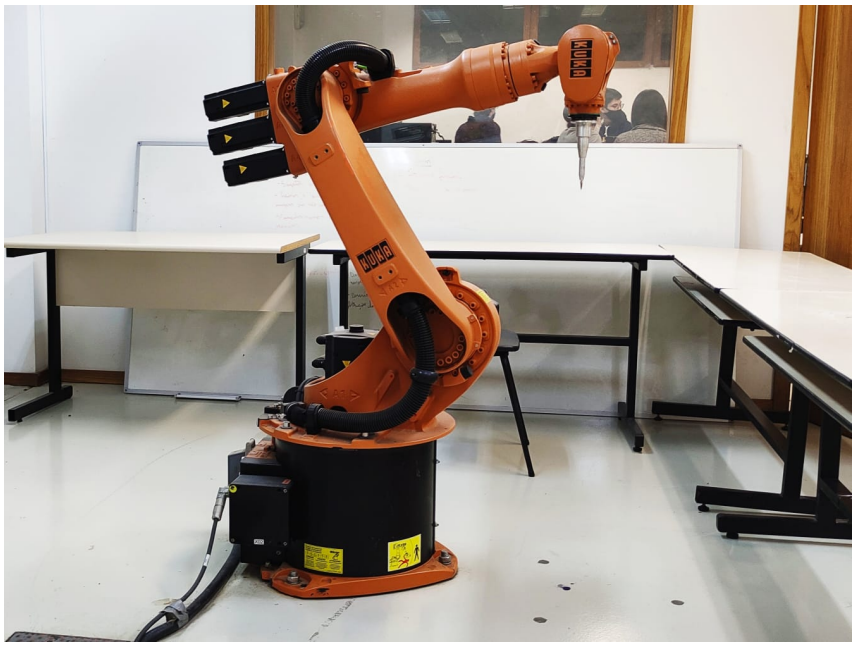


Figura 9: KUKA KR16-2 instalado na Escola Politécnica da USP (Fonte: Autores)

5.1.1 Geometria

A fim de simular corretamente as posições e colisões do robô é preciso ter um modelo tridimensional deste. A fabricante disponibiliza arquivos CAD .step e desenhos técnicos detalhados da geometria do robô via página de suporte do cliente. Entretanto, optou-se por utilizar os modelos disponíveis no repositório KUKA experimental do projeto ROS,

visto que as posições das juntas e seus limites cinemáticos também se encontram determinados no arquivo URDF (Arquivo Universal de descrição de robô) nele disponibilizado. Além disso, por se tratar de um projeto independente da fabricante, tomou-se o cuidado de validar cada uma das dimensões principais do modelo tomando como referência o manual fornecido pela KUKA.

O ambiente Gazebo utiliza modelos diferentes para renderização e visualização, sendo o primeiro um modelo geométrico detalhado no formato .dae e o segundo sendo um modelo simplificado (frequentemente convexo) do primeiro em formato .stl. O MuJoCo, ambiente escolhido para simulação e renderização neste trabalho utiliza apenas um modelo para ambos objetivos e em formato .stl, dessa forma foi-se necessário converter os arquivos detalhados .dae para .stl, para tanto utilizamos o software CAD AutoDesk Inventor.

Em seguida pode-se observar tanto as informações retiradas do manual do KUKA KR16 quanto os modelos tridimensionais encontrados.

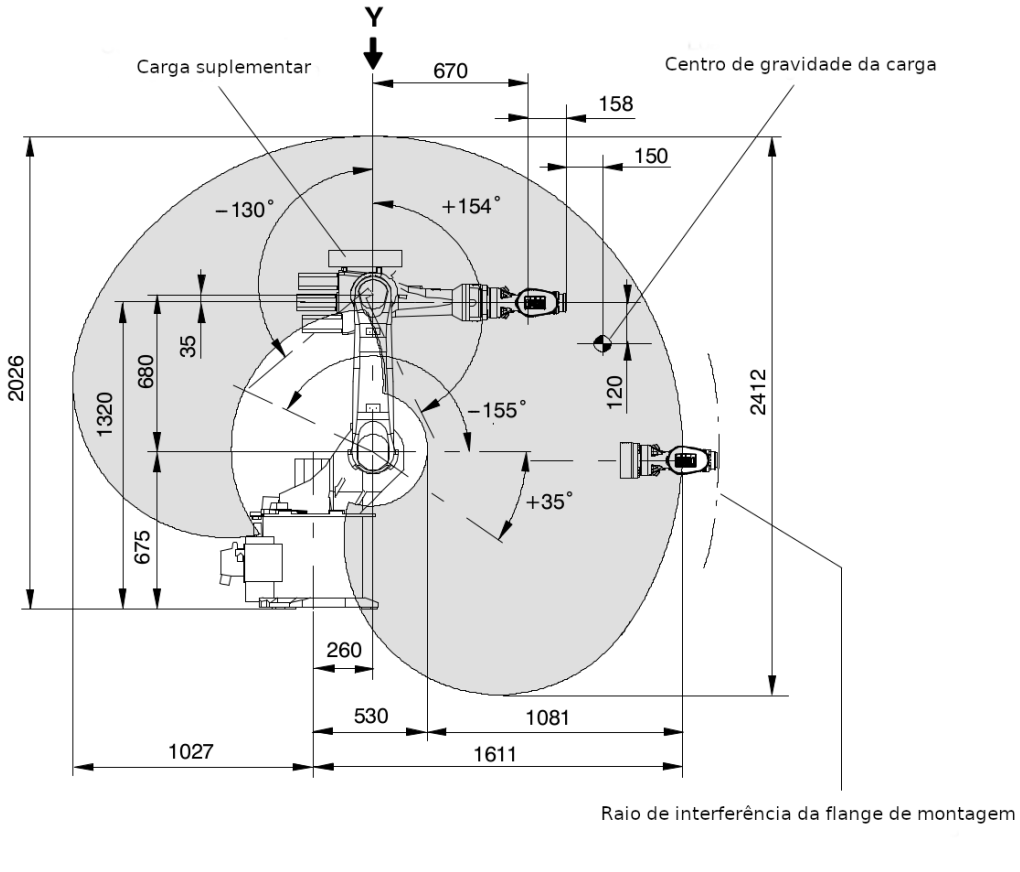
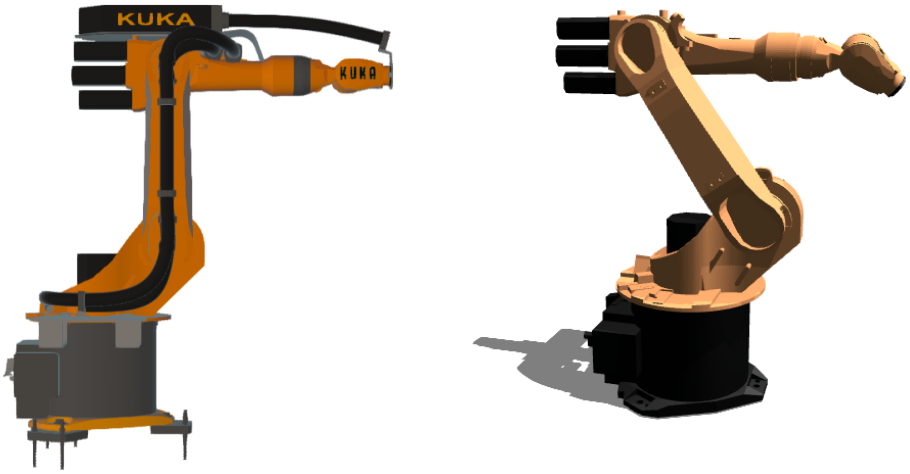


Figura 10: Dimensões principais e envelope de trabalho. (KUKA KR16 Specification - adaptação)



(a) Modelo da da fabricante (b) modelo do projeto ROS

Figura 11: Modelos 3D encontrados.(Fonte: Autores)

5.1.2 Cinemática

Visto que utilizamos o arquivo URDF como referência para o modelo do robô obtivemos simultaneamente os limites de cada junta de revolução e ainda suas respectivas velocidades máximas. Aqui novamente esses valores foram validados com as informações fornecidas pela fabricante.

Além disso, outra conversão de arquivos se faz necessária aqui, o ambiente MuJoCo é compatível com arquivos URDF, entretanto para ter acesso a todos os recursos que o ambiente proporciona é altamente recomendável representar o robô em um arquivo XML, seguindo a estrutura que a documentação da ferramenta chama de MJCF. Foi utilizado um tradutor automático, disponível no repositório URDF2MJCF de Eric Heiden para converter os arquivos.

Seguem abaixo as informações encontradas nos manuais:

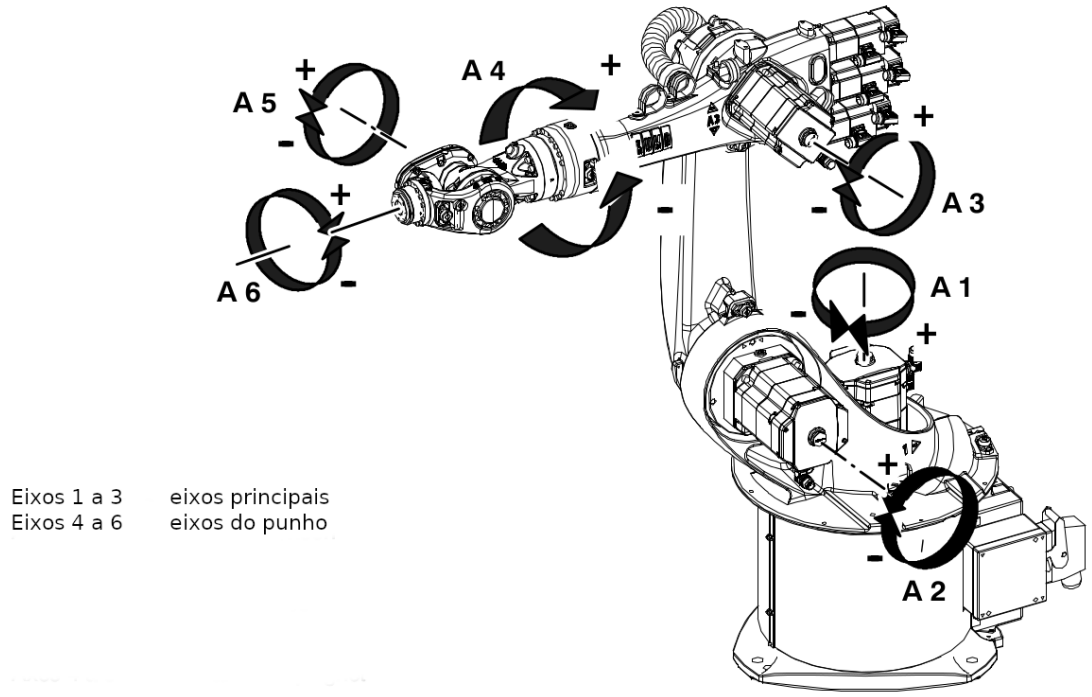


Figura 12: Eixos de rotação e direções de rotação no movimento do robô. (KUKA KR16 Specification - adaptação)

Eixo	**Amplitude de movimento**	**Velocidade**
A1	± 185°	156° /s
A2	+35° a -155°	156° /s
A3	+154° a -130°	156° /s
A4	± 350°	330° /s
A5	± 130°	330° /s
A6	± 350°	615° /s

Tabela 2: Limites cinemáticos do robô(KUKA KR16 Specification - adaptação)

5.1.3 Dinâmica

A fabricante não fornece dados suficientes para a modelagem da dinâmica do robô. Apesar de termos as especificações de cada um dos motores não há detalhes sobre redução e controle digital na atuação do robô. Dessa forma, este trabalho se propõe a modelar a etapa de planejamento de trajetória do robô, apenas.

Para que a transferência para o mundo real seja satisfatória, basta que o controle de velocidade e posição implementado pela KUKA tenha um baixo sobressinal na posição e um baixo tempo de acomodação na velocidade. Apesar de não termos os parâmetros de controle necessários para avaliar essas métricas pode-se assumir essas hipóteses como

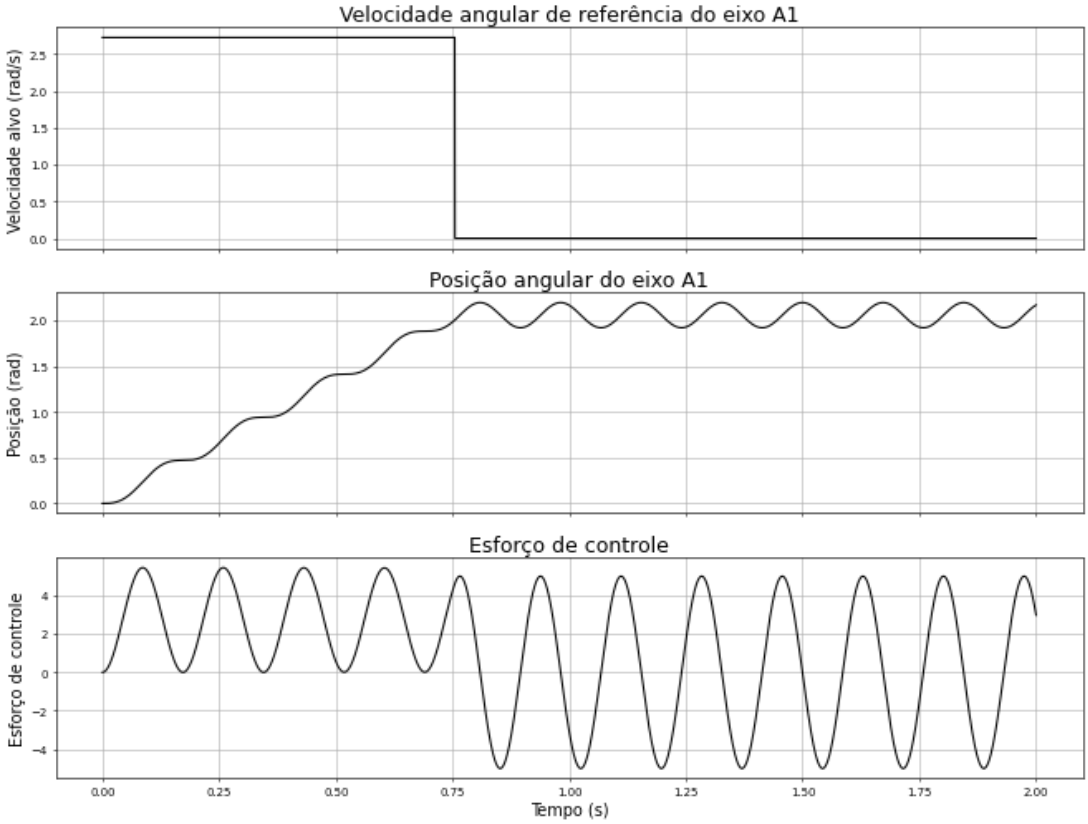
verdadeiras, dada a proposta de aplicação de um robô industrial. Além disso, segundo as especificações encontradas no manual o robô possui uma repetibilidade de ± 0.05mm

Modelagem dos Atuadores

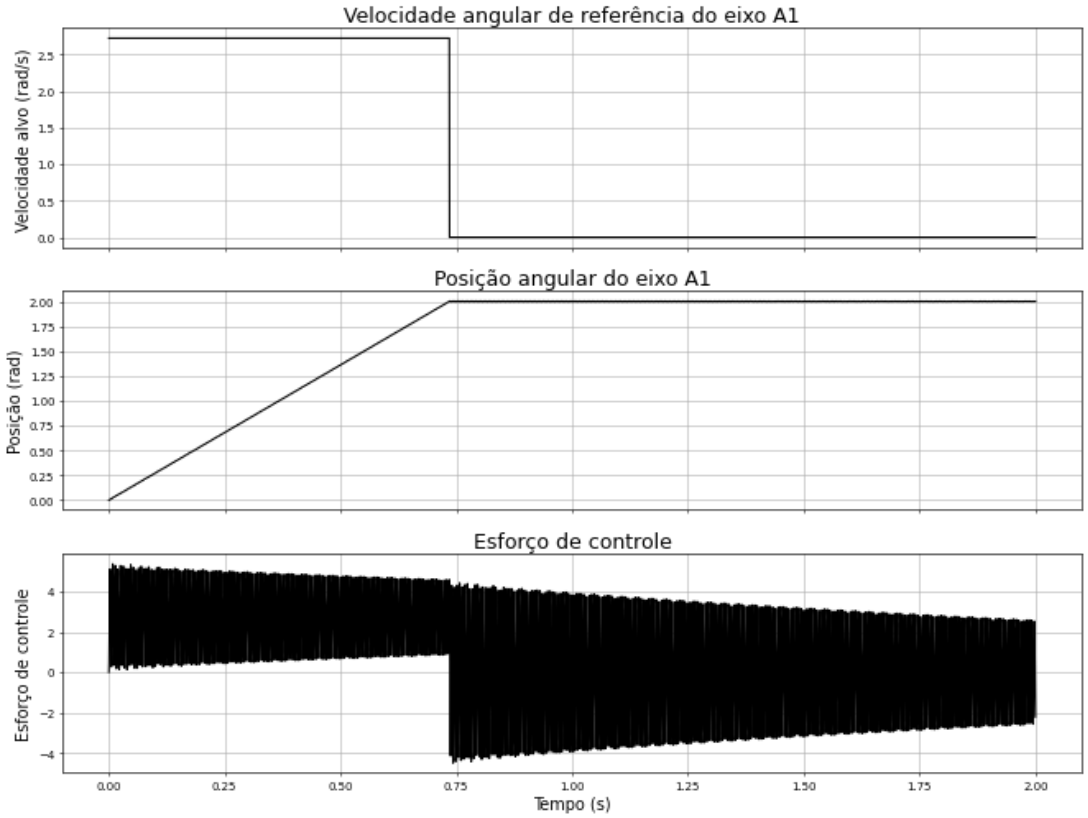
Visto que não temos disponíveis os parâmetros dinâmicos do sistema foi necessário modelar o sistema de forma simplificada. Em primeiro lugar foi definida uma densidade qualquer uniforme para os corpos que compõem braço robótico, ademais, estes foram considerados maciços. Com esta simplificação é possível obter as matrizes de inércia apenas com os modelos geométricos dos corpos.

Com o problema inercial resolvido, tem-se ainda o problema da modelagem dos motores, o MuJoCo possui apenas uma modelagem de atuação sobre o sistema, entretanto esta é genérica o suficiente para abarcar uma ampla gama de atuadores diferentes. Como pode ser visto na documentação do simulador, existem maneiras de ajustar os parâmetros dessa modelagem para que diversos tipos de atuadores sejam construídos, alguns destes possuem inclusive abstrações, ou seja, atalhos em programação para que sejam mais amigáveis ao usuário.

Aqui utilizou-se uma das abstrações disponíveis no MuJoCo, o servo de velocidade-integral, apesar do nome. Neste atalho o único parâmetro modificável é o coeficiente proporcional K_p do controlador. A fim de se obter um servo de posição com velocidade controlada utilizou-se o servo de velocidade-integral modificando-se a referência da velocidade para 0 quando a posição atinge a desejada. Vê-se abaixo um dos testes feitos com essa modelagem, o eixo se encontrava em repouso na posição 0; deseja-se levá-lo até a posição 2 rad com a velocidade máxima do eixo ($156^\circ/\text{s} \approx 2.72\text{rad/s}$).



(a) K_p baixo



(b) K_p alto

Figura 13: Comportamento do atuador(Fonte: Autores)

Encontrou-se um passo-a-passo de modelagem de robôs, e por consequência seus atuadores, fornecido na documentação do simulador. Este descrevia como os valores de amortecimento e ganho deveriam ser traduzidos para o modelo a fim se atingir um estado estático e dinâmico estável. Além disso, ainda cita que quando não há informações suficientes para se reproduzir os motores reais deve-se encontrar tais parâmetros por tentativa e erro.

No caso da modelagem com atuadores servos de velocidade integral o único de entrada é a constante proporcional do controlador. Sendo assim, fixava-se todos os 5 primeiros eixos e encontrava-se o parâmetro estável pra a junta A6. No passo seguinte repetia-se o processo com 4 juntas fixas e encontrava-se o parâmetro para a junta A5. Este processo foi repetido até que todos os parâmetros estivessem definidos e verificava-se a estabilidade do robô. Note que para uma junta An a massa movida por cada atuador aumenta conforme n diminui, isso faz com que o K_p precise aumentar (dado que a frequência natural também aumenta), portando é importante sempre escolher o menor K_p estável em cada passo, a fim de evitar que na junta A1 tenha-se um K_p muito elevado, o que leva a instabilidade.

Esta modelagem de atuação não é usual na comunidade de usuários de mujoco, tampouco nos modelos fornecidos de exemplo pela equipe de desenvolvedores e alguns motivos ajudam a explicar tal situação. Primeiramente, esta funcionalidade é nova no ambiente, tendo surgido em junho de 2022, por isso é esperado que tal recurso tenha uma documentação mais modesta e ainda baixo uso pela comunidade. Outro motivo, sobre o qual nem se encontram relatos nos fóruns relacionados, é que a estabilidade deste tipo de modelagem é limitada. Sendo que partindo-se da posição inicial para qualquer outra utilizando o método supracitado a simulação se mantém estável por todo o trajeto, entretanto, ao se definir uma pose inicial diferente da usada para escolher os parâmetros K_p a simulação se torna instável.

5.2 O ambiente

A sala em que o robô está instalado no Prédio da Engenharia Mecatrônica da Escola Politécnica da USP precisou ser reproduzida no ambiente virtual a fim de prever possíveis interferências no envelope de trabalho do robô dado que não há gaiola de isolamento do mesmo, sendo utilizadas cadeiras para cercar o perímetro. Além disso, para diminuir a necessidade de aprendizado por transferência na passagem para o mundo real, é preciso deixar o ambiente virtual o mais parecido possível com o ambiente real. Com isto posto, levantou-se as informações dimensionais da sala e propôs-se uma disposição ideal para a

realização dos experimentos.

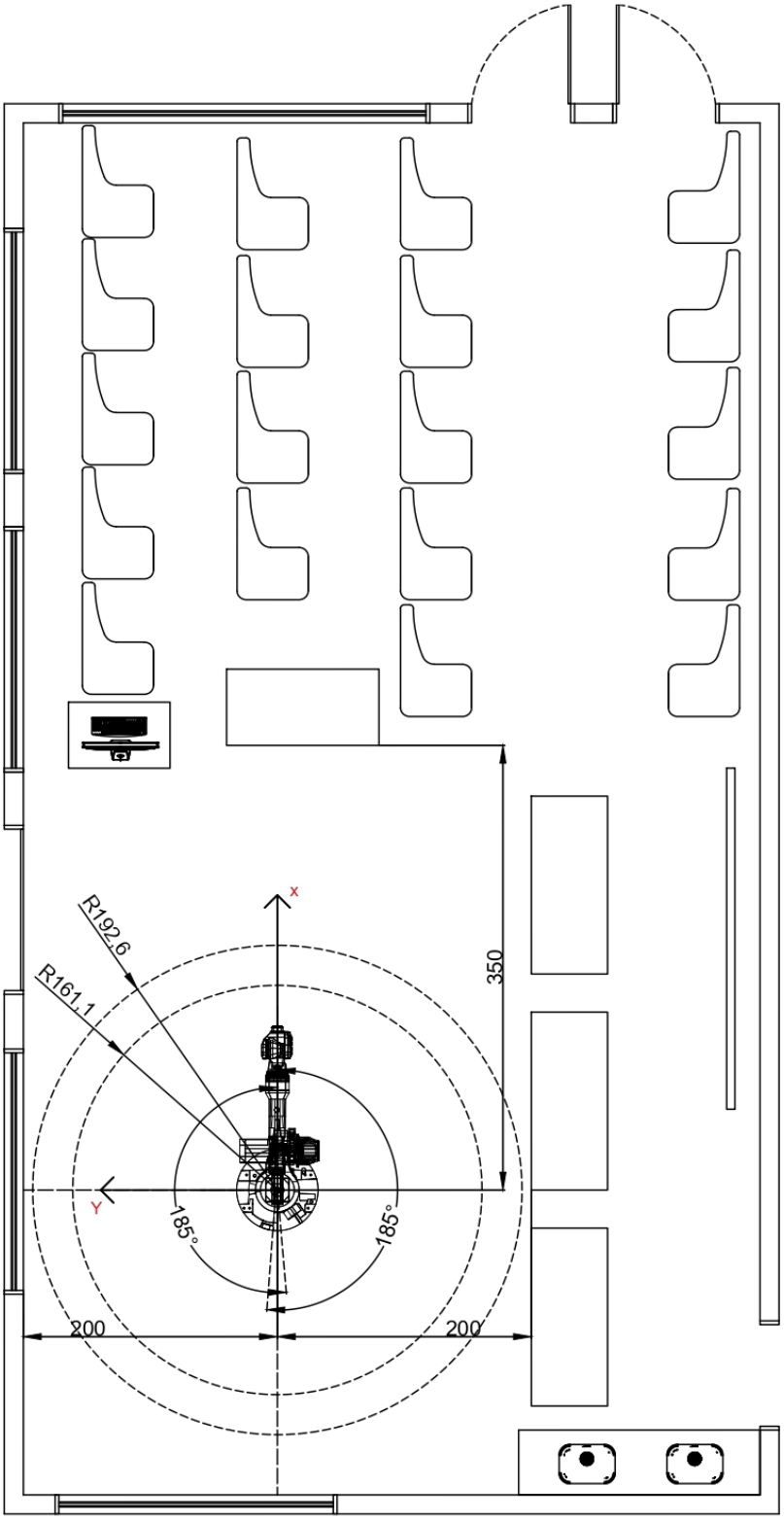


Figura 14: Planta baixa da sala, com a disposição final dos elementos. (Fonte: Autores)

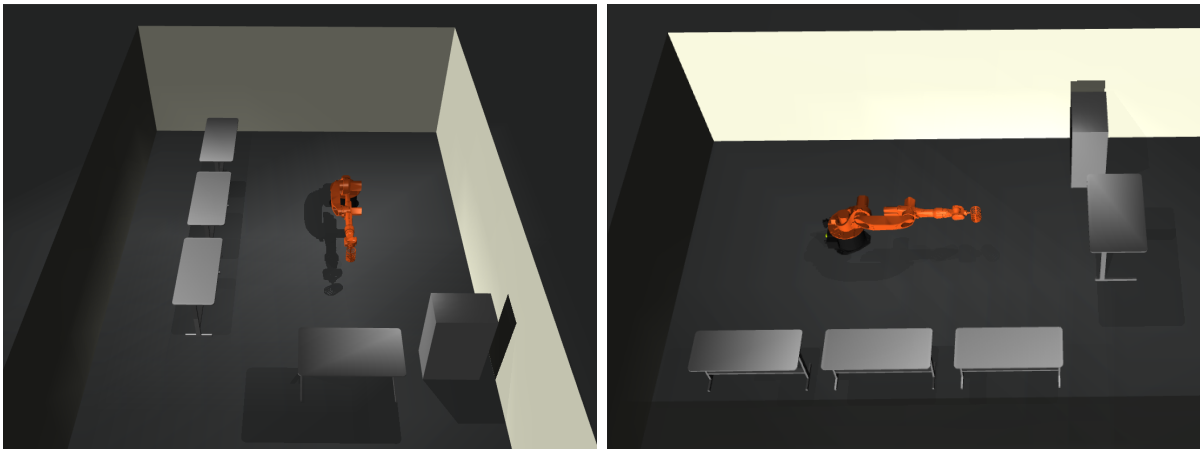
A circunferência tracejada menor se trata do envelope de trabalho do robô segundo o manual (o manual desconsidera o tamanho do último elo, como pode ser observado na

Figura 10). A circunferência tracejada maior representa o limite do envelope de trabalho do robô levando em conta a cesta coletora acoplada ao efetuador final incluindo o tamanho do último elo. Note também a posição do sistema global de coordenadas utilizado em todo o projeto.

O modelo virtual da sala foi construído utilizando elementos geométricos básicos, sendo as paredes, teto e piso semi-planos e o centro de controle do robô uma caixa, as exceções são as mesas e o robô em sí, que foram modelados em 3D e importados para o ambiente como um arquivo .stl capaz de representar sólidos. As descrições detalhadas dos parâmetros desses elementos pode ser encontradas no arquivo environment.xml

(a) Mesa real (b) Modelo 3D

Figura 15: Modelagem 3D das mesas(Fonte: Autores)



(a) visão frontal (b) visão lateral

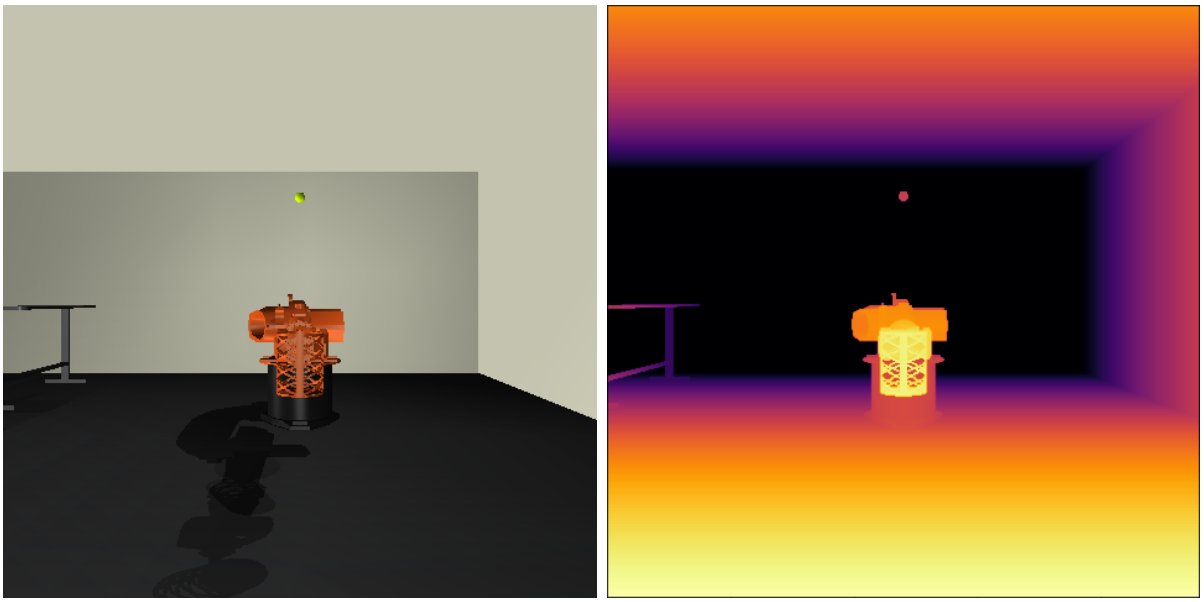
Figura 16: Renderização 3D do ambiente de simulação(Fonte: Autores)

Visto que o sensor utilizado é uma câmera foi necessário também definir as fontes de luz existentes. Estas, na sala original se tratam de lâmpadas fluorecentes, portanto fontes

extensas de luz, entretanto no ambiente utilizado neste trabalho estas lâmpadas foram substituídos por fontes pontuais de luz. As luzes configuradas tem um eixo principal e ângulo de corte, onde a partir dele objetos não são mais iluminados, dessa forma, utilizou-se uma fonte de luz a mais para iluminar adequadamente o teto; dado que as superfícies não refletem luz, não seria possível enxergá-lo.

5.2.1 Câmera simulada

O sensor escolhido para o projeto foi uma câmera de profundidade, a Intel RealSense Depth Camera D435, alguns parâmetros foram retirados do datasheet para representa-la virtualmente. Em especial o campo de visão, fov=58° e a distância inter-pupilar ipd=50mm. Por se tratar de uma câmera com propósito técnico seu software permite selecionar a taxa de atualização de quadros e resolução, inclusive podendo ser diferentes para os sensores RGB e o sensor de profundidade, este que possui uma taxa de atualização maior. Observe que exite um equilíbrio entre taxa de aquisição e resolução da imagem, dado o limite na transferência de dados via USB-A 3.0. Neste trabalho assumimos resoluções de 600x600 e quadros são atualizados em 60Hz para ambos sensores.



(a) Sensor de cor (b) Sensor de profundidade

Figura 17: Visão da câmera simulada.(Fonte: Autores)

A câmera foi posicionada sobre o eixo x global 3,6 metros distante do centro da base do robô, origem do sistema coordenado. na imagem abaixo é possível verificar o posicionamento e campo de visão da câmera.

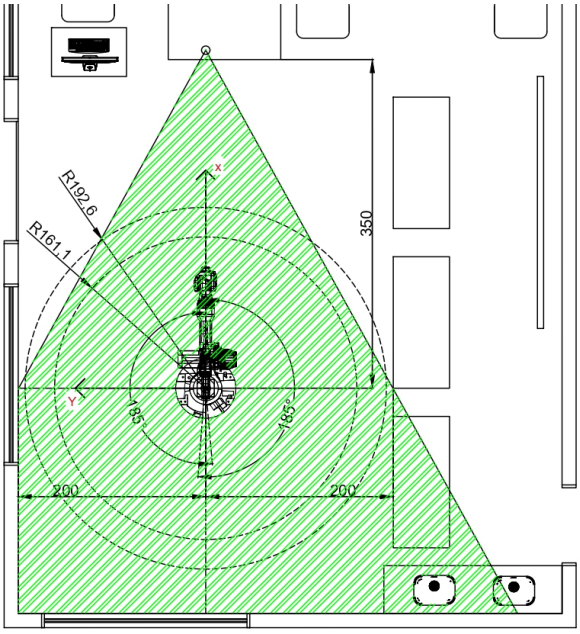


Figura 18: Campo de visão da câmera.(Fonte: Autores)

6 VISÃO COMPUTACIONAL

Como o sensor escolhido para captar os dados do sistema foi uma câmera de profundidade tornou-se imprescindível o entendimento e a posterior implementação de técnicas de visão computacional frente aos desafios impostas pelo problema. Este capítulo se propõe a detalhar as soluções escolhidas e implementadas para os desafios encontrados durante o desenvolvimento.

6.1 Reconhecimento da bola

O primeiro passo para saber a posição da bola é reconhece-la na imagem da câmera, para tanto utilizou-se alguma técnicas de visão computacional clássica.

Primeiramente, faz-se um filtro de cor na imagem, a fim de diferenciá-la do resto da imagem. O espaço de cores RGB não é conveniente no momento de se fazer um filtro desse tipo, por isso converte-se a imagem para o espaço HSV (matriz, saturação e valor/brilho) dessa forma escolhe-se um intervalo de verde (visto que é lançada uma bola de tênis) que será observado na imagem. Outros algoritmos também são aplicados na imagem , com a finalidade de redução de ruído, uma suavização gaussiana e iterações de erosão e dilatação. Segue abaixo alguns exemplos retirados da documentação da biblioteca OpenCV.

(a) Foto da câmera simulada (b) Mascara-resultado do filtro de cor

Figura 19: Aplicação do filtro de cor.(Fonte: Autores)

(a) original (b) erosão (c) dilatação (d) Suavização

Figura 20: Exemplos dos demais filtros aplicados.(OpenCV Docs)

O próximo passo é identificar um círculo e encontrar seu centro. Sendo assim executa-se um filtro identificador de bordas, associado a um algoritmo capaz de gerar uma lista de contornos. Dessa lista escolhe-se o menor contorno fechado. Dado o contorno, calcula-se seu centro em pixels.

6.2 Odometria Visual

Tendo solucionado o desafio de se obter a coordenada da bolinha em pixels. Passa-se agora para o desafio de encontrar a posição da bola em coordenadas da câmera e, em seguida, coordenadas globais. Já que estamos utilizando uma câmera 3D não haverá necessidade de utilizar o raio do circulo encontrado em conjunto com a dimensão real da bola para determinar uma solução única para o problema.

Em primeiro lugar, vale lembrar que estamos simulando uma câmera 3D real, a câmera real já conta um software/driver que já conta com todos estes algoritmos implementados. Entretanto, como este não é o caso de uso deste projeto foi necessário implementá-los novamente para que funcionem com a câmera simulada.

A mjr_readPixels(), fornecida pela API no mujoco, devolve um buffer de profundidade num formato normalizado, baseado na profundidade mínima Z_{perto} e máxima Z_{longe} configurada para a câmera. Para se obter a profundidade real d de um pixel (i, j) a partir da profundidade normalizada x faz-se necessária a seguinte conversão:

$$d(i,j) = \frac{Z_{perto}}{1 - x(i,j) * \left(1 - \frac{Z_{perto}}{Z_{longe}}\right)} \tag{6.1}$$

Seguem abaixa equação que representa a posição no sistema coordenado global a partir de um pixel (i,j) na imagem:

$$\begin{bmatrix} x \\ y \\ z \\ - \end{bmatrix} = d(i,j) \cdot H^{-1} \cdot \begin{bmatrix} i \\ j \\ 1 \\ 1/d(i,j) \end{bmatrix} \tag{6.2}$$

Com esta equação podemos obter a localização de qualquer ponto na imagem, a imagem abaixo são os pontos da Figura 19 projetados no espaço.

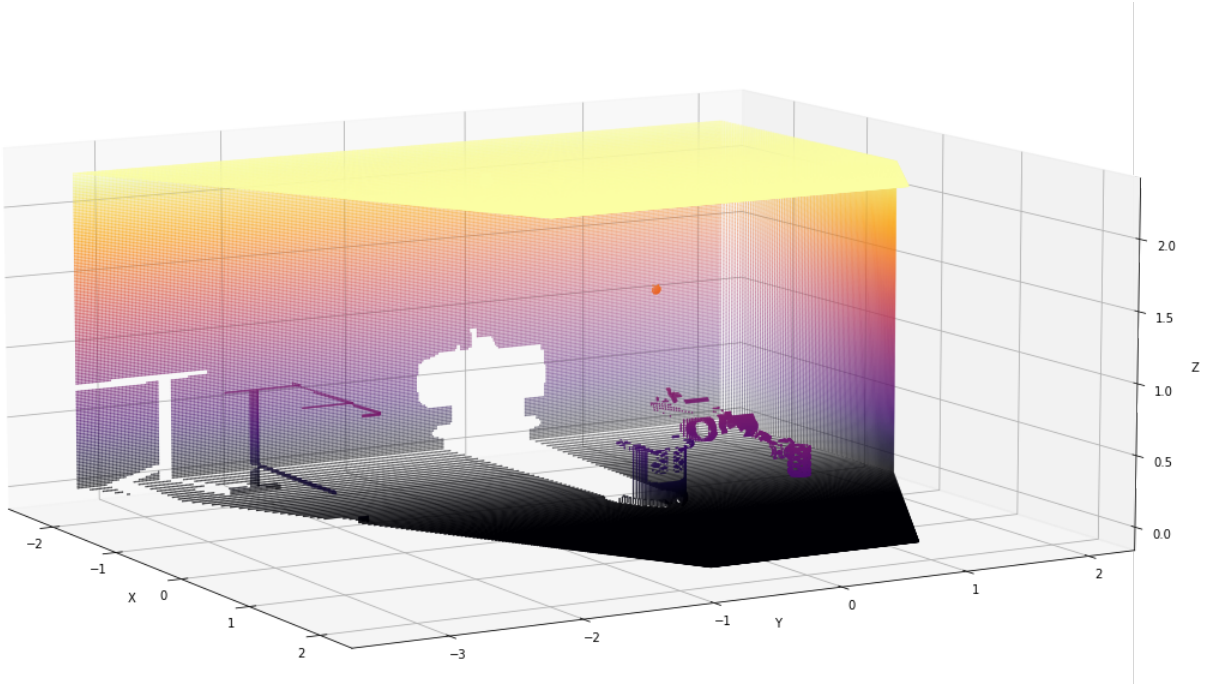


Figura 21: Projeção dos pontos no espaço.(Fonte: Autores)

6.2.1 Matrizes da câmera e de transformação

Para realizar os cálculos de odometria visual é necessário obter a matriz intrínseca da câmera. Normalmente essa etapa é realizada com o auxilio de um padrão de calibração posicionado em locais diferentes do campo de visão da câmera. Entretanto, como utilizamos uma câmera virtual foi necessário calcular a matriz a partir dos parâmetros da câmera.

Sendo, FOV_x e FOV_y os campos de visão nos eixos x e y da câmera, respectivamente; e H e W a altura e largura da imagem em pixels, respectivamente.

Calcula-se a distância focal f_x e f_y:

$$f_x = \frac{W}{2 \cdot tan\left(\frac{\pi \cdot FOV_x}{360}\right)} \qquad f_y = \frac{H}{2 \cdot tan\left(\frac{\pi \cdot FOV_y}{360}\right)} \tag{6.3}$$

Dado que não há distorções de fabricação pois a câmera é , c_x e C_y são simplesmente:

$$C_x = \frac{W}{2} \qquad C_y = \frac{H}{2} \tag{6.4}$$

E com isso tem-se a matriz da câmera K:

$$K = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

Para se obter as coordenadas no sistema global de coordenadas são necessários ainda as matrizes de rotação e o vetor de translação do sistema coordenado global para o da câmera. Com o ambiente descrito na na seção 5.2 nossa matriz R e vetor t são:

$$R = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \tag{6.6}$$

$$t = \begin{bmatrix} 0.825 \\ 0.0 \\ 3.6 \end{bmatrix} \tag{6.7}$$

Para finalmente calcular a matriz de transformação, define-se:

$$K_{fr} = \begin{bmatrix} \mathbf{K}_{[3\times3]} & \mathbf{0}_{[1\times3]} \\ \mathbf{0}_{[3\times1]} & 1 \end{bmatrix} \tag{6.8}$$

$$Rt_{fr} = \begin{bmatrix} \mathbf{R}_{[3\times3]} & \mathbf{t}_{[1\times3]} \\ \mathbf{0}_{[1\times3]} & 1 \end{bmatrix} \tag{6.9}$$

e assim tem-se a matriz de transformação H:

$$H = K_{fr} \cdot Rt_{fr} \tag{6.10}$$

6.3 Rastreamento da bola - Filtro de Kalman

Dadas as perturbações no mundo real ou ainda os obstáculos no visão da câmera, é necessária a construção de algum algoritmo que permita estimar a posição da bola mesmo quando esta não está visível. Para resolver este problema utilizamos um filtro de Kalman, que obedece as seguintes equações:

De predição:

$$\bar{\mathbf{x}} = \mathbf{F}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} \tag{6.11}$$

$$\bar{\mathbf{P}} = \mathbf{F}\hat{\mathbf{P}}\mathbf{F}^{\mathsf{T}} + \mathbf{Q} \tag{6.12}$$

De atualização:

$$\mathbf{K} = \bar{\mathbf{P}}\mathbf{H}^{\mathsf{T}}(\mathbf{H}\bar{\mathbf{P}}\mathbf{H}^{\mathsf{T}} + \mathbf{R})^{-1} \tag{6.13}$$

$$\mathbf{y} = \bar{\mathbf{x}} - H\mathbf{z} \tag{6.14}$$

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{K}\mathbf{y} \tag{6.15}$$

$$\hat{\mathbf{P}} = (\mathbf{I} - \mathbf{K}\mathbf{H})\bar{\mathbf{P}} \tag{6.16}$$

$$\tag{6.17}$$

Define-se como o vetor de estados $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ a_x \\ a_y \\ a_z \end{bmatrix} \tag{6.18}$$

Assumindo um modelo de aceleração constante, define-se F como a matriz de transição de estados, sendo:

$$dt = \frac{1}{\text{fps}} \tag{6.19}$$

$$F = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 & \frac{dt^2}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 & 0 & \frac{dt^2}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 & dt & 0 & 0 & \frac{dt^2}{2} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.20}$$

Define-se **H**, como sendo a máscara de medições:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.21}$$

A Matriz de covariância de medida R, obtida por uma amostra aleatória de 10.000 posições diferentes da bola:

$$\mathbf{R} = \begin{bmatrix} 3.0457e-03 & -8.60465e-04 & 1.2541e-04 \\ -8.6046e-04 & 5.4875e-04 & -3.1012e-05 \\ 1.2541e-04 & -3.1012e-05 & 1.0162e-04 \end{bmatrix} \tag{6.22}$$

Para o calculo de **Q**, define-se primeiramente o vetor τ:

$$\tau = \begin{bmatrix} 0.5 \cdot dt^2 \\ 0.5 \cdot dt^2 \\ 0.5 \cdot dt^2 \\ dt \\ dt \\ dt \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{6.23}$$

e finalmente, assumindo $var = 1$:

$$\mathbf{Q} = var \cdot \tau \cdot \tau^T = \tau \cdot \tau^T \tag{6.24}$$

Com o filtro de Kalman implementado o rastreamento da bolinha ficou muito mais suave e a previsão da aceleração muito mais precisa, também, isso foi extremamente importante para ter-se uma avaliação confiável de quando a bola está ou não em queda livre. Na imagem abaixo tem-se a trajetória da bolinha avaliada com o filtro de kalman, perceba que a bola passou por trás do braço robótico, portanto estava fora do campo de visão da câmera e mesmo assim o rastreamento continua bem comportado.

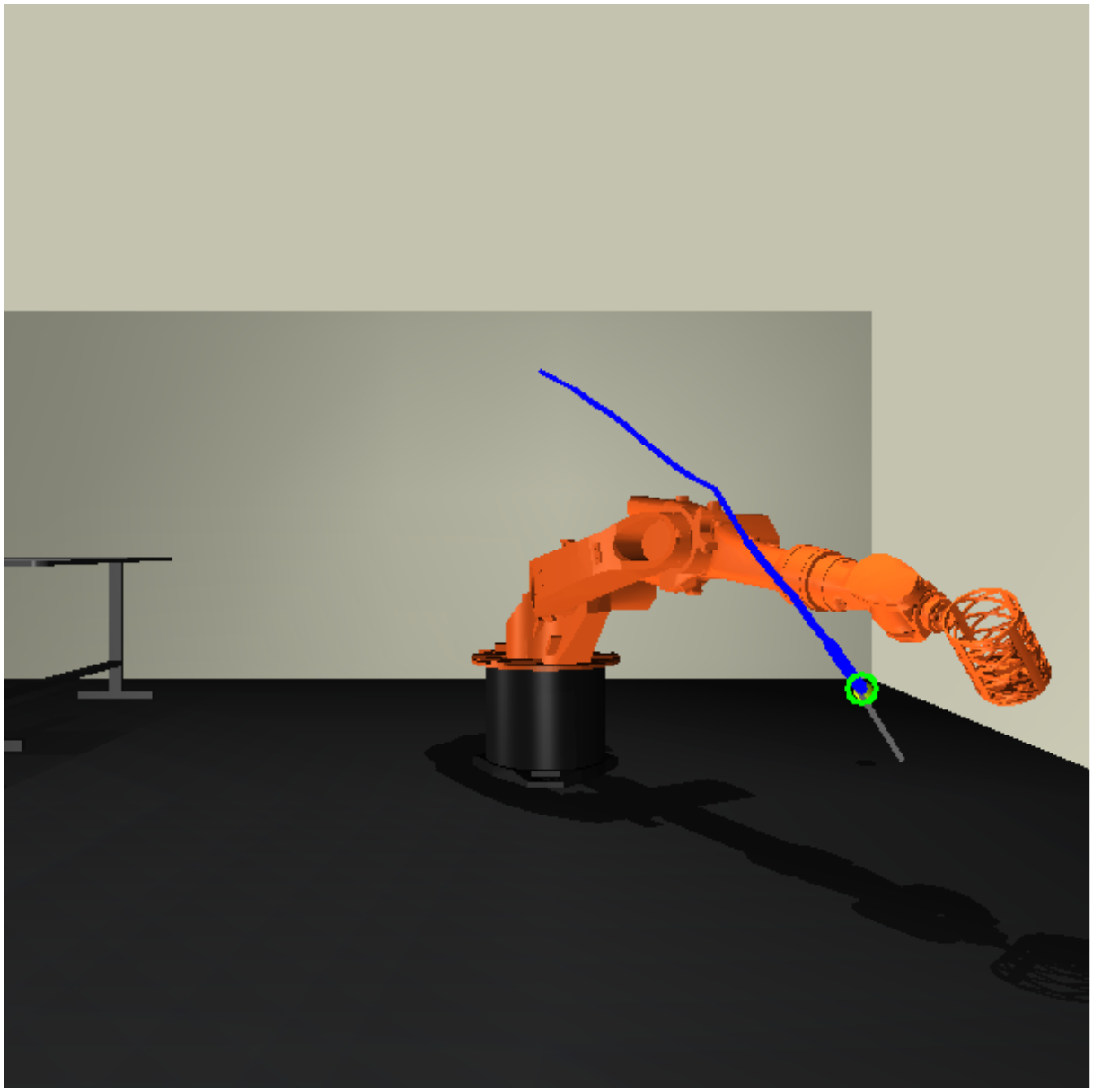


Figura 22: Rastreamento da bola, com obstáculo no campo de visão.(Fonte: Autores)

O pequeno calombo na rota em azul se refere ao fato de que a bola está parcialmente encoberta nesses ponto. Quando esta situação se apresenta o centro do contorno na máscara de cor e o centro da bolinha são divergentes.

7 MODELO DE APRENDIZADO POR REFORÇO

Conforme descrito na seção 3.1.1, o modelo do sistema pode ser descrito como processo de decisão de Markov, ou seja, pode ser descrito em função de seu espaço de estados, espaço de ações, função de transição de estados e função de recompensa. Nesta seção é explorada a definição destes elementos para o problema modelado.

7.1 Espaço de Estados

O espaço de estados adotado consiste em um vetor s composto pelo valor angular das articulações do robô, o vetor posição da bola $\vec{r_b} = (p_x, p_y, p_z)$, e o vetor velocidade da bola $\vec{V_b} = (v_x, v_y, v_z)$ e o vetor posição da cesta $\vec{r_c} = (x_c, y_c, z_c)$. O valor angular das articulações do robô obedece a um limite próprio conforme descrito na tabela 2.

$$s = \begin{bmatrix} p_x \\ p_y \\ p_z \\ v_x \\ v_y \\ v_z \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ x_c \\ y_c \\ z_c \end{bmatrix} \tag{7.1}$$

7.2 Espaço de Ações

O ambiente modelado é composto pelo robô, seu volume de trabalho e a bola arremessada. A ação tomada pelo robô consiste em mover suas articulações com o objetivo de que a posição da cesta montada em seu efetuador antecipe a trajetória da bola e a bola seja capturada. O controle do robô é feito através de sua velocidade. Considerando seis articulações e adotando os valores 1, 0 e -1 para representar, respectivamente, rotação no sentido positivo, rotação no sentido negativo e ausência de rotação, o número total de ações é de 3^6 e o espaço de ações pode ser descrito como $\mathcal{A} = [1, 0, -1]^6$. O espaço de ações completo pode ser descrito pela matriz A:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & 0 \\ 1 & 1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{7.2}$$

Cada linha da matriz A representa um vetor de ação A_i. O resultado da ação pode ser representado pelo vetor V:

$$V = kV_{max} \circ A_i \tag{7.3}$$

onde V_{max} representa um vetor contendo a velocidade máxima de cada uma das articulações, k representa um fator de velocidade, $V_{max} \circ A_i$ representa o produto elemento-a-elemento entre os vetores V_{max} e A_i e $V = [V_1, V_2, V_3, V_4, V_5, V_6]$ representa a velocidade imposta às articulações.

7.3 Função de transição de estados

Utilizando o vetor V definido na seção anterior em função da ação e o intervalo de tempo ΔT de cada passo da simulação é possível calcular os novos valores angulares das articulações do robô. Deste modo a função de transição de estados para um par estado-ação (s, a) é dada por:

$$f(s, a) = s + V_a \Delta T \tag{7.4}$$

em que V_a representa um vetor da forma:

$$V_a = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.5}$$

7.4 Função de recompensa

Entende-se que funções de recompensa descontínuas têm maior dificuldade de convergência, por isso optou-se por utilizar uma função logarítmica de barreira p(x) para os casos em que os limites de movimento são ultrapassados, sendo p um valor positivo que deverá ser subtraído da recompensa.

$$p(x) = min\left(p_{max}, C \cdot log(0,25(x_{max} - x_{min})^2) - log((x - x_{min})(x_{max} - x))\right)$$

(7.6)

A penalidade máxima p_{max} será aplicada caso x não esteja no intervalo $[x_{min}, x_{max}]$.

Esta função é utilizada como penalidade nos casos em que o limite de aproximação com o chão é excedido ou no caso em que os limites de rotação de uma ou maias juntas são excedidos. Os parâmetros x_{min} e x_{max} delimitam a região em que o movimento é permitido em cada caso, o parâmetro C é um parâmetro de curvatura, quanto maior for o seu valor, mais suavizada será a barreira.

A figura abaixo representa o comportamento da penalidade, para os parâmetros $x_{max} = -x_{min} = 3$, $C = 0.025$, $p_{max} = 1$:

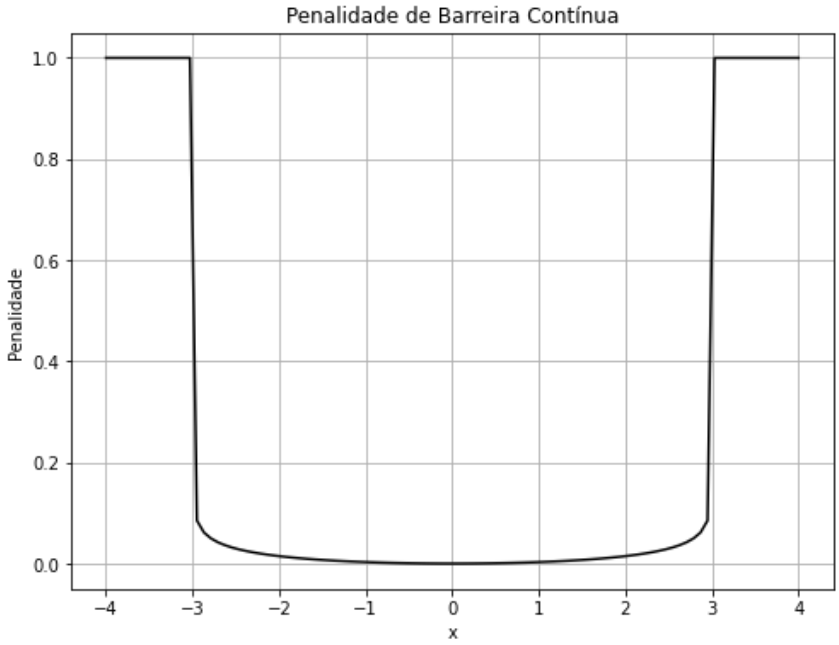


Figura 23: Função Penalidade contínua.(Fonte: Autores)

Outro parâmetro utilizado na função de recompensa foi a distância euclidiana entre o centro da cesta e a posição da bola.

Foram também utilizados os ângulos de Euler formados entre o sistema de coordenadas da cesta e o eixo Z global, ou seja, trata-se de uma forma de garantir que a cesta fique em posição vertical.

A última parte da função de recompensa tem o objetivo de fazer com que a cesta se aproxime da posição desejada antes da bola. Para tanto foram utilizadas as projeções dos

vetores em um plano, de modo a facilitar a análise. Primeiramente define-se os seguintes pontos:

$$A = \text{projeção da posição do alvo no plano xy} \tag{7.7}$$

$$B = \text{projeção da posição da bola no plano xy} \tag{7.8}$$

$$C = \text{projeção da posição da cesta no plano xy} \tag{7.9}$$

Em seguida calcula-se os seguintes vetores:

$$\vec{BA} = A - B \tag{7.10}$$

$$\vec{BC} = C - B \tag{7.11}$$

$$\vec{CA} = A - C \tag{7.12}$$

$$\vec{u} = \frac{\vec{BA}}{||\vec{BA}||} \tag{7.13}$$

Calcula-se a projeção dos vetores de interesse na direção bola-alvo e, a partir disso, calcula-se a função recompensa r:

$$a = \vec{CA} \cdot \vec{u} \tag{7.14}$$

$$b = \vec{BC} \cdot \vec{u} \tag{7.15}$$

$$r = 2(a - b) \tag{7.16}$$

7.5 Critérios de Parada

O treinamento do agente conta com diferentes critérios de parada. Esses critérios são utilizados pois os algoritmos adotados necessitam a simulação de episódios completos.

7.5.1 Limite de tempo alcançado

O primeiro critério e parada ocorre quando o é alcançado o limite de tempo máximo da simulação. Esse limite foi definido como $T = 6$ segundos. É importante que este parâmetro seja determinado de maneira que o robô possa alcançar a posição desejada.

7.5.2 Colisão do robô com o chão

De modo a garantir que a cesta não colida com o chão foi estabelecido uma altura mínima $H = 20$ cm entre o centro de massa da cesta e o chão. O valor de H é hiper-dimensionado por questão de segurança, tal medida foi adotada considerando que o robô é capaz de capturar a bola antes que ela chegue ao chão. O caso em que essa distância mínima não é respeitada consiste em um estado terminal que, quando alcançado resulta em punição para o robô.

7.5.3 Extrapolação dos limites de rotação das articulações

Conforme mostrado na tabela 2, as articulações do robô KUKA KR-16 apresentam um limite angular. Caso qualquer uma das seis articulações exceda o limite angular estabelecido, em qualquer sentido de rotação, este estado terminal é alcançado e o robô é punido como demonstrado na função recompensa.

7.5.4 O robô captura a bola dentro da cesta

Este critério de parada caracteriza o estado terminal em que houve sucesso, o robô foi capaz de atingir a posição desejada. A posição desejada é definida como o ponto da trajetória da bola contido no plano em que $h = 50$ cm, onde h denota a altura da bola em relação ao chão. Esta definição foi adotada pois permite a generalização do ponto desejado para diferentes trajetórias e garante que a bola caia dentro da cesta. Este estado é recompensado na função de recompensa.

7.5.5 Colisão da bola com o chão

O último estado terminal consiste na colisão da bola com o chão, indicando que o robô falhou na tarefa de capturar a bola.

PARTE IV

RESULTADOS

8 A FERRAMENTA

A construção da ferramenta com o objetivo de ser fácil de usar foi uma preocupação constante durante o desenvolvimento, dessa forma, explicita-se aqui a organização do código.

8.1 Parametrização

Como citado anteriormente o DQN tem uma natureza instável, por esse motivo ele é utilizado em conjunção com outras técnicas de modo a aumentar sua estabilidade e chance de convergência. Tal fato acaba por gerar uma quantidade muito grande de hiper parâmetros que precisam ser setados ao início do treinamento. Por fim, para facilitar a experiência do usuário com a ferramenta organizou-se todos os hiper-parâmetros do código em um arquivo de configuração *params.py*, que pode ser visualizado abaixo:

```
VALID_PARAMS = ("env_path","image_res","fps","cam_R","cam_t",
"num_episodes","max_sim_time","z_height","velocity_factor",
"floor_collision_threshold","render","plot",'gamma',"epsilon",
"epsilon_max","epsilon_min","epsilon_decay","tau",
'learning_rate',"memory_size",'batch_size',"update_gap",
"plot_interval")


params = {
# Environment parameters
    #Enviroment description filepath
        "env_path": "./sim_env/environment.xml",

# Camera parameters
    # Frame rate
```

```
        "fps": 60,
    # Image resolution
        "image_res": (600, 600), # (width, height)
        #must be square
        #TODO: make this work with non-square images
    # Camera rotation matrix
        "cam_R": [[0.0, 0.0, -1.0],
                  [0.0, 1.0, 0.0],
                  [-1.0, 0.0, 0.0]],
    # Camera translation vector
        "cam_t": [[0.825],
                  [0.0],
                  [3.6]],

# Simulation parameters
    # Maximum simulation time
        "max_sim_time": 6,
    # Velocity limit for the arm, factor of the maximum velocity
        "velocity_factor": 1.0,


# DQN parameters
    # Gamma discount factor
        "gamma": 0.99,
    # Learning rate
        "learning_rate": 0.0001,
    # Number of episodes to run
        "num_episodes": 700_000,

    #Replay Memory
        #Replay memory size
            "memory_size": 1_000_000,
        #Batch size
            "batch_size": 25000,

    #Epsilon Greedy
```

```
        #[max,mix,decay] epsilon values
            "epsilon": [1.0, 0.03, 0.999],
            #OR another way to set epsilon:
            # "epsilon_max": 1.0,
            # "epsilon_min": 0.03,
            # "epsilon_decay": 0.999,
            #The first is prioritised if both are set

    #Target and Policy network method
        #Updates target network every update_gap episodes
            "update_gap": 100,
        #Soft update method
            "tau": 1, #use 1.0 for hard update

# Reward parameters
    # Height of the basket to aim for when catching the ball
        "z_height": 0.5,
    # Threshold for the floor collision penalty
        "floor_collision_threshold": 0.2,

# Visualisation parameters
    # Render the simulation
        #(slows down the simulation way too much)
        "render": True,
    # Live plot of the rewards
        "plot": True,
    #Live plot interval
        "plot_interval": 10,
}
```

8.2 Execução

A execução do programa também é simplificada, via linha de comando basta utilizar:

Para treinar:

```
python3 main.py train
```

Para avaliar um modelo treinado:

```
python3 main.py eval -m "model.pth"
```

8.3 Visualização

Com o parâmetro render configurado é possivel assistir ao robô aprendendo.

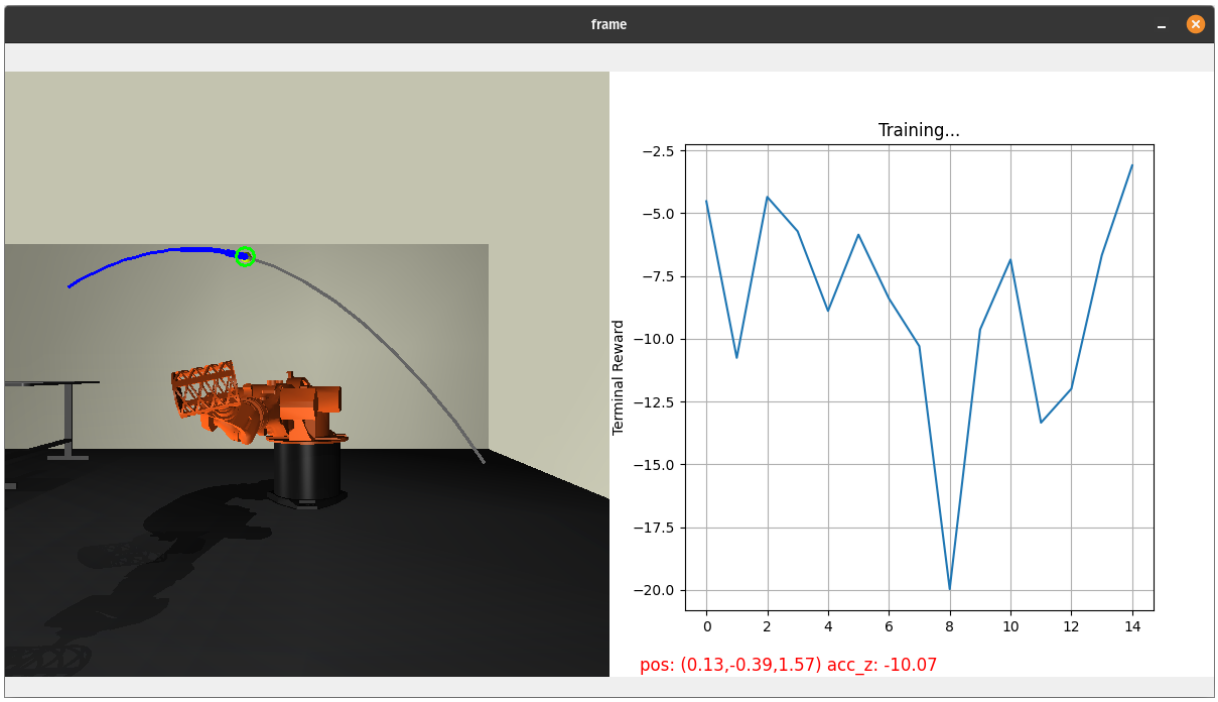


Figura 24: Janela de visualização do treinamento. (Fonte: Autores)

Com o parâmetro plot configurado é possível ver a evolução do treinamento ao vivo:

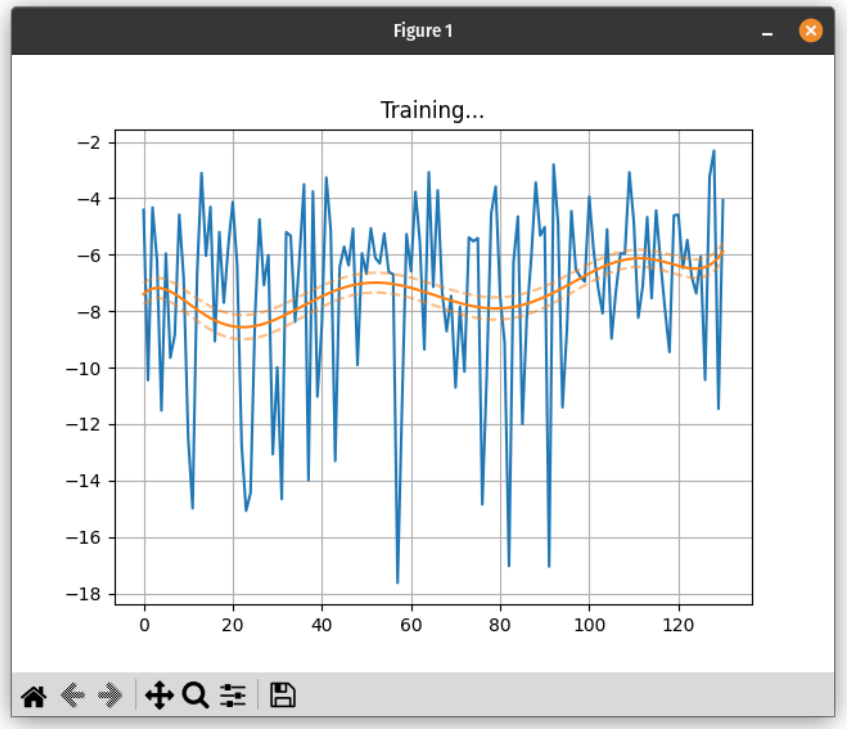


Figura 25: Plotagem ao vivo das recompensas terminais. (Fonte: Autores)

8.4 Orientação a Objetos

Houve a preocupação em modularizar toda a implementação de modo a facilitar que os futuros alunos modifiquem as algoritmos como preferirem. Para tanto foram utilizadas estruturas de classes com interações e bem definidas entre si. Ademais, foram também utilizadas *type-hints* para facilitar o entendimento sobre as entradas dos métodos contrutores.

9 SOLUÇÕES DESENVOLVIDAS

Infelizmente este trabalho não conseguiu encontrar os hiper-parâmetros que fizessem o modelo convergir para uma solução, discussões mais aprofundadas sobre isso podem ser encontradas na seção de conclusões.

PARTE V

CONCLUSÕES

A utilização de um modelo discreto para o espaço de ações permitiu a implementação do algoritmo DQN. No entanto, a complexidade do problema ao utilizar um espaço de ações de tamanho 729 junto com a instabilidade própria do algoritmo DQN dificultou a convergência. Com o intuito de garantir a convergência podem ser utilizadas outras técnicas de aprendizado por reforço que lidam com modelos não discretos como é o caso de algoritmos do tipo ator-crítico. Outro problema encontrado no treino é a grande quantidade de tempo e capacidade computacional demandados. Uma forma de lidar com isso é adotar simplificações para o modelo ou utilizar uma arquitetura de treino com múltiplos agentes.

Fazer o ambiente virtual de simulação foi um verdadeiro desafio, várias peças desconexas e mal documentadas apareceram pelo caminho. Entretanto, pode-se dizer que a criação do ambiente virtual foi um sucesso; a começar por ter conseguido inserir o robô dentro reprodução da sala real, em seguida, fomos capazes de criar um servo de velocidade no robô, lançar a bola de lugares diferentes e em condições diferentes programaticamente, visualizar tudo isso com uma câmera 3D simulada. Além de tudo isso, um dos integrantes do grupo colaborou com o desenvolvimento de uma função da biblioteca em python que estrutura a renderização do ambiente, visto que ela era necessária para o projeto e ainda não havia sido implementada(a colaboração foi aceita pelos criadores).

Em retrospecto, observando as várias peças desconexas e sem documentação encontradas, não há resultado mais bem sucedido que o atingido por este trabalho. As peças foram encaixadas, de forma coesa e funcional, com um programa bem estruturado e bem documentado. Claro que ainda há espaço para melhorias, a câmera simulada só consegue obter a profundidade correta de imagens quadradas, por exemplo. E isso torna o cenário futuro ainda mais animador, este projeto nasceu com o objetivo de criar um ambiente capaz de ensinar, ensinar um robô, e ensinar alunos, mas nunca se poderia o quanto ele ensinou seus autores e ainda ensinará seus futuros colaboradores.

REFERÊNCIAS BIBLIOGRÁFICAS

M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation.

P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. 10 2016.

H. Hu, K. Zhang, A. H. Tan, M. Ruan, C. G. Agia, and G. Nejat. A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain. *IEEE Robotics and Automation Letters*, 6:6569–6576, 10 2021.

A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager. 'good robot!': Efficient reinforcement learning for multi-step visual tasks with sim to real transfer. *IEEE Robotics and Automation Letters*, 5:6724–6731, 10 2020.

R. Jeong, Y. Aytar, D. Khosid, Y. Zhou, J. Kay, T. Lampe, K. Bousmalis, and F. Nori. Self-supervised sim-to-real adaptation for visual robotic manipulation. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2718–2724, 5 2020.

D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. 6 2018.

P. Kormushev, S. Calinon, and D. G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. 2015.

J. Luo, O. Sushkov*, R. Pevceviciute*, W. Lian, C. Su, M. Vecerik, N. Ye, S. Schaal, and J. Scholz. Robust multi-modal policies for industrial assembly via reinforcement learning and demonstrations: A large-scale study. 3 2021.

A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra. Benchmarking reinforcement learning algorithms on real-world robots. 9 2018.

D. J. Mankowitz, N. Levine, R. Jeong, A. Abdolmaleki, T. Springenberg, Y. Shi, J. Kay, T. Mann, T. Hester, and M. R. Deepmind. Robust reinforcement learning for continuous control with model misspecification. 6 2019.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning.

OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving rubik's cube with a robot hand. 10 2019.

X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3803–3810, 10 2017.

J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21, 5 2008.

D. Schwab, T. Springenberg, M. F. Martins, T. Lampe, M. Neunert, A. Abdolmaleki, T. Hertweck, R. Hafner, F. Nori, and M. Riedmiller. Simultaneously learning vision and feature-based control policies for real-world ball-in-a-cup.

H. Shen, J. Yosinski, P. Kormushev, D. G. Caldwell, and H. Lipson. Learning fast quadruped robot gaits with the rl power spline parameterization. *Cybernetics and Information Technologies*, 12:66–75, 2012.

A. Stooke and P. Abbeel. Accelerated methods for deep reinforcement learning. 2018. doi: 10.48550/ARXIV.1803.02811. URL `https://arxiv.org/abs/1803.02811`.

M. Vecerik, J.-B. R. Deepmind, O. S. Deepmind, D. B. Deepmind, R. P. Deepmind, T. R. Deepmind, C. S. Deepmind, R. H. Deepmind, and J. S. Deepmind. S3k: Self-supervised semantic keypoints for robotic manipulation via multi-view consistency. 9 2020.

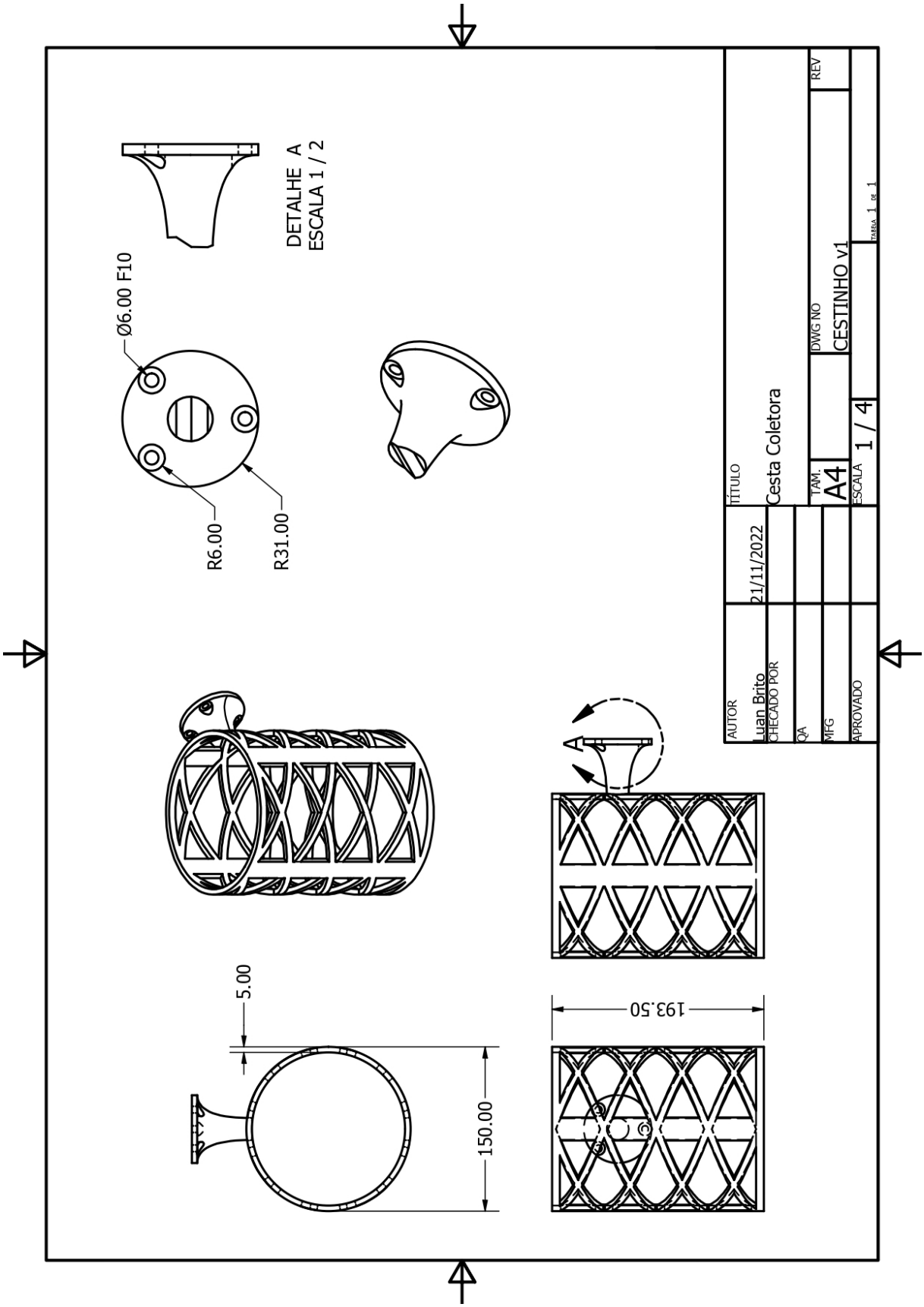
I. Zamalloa, R. Kojcev, A. Hernandez, I. Muguruza, L. Usategui, A. Bilbao, and V. Mayoral. Dissecting robotics - historical overview and future perspectives. 04 2017.

W. Zhu, X. Guo, D. Owaki, K. Kutsuzawa, and M. Hayashibe. A survey of sim-to-real transfer techniques applied to reinforcement learning for bioinspired robots. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–16, 9 2021.

PARTE VI

APÊNDICES

9.1 Desenho técnico da cesta



9.2 Repositório no GitHub

Os códigos por este trabalho desenvolvidos estão disponível no repositório no GitHub em:

`https://github.com/LuanGBR/Kuka_RL_Control`