UNIVERSIDADE FEDERAL DO AMAZONAS

FACULDADE DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUÇÃO EM ENGENHARIA ELÉTRICA

Fábson Gomes Nepomuceno

Investigação de algoritmos de cinemática inversa para resolução de trajetórias em sistemas cobot pick-and-place baseados em ROS

MANAUS-AM

2024

Fábson Gomes Nepomuceno

Investigação de algoritmos de cinemática inversa para resolução de trajetórias em sistemas Cobot pick-and-place baseados em ROS

Dissertação apresentada ao curso de Mestrado em Engenharia de Elétrica, área de concentração Controle e Automação de Sistemas, apresentada ao Programa de Pós-Graduação *Stricto Sensu* para obtenção do título de Mestre em Engenharia de Elétrica pela Universidade Federal do Amazonas – UFAM.

Área de Concentração: Desenvolvimento de Sistemas de Automação e Ambientes Inteligentes.

Linha de Pesquisa: Cyber-Physical Systems e Indústria 4.0.

Orientador: Prof. Dr. Vicente Ferreira de Lucena Júnior.

Coorientador: Prof. Dr. Iury Valente de Bessa.

MANAUS- AM

2024

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Nepomuceno, Fabson Gomes
N441i Investigação de algoritmos de cinemática inversa para resolução de trajetórias em sistemas cobot pick-and-place baseados em ROS / Fabson Gomes Nepomuceno . 2024
74 f.: il.; 31 cm.

Orientador: Vicente Ferreira de Lucena Júnior
Coorientador: Iury Valente de Bessa
Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Amazonas.

1. Industria 4.0. 2. Cinemática. 3. Cobot. 4. Pick-and-place. I. Lucena Júnior, Vicente Ferreira de. II. Universidade Federal do Amazonas III. Título

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Engenharia Elétrica

FOLHA DE APROVAÇÃO

Poder Executivo Ministério da Educação
Universidade Federal do Amazonas
Faculdade de Tecnologia
Programa de Pós-graduação em Engenharia Elétrica

Pós-Graduação em Engenharia Elétrica. Av. General Rodrigo Octávio Jordão Ramos, nº 3.000 - Campus Universitário, Setor Norte - Coroado, Pavilhão do CETELI. Fone/Fax (92) 99271-8954 Ramal:2607. E-mail: ppgee@ufam.edu.br

FÁBSON GOMES NEPOMUCENO

INVESTIGAÇÃO DE ALGORITMOS DE CINEMÁTICA INVERSA PARA RESOLUÇÃO DE TRAJETÓRIAS EM SISTEMAS COBOT PICK-AND-PLACE BASEADOS EM ROS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica na área de concentração Controle e Automação de Sistemas.

Aprovada em 20 de dezembro de 2024.

BANCA EXAMINADORA
Prof. Dr.-Ing. Vicente Ferreira de Lucena Junior - Presidente
Prof. Dr. Renan Landau Paiva de Medeiros - Membro Titular 1 - Interno
Prof. Dr. Cleonor Crescencio das Neves - Membro Titular 2 - Externo

Manaus, 10 de dezembro de 2024.

Documento assinado eletronicamente por **Renan Landau Paiva de Medeiros, Professor do Magistério Superior**, em 20/12/2024, às 16:11, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Vicente Ferreira de Lucena Júnior, Professor do Magistério Superior**, em 20/12/2024, às 16:12, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Cleonor Crescêncio das Neves, Usuário Externo**, em 20/12/2024, às 16:51, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufam.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2366442** e o código CRC **D470FFBB**.

Av. Octávio Hamilton Botelho Mourão - Bairro Coroado 1 Campus Universitário Senador Arthur Virgílio Filho, Setor Norte - Telefone: (92) 3305-1181
CEP 69080-900 Manaus/AM - mestrado_engeletrica@ufam.edu.br

Referência: Processo nº 23105.051816/2024-12 SEI nº 2366442

Criado por 31183646291, versão 3 por 22464247200 em 11/12/2024 13:18:37.

Agradecimentos

Agradeço primeiramente à Deus, pela jornada da vida.

Agradeço a meus familiares e amigos que torceram e me apoiaram durante o mestrado, onde sempre me incentivaram a ir até o fim.

Agradeço ao Professor Vicente pelo trabalho de orientação técnica e por todo apoio fornecido durante a jornada do programa.

Agradeço aos amigos da UFAM principalmente do grupo de pesquisa, ao Álvaro e Davi pelas trocas de conhecimento e experiência ao longo desses anos.

Agradeço aos professores do PPGEE pelo compartilhamento do conteúdo técnico científico do programa e conselhos e sempre dispostos a oferecer qualquer ajuda.

Agradeço à UFAM, pelo aprendizado construído no decorrer do mestrado.

Resumo

A Indústria 4.0 exige automação para melhorar eficiência e precisão, tornando os robôs colaborativos essenciais pela sua adaptabilidade e integração com IoT e inteligência artificial. Neste contexto, a resolução da cinemática inversa em manipuladores, especialmente para pick-and-place, é um desafio relevante. Esta dissertação investiga métodos numéricos para resolver a cinemática inversa em um ambiente simulado utilizando o software GAZEBO, organizando peças de LEGO com um Cobot integrado ao ROS e visão computacional via YOLO. Foram comparados algoritmos como Gradiente Descendente (baseado na pseudo-inversa do Jacobiano), que ajusta iterativamente as variáveis articulares, e os métodos de otimização BFGS, L-BFGS-B e Nelder-Mead, que utilizam aproximações da Hessiana para melhorar a convergência. Os resultados mostram que, embora os métodos analíticos sejam superiores ao tempo de execução, os métodos numéricos oferecem maior flexibilidade e são mais fáceis de implementar, no entanto a complexidade e velocidade computacional carecem de desempenho em relação ao analítico, tornando-se uma alternativa viável para resolver problemas de cinemática inversa em cenários onde soluções analíticas são inviáveis ou inexistentes.

Palavras-chave: Cobot, Industria 4.0, Cinemática, ROS, Pick-and-place.

Abstract

Industry 4.0 demands automation to enhance efficiency and precision, making collaborative robots essential due to their adaptability and integration with IoT and artificial intelligence. In this context, solving inverse kinematics in manipulators, particularly for pick-and-place tasks, is a significant challenge. This dissertation investigates numerical methods for solving inverse kinematics in a simulated environment using the GAZEBO software, organizing LEGO pieces with a Cobot integrated into ROS and computer vision via YOLO. Algorithms such as Gradient Descent (based on the pseudo-inverse of the Jacobian), which iteratively adjusts joint variables, and optimization methods like BFGS, L-BFGS-B, and Nelder-Mead, which use Hessian approximations to improve convergence, were compared. The results indicate that while analytical methods are superior in execution time, numerical methods offer greater flexibility and are easier to implement. However, their computational complexity and speed lag behind analytical approaches, making them a viable alternative for solving inverse kinematics problems in scenarios where analytical solutions are infeasible or nonexistent.

Keywords: Cobot, Industry 4.0, Kinematics, ROS, pick-and-place.

LISTA DE NOMENCLATURA

Abreviações

ROS - *Robot Operating System.*

YOLO – You Only Look Once (Usado no contexto de visão computacional).

BFGS - *Broyden–Fletcher–Goldfarb–Shanno*.

L-BFGS - *Limeted Memory Quase-Newton.*

DH - *Denavit-Hartenberg.*

DOF – *Degrees of Freedom* (Graus de Liberdade)

IK – *Inverse Kinematics* (Cinemática inversa).

FK – Forward Kinematics (Cinemática Direta).

FPS – *Frames per Second.*

SVD – *Singular Value Decomposition* (Decomposição em Valores Singulares, usada para pseudo-inversa do Jacobiano).

CNN – *Convolution Neural Network*

COBOT – *Collaborative Robot*

TCP – *Tool Center Point*

RGB – *Red, Green and Blue.*

Outros

Gazebo – Simulador 3D para robótica.

Pose – Posição e orientação de objeto.

Blender - Ferramenta que permite a criação de vastos conteúdos de 3D. Oferece funcionalidades completas para modelagem, renderização e visualização de conteúdo 3D.

Pyquaternion - módulo Python completo para representar e usar quaternions.

Pipeline - Técnica de programação que divide tarefas sequenciais em estágios, permitindo que várias instruções sejam executadas ao mesmo tempo.

Índice de Figuras

Figura 1.1 - Diferentes estágios da produção industrial 3
Figura 1.2 - Utilização de robôs industriais no mundo 4
Figura 2.1 - Anatomia de um robô manipulador 11
Figura 2.2 - Tipos de juntas utilizadas em robôs 12
Figura 2.3 - Robô colaborativo com modulo pick-and-place da UR5 15
Figura 2.4 - Sistema de orientação vetor-ângulo 16
Figura 2.5 - Representação das rotações em torno dos eixos no UR5 17
Figura 2.6 - Representação das transformações das juntas e posição de um robo manipulador 20
Figura 2.7 - Representação dos parâmetros de DH 23
Figura 2.8 - Tipos de singularidade no sistema colaborativo 26
Figura 2.9 - Ecossistema do ROS 34
Figura 2.10 - Sistema de arquivos ROS 35
Figura 2.11 - Processo de comunicação entre nós 35
Figura 2.12 - Exemplo de classificação usando redes neurais 37
Figura 2.13 - Arquitetura de um YOLO com 24 camadas convolucionais 38
Figura 3.1 - Triagem para a revisão literária 40
Figura 4.1 - Etapas do fluxo do ciclo completo de pick-and-place do robô 43
Figura 4.2 - Fase pick-up robô 44
Figura 4.3 - Fase placed do robô 44
Figura 4.4 - Arquitetura do sistema pick-and-place 45
Figura 4.5 - Visão simplificada das etapas do processo da resolução da cinemática 46
Figura 4.6 - Ambiente de simulação do sistema proposto 47
Figura 4.7 - Referências para cinemática 48
Figura 4.8 - Gráfico de comparação dos desempenhos das versões YOLO 51
Figura 4.9 - Aprendizagem baseada em YOLOV5 para detecção e estimativa da pose 53
Figura 4.10 - Detecção das peças LEGO no ambiente gazebo 53
Figura 5.0 - Ambiente de simulação gazebo 57
Figura 5.1 - Curva de convergência de um ponto ao longo da trajetória.....58
Figura 5.2 - Posição das juntas ao longo de um movimento 60

Índice de Tabelas

Tabela 4.1 - Parâmetros da cinemática do manipulador UR5e 48
Tabela 4.2 - Especificação do dispositivo utilizado para executar os testes 54
Tabela 5.1 - Tempo para resolver a cinemática inversa para diferentes movimentos 55
Tabela 5.2 - Média de erro e iteração ao longo de um movimento 56

Sumário

1 Introdução 1
1.1 Contexto 1
1.1.1 Indústria 4.0 2
1.1.2 Robôs colaborativos 3
1.1.3 Machine Vision 4
1.1.4 Modelos digitais 5
1.2 Problema 6
1.3 Motivação 7
1.4 Objetivo geral 8
1.5 Objetivo específico 8
1.6 Estrutura do documento 9
2 Fundamentação Teórica 10
2.1 Robôs manipuladores 10
2.1.1 Elementos básicos de um robô manipulador 10
2.1.2 Planejamento de movimento de um robô manipulador 13
2.1.3 Operações de pick-and-place 13
2.2 Robô UR5E 14
2.2.1 Sistema de coordenadas e orientação do robô UR5e 15
2.3 Cinemática 19
2.3.1 Cinemática direta 20
2.3.2 Cinemática inversa 24
2.4 Matrizes jacobiana e singularidades 25
2.5 Métodos numéricos 26
2.5.1 Gradiente descendente 26
2.5.2 Métodos Quase-Newton 28
2.5.3 BFGS 30
2.5.4 Limited memory Quase-Newton L-BFGS-B 32
2.5.5 Nelder-Mead 33
2.5 Robot Operate System (ROS) 33
2.7 Visão computacional 36
2.7.1 YOLO 37
2.7.2 Estrutura da YOLOv5: Tronco, Pescoço e Cabeça 38

3 Revisão da Literatura e Trabalhos Correlatos 39
3.1 Revisão Literária 39
3.2 Trabalhos Correlatos 40
4 Arquitetura e Metodologia 42
4.1 Arquitetura Proposta 43
4.2 Metodologia 46
4.2.1 Ambiente de simulação 47
4.2.2 Definição e modelo matemático para o manipulador UR5e 48
4.2.3 Cinemática do manipulador UR5e 48
4.2.4 Reconhecimento e classificação 51
4.2.5 Segmentação de alvos e determinação da pose do ponto de captura 52
4.2.6 Processamento e cálculo do tempo 53
5 Resultados 56
6 Conclusão 61
Referências Bibliográficas 62
Apêndice 68

1. Introdução

Este capítulo contextualiza o tema da dissertação, fornecendo uma base para a compreensão do problema de pesquisa. Além disso, apresenta a problemática da pesquisa, o objetivo geral e os objetivos específicos traçados para responder ao problema descrito. Além disso explora-se a motivação central deste trabalho e sua estruturação. A contextualização abrange a origem e a relevância do tema, proporcionando uma compreensão clara e completa do problema abordado, enquanto a problemática destaca as lacunas sobre o tema proposto.

1.1Contexto

Os robôs têm sido amplamente utilizados nas indústrias desde a década de 1960 para realizar tarefas repetidas e perigosas que seriam difíceis para os seres humanos. Além disso, estes são mais eficazes por terem a possibilidade de trabalharem continuamente e sem descanso, o que aumenta a eficácia e a produção das fábricas.

A automação de processos industriais com robôs apresenta um grande potencial para melhorar a qualidade dos produtos e reduzir o tempo e os custos de produção [7]. Além disso, a utilização de robôs na indústria pode liberar trabalhadores humanos para desempenhar tarefas mais criativas e com alto grau de complexidade no manuseio, aumentando assim a satisfação no trabalho e a produtividade. Dessa forma, a adoção da automação industrial com robôs pode trazer benefícios significativos para a indústria, como aumento da eficiência operacional e melhoria da competitividade no mercado.

Os robôs colaborativos industriais, também conhecidos como Cobots, são amplamente utilizados na automação de tarefas repetitivas, como operações de pick-and-place em linhas de produção e armazéns [3]. Sua capacidade de operar de forma autônoma, segura e eficiente os torna essenciais na indústria moderna [1]. Além disso, eles podem manipular materiais pesados e perigosos, como metais quentes e produtos químicos, reduzindo os riscos para os trabalhadores humanos.

Entretanto, a precisão no manuseio de objetos pode ser um desafio significativo, especialmente em ambientes de manufatura onde a variabilidade da matéria-prima resulta em mudanças frequentes de formas e tamanhos [6]. Para que um Cobot realize operações de agarrar objetos com eficiência, é necessário um sistema integrado de visão computacional que reconheça e classifique os objetos corretamente, além de um controle otimizado do manipulador para garantir o posicionamento adequado do módulo de pick-and-place [5]. O cálculo do

planejamento de movimento envolve o controle coordenado de múltiplas juntas, assegurando que o Cobot se mova de forma precisa no espaço tridimensional para alcançar seu objetivo sem colisões.

Esta pesquisa investiga e compara métodos numéricos para a resolução da cinemática inversa em um Cobot de seis graus de liberdade (6-DOF), analisando sua precisão, eficiência computacional e aplicabilidade em operações de pick-and-place. A complexidade desse problema se deve à natureza não linear das equações da cinemática inversa, cuja resolução exige estratégias computacionais robustas. Assim, são avaliados diferentes algoritmos, incluindo Gradiente Descendente (usando a pseudo-inversa do Jacobiano), BFGS, L-BFGS-B e Nelder-Mead, explorando suas vantagens e limitações em termos de convergência e desempenho computacional.

Além disso, está dissertação propõe um sistema integrado de visão computacional, utilizando o YOLO para detecção e classificação de objetos, permitindo a organização dinâmica de peças de LEGO. Através de simulações no Gazebo, o estudo analisa o comportamento cinemático do Cobot ao longo de trajetórias planejadas, fornecendo uma base comparativa entre os métodos numéricos implementados. Dessa forma, esta pesquisa contribui para o avanço da robótica industrial, avaliando a viabilidade do uso de métodos numéricos na resolução da cinemática inversa e sua integração com sistemas de visão computacional em um ambiente de simulação industrial.

1.1.1. Indústria 4.0

A indústria 4.0 é uma evolução da indústria manufatureira que tem por objetivo aumentar a eficiência, a qualidade e a personalização da produção. Isso é alcançado pela combinação de tecnologias avançadas, como inteligência artificial, Internet das coisas (IoT) e robótica, para criar uma rede de inteligência artificial que integra sistemas e processos de produção.

De acordo com [9], as atualizações tecnológicas continua sendo um dos principais elementos de desenvolvimento para a indústria. Assim, sendo necessário as indústrias acompanharem o ritmo do mercado bastante competitivo, para modernizar os sistemas e processos de produção [10]. Permitindo que as fábricas do futuro integrem novos tipos de sistemas de informação e automação inteligentes, além de robôs colaborativos mais flexíveis, conhecidos como "Cobots". Esses avanços visam alcançar uma automação mais completa, reduzindo ou eliminando a necessidade de mão de obra humana em tarefas consideradas

inviáveis ou inadequadas para trabalhadores [11]. A figura 1.1 ilustra a evolução da produção industrial em seus estágios.

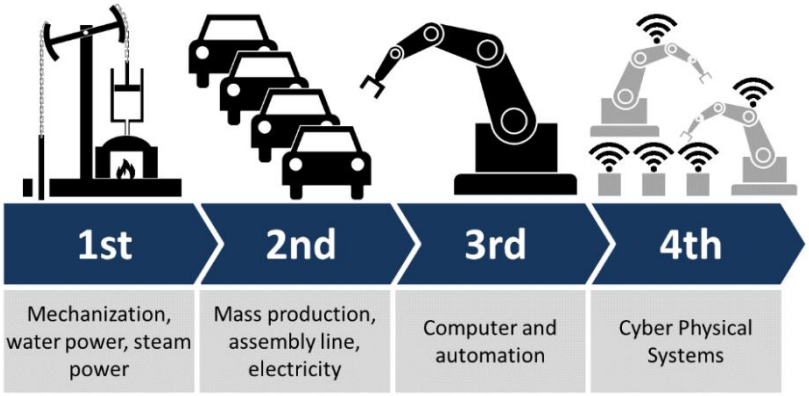

Figura 1.1- Diferentes estágios da produção industrial. Fonte: [12].

1.1.2. Robôs colaborativos

Robôs colaborativos, segundo a norma ISSO/TS 15.066 que apresenta a definição de um sistema de robótica colaborativa como um estado no qual um robô projetado para esse tipo de função e um operador trabalham em um espaço colaborativo. A utilização de robôs colaborativos na indústria é feita de diferentes formas, para [25] essa classificação se dá em quatro modalidades, sendo elas:

- **Coexistência**, quando robô e humano se encontram em um mesmo ambiente, porém sem interação entre eles.
- **Sincronizada,** quando o robô e o humano interagem em um mesmo ambiente, porém em tempos diferentes.
- **Cooperativa**, quando robô e o humano coexistem em um mesmo ambiente e realizam tarefas distintas ao mesmo tempo.
- **Colaborativa,** quando o robô e o humano executam a mesma tarefa ao mesmo tempo em um ambiente.

Os robôs colaborativos, ou Cobots, desempenham um papel crucial na Indústria 4.0, também conhecida como a quarta revolução industrial. Esses robôs são projetados para executar tarefas repetitivas, como operações de *pick-and-place*, em que peças ou objetos são manipulados e posicionados em locais específicos. Segundo [26], os robôs *pick-and-place*, são exemplos de sistemas de automação flexíveis, que podem ser rapidamente adaptados a mudanças na produção ou a novos produtos. A integração de robôs com módulos *pick-and-place* na produção industrial é uma tendência global crescente [2], permitindo que os processos

de produção se tornem mais eficientes, precisos e ágeis, além de liberar os trabalhadores de tarefas repetitivas e potencialmente perigosas. Atualmente, as pesquisas têm focado na colaboração entre humanos e robôs, com o objetivo de capacitar operadores a trabalharem de forma integrada com esses sistemas. Em especial, busca-se identificar e otimizar os níveis adequados de colaboração e automação em operações com Cobots [11].

Os robôs colaborativos são uma tecnologia revolucionária fundamental da Indústria 4.0, que está mudando a forma como trabalhamos e produzimos bens. Além disso, os autômatos colaborativos oferecem muitos benefícios, incluindo melhoria da eficiência, segurança e qualidade, além de liberar os trabalhadores para tarefas de maior valor agregado. Ademais, o setor de robôs colaborativos tem apresentado um crescente aumento nos últimos anos, o que demonstra a sua importância e potencial para o futuro da produção industrial. conforme mostra a figura 1.2.

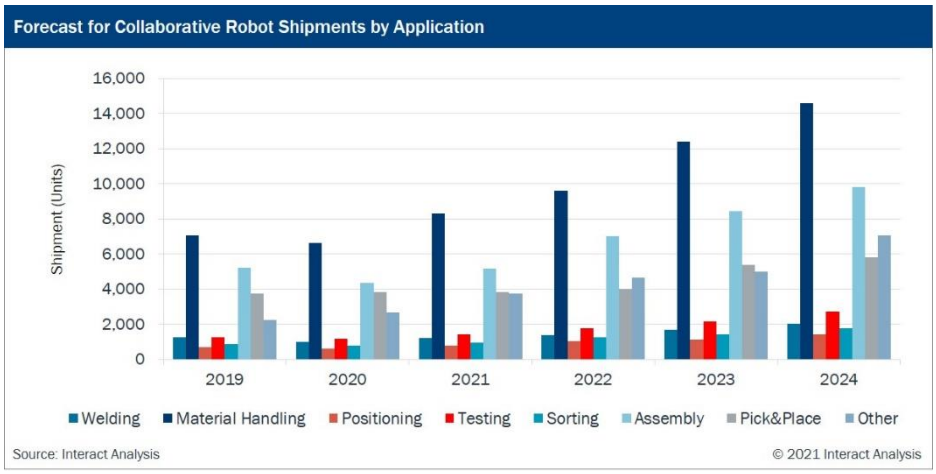

Figura 1.2- Utilização de robôs industriais no mundo. Fonte: [27].

1.1.3. Machine Vision (visão de máquina)

Machine vision é uma tecnologia que permite que máquinas, como robôs ou sistemas automatizados, percebam e interpretem informações visuais do ambiente usando câmeras, sensores e algoritmos. É um subconjunto da visão computacional, que é o campo da inteligência artificial e da ciência da computação que se concentra em permitir que computadores interpretem e entendam dados visuais do mundo, possibilitando melhor sensibilidade e resolução. Isto possibilita para a aplicação identificação e inspeção de peças de vários tamanhos e forma.

Em geral, a *machine vision* é uma tecnologia cada vez mais importante na era da Indústria 4.0, elas resolvem vários desafios do uso da robótica, como a variabilidade as posições

das peças e ferramentas, na movimentação do robô e nos pontos de operação do robô, por isso, o use de técnicas de *machine vision* em todas as suas variantes (2D, 3D, nuvens de pontos) são fundamentais para o processo industrial [14]. Ela tem o potencial de melhorar a eficiência, precisão e segurança em muitas aplicações, e seu desenvolvimento e integração contínuos com outras tecnologias provavelmente levarão a aplicações ainda mais avançadas no futuro.

1.1.4. Modelos digitais

No crescente corpo de literatura, termos relacionados a simulações, *como digital twin* (DT), têm sido usados de forma inconsistente para definir diferentes relações entre componentes físicos e digitais. Em [36], é apresentada uma revisão aprofundada categorizando conceitos associados às DTs. Termos como Modelo Digital (DM) e Sombra Digital (DS), diferem no que diz respeito à infraestrutura de comunicação entre os componentes físicos e digitais.

Um modelo digital é uma representação digital de um modelo existente ou objeto físico onde não possui qualquer troca automatiza de dados entre o objeto físico e o objeto digital. Esse tipo de modelo pode incluir uma simulação de modelos de fabricas planejadas, modelos matemáticos de novos produtos, ou qualquer outro modelo de objeto físico.

A sombra digital, com base na definição de modelo digital, se houver um fluxo de dados unidirecional automatizado entre o estado do objeto físico existente e um objeto digital pode-se classificar essa combinação como sombra digital, pois uma mudança de um estado do objeto físico leva a uma mudança de estado no objeto digital, mas não vice-versa.

Por fim, o *digital twin* leva o nível de integração mais profundo, permitindo que informações sobre o estado dos componentes físicos e digitais sejam analisados simultaneamente e comparados em tempo real. Com isso, um DT leva o nível de integração ainda mais longe, apesar de uma série de desafios técnicos que devem ser superados para poder representar eficazmente as suas capacidades e limitações, para produzir um DT de alta fidelidade.

Esses gêmeos digitais oferecem a vantagem de monitorar e analisar continuamente o desempenho dos sistemas físicos, fornecendo informações valiosas sobre sua performance, saúde e riscos potenciais. Além disso, eles também possibilitam a simulação de cenários de operação e a otimização de processos, ajudando a melhorar a eficiência, a segurança e a sustentabilidade da indústria. A tecnologia de gêmeos digitais desempenha um papel central na

implementação de fábricas inteligentes, ao integrar dados em tempo real com análises preditivas e simulações [4].

Na robótica colaborativa, os gêmeos digitais desempenham um papel ainda mais relevante. Eles possibilitam a simulação do comportamento dos robôs em diversos cenários, permitindo a otimização da programação antes da implantação física. Essa aplicação não apenas reduz os custos e o tempo de desenvolvimento e testes, mas também melhora a segurança e a eficiência operacional dos robôs colaborativos [17]. Além disso, os gêmeos digitais facilitam o monitoramento em tempo real da performance do robô, identificando potenciais falhas e possibilitando a manutenção preventiva.

Assim, é notável que a tecnologia de gêmeos digitais tem um papel fundamental na Indústria 4.0 e na robótica colaborativa, promovendo uma integração entre o mundo físico e o mundo digital e proporcionando vantagens competitivas para as empresas que a utilizam.

1.2. Problema

Os robôs colaborativos (Cobots) têm sido amplamente estudados na indústria, especialmente em aplicações de visão computacional e automação de tarefas. No entanto, a literatura apresenta uma lacuna significativa no que diz respeito à resolução da cinemática inversa para controle de posição em operações de pick-and-place, particularmente quando integrados ao sistema Robot Operating System (ROS) e sistemas de visão computacional.

A transição das indústrias para ambientes colaborativos com Cobots exige soluções eficientes para o planejamento de movimento, mas os métodos numéricos para a resolução da cinemática inversa ainda carecem de estudos aprofundados em cenários industriais específicos. As pesquisas sobre Cobots geralmente se dividem em três áreas principais: implementações e modulações [15] [16], estudos sobre a importância e o impacto dos Cobots na indústria [13] e estudos de casos [14]. Embora existam abordagens analíticas que resolvem esse problema, sua aplicabilidade é limitada a configurações específicas de robôs, tornando necessário investigar métodos numéricos alternativos que ofereçam maior flexibilidade e adaptação a diferentes tarefas.

Dessa forma, este estudo busca preencher essa lacuna, investigando e comparando diferentes métodos numéricos para a resolução da cinemática inversa em um Cobot de 6 DOFs, avaliando seu desempenho em termos de precisão e eficiência computacional em um ambiente de simulação industrial, no contexto da organização de peças.

1.3. Motivação

As linhas de produção modernas, em setores como o automotivo, eletrônico e de bens de consumo, buscam constantemente otimizar processos produtivos para reduzir o tempo de execução e aumentar a eficiência. Tarefas como organização e manipulação de peças exigem precisão e são frequentemente executadas por operadores humanos, o que pode resultar em riscos ergonômicos, fadiga e exposição a condições adversas.

O avanço dos robôs colaborativos (Cobots) representa uma solução eficaz para essas operações, oferecendo maior segurança e produtividade. A integração desses sistemas com visão computacional possibilita a automatização de tarefas pick-and-place, permitindo a manipulação eficiente e segura de objetos em ambientes industriais. No entanto, para que os Cobots realizem esses movimentos de forma precisa, é fundamental um controle adequado de seus múltiplos graus de liberdade (DOF).

A resolução da cinemática inversa é um desafio essencial nesse contexto, pois determina como o manipulador deve ajustar suas juntas para alcançar posições e orientações desejadas. Métodos analíticos são tradicionalmente utilizados, mas possuem limitações quando aplicados a robôs com configurações mais complexas. Métodos numéricos surgem como uma alternativa viável, oferecendo maior flexibilidade para lidar com diferentes configurações do manipulador e restrições do ambiente.

Esta pesquisa investiga e compara diferentes métodos numéricos para a resolução da cinemática inversa, avaliando sua eficiência e precisão na manipulação de peças em um ambiente simulado. Para isso, será implementado um Cobot integrado ao ROS, equipado com um módulo de visão computacional baseado no YOLO, para reconhecimento e classificação de peças de LEGO. Os algoritmos analisados incluem Gradiente Descendente (com pseudo-inversa do Jacobiano), BFGS, L-BFGS-B e Nelder-Mead, explorando suas características em termos de convergência, estabilidade e aplicabilidade no controle de manipuladores.

A contribuição deste estudo está no aprofundamento da compreensão sobre o impacto dos métodos numéricos na cinemática inversa de robôs colaborativos, possibilitando a aplicação dessas técnicas em cenários industriais que demandam alta precisão, flexibilidade e adaptação a diferentes tarefas.

1.4. Objetivo geral

O objetivo principal desta dissertação é investigar métodos numéricos para a resolução do problema da cinemática inversa em um robô colaborativo aplicado a operações de pick-and-place ao longo de uma trajetória. O estudo avalia a precisão, eficiência e tempo de execução dos algoritmos, buscando identificar a abordagem que melhor equilibre desempenho computacional e comportamento do manipulador em um robô de 6 graus de liberdade (6-DOF) no contexto da manipulação robótica

1.5. Objetivo específico

Esta dissertação visa investigar o comportamento cinemático de um sistema robótico por meio da implementação de algoritmos de métodos numéricos capazes de controlar o manipulador em operações de controle de posição. O sistema deverá receber como entrada uma pose desejada e, por meio de diferentes métodos numéricos, calcular os ângulos das juntas do manipulador. Como caso de estudo, será utilizado um robô colaborativo UR5e para organizar peças de LEGO dispersas em uma esteira industrial, integrando recursos de visão computacional para reconhecimento e classificação das peças. Para atingir esse objetivo, serão realizadas as seguintes etapas:

- Criação de um ambiente de desenvolvimento simulado no software Gazebo para um sistema Cobot com um módulo *pick-and-place*, e a utilização de um módulo de câmera RGB-D para realizar operações de classificação e geração de coordenadas para realizar o ordenamento das peças em diferentes posições dinamicamente.
- Desenvolvimento de um algoritmo de visão computacional para reconhecimento e classificação das peças de LEGO utilizando uma câmera do tipo RGB-D.
- Desenvolvimento de algoritmos para a resolução da cinemática inversa aplicado no Cobot utilizando o framework ROS para a implementação dos métodos e cálculo de posição de coordenadas para organização e posicionamento de peças, utilizando as peças de LEGO como exemplo de aplicação.
- Investigar a técnicas de algoritmos implementadas e realizar simulações.
- Obtenção de métricas do comportamento através de gráficos de convergência e estabilidade das juntas.
- Avaliação da performance do comportamento cinemático das operações *pick-and-place* do robô utilizando simulação e comparação com a literatura.

1.6. Estrutura do documento

O conteúdo deste trabalho está divido entre capítulos onde abordam sobre o desenvolvimento do sistema, uma revisão literária das principais tecnologias e metodologias utilizadas para a resolução problema da cinemática inversa em robôs colaborativos.

O capítulo 2, Fundamentação Teórica, são apresentados os principais conceitos e teorias usados para a pesquisa e desenvolvimento do sistema proposto. O capítulo 3, Revisão da Literatura e Trabalhos correlatos, descreve uma pesquisa sobre o tema e os principais artigos que abordam os aspectos da pesquisa, tais como, sistema operacional (ROS), visão computacional, algoritmos de cinemática inversa, para serem usados como referencial teórico e empregando alguns conceitos de sistemas robóticos que servem como base para a pesquisa. O capítulo 4, Método Proposto, descreve a proposta e os módulos que irão compor a arquitetura do sistema, descrevendo de forma detalhada cada um dos módulos que compõem a arquitetura. No capítulo 5, Resultados, são mostrados os resultados da pesquisa e discussões dos experimentos. O capítulo 6, Conclusão, é feito um fechamento do documento com conclusões e considerações finais, por último são apresentadas as referências utilizadas para fundamentar o trabalho.

2.0. Fundamentação teórica

Neste capítulo, apresenta-se um conjunto de base conceitual que serve como embasamento para o desenvolvimento deste projeto acadêmico, consistindo em uma revisão bibliográfica sobre os principais temas desta pesquisa, que visa fornecer um suporte teórico para a compreensão do tema em questão.

2.1. Robôs manipuladores

Segundo [40], um robô manipulador é definido como "uma máquina manipuladora com múltiplos graus de liberdade, controlada automaticamente, reprogramável e multifuncional, que pode ter uma base fixa ou móvel, destinada a aplicações em automação industrial". Dessa forma, esses robôs podem ser projetados para realizar operações colaborativas, atuando em ambientes de trabalho compartilhados de maneira eficiente e segura.

A estrutura mecânica de um robô manipulador é composta por uma série de corpos rígidos, chamados elos, interligados por articulações ou juntas. Essa configuração permite que o robô manipule objetos com precisão e mobilidade. Em geral, o manipulador é composto por três partes principais: um braço, que proporciona movimento; um punho, que garante destreza; e um efetuador, que realiza a tarefa para a qual o robô foi programado [41].

2.1.1. Elementos básicos de um robô manipulador

Um robô é composto basicamente por uma base fixa, um braço articulado constituído por elos mecânicos, um controlador e sensores. O braço do manipulador robótico é responsável pelo posicionamento do robô no espaço físico cartesiano, ou seja, é a parte que se move nos eixos x, y e z. Ele é formado por uma série de elos conectados por juntas, sendo fixado à base em uma extremidade (elo de entrada) e a um punho na outra (elo de saída). A Figura 1.2 ilustra a configuração de um braço robótico.

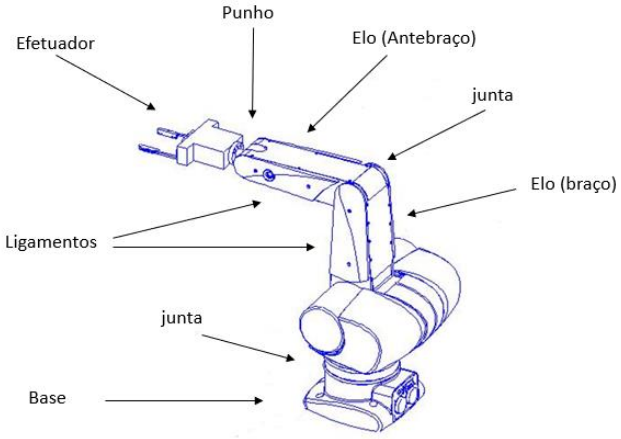

Figura 2.1 - Anatomia de um robô manipulador. Fonte: Adaptado de [39].

Um Robô Manipulador Geralmente possui os elementos básicos descritos abaixo:

- **Ligamento:** Seria os elos braços e antebraços
- **Punho:** O punho do manipulador robótico é ligado ao último elo do manipulador, ou seja, está conectado com uma ferramenta de trabalho ou apenas uma garra. O que vai determinar qual o tipo de ferramenta o manipulador deve possuir, é a função na qual o manipulador foi produzido para executar.
- **Atuador:** São elementos que convertem a energia elétrica, hidráulica e pneumática em potência mecânica que são enviadas aos elos para que os mesmos e movimentem.
- **Controlador:** É a unidade responsável pelo gerenciamento e monitoração dos parâmetros operacionais requeridos para realizar as tarefas dos robôs.
- **Efetuador:** É o elemento que executa e dá função de uso ao robô, já que o robô apenas transporta o efetuador a algum lugar, contudo é o efetuador que executa o trabalho. Geralmente cada robô trabalha com apenas um efetuador. A ferramenta do efetuador mais simples é a garra, no qual possui a função de abrir e fechar. Outros tipos de efetuador são pistolas, furadeiras, polidoras ou soldadoras.
- **Sensores:** São elementos que fornecem as informações, parâmetros externos aos elementos de controle. Leva ao conhecimento do controlador o estado de cada junta, (posição, velocidade, aceleração).
- **Juntas:** As juntas em manipuladores robóticos são essenciais para permitir o movimento em várias direções e constituem uma das partes fundamentais da cinemática do robô [37]. A mobilidade do manipulador resulta de uma série de movimentos elementares e independentes, conhecidos como graus de liberdade (GL/DOF). A presença de diferentes tipos de juntas permite ao manipulador robótico

ajustar-se a diversas posições e executar uma ampla variedade de tarefas. A figura 1.3 ilustra os tipos de juntas.

Junta rotativa – gira em torno de linha imaginaria estacionária chamada de eixo de rotação.

Junta prismática – move em linha reta. Translação relativa entre dois elementos.

Junta esférica – Funciona com a combinação de três juntas de rotação, realizando a rotação em torno de três eixos.

Junta cilíndrica – é composta por duas juntas, uma rotacional e uma prismática;

Junta planar – é composta de duas juntas prismáticas, realiza movimento em duas direções.

Junta parafuso – é constituída de um parafuso que contém uma porca ao qual executa um movimento semelhante ao da junta prismática, com movimento do parafuso.

Essas juntas conferem ao manipulador robótico a capacidade de se mover com precisão no espaço tridimensional, possibilitando a execução de tarefas variadas e adaptando-se a diferentes configurações e necessidades de operação.

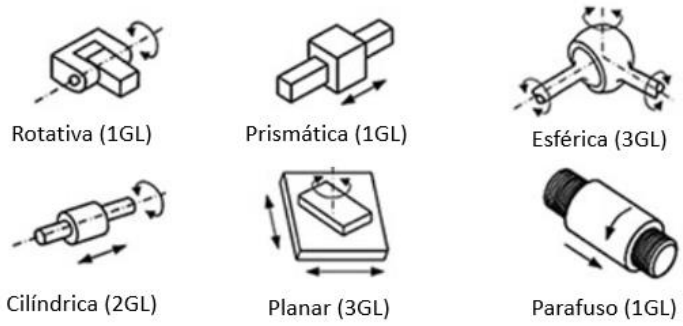

Figura 2.2 - Tipos de juntas utilizadas em robôs. Fonte: [27].

O espaço de trabalho de um robô manipulador é definido como o volume total que seu efetuador pode alcançar ao executar todos os movimentos possíveis. Esse espaço é limitado pelas restrições geométricas e mecânicas das juntas do robô, que podem afetar sua amplitude de movimento. Em manipuladores com seis graus de liberdade (DOF), sendo três para posicionamento e três para orientação, o robô é capaz de realizar qualquer tarefa dentro dos limites do seu espaço de trabalho, pois possui flexibilidade completa para alcançar qualquer ponto e orientar-se em qualquer direção nesse volume. Em contrapartida, um manipulador com menos de seis graus de liberdade não consegue alcançar todos os pontos possíveis no espaço

tridimensional, limitando sua capacidade de movimentação e tornando-o menos versátil para tarefas complexas [42].

A configuração do espaço de trabalho é um fator crítico no planejamento e controle de robôs, especialmente em aplicações industriais onde a precisão e a adaptabilidade são essenciais. O design adequado das juntas e dos elos garante que o manipulador possa operar eficientemente dentro do espaço de trabalho necessário para a aplicação, maximizando o alcance e a eficácia em tarefas como montagem, manuseio de materiais e operações de *pick-and-place*.

2.1.2. Planejamento de movimento de um robô manipulador

O planejamento de movimento é uma etapa essencial na robótica, especialmente para manipuladores robóticos que precisam executar tarefas com precisão em um ambiente tridimensional [41]. Essa etapa consiste em definir o conjunto de movimentos necessários para que um robô vá de uma posição inicial a uma posição-alvo, evitando obstáculos e respeitando restrições físicas e de segurança. No contexto de robôs colaborativos, como braços robóticos em ambientes industriais, o planejamento de movimento deve ser tanto eficiente quanto adaptável, garantindo que o robô atue de forma segura e colaborativa ao lado de operadores humanos.

Uma das tarefas centrais no planejamento de movimento é resolver a cinemática inversa, que consiste em determinar os ângulos das juntas de um robô para que ele alcance uma posição e orientação específicas no espaço. Esse problema é essencial, pois controla como o manipulador deve ajustar suas juntas para que seu *"end-effector"* (ferramenta ou garra) atinja um alvo específico ou levar o efetuador para determinados pontos do plano cartesiano, respeitando as restrições de movimento do robô [43]. A resolução da cinemática inversa pode ser abordada por métodos analíticos e numéricos, cada um com suas vantagens e desafios específicos.

2.1.3. Operações de pick-and-place

Técnicas de otimização para operações de *pick-and-place* têm sido amplamente estudadas ao longo dos anos, com o objetivo de melhorar a eficiência e precisão desse tipo de tarefa. Operações de *pick-and-place* referem-se à movimentação de objetos por meio de manipuladores robóticos, abrangendo atividades como transporte de peças, montagem de componentes e organização de produtos em linhas de produção. Esses processos são

fundamentais em ambientes industriais, onde a precisão, velocidade e repetibilidade são essenciais para garantir a produtividade e a qualidade dos produtos.

Para definir uma operação *pick-and-place*, [44] descrevem dois problemas fundamentais inerentes a essas técnicas, separando-os na forma mais eficiente:

1. Sensorizar, detectar e classificar os objetos a serem manipulados;
2. *Gripping*, que envolve a atuação de uma garra robótica capaz de coletar peças.

Para o processo de *pick-and-place* vários autores desenvolveram métodos para esta operação, como [45], que desenvolveu um algoritmo em tempo real, usando processamento de imagem de várias câmeras estacionárias. Estratégias de *gripping* também foram aprimoradas por diversos pesquisadores. Em [46] propuseram uma estratégia dividida em três passos:

1. Alinhamento do *gripper* com o objeto no plano da imagem capturada por um sistema de câmera;
2. Alinhamento do *gripper* com a peça na direção normal ao plano da imagem;
3. Fecho do *gripper* até a peça estar segura pelo mesmo. [46] realizou esta etapa para micropeças, mas pode ser estendida para todas as operações.

Para além dos dois passos principais do processo de pick-and-place (1. Sensorização, deteção e classificação dos objetos a serem manipulados; e (2. *Gripping)* é essencial considerar o controle cinemático para garantir o posicionamento preciso do manipulador. A resolução da cinemática inversa é crucial para calcular a sequência de movimentos das juntas do robô, permitindo que ele atinja a posição desejada de forma eficiente e segura. Isso assegura que o robô seja capaz de adaptar sua trajetória e realizar a operação de *pick-and-place* com precisão, mesmo em cenários dinâmicos e ambientes industriais complexos.

2.2. Robô UR5E

Robôs industriais colaborativos, como o UR5e da *Universal Robots*, destacam-se por sua flexibilidade em operações com ou sem interação humana. O UR5e possui um design leve e compacto, com seis graus de liberdade, permitindo movimentos ágeis e precisos. As articulações podem girar até 360°, e a última junta é ilimitada, aumentando a versatilidade em operações complexas [28].

O UR5e é conhecido por sua interface de programação intuitiva, que permite a criação de rotinas de movimento de maneira visual e fácil de entender, mesmo para usuários sem

experiência em robótica. Sua central de controle oferece conectividade Ethernet, permitindo comunicação com outros dispositivos, como computadores e sensores externos. Essa conectividade facilita a integração do UR5e em sistemas de automação e monitoramento, tornando-o ideal para aplicações em ambientes industriais modernos.

Seu ambiente de desenvolvimento suporta várias linguagens de programação como *python*, que é amplamente utilizado para visão computacional e C/C++, além de possuir a integração com o framework ROS onde possibilita a criação de nós ROS, onde cada nó pode ser responsável para executar uma determinada operação, podendo os nós compartilharem recursos e comunicarem entre si, além desse framework está sempre em constante atualização pela comunidade. A figura 2.2 ilustra o modelo de robô utilizado neste trabalho.

Figura 2.3 – Robô colaborativo com módulo *pick-and-place* da UR5. Fonte: [28].

Além disso, a capacidade de colaborar com operadores humanos torna o UR5e uma escolha segura para diversas tarefas no chão de fábrica, incluindo operações de *pick-and-place*, montagem e manipulação de peças. A presença de sensores avançados de força e torque na estrutura do Cobot permite ajustes automáticos em resposta a alterações no ambiente de trabalho, garantindo segurança e eficiência durante a execução das tarefas.

2.2.1. Sistema de coordenadas e orientação do robô UR5e

O robô UR5e utiliza sistemas de coordenadas cartesianas para definir poses (posição e orientação) no espaço de trabalho. Esses sistemas de coordenadas são estabelecidos na base do robô (referencial de base) e no centro da ferramenta (referencial da ferramenta). A posição da ferramenta em relação à base é definida pelas coordenadas *X*, *Y* e *Z*, enquanto a orientação é representada pelo vetor-ângulo, que é uma forma eficiente de representar rotações no espaço tridimensional, evitando problemas numéricos comuns em matrizes de rotação.

O conceito de orientação por vetor-ângulo é baseado no teorema de rotação de corpos rígidos de Euler, que estabelece que a orientação de um corpo rígido pode ser representada por uma rotação em torno de um vetor unitário fixo, que serve como eixo de rotação. A Figura 2.3 ilustra como o eixo de rotação inicial (em azul) se transforma após a rotação de θ graus ao redor do vetor u, resultando no eixo final (em vermelho).

Essa abordagem é útil em aplicações robóticas, especialmente para controlar a orientação do ponto central da ferramenta (Tool Center Point, TCP) e é particularmente útil para contornar problemas numéricos que ocorrem em algumas configurações singulares ao utilizar matrizes de rotação para definir a orientação.

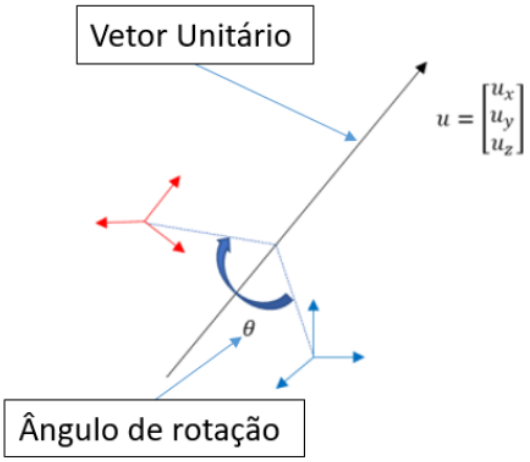

Figura 2.4 – Sistema de orientação vetor-ângulo. Fonte: adaptado de [38].

No caso do UR5e, definindo-se como eixo de referência a base do robô, a orientação da ferramenta é definida com apenas três parâmetros, rx, ry e rz, ou seja, o vetor de rotação é calculado a partir das componentes espaciais x, y e z do vetor unitário u e de sua rotação θ:

$$r = \begin{bmatrix} rx \\ ry \\ rz \end{bmatrix} = \begin{bmatrix} \theta u_x \\ \theta u_y \\ \theta u_z \end{bmatrix} \tag{2.1}$$

Apesar de contornar problemas numéricos, o método do vetor de rotação é de certa forma mais abstrato, já que é mais complexo correlacionar a disposição física de elementos no espaço com os valores numéricos dos vetores de rotação. Sendo assim, a fim de tornar o cálculo do ângulo de rotação do robô menos abstrato, utiliza-se um sistema de orientação baseado em matriz de rotação. Esse sistema de orientação é baseado na definição de ângulos de rotação em torno dos eixos cartesianos do robô z, y e x, respectivamente α, β, e γ, tendo como origem a base do Cobot. Isso permite calcular a matriz de rotação completa como o produto das rotações

em cada eixo, possibilitando o controle preciso do TCP. Essa representação é vantajosa para implementar a cinemática direta e inversa do manipulador. A Figura 2.4 ilustra a definição dos ângulos α, β, γ em um manipulador robótico.

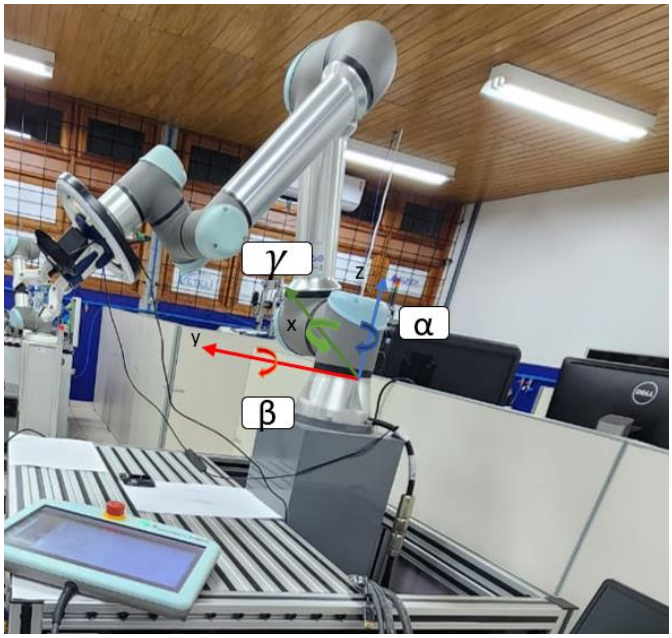

Figura 2.5 – Representação das rotações em torno dos eixos no UR5. Fonte: Autor.

Pode-se observar que uma rotação de um ângulo α com relação ao eixo z da base do robô corresponde a uma rotação dos eixos X e Y do TCP. Essa rotação pode ser representada pela seguinte matriz:

- Rotação ao redor do eixo z por um ângulo α:

$$R_z(\alpha) = \begin{bmatrix} cos\alpha & -sen\alpha & 0 \\ sen\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.2)$$

Consequentemente para os eixos x e y tem-se as seguintes matrizes:

- Rotação ao redor do eixo x por um ângulo γ:

$$R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -sen\gamma \\ 0 & sen\gamma & cos\gamma \end{bmatrix} \quad (2.3)$$

- Rotação ao redor do eixo y por um ângulo β:

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & sen\beta \\ 0 & 1 & 0 \\ -sen\beta & 0 & cos\beta \end{bmatrix} \quad (2.4)$$

A matriz de rotação completa, R, é o produto das rotações individuais. Sendo assim:

$R(\alpha, \beta, \gamma) = R_z(\alpha) * R_y(\beta) * R_x(\gamma)$

Chegando-se a:

$$R(\alpha, \beta, \gamma) = \begin{bmatrix} cos\alpha.cos\beta & cos\alpha.sen\beta - sen\alpha.cos\,\gamma & cos\alpha.sen\beta.cos\gamma + sen\alpha.sen\gamma \\ sen\alpha.cos\beta & sen\alpha.sen\beta + cos\alpha.cos\gamma & sen\alpha.sen\beta.cos\gamma + cos\alpha.sen\gamma \\ -sen\beta & cos\beta.sen\gamma & cos\beta.cos\gamma \end{bmatrix} \quad (2.5)$$

As três componentes de orientação *rx*, *ry* e *rz* da ferramenta do robô UR5e podem ser definidas a partir da sua matriz de rotação R (α, β, γ) da seguinte forma:

$$\theta = cos^{-1}\left(\frac{R_{11} + R_{22} + R_{33} - 1}{2}\right) \quad (2.6)$$

$$w = \frac{1}{2sen\theta}\begin{bmatrix} R_{32} - & R_{23} \\ R_{13} - & R_{31} \\ R_{21} - & R_{12} \end{bmatrix} \quad (2.7)$$

$$rx = W_1 = \frac{1}{2sen\theta}(R_{32} - R_{33}) * \theta \quad (2.8)$$

$$ry = W_2 = \frac{1}{2sen\theta}(R_{13} - R_{31}) * \theta \quad (2.9)$$

$$rz = W_3 = \frac{1}{2sen\theta}(R_{21} - R_{12}) * \theta \quad (2.10)$$

Dessa forma tem-se a relação entre os inputs que determinam a orientação do TCP do robô *rx, ry* e *rz*, com os valores dos ângulos de rotação nos eixos de cada eixo do UR5 [47].

2.3. Cinemática

A Cinemática é a ciência que estuda o movimento das estruturas, focando na análise de posição, velocidade e aceleração, sem considerar as forças ou torques que o provocam. A cinemática envolve todas as propriedades geométricas relacionadas ao tempo, abrangendo a posição e orientação das ligações dos manipuladores em condições estáticas. Diferentemente da cinemática, a dinâmica do movimento leva em consideração as interações de forças e torques que afetam a estrutura do robô.

A cinemática inversa é uma tarefa essencial para o controle de posição de qualquer robô e para o desenvolvimento de simuladores. Ela consiste em calcular as coordenadas das articulações que correspondem a uma determinada configuração do *"end-effector"* (posição e orientação desejadas no espaço). Por outro lado, a cinemática direta envolve determinar a posição e orientação do *"end-effector"* a partir dos ângulos atuais das juntas do manipulador. Ambas as cinemáticas podem ter múltiplas soluções matemáticas, especialmente em robôs com seis ou mais graus de liberdade, exigindo uma escolha da solução mais adequada para evitar colisões e maximizar a eficiência.

Um robô manipulador é modelado como uma cadeia de corpos rígidos, chamados de elos, que estão interconectados por articulações. Essa cadeia começa em uma base fixa e termina no *"end-effector",* que é a parte móvel do sistema. O objetivo é controlar tanto a posição quanto a orientação do *"end-effector"* no espaço de trabalho. Para isso, uma trajetória é planejada e programada, envolvendo o ajuste das variáveis de junta para alcançar o movimento desejado. A cinemática direta e inversa estabelece a relação matemática entre as variáveis das juntas, a posição e a orientação do *"end-effector"* conforme representado na figura 2.6, permitindo que o robô manipule objetos de maneira precisa no espaço tridimensional [32].

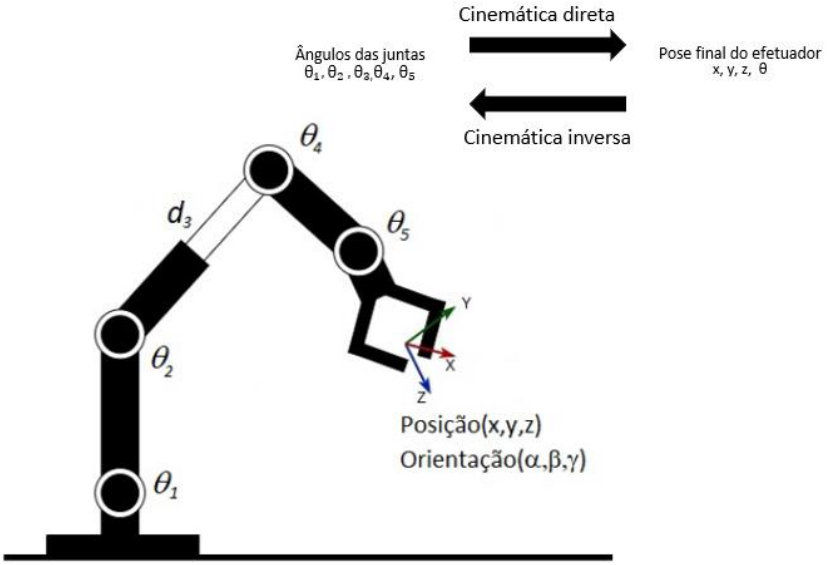

Figura 2.6 – Representação das transformações das juntas e posição de um robô manipulador. Fonte: Adaptado de [5].

2.3.1. Cinemática direta

A cinemática direta é responsável por determinar a posição e a orientação do *"end-effector"* com base nas variáveis das juntas do manipulador, como os ângulos das juntas rotacionais e os deslocamentos das juntas prismáticas. Esse cálculo é feito em relação a um sistema de coordenadas de referência, geralmente a base do robô [32]. A cinemática direta descreve o movimento do manipulador sem considerar as forças ou torques que o causam, focando apenas na geometria do sistema.

No caso do UR5e, a cinemática direta é modelada por meio de uma sequência de transformações homogêneas, que representam as rotações e translações associadas a cada elo do manipulador. A matriz de transformação homogênea T é usada para definir a posição e orientação de um ponto no espaço 3D em relação a um sistema de coordenadas de referência. A matriz é composta por uma matriz de rotação R e um vetor de translação d:

$$T = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix} \tag{2.11}$$

Onde:

- R é a matriz de rotação $3x3$ que representa a orientação do *"end-effector"* em relação ao sistema de coordenadas de base.

- d é o vetor de translação $3x1$ que representa a posição do *"end-effector"* nas coordenadas $x, y\ e\ z$.

Essa abordagem permite calcular a posição final do *"end-effector"* de maneira precisa, utilizando a seguinte fórmula geral:

$$T_{total} = T_1 * T_2 * T_3 * T_4 * T_5 * T_6 \quad (2.12)$$

Onde T_i é a matriz de transformação homogênea associada a cada junta i, incorporando tanto a matriz de rotação quanto o vetor de translação do elo correspondente. Essa matriz final, T_{total}, contém a posição (coordenadas x, y, z) e a orientação (representada pela matriz de rotação) do TCP no espaço tridimensional.

Portanto, a cinemática direta consiste em encontrar a matriz da equação (2.5), a fim de obter as informações do efetuador com base nas posições das juntas. Entretanto, esses cálculos são trabalhosos e intensificam-se à medida que o manipulador apresenta mais juntas, pois torna-se necessário encontrar uma matriz de transformação homogênea para cada elo e multiplicá-las.

Diante disso, existem algumas convenções que tornam a obtenção das matrizes de transformação homogênea um processo sistemático, sendo a notação de Denavit-Hartenberg a mais utilizada.

Convenção de Denavit-Hartenberg

A Transformação homogênea é utilizada para descrever a relação entre o plano de coordenadas $o_i x_i y_i z_i\ para\ i \in \{0, \dots, n\}$ que são atribuídos aos elos e juntas do robô. Para definir uma movimentação rígida, seis parâmetros são necessários: três parâmetros para definir a rotação e três parâmetros para definir a translação. A convenção de DH reduz o número de parâmetros que são necessários para definir uma transformação homogênea de seis para quatro, se aproveitando da geometria em comum dos manipuladores e fazendo escolhas inteligentes para a origem e os eixos de coordenada. A convenção de DH é o procedimento padrão usado na derivação da cinemática direta de manipuladores industriais.

Na convenção de DH cada transformação homogênea A_i é um produto de quatro transformações básicas [32]:

$$A_i = Rot_{z_i,\theta_i}.Trans_{z_i,d_i}.Trans_{x_i,a_i}.Rot_{x_i,\alpha_i}=\begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}c_{\alpha_i} & a_i c_{\alpha_i} \\ -s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.13)$$

Em que:

- $\boldsymbol{Rot_{z,\theta_i}}$ *:* representa uma rotação θ_i em torno do eixo z_i;
- $\boldsymbol{Trans_{z,d_i}}$: representa um deslocamento d_i ao longo do eixo z_i;
- $\boldsymbol{Trans_{x,a_i}}$: representa um deslocamento a_i ao longo do eixo x_i;
- $\boldsymbol{Rot_{x,\alpha_i}}$: representa uma rotação α_i em torno do eixo x_i.

Assim tem suas respectivas matrizes:

$$Rot_{x_i,\theta_i}=\begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.14)$$

$$Trans_{z_{n-1}}(d_n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.15)$$

$$Trans_{x_n}(r_n) = \begin{bmatrix} 1 & 0 & 0 & r_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.16)$$

$$Rot_{x_n}(\alpha_n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.17)$$

Onde os quatro parâmetros θ_i, a_i, d_i, α_i são ângulo da junta, comprimento do elo, distância do elo e torção da junta j_i, e elo l_i,. Como A_i, é uma função de θ_i, os outros três parâmetros são constantes. De acordo com a convenção DH, os planos de coordenadas $o_i x_i y_i z_i$ atribuídos a junta j_i e ao elo l_i devem possuir os seguintes atributos:

1. O Eixo z_i deve ser o eixo de revolução da junta j_{i+1}.

2. O Eixo x_i deve ser perpendicular aos eixos z_{i-1} e z_i.
3. O eixo x_i deve interceptar o eixo z_{i-1}.
4. Todos os planos de coordenadas devem seguir a regra da mão direita.

Na Figura 2.7, apresenta-se uma interpretação visual das 4 matrizes da equação (2.13). A partir dessa interpretação pode-se obter os parâmetros θ_i, d_i, a_i e α_i que são chamados de parâmetros membros ou parâmetros DH, os quais recebem as seguintes denominações:

- θ_i é o ângulo da junta i, este ângulo é medido entre θ_{i-1} e $x_i$em um plano normal ao z_{i-1};
- d_i é o elo de deslocamento, encontrado a partir da distância perpendicular a partir da origem da junta anterior o_{i-1} à intersecção do eixo x_i com z_{i-1}, medido ao longo do eixo z_{i-1};
- a_i é o elo de comprimento, encontrado a partir da distância entre os eixos z_{i-1} e z_i, e é medida ao longo do eixo x_i;
- α_i é o elo de torção, encontrado a partir do ângulo entre os eixos z_{i-1} e z_i, medido em um plano normal a x_i. O sentido positivo para α é determinado a partir de z_{i-1} a z_i pela regra da mão direita.

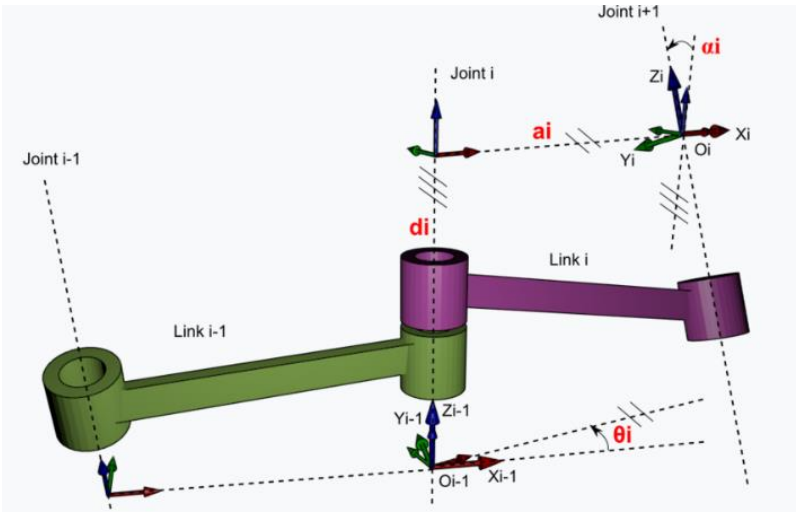

Figura 2.7 – Representação dos parâmetros de DH. Fonte: [48].

A Figura 2.7 demonstra o uso dos parâmetros de DH com dois eixos de coordenadas de exemplo.

Um conjunto de n matrizes de transformação homogêneas, denotadas por $A_1, A_2, \ldots, A_n$, é obtido a partir da substituição dos parâmetros na Equação (2.13). Essas matrizes são fundamentais para calcular a cinemática direta do manipulador.

Essas matrizes de transformação são combinadas para formar uma matriz única, que permite determinar a posição e orientação do *"end-effector"* para qualquer configuração das variáveis de junta. Dessa forma, ao obter a posição no espaço de trabalho do manipulador com base nos valores das variáveis das juntas, é possível representar o manipulador como um ponto no espaço de configurações, abstraindo suas características físicas.

Para calcular a cinemática direta, as matrizes de transformação homogêneas $A_1, A_2, \ldots, A_n$ são aplicadas sequencialmente na Equação (2.12). A matriz de transformação resultante descreve a posição e orientação do *"end-effector"* em relação à base do manipulador, como uma função das variáveis das juntas q_i, resolvendo assim o problema da cinemática direta.

2.3.2. Cinemática inversa

A cinemática inversa consiste em calcular os ângulos das juntas necessários para que o *"end-effector"* alcance uma posição e orientação específicas. No caso do UR5e, esse cálculo é particularmente complexo, pois o robô possui seis graus de liberdade, o que permite múltiplas configurações das juntas para alcançar a mesma posição final do *"end-effector"*. Além disso, há restrições impostas pelos limites dos ângulos das juntas. Em aplicações práticas, são frequentemente aplicados critérios de seleção para escolher a solução mais adequada, como a proximidade em relação à posição atual do *"end-effector"* ou a ausência de obstáculos ao longo do trajeto até o destino. Em manipuladores mais complexos, com muitos parâmetros DH diferentes de zero, o número de soluções possíveis pode ser ainda maior. A resolução da cinemática inversa pode ser feita iterativamente, utilizando métodos numéricos ou analíticos:

- Métodos Analíticos: São aplicados em robôs cuja configuração permite obter soluções fechadas para a cinemática inversa, oferecendo uma resolução direta das equações que relacionam os ângulos das juntas à posição e orientação do *"end-effector"* (TCP). Dentro dessa categoria, distinguem-se dois métodos principais: o método algébrico e o geométrico. O método algébrico utiliza manipulações matemáticas para resolver as equações derivadas da cinemática direta, enquanto o método geométrico faz uso intensivo de técnicas de geometria plana para encontrar as soluções. Em ambos os casos, as expressões envolvidas são geralmente transcendentais, ou seja, compostas por funções como de $sen(\theta)$ e $cos(\theta)$, em vez de funções lineares de θ.

- Métodos Numéricos: Para situações em que uma solução analítica não é viável, são aplicados métodos como gradiente descendente com pseudo-inversa da matriz Jacobiana ou Levenberg-Marquardt, que iterativamente ajustam os ângulos das juntas até alcançar a pose desejada. Esses métodos são bem mais custosos do ponto de vista computacional, devido a sua natureza iterativa, e são bem mais lentas do que as versões correspondentes de forma fechada porem não sofrem com problemas de singularidade [51].

Para os algoritmos de planejamento e outras análises do espaço de trabalho, resolver o problema inverso, que consiste em converter as coordenadas desejadas do ambiente para os ângulos das articulações, é crucial. Embora a cinemática inversa em geral busque calcular todos os ângulos das articulações, q_{arm} , que posicionam um elo em uma translação e orientação específicas, $T = [\,R\ t]$, as abordagens analíticas frequentemente falham ao lidar com as singularidades, resultando em soluções subótimas [33]. Devido a essas limitações, diversos métodos numéricos, como o gradiente descendente baseado em $\frac{\delta T}{\delta q}$, foram desenvolvidos. A maioria da literatura sobre cinemática inversa generalizada tende a focar nesses métodos numéricos, em parte porque resolver o problema analiticamente exige um esforço considerável e nem sempre é viável [49].

2.4. Matrizes jacobiana e singularidades

Quando o determinante da matriz Jacobiana é zero, o sistema de equações que relaciona as forças aplicadas na plataforma móvel com as forças nos atuadores se torna insolúvel. Isso indica que a rigidez da plataforma é zero em uma direção específica ou que a tensão em uma das ligações do robô se torna infinita [50]. Em manipuladores robóticos, é fundamental compreender as configurações singulares, pois essas configurações limitam os movimentos possíveis do braço e resultam em perda de graus de liberdade. Na prática, sinais de singularidade incluem a incapacidade do manipulador de se mover em uma determinada direção ou o movimento extremamente rápido de algumas articulações em resposta a pequenos deslocamentos, o que pode comprometer o controle e a estabilidade do sistema.

Uma singularidade ocorre sempre que *det(J) = 0*. Quando o Jacobiano perde um rank, ou seja, uma linha se torna nula, um ponto de singularidade ocorre. Há dois tipos de singularidades, são as singularidades internas e de limites. As singularidades de limites surgem tipicamente quando o braço robótico é esticado para seu tamanho máximo perdendo assim

liberdade de se mover para qualquer uma direção. As singularidades internas não costumam ser simples, seus pontos são obtidos pela resolução do determinante.

Há três tipos de principais de singularidade: singularidades no pulso, no ombro e no cotovelo, conforme representados pela figura 2.8.

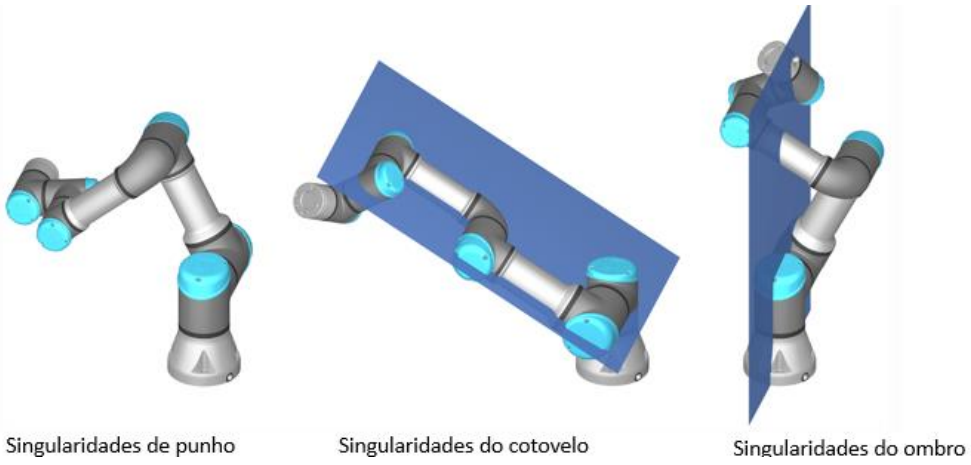

Figura 2.8 – Tipos de singularidade no sistema colaborativo. Fonte: [52].

- Singularidades do punho: Ocorrem quando os eixos das articulações 4 e 6 ficam paralelos e podem fazer com que essas articulações girem 180 graus instantaneamente. Nos robôs UR, isso corresponde a $\theta 5 = 0°$, $\theta 5 = \pm 180°$ ou $\theta 5 = \pm 360°$;

- Singularidades do cotovelo: Ocorrem quando os eixos das articulações 2, 3 e 4 são coplanares, fazendo com que o cotovelo trave na posição. Nos robôs UR, isso corresponde a $\theta 3 = 0°$.

- Singularidades do ombro: Ocorrem quando o ponto de interseção dos eixos das articulações 5 e 6 se alinha com o plano que passa pelos eixos das articulações 1 e 2, fazendo com que as articulações 1 e 4 girem 180 graus instantaneamente.

2.5. Métodos numéricos

Os métodos numéricos para a resolução da cinemática inversa são amplamente utilizados quando não há uma solução analítica direta ou quando o manipulador possui uma estrutura complexa que inviabiliza a obtenção de expressões fechadas para os ângulos articulares. Esses métodos iterativos buscam minimizar o erro entre a pose desejada e a alcançada pelo robô, ajustando as variáveis articulares a cada iteração. Entre os algoritmos numéricos mais utilizados, destacam-se o Gradiente Descendente, que ajusta as juntas com base na direção do gradiente da função de erro; o BFGS e o L-BFGS-B, métodos quasi-Newton que

utilizam aproximações da matriz Hessiana para otimizar a convergência e lidar com restrições de juntas; e o Nelder-Mead, um método baseado em simplex que não utiliza derivadas e é útil em problemas de difícil modelagem diferencial. Esses métodos permitem maior flexibilidade para a resolução da cinemática inversa, tornando-os viáveis para aplicações em que soluções analíticas são inviáveis ou inexistentes.

2.5.1. Gradiente descendente

Em robótica, o método do gradiente descendente, frequentemente referido como método de transposição jacobiana, é uma abordagem numérica para resolver um conjunto de n equações algébricas não lineares com n incógnitas. Este método foi aplicado à cinemática inversa por Balestrino [34] e Wolovich [35] em 1984. Baseia-se no algoritmo original de otimização de descida de gradiente, porém com ajustes específicos para adaptá-lo aos desafios da robótica.

O método do gradiente descendente utiliza apenas a derivada de primeira ordem da expansão de Taylor e pode ser geralmente formulado conforme a equação (2.18), onde n é a n-ésima iteração, x a variável de otimização, $f(x_n)$ o gradiente da função de otimização, e $\propto$ a taxa de aprendizado, um parâmetro ajustável do algoritmo.

$$x_{n+1} = x_n - \propto \nabla f(x_n) \tag{2.18}$$

O gradiente descendente aborda o problema da cinemática inversa ajustando iterativamente os ângulos das articulações para minimizar o erro entre as posições desejada e real do efetuador final. Para realizar isso, o método emprega a transposição da matriz jacobiana, que mapeia o erro de posição entre a localização atual e a desejada do efetuador final no espaço dos ângulos das articulações. Para fins de cinemática inversa, x_n e x_{n+1} denotariam ângulos de articulação, enquanto o resíduo, $r(x)$, denotaria um vetor de erro para a pose atual e a pose desejada.

O pseudocódigo do método do gradiente descendente, também conhecido como método de transposição jacobiana é fornecido no algoritmo abaixo.

Algoritimo - Gradiente Descendente

```
goal_pose = y
q = current joint angles
step_size = desired step size
tolerance = set tolerance
e = goal_pose - current_pose
while norm(e) >= tolerance do
    J = Jacobian(q)                    // Compute Jacobian with method
```

```
    J_T = J.transpose()                                  // Compute Jacobian transpose
    gradient = alpha * J_T * e
    q += step_size * gradient
    q = check_joint_limits(q)              // Adjust to max/min value if above/under limit
    e = goal_pose - ForwardKinematics(q)               // Compute pose with FK method
end while
```

2.5.2. Métodos Quase-Newton

Para compreender os métodos Quase-Newton, é importante primeiro entender o método de Newton. O método de Newton, também conhecido como método de Newton multidimensional, busca minimizar uma função $f: \mathbb{R}^n \rightarrow \mathbb{R}$ aproximando-a localmente por uma função quadrática e, em seguida, minimizando essa função aproximada. Essa abordagem utiliza uma expansão em série de Taylor da função f até a segunda ordem, mostrada na equação (2.19), o que permite construir uma representação quadrática próxima da função original ao redor de um ponto inicial. O método de Newton é conhecido por sua convergência quadrática, ou seja, ele converge rapidamente para a solução. No entanto, essa rapidez de convergência depende fortemente da escolha de uma boa estimativa inicial para a solução, pois uma estimativa distante pode comprometer a eficiência do método. Com essa aproximação quadrática, o método de Newton utiliza informações tanto do gradiente quanto da matriz Hessiana de f para calcular as atualizações na direção que minimiza a função de forma mais e eficiente [54].

$$f(x) \cong q(x) = f(x_k) + \nabla f^T(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k), \tag{2.19}$$

Em que $x \in \mathbb{R}^n, \nabla f(x_k)$ são o vetor gradiente de f no ponto $x_k \in \mathbb{R}^n$ e $\nabla^2 f(x_k)$ é a matriz Hessiana de $f$no ponto x_k. A condição necessária para existência de ponto mínimo é que

$$\nabla f(x) = 0 \tag{2.20}$$

Assim, considerando a aproximação quadrática, temos,

$$\nabla q(x) = 0 \tag{2.21}$$

Com isso, temos que,

$$\nabla q(x) = \nabla f(x_k) + \nabla^2 f(x_k)(x - x_k) = 0, \tag{2.22}$$

$$\nabla^2 f(x_k)(x - x_k) = -\nabla \mathrm{f}(x_k) \tag{2.23}$$

Logo, a expressão que define o método Newton é dada por:

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla \mathrm{f}(x_k), \tag{2.24}$$

Sendo que o critério de parada do método pode ser estabelecido como:

$$\big|\big|\nabla f(x_k)\big|\big| < \epsilon, \tag{2.25}$$

Definindo ϵ como um erro pré-estabelecido.

A convergência do método de Newton é garantida quando a matriz Hessiana é definida positiva e não singular, condições que devem ser verificadas em cada iteração do método. No entanto, para melhorar a eficiência e garantir a convergência em casos onde essas condições não são plenamente atendidas, algumas modificações podem ser implementadas. Uma dessas modificações é a introdução de um parâmetro de busca β_k, que é ajustado por meio de uma busca unidimensional. Esse parâmetro controla o tamanho do passo dado na direção calculada pelo método de Newton, permitindo um ajuste mais refinado e garantindo que cada iteração leve a uma redução consistente na função objetivo, assim:

$$\mathrm{minimizar}_{\beta} f(x_k + \beta d_k) \tag{2.26}$$

Assim, o método de Newton com busca unidimensional é dado por:

$$x_{x+1} = x_k + \alpha_k d_k, \tag{2.26}$$

sendo $d_k = -(\nabla^2 f(x_x))^{-1} \nabla f(x_k)$ uma direção de descida.

O algoritmo do método Newton para a minimização do f é representado abaixo.

Algoritimo – Método de Newton

Dados: $\boldsymbol{f}$: $\mathbb{R}^{\boldsymbol{n}} \rightarrow \mathbb{R}$ continuamente diferençável, $\boldsymbol{\epsilon}$ a precisão desejada e $\boldsymbol{x_0}$o ponto inicial
K-> 0;
Output: Ponto de mínimo;
while $\left|\left|\nabla \boldsymbol{f}(\boldsymbol{x_k})\right|\right| \geq \boldsymbol{\epsilon}$
$\boldsymbol{d_k} \leftarrow [\nabla^{\boldsymbol{2}} \boldsymbol{f}(\boldsymbol{x_k})]^{-\boldsymbol{1}} \nabla \boldsymbol{f}(\boldsymbol{x_k})$;
$\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \boldsymbol{d_k}$;
$\boldsymbol{k} \leftarrow \boldsymbol{k} + \boldsymbol{1}$;
end while
Return $\boldsymbol{x_k}$

Os métodos Quase-Newton, em vez de calcular diretamente a inversa da matriz Hessiana, como é feito no método de Newton, aproximam essa inversa de forma iterativa e eficiente, utilizando apenas derivadas de primeira ordem. Esses métodos são considerados entre os mais sofisticados para resolver problemas de otimização irrestrita, pois evitam o alto custo computacional associado à avaliação e inversão da matriz Hessiana. A principal vantagem dos métodos Quase-Newton é sua capacidade de construir uma aproximação da inversa da Hessiana com base apenas nas informações da função erro, reduzindo significativamente o esforço computacional enquanto mantém uma boa precisão na direção de descida.

2.5.3. BFGS (Broyden-Fletcher-Goldfarb-Shanno)

O algoritmo de projeção de gradiente Broyden-Fletcher-Goldfarb-Shanno (BFGS) é um método quase-Newton que usa os gradientes da função de custo de iterações passadas para gerar informações aproximadas da segunda derivada. O algoritmo usa essas informações de segunda derivada para determinar a etapa a ser executada na iteração atual [53]. Um método de projeção de gradiente é usado para lidar com os limites da função de custo criados pelos limites das articulações do modelo do robô. A direção calculada é modificada para que a direção da pesquisa seja sempre válida. O objetivo do método BFGS na cinemática inversa é minimizar uma função de custo que representa o erro entre a posição e orientação desejadas do *"end-effector"* e a posição e orientação atuais, dadas pelos ângulos das juntas $\theta = [\theta_1, \theta_2, \ldots, \theta_6]$.

O método BFGS para cinemática inversa são utilizados os seguintes passos:

1. **Definir a Função de Custo**: A função de custo f(θ) mede a diferença entre a posição e orientação desejadas e a posição e orientação atuais do *"end-effector"*. A função de custo pode ser formulada como:

$$f(\theta) = ||p(\theta) - [p_{target}||^2 + R(\theta) - R_{target}||^2 \quad (2.15)$$

onde $p(\theta)\ e\ R(\theta)$ são a posição e matriz de rotação atuais calculadas pela cinemática direta usando θ e p_{target}, e R_{target} são a posição e orientação desejadas.

2. Inicializar Valores: geralmente utilizando uma estimativa inicial $\theta^{(0)}$para os ângulos das juntas e uma matriz hessiana aproximada $H^{(0)}$ (geralmente, a matriz identidade *I*).
3. Iteração:

- Calcule o gradiente da função de custo $\nabla f(\theta)$, que representa a direção do maior aumento da função de custo em relação aos ângulos das juntas. O gradiente pode ser calculado numericamente ou simbolicamente.
- Use o atualizador BFGS para ajustar a matriz Hessiana aproximada. A atualização é dada por:

$$H^{(k+1)} = H^{(k)} + \frac{y^{(k)}(y^{(k)})^T}{(y^{(k)})^T s^{(k)}} - \frac{H^{(k)} s^{(k)} (s^{(k)})^T H^{(k)}}{(y^{(k)})^T H^{(k)} s^{(k)}} \quad (2.15)$$

Onde:

- $s^{(k)} = \theta^{(k+1)} - \theta^{(k)}$ é a mudança nos ângulos das juntas.
- $y^{(k)} = \nabla f(\theta^{(k+1)}) - \nabla f(\theta^{(k)})$ é a mudança do gradiente.

- Calcule a direção de atualização Δθ usando:

$$\Delta\theta = -H^{(k)} \nabla f(\theta) \quad (2.15)$$

- Atualize os valores das juntas:

$$\theta^{(k+1)} = \theta^{(k)} + \alpha\, \Delta\theta \quad (2.15)$$

onde α é o tamanho do passo, geralmente ajustado para garantir a convergência.

4. Critério de parada: O algoritmo é iterado até que o gradiente ∇f(θ) seja suficientemente pequeno, indicando que a função de custo está minimizada e que a posição e orientação do "*end-effector"* estão próximas do alvo.

O método BFGS é vantajoso para resolver a cinemática inversa de manipuladores com 6 graus de liberdade (DOF), pois oferece convergência rápida e maior estabilidade, especialmente em configurações complexas ou próximas aos limites das juntas. A atualização adaptativa da matriz Hessiana permite que o BFGS lide eficientemente com regiões de

múltiplos mínimos locais, onde outros métodos poderiam apresentar dificuldades. Esse método também é eficaz quando a estimativa inicial está distante da solução.

2.5.4. Limited Memory Quase-Newton L-BFGS-B

O L-BFGS-B é um algoritmo de memória limitada, esse método baseado em gradiente que tem como objetivo encontrar o valor ótimo de uma função, onde resolve grandes problemas de otimização não linear sujeitos a limites simples nas variáveis. Ele se destina a problemas nos quais é difícil obter informações sobre a matriz Hessiana ou a problemas densos de grande porte. O método faz parte da família dos métodos Quasi-Newton, e visa resolver o problema do custo computacional. L-BFGS é o único método Quasi-Newton possibilita lidar com limites nas variáveis [55].

Nesse método, ao invés de armazenar aproximações densas de dimensões *n* da matriz Hessiana associada ao problema, esse método armazena apenas alguns vetores de dimensão *n*. A ideia do método L-BFGS é armazenar informação de curvatura apenas das últimas iterações. Já que provavelmente serão as mais relevantes para a construção da aproximação do Hessiano do que as iterações mais antigas.

As primeiras m-1 iterações, o algoritmo do L-BFGS-B se equivale ao BFGS, com $H_k^0 = H_0^{BFGS}$. No entanto quando $m > \frac{n}{2}$, o L-BFGS se torna mais complexo que o BFGS. O algoritmo abaixo demostra a construção do método.

Algoritimo – Método L-BFGS-B

Entrada: $\boldsymbol{T_{desejado}}$: $\boldsymbol{Pose\ desejada\ do"end-effector"}$
$\boldsymbol{\theta^{(0)}}$: Estimativa dos ângulos das juntas
m: número de pares de gradiente e deslocamento a serem armazenados
Para k=0 até convergência faça
  Escolha $\boldsymbol{H_k^0}$
  $\boldsymbol{Calcule\ p_k \leftarrow -H_k \nabla f_k}$
  $\boldsymbol{\theta_{k+1} \leftarrow \theta_k + \alpha_k p_k}$
  Se $\boldsymbol{k > m}$,
    Descarte o par de vetores { $\boldsymbol{s_{k-m} > y_{k-m}}$ }
  Fim Se
  Calcule e armazene $\boldsymbol{s_k \leftarrow \theta_{k+1} - \theta_k, y_k \leftarrow \nabla f_{k+1} - \nabla f_k}$
  $\boldsymbol{k \leftarrow k + 1}$;
end while
Fim Para
Retorne $\boldsymbol{\theta_{final} = \theta^{(k+1)}}$

A estratégia de armazenar os últimos m pares de vetores (gradientes e deslocamentos) no método L-BFGS é particularmente eficiente na prática para robôs de 6 DOF, pois reduz

significativamente os requisitos de memória e o custo computacional ao calcular a cinemática inversa. Essa abordagem permite otimizar a trajetória do manipulador sem a necessidade de armazenar toda a matriz Hessiana. No entanto, essa técnica pode apresentar limitações em situações onde o problema é mal condicionado, como em configurações próximas a singularidades, onde os autovalores da matriz Hessiana aproximada são muito distantes entre si. Em tais casos, seria mais adequado armazenar apenas os pares de vetores que contribuem para a geração de uma matriz Hessiana bem condicionada, melhorando a estabilidade e precisão do algoritmo.

2.5.5. Nelder-Mead

O Método Nelder-Mead, também conhecido como método do simplex, é uma técnica de otimização numérica que busca minimizar uma função objetivo em múltiplas dimensões sem a necessidade de derivadas. Ele é particularmente útil para resolver problemas onde a função objetivo não é diferenciável, ou onde o cálculo de gradientes é inviável ou computacionalmente caro [63]. Por isso, ele pode ser utilizado para resolver o problema de cinemática inversa em manipuladores robóticos, como o UR5e.

No contexto da cinemática inversa, a função objetivo do Nelder-Mead pode ser definida como o erro entre a pose desejada (posição e orientação do efetuador final) e a pose alcançada a partir de uma estimativa inicial dos ângulos das juntas. O método utiliza um conjunto de vértices que formam um simplex no espaço das variáveis articulares (θ), adaptando iterativamente os vértices com operações como reflexão, expansão, contração e redução.

As principais etapas do método no problema de cinemática inversa incluem:

1. **Definição do Simplex Inicial**: Um conjunto inicial de soluções (ângulos das juntas) é gerado. Essas configurações servem como os vértices iniciais do simplex.
2. **Avaliação da Função Objetivo**: Para cada vértice, a função objetivo é avaliada, comparando a pose alcançada pela cinemática direta com a pose desejada.
3. **Atualização Iterativa**: O simplex é atualizado com base em operações de reflexão, expansão, contração ou redução, dependendo da qualidade das soluções encontradas.
4. **Convergência**: O método converge para uma configuração articular onde o erro entre a pose desejada e a alcançada é minimizado.

Uma vantagem significativa do método Nelder-Mead é sua simplicidade e a ausência de necessidade de gradientes, o que o torna adequado para problemas de cinemática inversa em

robôs com configurações complexas ou em situações onde os gradientes são difíceis de calcular. No entanto, como é um método de busca local, sua eficácia pode depender da escolha do simplex inicial e da topologia da função objetivo, exigindo atenção para evitar mínimos locais.

2.6. Robot Operate System (ROS)

ROS é um framework de software de código aberto que é amplamente utilizado na robótica e em outras áreas da robótica móvel. O ROS foi desenvolvido originalmente em 2007 pela equipe de robótica do Instituto de Tecnologia da Geórgia e, desde então, tornou-se um projeto de código aberto com uma grande comunidade de desenvolvedores que contribuem para o seu desenvolvimento e uso.

O ROS é projetado para ser uma plataforma de desenvolvimento de robôs modular e flexível, que fornece uma série de bibliotecas, ferramentas e recursos para facilitar o desenvolvimento de aplicativos de robôs. Ele é executado em sistemas operacionais baseados em Linux e Windows e suporta uma ampla variedade de linguagens de programação, incluindo C++, Python e Java.

Uma das principais vantagens do ROS é a sua arquitetura distribuída, que permite que diferentes componentes do robô (como sensores, atuadores e processadores) sejam executados em diferentes computadores, e se comuniquem entre si usando um sistema de mensagens assíncrono. Isso torna mais fácil para os desenvolvedores de robôs criar aplicativos complexos e distribuídos que envolvem múltiplos sensores, atuadores e outros componentes.

Outra vantagem do ROS é a sua ampla gama de pacotes e bibliotecas disponíveis para os desenvolvedores de robôs. Esses pacotes incluem drivers de dispositivo, algoritmos de navegação, pacotes de visão computacional e muitos outros. Além disso, a comunidade do ROS é muito ativa, e existem muitos recursos on-line disponíveis para ajudar os desenvolvedores a aprenderem a usar o ROS conforme ilustrado na figura 2.9.

Figura 2.9- Ecossistema do ROS**.** Fonte:[31].

Com base nisso, o ROS é uma plataforma de software modular, flexível e distribuída que fornece uma ampla gama de recursos e ferramentas para facilitar o desenvolvimento de aplicativos de robôs. Ele é amplamente utilizado na robótica e em outras áreas da robótica

móvel, e sua arquitetura distribuída e a diversidade de pacotes e bibliotecas o tornam uma escolha popular entre os desenvolvedores de robôs.

A estrutura do framework suporta modularização e reutilização de código por meio de pacotes (*packages*), nós (nodes), bibliotecas (*libraries*), tópicos (*topics*) e mensagens (*messages*). Eles possuem as seguintes definições:

- Um pacote pode conter nós ROS, código de biblioteca e arquivos de dados e configuração conforme representado na figura 2.10. As informações sobre um pacote e as dependências entre pacotes são descritas por meio de arquivos de manifesto. Pacotes ROS são normalmente construídos usando sistemas de compilação específicos do ROS (como *catkin*, baseado em CMake).

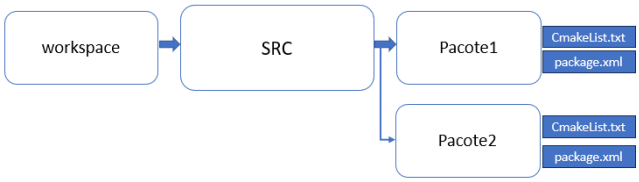

Figura 2.10- Sistema de arquivos ROS. Fonte: Autor.

- Os nós ROS são unidades executáveis. Um sistema baseado em ROS normalmente consiste em vários nós rodando em paralelo e se comunicando entre si por meio de mensagens ROS. Um nó ROS também pode carregar e usar bibliotecas fornecidas por outros pacotes.
- As mensagens fornecem uma maneira para os nós se comunicarem entre si. As mensagens são organizadas por tópicos. Um nó pode publicar mensagens sobre um tópico específico. Outros nós inscritos no mesmo tópico podem receber as mensagens publicadas. As mensagens são automaticamente serializadas e desserializadas. As estruturas de dados são definidas por meio de arquivos de descrição de mensagens e traduzidas para diferentes linguagens de programação durante a compilação. A figura 2.11 mostra a troca de informações entre dois nós.

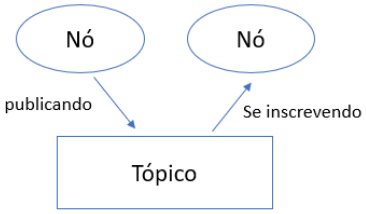

Figura 2.11- Processo de comunicação entre nós. Fonte: Autor.

- *ROSservices* implementam semântica de chamada de procedimento sobre mensagens ROS. Um servidor pode anunciar um serviço. Quando um cliente chama o serviço, ele envia uma mensagem de solicitação ao servidor e, quando o servidor atende a solicitação, ele envia de volta uma mensagem de resposta.

2.7. Visão computacional

A visão computacional é uma área de pesquisa no ramo da robótica que tem como objetivo desenvolver algoritmos e técnicas capazes de permitir que os robôs "vejam" e compreendam o ambiente ao seu redor por meio de imagens e dados visuais. Essa tecnologia é amplamente utilizada em robôs colaborativos, especialmente em áreas de pesquisa que exigem reconhecimento e classificação de objetos, tais como peças de maquinaria ou ferramentas. É fundamental que um robô colaborativo possa reconhecer objetos para trabalhar de forma eficaz com humanos e ajustar seu comportamento às necessidades ou alterações de uma linha de produção.

Dentro do campo da visão computacional, existem diversas ferramentas disponíveis. Uma das ferramentas mais importantes em relação à aprendizagem de máquina são os classificadores [30]. Esses classificadores permitem resolver diversos problemas conhecidos na indústria por meio de várias etapas de tratamento e processamento conforme mostrado na figura 2.12, resumidamente, embora varie de acordo com a aplicação, a visão computacional possui em sua maioria as seguintes funções típicas.

- **Aquisição da imagem:** por meio de câmeras e sensores, realiza a aquisição da imagem, no qual pode ser bi ou tridimensional, o dispositivo intercepta os pixels e assim identifica a intensidade da luze cores, além de outros aspectos como profundidade.
- **Pré-processamento:** o sistema realiza o processamento da imagem para assegurar a qualidade das análises posteriores. Isso envolve etapas como redução de ruídos, ajuste de brilho entre outros.
- **Extração de características**: reconhecimento de características fundamentais de uma imagem, tais como, texturas, formatos e movimentos. Isso é o que possibilita por exemplo, que o robô identifique bordas e campos onde servirão de base para a etapa posterior.

- **Deteção e segmentação:** realiza a priorização da relevância de cada região da imagem adquirida para o processamento final.
- **Processamento:** etapa final onde é realizado a classificação dos objetos.

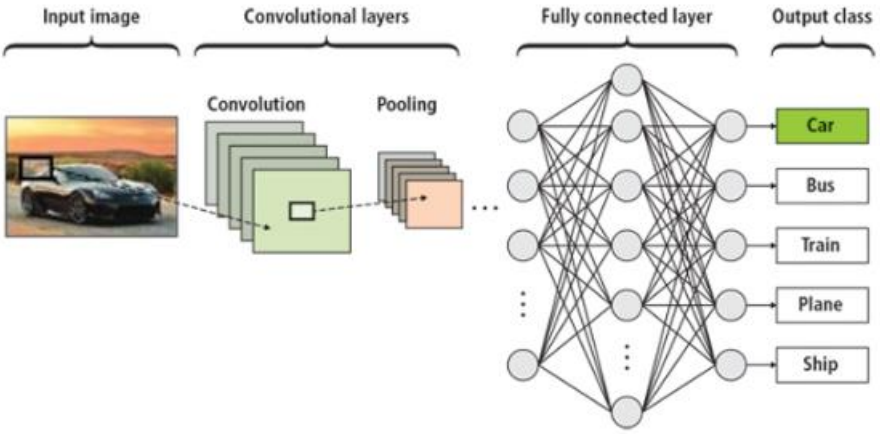

Figura 2.12 - Exemplo de classificação usando redes neurais. Fonte: [30].

2.7.1. YOLO

O termo YOLO (*You Only Look Once*) refere-se a uma técnica de detecção de objetos que analisa toda a imagem em uma única passagem pela rede, ou seja, "olha apenas uma vez". Essa abordagem permite identificar e classificar objetos rapidamente, dividindo a imagem em uma grade de células menores. Cada célula é analisada para detectar a presença de objetos, determinando suas *bounding boxes* — as caixas que delimitam o objeto. Diferentemente de outras arquiteturas, como a *mask* R-CNN, que cria uma máscara sobre o objeto, a YOLO foca apenas na criação das *bounding boxes*. Esse método é amplamente utilizado em sistemas de visão que exigem classificação em tempo real, por oferecer alta precisão e baixo custo computacional em comparação a redes mais antigas [29].

A primeira versão do YOLO foi introduzida em 2015 e vem evoluindo constantemente desde então, com cada nova versão trazendo aprimoramentos em precisão e velocidade. A arquitetura do YOLO possui semelhanças com a GoogleNet, contando com 24 camadas convolucionais, quatro camadas de *max-pooling* e duas camadas totalmente conectadas. A YOLO redimensiona a imagem de entrada para 448x448 antes de processá-la, aplica convoluções de 1x1 para reduzir os canais e utiliza convoluções de 3x3 para gerar uma saída tridimensional [56]. As camadas convolucionais empregam a função de ativação ReLU, exceto a camada final, que utiliza uma função de ativação linear. Além disso, o modelo faz uso de técnicas como normalização em lote *batch normalization* e *dropout*, para evitar sobreajuste. A rede completa é mostrada na figura 2.13.

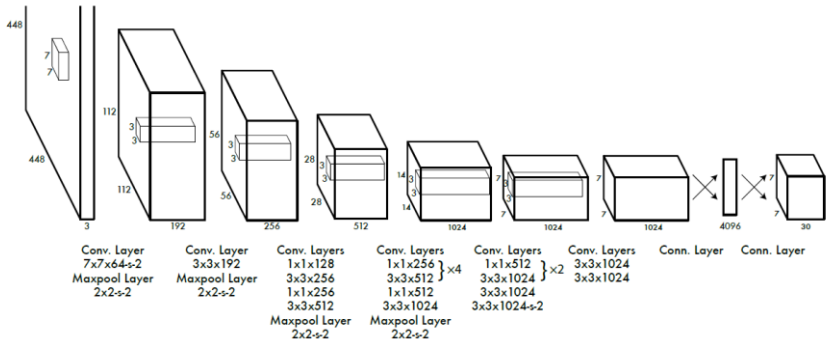

Figura 2.13- Arquitetura de um YOLO com 24 camadas convolucionais, quatro camadas de pooling máximo e duas camadas totalmente conectadas. Fonte: [56].

A arquitetura funciona da seguinte forma:

- Redimensiona a imagem de entrada para 448x448 antes de passar pela rede convolucional.
- Uma convolução 1x1 é aplicada primeiro para reduzir o número de canais, seguida por uma convolução 3x3 para gerar uma saída cuboidal.
- A função de ativação subjacente é ReLU, exceto pela camada final, que usa uma função de ativação linear.
- Algumas técnicas adicionais, como normalização em lote e abandono, regularizam o modelo e evitam que ele fique sobreajustado.

2.7.2. Estrutura da YOLOv5: Tronco, Pescoço e Cabeça

A YOLOv5 é composta de três componentes principais: o tronco (*backbone*), o pescoço (*neck*) e a cabeça (*head*). Esses três componentes trabalham de maneira integrada, como em um sistema biológico, onde o tronco fornece suporte ao pescoço, que sustenta a cabeça [57].

Tronco (*Backbone*): O tronco é responsável pela extração das principais características da imagem de entrada. A YOLOv5 utiliza redes do tipo *Cross Stage Partial Networks* (CSP) como *backbone*, que particionam o mapa de características e o fundem novamente, permitindo que o gradiente se propague por diferentes caminhos na rede. Essa estrutura reduz o custo computacional da extração de características, aumentando a velocidade de inferência [58].

Pescoço (*Neck*): O pescoço da YOLOv5 é responsável pela criação das pirâmides de características, estruturas que permitem a detecção de objetos em diferentes escalas, garantindo que objetos pequenos, médios e grandes sejam detectados de forma eficiente. A YOLOv5 adota

a *Feature Pyramid Network* (FPN), usada também em modelos como *Faster* R-CNN e *Mask* R-CNN, o que possibilita maior robustez na detecção multiescala [57].

Cabeça (*Head*): A cabeça é a etapa final, onde as *bounding boxes* e as probabilidades de detecção são geradas. A YOLOv5 cria três tamanhos de mapas de características (18×18, 36×36 e 72×72), que permitem a previsão de objetos em várias escalas. Essa estrutura possibilita ao modelo lidar com uma variedade de tamanhos de objetos na mesma imagem, mantendo a precisão e eficiência na detecção [59].

3.0. Revisão da Literatura e Trabalhos Correlatos

Neste tópico são descritos um conjunto de produção científica gerada em torno do tema deste trabalho e os principais trabalhos correlatos, apresentando uma visão geral das pesquisas existentes relevantes para o estudo deste documento, as estruturas referentes a arquitetura e desenvolvimento estão divididas em: modelagem da cinemática para um robô *pick-and-place* e sistema de visão.

3.1. Revisão Literária

Para a revisão literária deste trabalho, foram coletadas publicações, considerando apenas artigos em periódicos e publicações em conferências, com a maioria entre o período de outubro de 2018 até abril de 2024 sobre assuntos que envolvem o trabalho proposto principalmente em relação a modelagem de robôs colaborativos integrados com visão computacional, nas seguintes bibliotecas digitais:

- IEEE Xplore – Biblioteca digital de periódicos da Institute of Electrical and Electronics Engineers.
- Direct Science – Biblioteca digital de periódicos publicados pela Elsevier.

Para a seleção dos artigos, foi utilizado como critério a busca por estudos com o termo *"Robot pick-and-place"* nos títulos e resumos. Além disso, foram aplicados critérios de inclusão e exclusão. Na etapa seguinte, realizou-se uma leitura ativa dos artigos para analisá-los de forma mais aprofundada e selecionar aqueles que apresentavam um desenvolvimento claro sobre robôs colaborativos que utilizam visão computacional em suas operações.

No IEEE Xplore, foi encontrado publicações que abrangem operações de *pick-and-place* de robôs em conjunto com a visão computacional. Já no Science Direct, foi limitado a busca para engenharia e ciência da computação, utilizando o termo *"robot pick-and-place computer vision"*. A Figura 3.1 ilustra o processo de seleção dos trabalhos escolhidos para fundamentar e apoiar este trabalho.

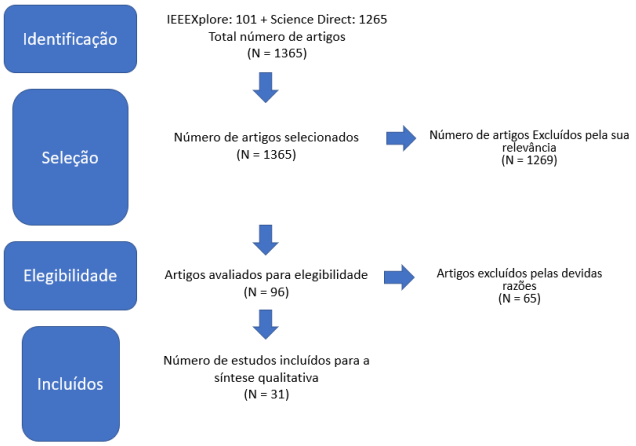

Figura 3.1- Triagem para a revisão literária. Fonte: Autor, 2024.

Para atender aos requisitos definidos na seção 1.1, que descreve robôs colaborativos com módulo *pick-and-plac*e que utilizam visão computacional, foram analisados 1365 trabalhos. Desses, foram selecionados 96 que tratavam de operações *pick-and-place* implementando o ROS para controle cinemático das operações do robô. Em seguida, foi realizada uma análise ativa do processo de desenvolvimento, com foco nas tecnologias utilizadas para implementação da visão computacional nos robôs colaborativos para detecção de objetos. Dentre os trabalhos analisados, 31 utilizavam técnicas avançadas para a detecção de objetos. Portanto este trabalho utiliza esses 65 trabalhos como referência sobre o tema, com foco nas 31 publicações que tem como foco principal detecção de objetos utilizando técnicas avançadas como YOLO e utilizando ROS para controle e modelagem de operações *pick-and-place* como ponto central do trabalho.

3.2. Trabalhos Correlatos

Para a pesquisa da modelagem da cinemática inversa, conforme visto nesse documento, o principal fator desta pesquisa é a avaliação e a implementação de algoritmos numéricos para comparação da performance e comportamento, alguns estudos discutiram ,conforme [18], a implementação de algoritmos por métodos numéricos para a simulação da cinemática inversa de um robô SCARA implementando métodos tais como o Newton-Raphson, Levenberg-Marquardt e Halley para resolver equações não lineares associadas à cinemática inversa destacando que o método Newton-Raphson demonstrou maior eficiência com menor tempo de execução e precisão elevada onde se utilizou a ferramenta MATLAB para realizar as comparações entre os métodos. Outros estudos exploraram métodos de detecção de objetos baseados em aprendizado de máquina, que se destacam por sua maior generalidade e capacidade de lidar tanto com objetos texturizados quanto não texturizados. Esses métodos demonstram

desempenho superior em operações *pick-and-place*. No trabalho de [22], por exemplo, foi utilizado um método de aprendizado profundo para detectar e segmentar diferentes tipos de mantimentos. O robô foi então empregado para realizar tarefas automatizadas de coleta e organização de itens em lixeiras específicas, onde se utilizou o algoritmo *Iterative Closest Point* (ICP) para a manipulação do robô para obter os pares de pontos correspondentes para o objeto-alvo.

A dissertação [19] apresenta um estudo sobre a aplicação de métodos de otimização híbridos para resolver problemas de despacho econômico em sistemas de geração de energia elétrica. O trabalho foca em uma variante do problema chamada PDE-PCV, que incorpora os efeitos do ponto de carregamento de válvula, tornando a função objetivo não convexa e não diferençável. Para resolver essa complexidade, o autor propôs um método híbrido que combina a evolução diferencial, conhecida por sua eficiência em busca global, com o método *Quase-Newton* BFGS, que otimiza a busca local utilizando gradientes aproximados. A implementação computacional foi desenvolvida em MATLAB e testada em cenários com diferentes números de geradores, onde destacam a eficácia do método em atender às restrições do sistema e reduzir os custos operacionais, contribuindo para o avanço de estratégias de otimização aplicadas à geração termoelétrica.

Em [23] explora a minimização de funções objetivas suaves e não convexas por meio de um algoritmo iterativo baseado no método de Newton combinado com gradiente conjugado (CG). O algoritmo é projetado para detectar e utilizar direções de curvatura negativa na Hessiana, permitindo melhorar a eficiência computacional e garantir resultados teóricos de complexidade, onde se utiliza aprimoramentos como CG limitado e busca linear para garantir redução significativa da função objetiva em cada iteração, mantendo os custos computacionais baixos. Além disso, o artigo discute a aplicabilidade do método em contextos que envolvem a minimização de problemas sem restrições em larga escala, especialmente quando a avaliação completa da matriz Hessiana é inviável.

Por sua vez, [24] apresenta uma abordagem avançada para a solução de problemas de otimização irrestrita de grande porte. A autora, explora a aplicação do método Quasi-Newton L-BFGS (*Limited-memory* BFGS) e propõe uma nova estratégia que utiliza a fatoração incompleta de Cholesky para aprimorar a convergência do método em problemas envolvendo matrizes Hessianas de grande dimensão e esparsas. A dissertação aborda aspectos teóricos, algoritmos de otimização e resultados numéricos.

Em [64] apresenta uma análise comparativa entre seis modelos locais aplicados à aproximação da cinemática inversa de manipuladores robóticos redundantes. O estudo envolve os robôs planar, PUMA 560 e Motoman HP6, utilizando métodos como redes de funções de base radial (RBFN), redes de modelos locais (LMN), regressão ponderada local (LWR), entre outros. Os modelos foram avaliados quanto à acurácia na estimativa dos ângulos das juntas em diversas trajetórias no espaço de trabalho. Além disso, a dissertação realiza testes de hipóteses para validar as diferenças estatísticas entre os métodos mais eficazes. Os resultados destacam a importância de métodos locais na resolução da cinemática inversa em robôs redundantes.

O trabalho de [65] investigou métodos analíticos e numéricos para a resolução da cinemática inversa em manipuladores robóticos industriais, com foco na implementação computacional e na análise de estabilidade em regiões singulares. Entre os métodos abordados estão Newton-Raphson, pseudo-inversa e a análise da Posição-Zero. A pesquisa destacou as vantagens dos métodos analíticos em termos de velocidade e precisão, mas apontou limitações em cenários específicos. Por outro lado, os métodos numéricos demonstraram maior flexibilidade, embora com um custo computacional mais elevado. O trabalho também apresentou uma aplicação prática em um programa computacional para análise cinemática, incluindo representação gráfica e cálculos baseados nos parâmetros Denavit-Hartenberg, oferecendo uma base sólida para o estudo e comparação de métodos numéricos aplicados à cinemática inversa, alinhando-se aos objetivos da presente dissertação.

A dissertação [66] apresenta o desenvolvimento e otimização de um resolvedor de cinemática inversa (IK) baseado em algoritmos meméticos para o framework de planejamento de movimento *MoveIt*, integrado ao sistema operacional de robôs (ROS). O resolvedor suporta árvores cinemáticas com múltiplos efetores finais, indo além das cadeias cinemáticas com um único *efector* final disponíveis nos resolvedores convencionais. O algoritmo combina técnicas de otimização evolutiva, otimização por enxame de partículas e métodos baseados em gradiente, permitindo lidar com mínimos locais, limites de juntas e alcançar convergência rápida e precisa. Testes experimentais mostraram que o resolvedor memético redesenhado superou os métodos baseados em gradiente, incluindo implementações existentes no MoveIt, em cenários de IK com múltiplos objetivos.

Em [67] o autor explora o planejamento de trajetórias de manipuladores robóticos em ambientes industriais, abordando o desenvolvimento de um planeador de trajetórias baseado na família de algoritmos *Roadmaps*. O trabalho destaca a necessidade de discretização espacial e

da definição de equações cinemáticas para guiar o movimento do robô, propondo melhorias e comparando o desempenho do novo algoritmo com ferramentas existentes, como o *MoveI*t. As validações são realizadas em dois cenários industriais relevantes: o *European Robotic Challenge* (EuRoC) e o *Amazon Picking Challenge*. Os resultados evidenciam a aplicabilidade do método desenvolvido, oferecendo uma solução flexível e genérica para operações de pick-and-place.

Já em [68], O artigo apresenta uma abordagem robusta para a resolução da cinemática inversa baseada no método de Levenberg-Marquardt (LM), destacando sua capacidade de lidar com problemas de solvabilidade, singularidades e sistemas com múltiplas ou nenhuma solução. Esse método numérico minimiza o erro residual entre a posição desejada e a posição real do end-effector, garantindo uma solução otimizada com pequenas variações nas articulações. Além disso, o uso de um fator de amortecimento robusto melhora a estabilidade numérica e a velocidade de convergência, tornando-o adequado para cadeias cinemáticas de grande escala com graus de liberdade variáveis.

Já na parte de visão computacional, diversos estudos têm abordado a detecção de objetos utilizando técnicas baseadas em aprendizado de máquina [20] [21] para integração com operações *pick-and-place*. Esses métodos são especialmente adequados para a proposta deste trabalho, pois oferecem maior flexibilidade e facilidade de implementação na tarefa de detecção de peças de LEGO. No contexto desta pesquisa, a visão computacional desempenha um papel secundário, sendo utilizada principalmente para contextualizar o cenário industrial simulado, enquanto o foco principal permanece na resolução da cinemática inversa para o planejamento de movimento.

Com base nas referências teóricas revisadas nesta pesquisa, é frequentemente observado que os métodos numéricos não apresentam desempenho suficientemente rápido, o que levou os métodos analíticos a se tornarem uma escolha popular na resolução do problema de cinemática inversa na comunidade acadêmica. A desvantagem de uma solução analítica, comprovada historicamente por meio de artigos e a comunidade robótica, é que ela não tem uma solução geral, pois a expressão de forma fechada é baseada no projeto do manipulador robótico, já que é necessário saber a quantidade de graus de liberdade e os parâmetros do robô desejado. Como uma solução analítica (expressão de forma fechada) foi historicamente comprovada como sendo mais rápida do que os métodos numéricos, foram publicadas poucas pesquisas sobre o uso de métodos numéricos para sistemas robóticos em tempo real, apesar de seu potencial para lidar com problemas mais generalizados e flexíveis.

Assim, este estudo implementa uma abordagem modular para o sistema *pick-and-place*, utilizando nós do ROS como elementos-chave dessa arquitetura. Cada nó é projetado para ser flexível, permitindo substituições ou atualizações futuras nos métodos aplicados para a cinemática inversa. Essa modularidade torna possível adaptar o sistema para atender a diferentes cenários específicos ou mais complexos. Por exemplo, na parte de detecção de objetos, este trabalho explora técnicas baseadas em redes neurais convolucionais (CNNs), que se mostraram altamente eficazes na identificação de objetos em imagens. As CNNs oferecem maior precisão em comparação com técnicas baseadas em características, como o SIFT, e maior eficiência em relação a métodos tradicionais de aprendizado de máquina. Essa escolha de abordagem reforça a adaptabilidade do sistema e possibilita sua expansão para aplicações futuras em ambientes industriais.

4.0. Arquitetura e Metodologia

Esta pesquisa propõe a investigação de métodos numéricos para a resolução da cinemática inversa em um robô colaborativo de 6 graus de liberdade (6-DOF), aplicado a operações pick-and-place. O estudo compara diferentes abordagens computacionais, avaliando sua precisão, eficiência e tempo de execução, a fim de identificar a técnica mais adequada para controle de movimento em sistemas robóticos industriais.

Nesta seção, são apresentadas as técnicas e procedimentos detalhados que nortearão as atividades desta pesquisa, garantindo a validação dos métodos numéricos investigados para o controle cinemático inverso. Dessa forma, são abordados os tópicos relacionados à arquitetura proposta e ao desenvolvimento do ambiente de simulação, incluindo a integração do Sistema Operacional de Robôs (ROS), o manipulador robótico UR5e e os algoritmos numéricos estudados.

4.1. Arquitetura proposta

Para desenvolver uma aplicação de reposicionamento de peças de LEGO utilizando um robô manipulador UR5e equipado com um módulo *pick-and-place,* o sistema foi estruturado em duas seções principais. Depois que o robô move para a posição inicial, a primeira parte é dedicada a fase de identificação das peças, empregando um algoritmo de detecção e classificação baseado em imagens capturadas para fornecer as poses das peças de LEGO e assim determinar onde colocá-las. A segunda parte concentra-se no fluxo de trabalho do *pick-and-place*, no qual o movimento do robô é planejado e executado utilizando os métodos estudados nesta pesquisa, com o controle realizado através do ROS. Nessa etapa, serão

implementados algoritmos numéricos desenvolvidos para resolver o problema da cinemática inversa, conforme proposto neste estudo.

O fluxo de trabalho em torno do robô *pick-and-place* para realizar a solicitação de ação para o controlador realizar o movimento é mostrado conforme a figura 4.1.

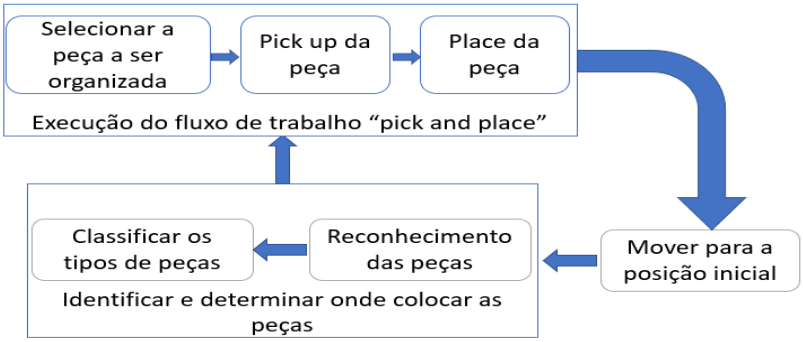

Figura 4.1 – Etapas do fluxo do ciclo completo de pick-and-place do robô. Fonte: Autor, 2024.

Na etapa do fluxo *pick-and-place*, ilustrada na Figura 4.1, o sistema ROS é utilizado em conjunto com um sistema de planejamento de movimento para peças de LEGO. Esse sistema elabora e manipula a rota de cada peça, permitindo que ela seja movida até sua posição designada na respectiva caixa.

A fase *pick up* move o robô UR5e até a peça, pega ela e move para uma posição estabelecida conforme mostrado na imagem 4.2.

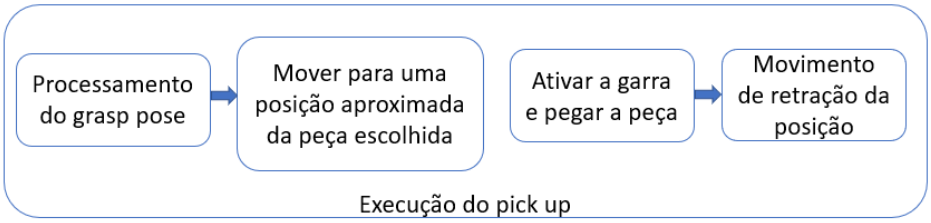

Figura 4.2 – Fase pick-up robô. Fonte: Autor, 2024.

Na fase *placed* mostrada na imagem 4.3, o robô coloca o objeto na caixa apropriada, onde a abordagem de posicionamento e as posições de retração são computadas em relação a posição de posicionamento desejada.

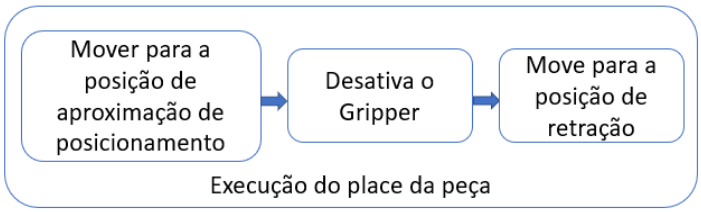

Figura 4.3 – Fase placed do robô. Fonte: Autor, 2024.

A Figura 4.4 ilustra a arquitetura do sistema pick-and-place. O robô é composto por um braço robótico UR5 de seis graus de liberdade (6 DOFs) e opera em um ambiente com kernel Linux (Ubuntu 20.04), que fornece a infraestrutura necessária para o middleware ROS. Esse middleware gerencia a comunicação entre os diferentes componentes do sistema por meio do ROS Communication BUS, utilizando o modelo de publicação e assinatura *(Publisher/Subscriber)*, proporcionando uma interface de desenvolvimento rápida e robusta para desenvolvimento com codigo *c++* e *python*. A arquitetura é baseada no ROS *Noetic*, integrado ao simulador *Gazebo* 11. O sistema de visão e o controle do movimento do robô comunicam-se pelo barramento ROS, com o sistema de visão fornecendo a localização das peças, permitindo ao controlador do braço robótico planejar as coordenadas da trajetória para alcançar e agarrar as peças.

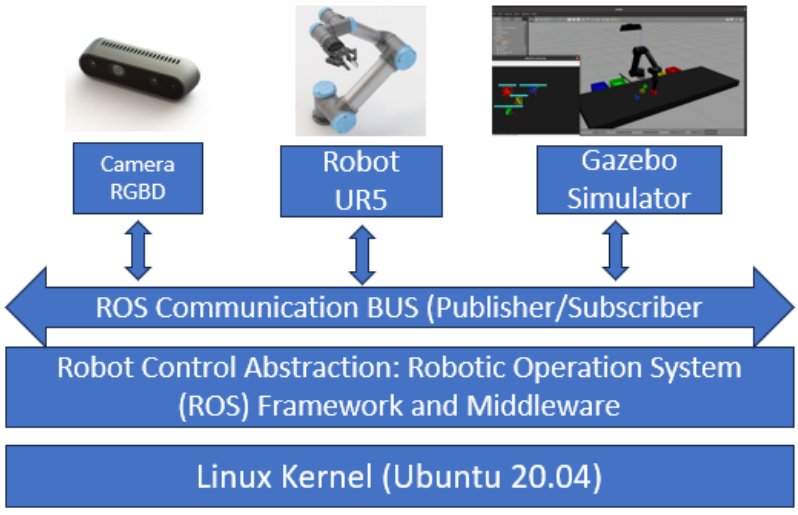

Figura 4.4 – Arquitetura do sistema pick-and-place**.** Fonte: Autor, 2024.

Os principais componentes conectados incluem:

1. *Câmera RGBD*: Responsável pela captura de imagens, a biblioteca do Gazebo "openni_kinect" simula o comportamento de uma câmera Kinect RGBD, permitindo obter imagens e informações de profundidade do ambiente montado. Esses dados são essenciais para a detecção e reconhecimento de peças no cenário.
2. *Modelo do Robô UR5e*: Manipulador colaborativo cuja movimentação e controle são gerenciados pelo ROS, permitindo a realização de operações pick-and-place.
3. *Simulador Gazebo*: Ambiente virtual utilizado para simular o robô e o cenário operacional, possibilitando testes e validações antes da execução em um ambiente físico.

Esses elementos são coordenados pelo ROS Communication BUS, que permite a troca de mensagens entre os nós responsáveis pelo processamento das imagens capturadas, o controle do movimento do robô e a integração com o simulador. Essa abordagem modular e centralizada no ROS facilita a interoperabilidade entre hardware, simulação e algoritmos de controle.

Dentro dessa estrutura, o sistema de controle robótico utiliza mensagens do ROS para publicar e assinar tópicos. Um dos principais tópicos é o *"MotionControl"*, que coordena a comunicação entre os nós do sistema. Nesse contexto, um nó é responsável por executar as ações de controle do movimento do manipulador robótico, enquanto outro nó realiza a detecção e o reconhecimento das peças de LEGO, publicando suas localizações no tópico correspondente.

4.2. Metodologia

Esta seção mostra uma descrição detalhado do processo e das etapas envolvidas na resolução da cinemática inversa, e assim como a implementação dos referidos métodos que serão realizados. O objetivo deste capitulo é apresentar uma explicação clara da metodologia e fornecer uma compreensão abrangente das considerações que devem ser levadas em conta na implementação das soluções.

A Figura 4.5 apresenta uma visão simplificada do processo de resolução de trajetórias, onde o ROS é utilizado para publicar tópicos contendo os ângulos articulares calculados. O controle das juntas é gerenciado pelo *"Move Controller"*, responsável por atualizar os estados articulares em tempo real, garantindo a execução precisa dos movimentos do manipulador. Na etapa de cinemática inversa, são implementados os algoritmos Gradiente Descendente, BFGS e Nelder-Mead para calcular os ângulos articulares necessários, permitindo o controle eficiente do manipulador. Esses valores são então validados no ambiente de simulação Gazebo. Os

métodos numéricos selecionados foram escolhidos devido à sua capacidade de abordar diferentes aspectos da eficiência computacional, precisão e estabilidade no controle de manipuladores robóticos, oferecendo uma comparação abrangente de técnicas numéricas aplicáveis ao robô UR5e

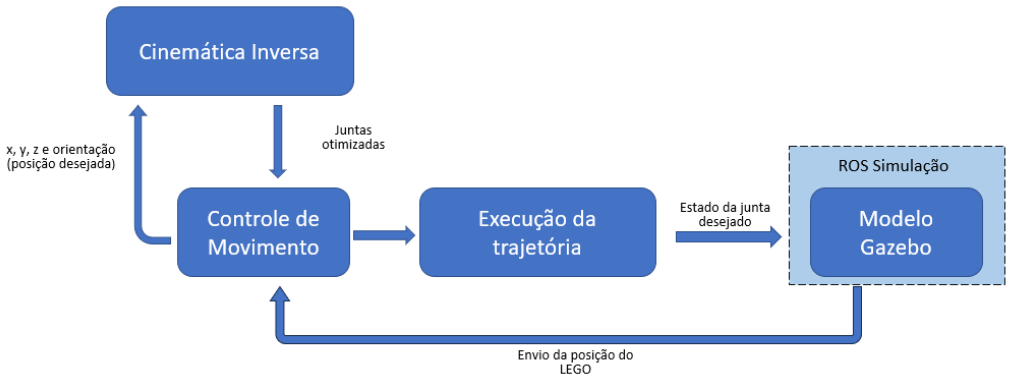

Figura 4.5 – Visão simplificada das etapas do processo da resolução da cinemática. Fonte: Autor, 2024.

4.2.1. Ambiente de simulação

O ambiente de simulação foi desenvolvido utilizando o software Gazebo, conforme descrito no Apêndice D, com o objetivo de criar uma plataforma virtual que integre e teste as funcionalidades do sistema proposto. Dois módulos principais foram implementados: uma câmera RGBD para o reconhecimento de peças de LEGO e um modelo detalhado do manipulador UR5e, utilizado para controlar o movimento do robô no espaço cartesiano. A configuração do ambiente está ilustrada na Figura 4.6.

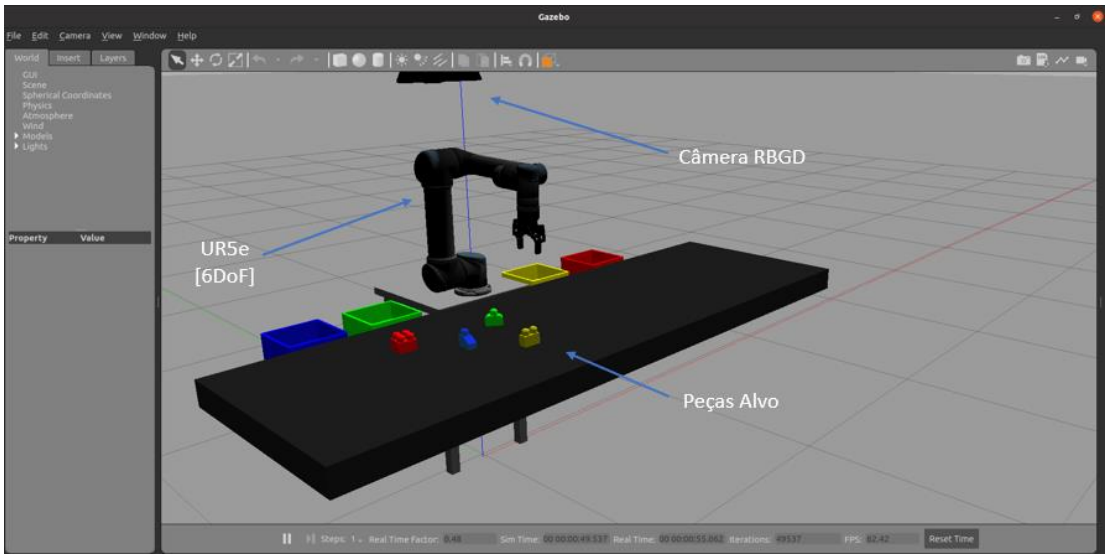

Figura 4.6 – Ambiente de simulação do sistema proposto**.** Fonte: Autor, 2024

A câmera RGBD foi configurada para capturar imagens em RGB e informações de profundidade do ambiente. Esses dados são processados para identificar o tipo e a localização das peças de LEGO dispersas em uma esteira virtual. A posição e a orientação aproximadas de cada peça são estimadas com base nas coordenadas espaciais e na profundidade capturada pelo sensor. A classificação das peças é feita utilizando o modelo YOLOv5, previamente treinado em um conjunto de dados. Esse processo permite que o sistema determine as posições-alvo para o manipulador robótico realizar as operações de *pick-and-place*.

O manipulador UR5e foi modelado no Gazebo com todos os seus graus de liberdade (6-DOF), respeitando as especificações técnicas reais do robô. Durante a simulação, o efetuador final do robô se move até a posição da peça identificada pela câmera, realiza o agarre utilizando o *gripper*, e transporta a peça até a caixa correspondente, conforme a classificação fornecida pelo módulo de visão computacional. Esse fluxo operacional assegura que os diferentes subsistemas do sistema proposto trabalhem de forma integrada e coordenada.

A resolução da cinemática inversa desempenha um papel central no controle do manipulador. Para isso, foram implementados diferentes métodos numéricos, incluindo gradiente descendente, BFGS, L-BFGS-B e Nelder-Mead. Esses algoritmos calculam os ângulos das juntas necessários para que o efetuador alcance a posição e orientação desejadas no espaço cartesiano. Os valores das juntas calculados são transmitidos ao manipulador no ambiente de simulação através do ROS, que atua como um middleware para a comunicação entre os módulos de visão computacional e controle do manipulador.

4.2.2. Definição e modelo matemático para o manipulador UR5e

Esta parte foca na definição da cinemática inversa do manipulador robótico UR5e. As propriedades geométricas do manipulador servem como base para a definição dos parâmetros Denavit-Hartenberg (DH), que são fundamentais para descrever as relações entre as juntas e os elos do robô. Esses parâmetros também são utilizados como ponto de partida para as transformações aplicadas pelos métodos numéricos implementados. O código, desenvolvido em Python, realiza os cálculos da cinemática inversa e permite a comparação entre diferentes métodos, como gradiente descendente, BFGS, L-BFGS e Nelder-Mead. Os ângulos articulares calculados são testados em um manipulador UR5e simulado no ambiente Gazebo,

proporcionando não apenas uma análise matemática dos métodos, mas também uma validação visual das soluções, que facilita a verificação e interpretação das implementações.

4.2.3. Cinemática do manipulador UR5e

Com base na seção 2.3, a notação de Denavit Hartenberg é uma ferramenta utilizada para sistematizar a descrição da cinemática de sistemas mecânicos articulados com n graus de liberdade. Para descrever a translação e rotação entre dois eixos adjacentes, Denavit e Hartemberg propuseram um método matricial para estabelecimento sistemático de um sistema de coordenadas fixo para cada eixo de uma cadeia cinemática articulada. A figura 4.7 e a tabela 4.1 apresentam os parâmetros de DH de um manipulador UR5e.

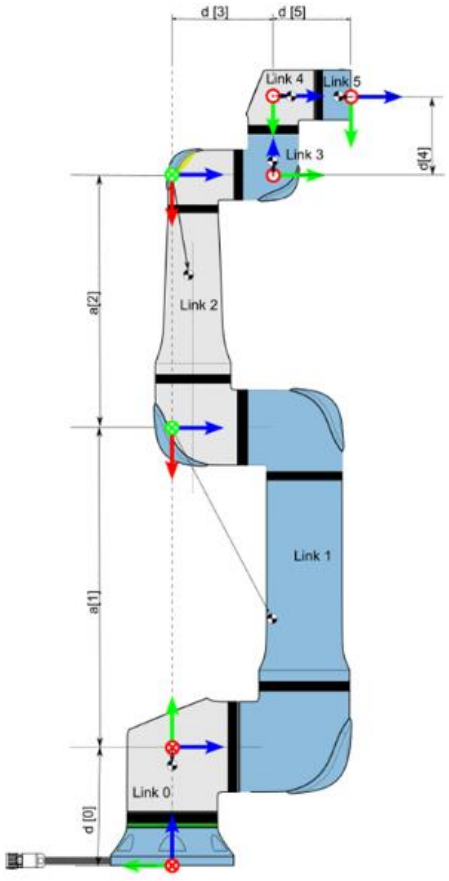

Figura 4.7 – Referências para cinemática**.** Fonte: [62].

Tabela 4.1 - Parâmetros da cinemática do manipulador UR5e**.** Fonte: Autor, 2024]

Junta	a	α	d	θ
1	0	$\pi/2$	0.089159	θ_1
2	-0.425	0	0	θ_2
3	-0.39225	0	0	θ_3
4	0	$\pi/2$	0.10915	θ_4
5	0	$-\pi/2$	0.09465	θ_5
6	0	0	0.0823	θ_6

O código apresentado no apêndice A implementa o cálculo da cinemática direta para o manipulador robótico UR5e. O objetivo da cinemática direta é determinar a posição e a orientação do efetuador final *"end-effector"* do robô em relação ao referencial base, dado um conjunto de ângulos das juntas (θ). O código apresentado tem as seguintes etapas:

1. Definição dos Parâmetros do Robô

O manipulador UR5 é descrito por seis parâmetros DH que são usados para definir a geometria do robô:

- d: Deslocamentos ao longo do eixo Z para cada junta.
- a: Distâncias ao longo do eixo X para cada junta.
- α (alpha): Ângulos de rotação ao redor do eixo X para cada junta.

Esses parâmetros são necessários para definir a transformação homogênea que descreve a relação entre as diferentes partes do robô.

2. Função de Transformação Homogênea (HT)

A função *HT(n, th)* calcula a matriz de transformação homogênea para o elo *n* do robô, utilizando os parâmetros DH. A transformação homogênea é composta por:

- T_a: Matriz de deslocamento, que move o sistema de coordenadas da junta.
- T_d: Matriz de translação, que aplica o deslocamento ao longo do eixo Z.
- R_{zt}: Matriz de rotação ao redor do eixo Z.
- R_{xa}: Matriz de rotação ao redor do eixo X.

Essas matrizes são multiplicadas na ordem correta para gerar a transformação homogênea que relaciona a posição e a orientação do efetuador em relação à base do robô.

2. Função *"forward"*

A função *forward(th*) calcula a cinemática direta total do manipulador UR5, multiplicando as transformações homogêneas de cada junta (de 1 a 6) para obter a transformação final T_6, que descreve a posição e a orientação do efetuador final no espaço tridimensional. Para isso, a função *HT* é chamada para cada junta, e as transformações resultantes são multiplicadas em sequência. Os passos do cálculo da transformação total são os seguintes:

1. Calcula as matrizes de transformação homogênea HT_1, $HT_2, \dots, HT_6$ para cada elo do robô, utilizando a função *HT*
2. Multiplica sequencialmente essas matrizes:

$$T_6 = HT_1 . HT_2 . HT_3 . HT_4 . HT_5 . HT_6$$

3. Retorna T_6, que contém a posição do efetuador final *(x, y, z)* e sua orientação no espaço tridimensional multiplicando as transformações homogêneas das seis juntas, em sequência.

O cálculo da cinemática direta neste projeto é importante para saber a posição do efetuador final na inicialização do sistema, e para saber onde está a pose do efetuador com base nas juntas atuais.

4.2.4. Reconhecimento e classificação

Para o reconhecimento de peças de LEGO em uma imagem, é possível dividir essa tarefa em etapas menores para torná-lo menos complexo e gerenciável. Para o método proposto, foi abordado as seguintes etapas:

1. Instalação de bibliotecas necessárias para fazer o processamento da imagem com o *python* juntamente com o modulo *pyTorch* já que fornece suporte para redes neurais profundas para classificação das peças.
2. Configuração de um ambiente simulado para implementar um sistema para gerar dados e testar modelos de detecção em condições controladas.
3. Criação de um modelo de reconhecimento de imagem para identificar as diferentes peças de LEGO dentro do ambiente industrial simulado. A biblioteca utilizada foi o *pyTorch*

para criação de um modelo de aprendizado de máquina para conseguir aprender a identificar as diferentes peças de LEGO localizadas na esteira, para isso, foi utilizado um conjunto de dados para detecção de peças de LEGO [61].

4. Em seguida, após a criação do modelo é necessário avaliar o desempenho de reconhecimento do modelo treinado executando inferências em imagens de testes. Durante a inferência, a YOLO processa as imagens em tempo real, gerando caixas delimitadoras e classificações para os objetos detectados.

5. Os resultados são exibidos em uma imagem de saída, desenhando caixas delimitadoras ao redor de cada peça identificada.

No gráfico da imagem 4.8, mostra o objetivo da YOLO de produzir um modelo de detector de objetos muito eficiente (eixo Y) em relação ao seu tempo de inferência (eixo X). Os resultados preliminares mostram que o YOLOv5 se sai muito bem para esse fim em relação a outras técnicas de ponta. É possível notar que todas as variantes do YOLOv5 treinam mais rápido que o *EfficientDet*. O modelo YOLOv5 mais preciso, YOLOv5x, pode processar imagens várias vezes mais rápido com um grau de precisão semelhante ao do modelo *EfficientDet D4*.

O YOLOv5 deriva a maior parte de sua melhoria de desempenho dos procedimentos de treinamento do *PyTorch*, enquanto a arquitetura do modelo permanece próxima ao YOLOv4.

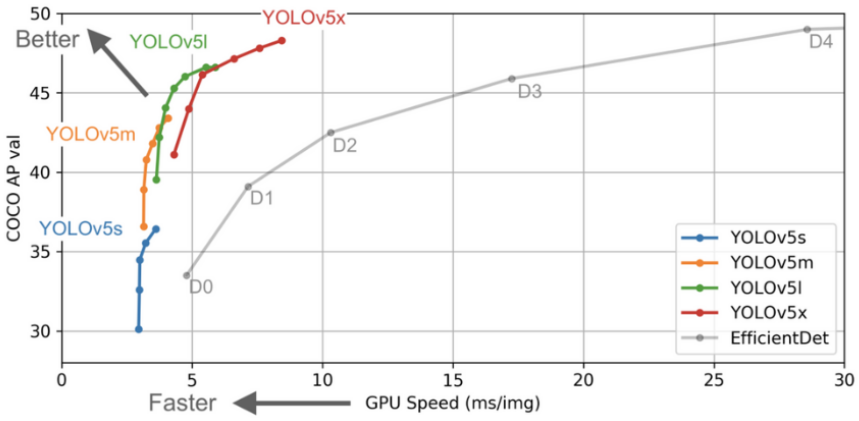

Figura 4.8 – Gráfico de comparação dos desempenhos das versões YOLO. Fonte [29].

4.2.5. Segmentação de alvos e determinação da pose do ponto de captura

A detecção das peças de LEGO é realizada com base em visão computacional, utilizando a rede neural de aprendizagem profunda YOLOv5 [60] para treinar um conjunto de dados e gerar um modelo capaz de detectar diferentes tipos de peças LEGO. Para alcançar

melhores resultados de reconhecimento, o sistema requer um número significativo de dados de suporte. Nesta pesquisa, foi utilizado a ferramenta Blender para gerar um conjunto de dados de peças LEGO devidamente rotuladas. Após 300 épocas de treinamento, os resultados experimentais demonstram que o modelo foi eficaz em identificar as peças na esteira e em marcar retângulos ao redor delas.

Para aprimorar o processo de seleção e posicionamento das peças, foi adotado o uso de uma garra robótica. O quadro retangular detectado pelo algoritmo YOLOv5 é utilizado para determinar o ponto central da caixa delimitadora, onde é extraído com base nos contornos da máscara. Ajustes de distorção da câmera são aplicados para alinhar o centro ao referencial global. Com isso, uma análise dos resultados de detecção revela que o desvio entre o ponto central do retângulo previsto e o ponto central real da peça é pequeno. Assim, neste trabalho, o ponto central do quadro retangular é utilizado para a determinação da orientação da peça, com base em imagens de profundidade e refinada utilizando vetores diretores calculador a partir dos contornos, para assim, realizar a ação de agarre da peça.

O mapa de cores da câmera é registrado com o mapa de profundidade obtido pela câmera Kinect. A posição *(X, Y, Z)* da peça é transformada para o referencial do robô com base na posição da câmera *kinect* e correções de escala. A orientação é convertida para *quaternions* utilizando a biblioteca *PyQuaternion*. Após os testes, verificou-se que o erro entre os pontos previstos e os reais está dentro de uma margem aceitável, garantindo a precisão necessária para o processo de manipulação.

A processo de treinamento e detecção das peças envolve duas etapas principais conforme mostra a figura 4.9. onde temos:

1. Processo de treinamento: Treinamento da rede de detecção de objetos baseada em RGB (YOLOv5) e sua validação em relação ao conjunto de dados de teste.
2. Processo de detecção de peças: Estimativa de pose usando dados RGB-D brutos em tempo real usando a rede YOLOv5 pré-treinada obtendo as posições das peças e realizando posteriormente a estimativa da pose.

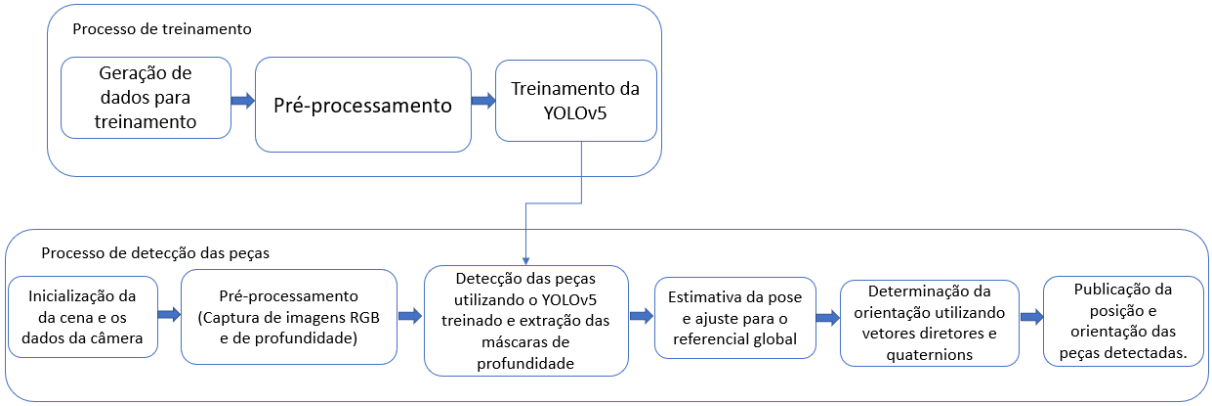

Figura 4.9 – Aprendizagem baseada em YOLOV5 para detecção e estimativa da pose. Fonte [29].

Essa pipeline representa a solução completa que combina visão computacional, aprendizado de máquina e transformações geométricas para localizar peças de Lego na cena. O resultado da detecção das peças no ambiente de simulação com base no método descrito é mostrado na figura 4.10.

Figura 4.10 – Detecção das peças LEGO no ambiente gazebo. Fonte: Autor.

4.2.6. Processamento e cálculo do tempo

Os tempos de cálculo da resolução dos métodos são comparados entre cada um dos algoritmos numéricos e o analítico. Para garantir que os resultados desses testes sejam confiáveis, cada caso de teste foi executado um determinado número de vezes para ter referência de um tempo médio computacional, isso é realizado pois se deve levar em consideração que os algoritmos de otimização têm uma tendência a diferir ligeiramente no tempo de convergência. O número máximo de iterações foi definido como 5000 e tolerância de

1e-6 para métodos que aceitam delimitações. As especificações do ambiente de benchmark de teste são apresentadas na Tabela 4.2.

Computador	Alienware M15R2
Processador	11th Gen Intel Core i7-11800H @ 2.30GHz
GPU	Nvidia 3070 8GB
Memoria	40,0 GB
Sistema	Ubuntu 20.04

Tabela 4.2 – Especificação do dispositivo utilizado para executar os testes. Fonte: Autor.

5.0. Resultados

5.1 Simulação

Nesta seção, descreve-se a implementação do ambiente de simulação utilizado para testar os métodos numéricos de resolução da cinemática inversa. A simulação foi realizada no software Gazebo, integrado ao ROS, para permitir a interação entre os módulos de visão computacional e controle do manipulador. Foram modeladas peças de LEGO em um cenário industrial virtual, e os métodos estudados (Gradiente Descendente, BFGS, L-BFGS-B e Nelder-Mead) foram aplicados para avaliar o comportamento do robô colaborativo UR5e ao longo de diferentes trajetórias. Os testes buscaram analisar a precisão da solução numérica e a estabilidade das juntas ao longo do movimento.

5.2 Análise dos Resultados Experimentais

O modelo virtual desenvolvido conforme mostra a figura 5.0, simula uma esteira com peças de LEGO dispersas em uma área específica. Para isso, foi criado um programa para realizar a dispersão das peças na esteira e um nó ROS dedicado ao sistema de visão computacional. O objetivo é identificar e classificar as peças para organizá-las em suas respectivas caixas. O movimento completo do robô é realizado com base em diferentes métodos numéricos para a solução da cinemática inversa, vistos na seção 2, que é o foco principal desta pesquisa. A validação e comparação foram realizadas exclusivamente para os métodos numéricos, dado que eles são o objeto de estudo. Os testes para o cálculo do tempo do movimento foram divididos em dois modos: o primeiro teste avalia o movimento de retorno à posição inicial *"home*" do robô, e o segundo teste representa o processo *pick-and-place*

completo de uma peça de LEGO para sua respectiva caixa. O ambiente que será realizado os testes é mostrado na figura abaixo.

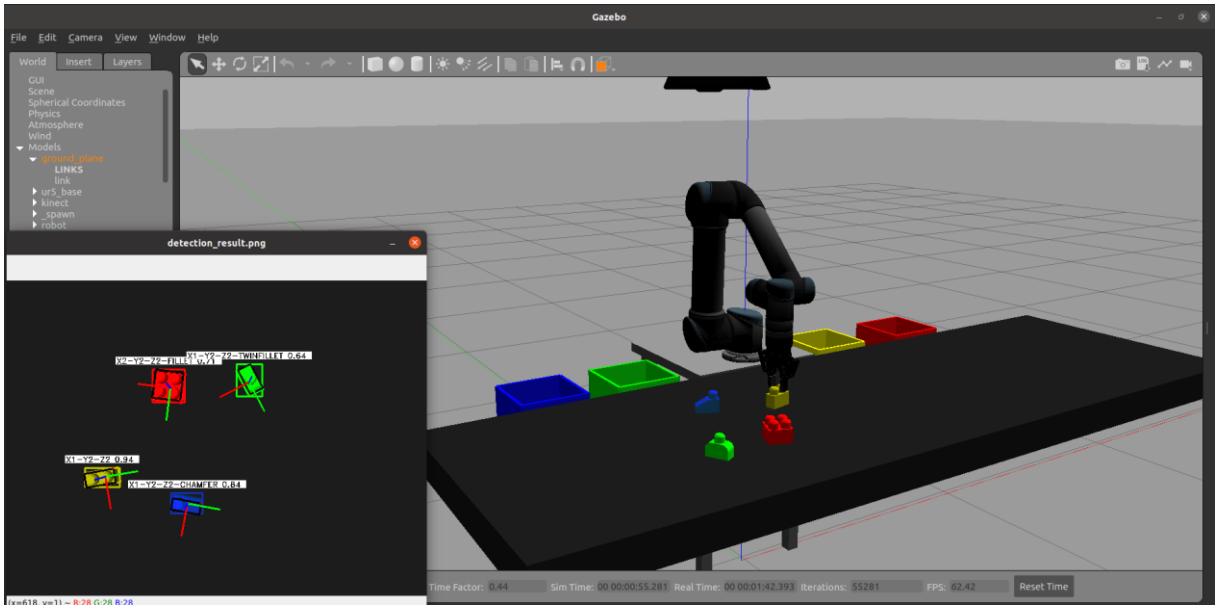

Figura 5.0 – Ambiente de simulação gazebo para a realização dos testes de comportamento cinemático. Fonte: Autor.

Para cada método foram realizados 5 testes e feito a média de duração do processo do movimento, os resultados dos testes podem ser vistos na tabela abaixo.

Método	Teste 1 duração (s)	Teste 2 duração (s)
Gradiente-Descendente	28.95	235.17
BFGS	18.97	189.55
L-BFGS-B	11.02	143.90
Nelder-Mead	37.67	343.74
Analítico	3.71	33.80

Tabela 5.1 – Tempo para resolver a cinemática inversa para diferentes movimentos. Fonte: Autor.

Os resultados indicam que métodos baseados na matriz Hessiana, como o BFGS e o L-BFGS-B, oferecem uma duração de movimento mais rápido que os demais, já o método Gradiente Descendente se torna mais rapido que Nelder-Mead devido ao uso da pseudo inversa de Jacobiano, que impacta diretamente no cálculo do gradiente e na eficiência do método. O Nelder-Mead requer um tempo médio maior do que BFGS e L-BFGS-B, isso é devido a sua eficiência ser limitada por depender exclusivamente da geometria do simplex para explorar o

espaço de busca. Já o método analítico como esperado, apresenta os tempos mais curtos que os métodos numéricos, devido à sua solução fechada, que elimina a necessidade de iterações para resolver o problema. A curva de convergência dos métodos é mostrada na figura 5.0, onde é obtido essa convergência para um ponto da trajetória do movimento realizado.

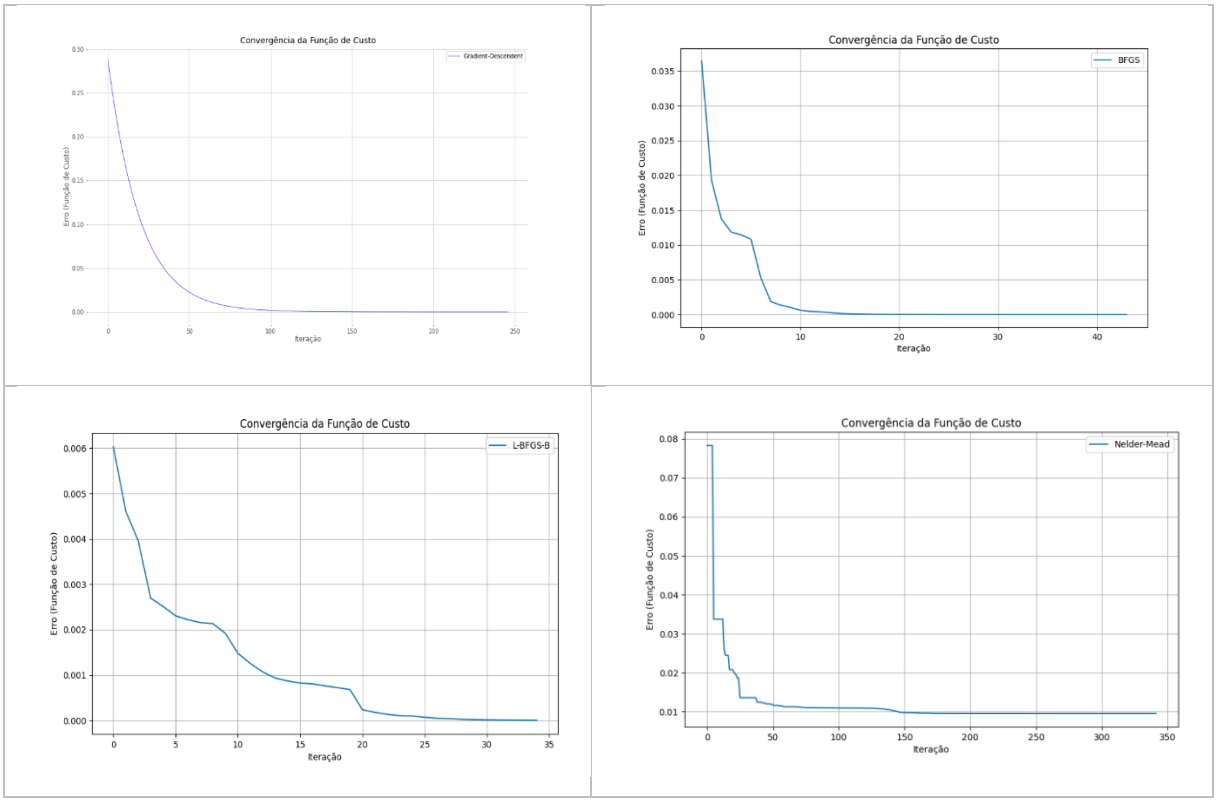

Figura 5.1 – Curva de convergência de um ponto ao longo da trajetória. Fonte: Autor.

Para avaliar a eficiência dos algoritmos numéricos em relação a precisão e eficiência, foram considerados dois critérios principais: o número médio de iterações necessárias para que o algoritmo convergisse ou atendesse ao critério de parada, e o erro médio ao longo de um movimento completo do robô. Para calcular o erro médio, foram somados os erros resultantes de cada ponto interpolado ao longo da trajetória do movimento, e, em seguida, foi calculada a média desses valores. A tabela 5.2 mostra o resultado do teste realizado no ambiente implementado.

Método	**Erro médio**	**Iterações Médias por Ponto**	**media de custo computacional por iteração**	**Tempo médio para convergência de uma iteração**

Gradiente-Descendente	9.75e-07	214	54.834	11.728
BFGS	8.83e-09	42	652.64	27.411
L-BFGS-B	0.0002267	35	357	12.495
Nelder-Mead	0.002239	261	423.82	110.620

Tabela 5.2 – Média de erro e iteração ao longo de um movimento. Fonte: Autor.

Os resultados apresentados na Tabela 5.2 mostram que o Gradiente Descendente requer um número significativamente maior de iterações para convergir (214,71 em média) em comparação com os métodos BFGS e L-BFGS-B. Isso está alinhado com a literatura, já que o Gradiente Descendente possui convergência linear, enquanto BFGS e L-BFGS-B apresentam convergência quase quadrática, explorando o espaço de busca de forma mais eficiente. Enquanto o Gradiente Descendente utiliza apenas o gradiente para determinar a direção de busca, o BFGS e o L-BFGS-B incorporam uma aproximação da matriz Hessiana, permitindo uma melhor exploração do espaço de busca e um número menor de iterações para alcançar a solução.

Os métodos BFGS e L-BFGS-B realizam cálculos mais complexos por iteração devido à aproximação da matriz Hessiana, mas compensam essa complexidade com menor número de iterações. O BFGS, por exemplo, apresenta um erro médio muito baixo $8{,}83 \times 10^{-9}$ e requer apenas 42,74 iterações em média, demonstrando sua eficiência em problemas de alta precisão. O L-BFGS-B, uma variante de memória limitada, equilibra eficiência computacional e precisão ao apresentar um erro médio de 0,00022 e uma média de 35,78 iterações, sendo particularmente útil em problemas de alta dimensionalidade ou com restrições de limite.

Por outro lado, o Nelder-Mead, que não utiliza informações de gradiente ou Hessiana, depende de ajustes no simplex, resultando em um número elevado de iterações (261,516 em média) e um erro maior (0,00223). Apesar de sua simplicidade e adequação a problemas sem gradiente disponível, sua eficiência em termos de convergência é limitada, tornando-o menos competitivo em aplicações que demandam alta precisão.

O Gradiente Descendente com pseudo-inversa da Jacobiana, embora apresente cálculos mais simples por iteração, requer um número elevado de iterações para alcançar uma solução, refletindo sua menor eficiência em explorar o espaço de busca. Esses resultados

indicam que o Gradiente Descendente, apesar de seu baixo erro médio $9{,}75 \times 10^{-7}$, é menos adequado para aplicações em tempo real devido ao seu elevado número de iterações.

Uma representação visual do comportamento dos erros de cada método em relação a estabilidade das juntas ao longo do movimento *"home*" pode ser vista na figura 5.2 onde, no método Nelder-Mead, a ausência de gradientes e Hessianas faz com que o método dependa unicamente do ajuste do simplex. A posição das juntas ao longo da trajetória de cada método pode ser visto na figura 5.2, onde quando mais suave é a curvatura das posições mais estável é o movimento do robô ao longo da trajetória.

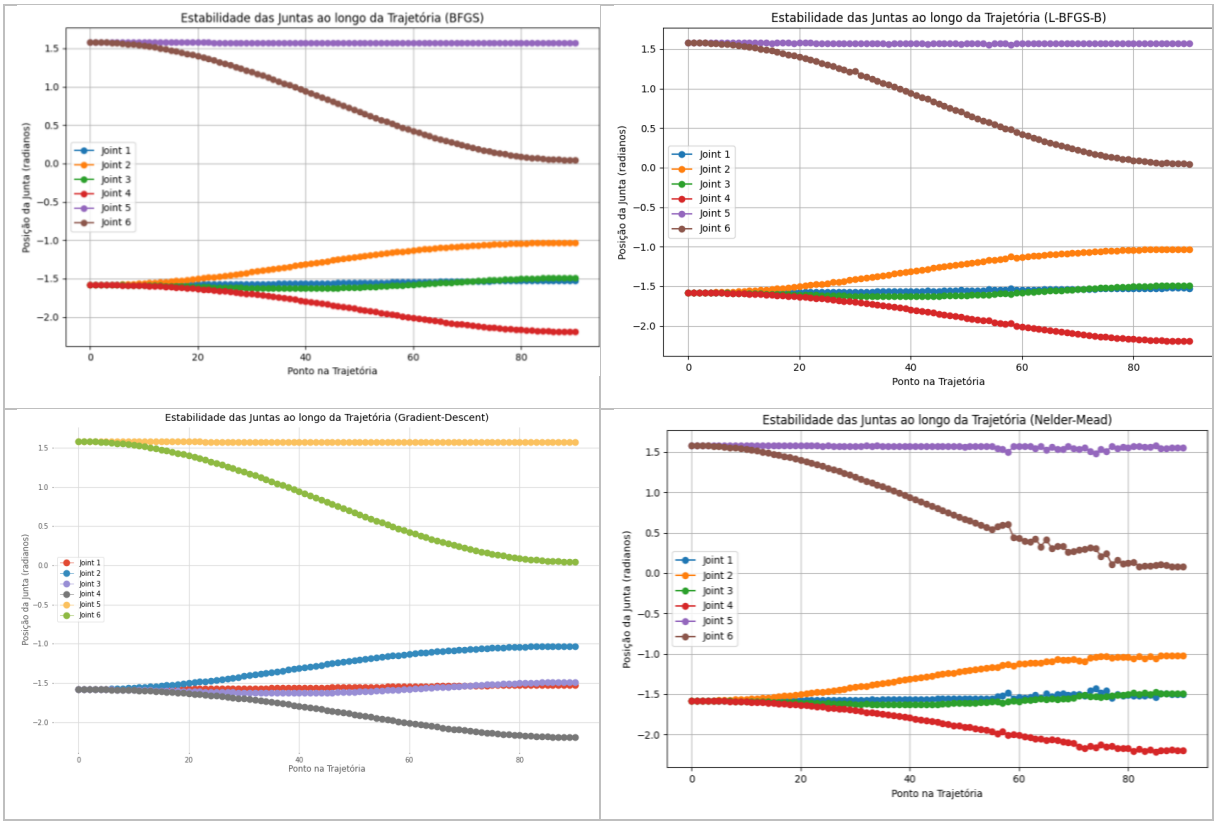

Figura 5.2 – Posição das juntas ao longo de um movimento. Fonte: Autor.

No geral, a análise das métricas estudadas confirma que os métodos baseados na matriz Hessiana, como BFGS e L-BFGS-B, são mais eficientes para resolver problemas de cinemática inversa, especialmente em cenários que exigem precisão e menor tempo computacional. Esses métodos demonstram um equilíbrio entre precisão, número de iterações e avaliações da função de custo, enquanto os métodos como Nelder-Mead e Gradiente Descendente são mais adequados para aplicações específicas ou como soluções iniciais para algoritmos mais avançados.

6.0. Conclusão

Com base na seção 1.2, esta pesquisa atingiu seus objetivos ao investigar e implementar diferentes métodos numéricos para a solução da cinemática inversa em um robô manipulador UR5e, integrando-os a um sistema baseado no ROS. O foco esteve na análise e implementação de abordagens baseadas em métodos numéricos. Esta pesquisa demonstra que as soluções numéricas são eficazes, mas possuem características específicas que influenciam seu desempenho, dependendo do método utilizado. Por exemplo, métodos baseados na matriz Hessiana, como o BFGS e o L-BFGS-B, oferecem uma relação ideal entre eficiência e precisão, sendo as mais adequadas para a aplicação de cinemática inversa no robô UR5 caso tenha que ser escolhido uma solução numérica para resolver problema de cinemática. Apesar do método analítico ser superior a ela em desempenho, carece de flexibilidade para lidar com configurações mais complexas em alguns robôs, enquanto o Gradiente Descendente com pseudo-inversa do Jacobiano apresenta limitações para aplicações em tempo real, devido ao elevado número de iterações necessárias para convergir em comparação com métodos baseados em curvatura, como o BFGS. Apesar de a pseudo-inversa do Jacobiano permitir uma maior estabilidade e robustez em situações de redundância ou singularidade, o método ainda utiliza apenas informações do gradiente, sem considerar a curvatura da função de custo. O Nelder-Mead, embora útil em casos específicos, não apresenta desempenho competitivo frente aos métodos baseados em Hessiana.

A contribuição desta pesquisa está na demonstração da viabilidade de aplicações numéricas para a solução de problemas complexos na robótica colaborativa, assim como na integração de módulos de visão computacional e controle. Sugere-se, como trabalhos futuros, a investigação de híbridos entre métodos numéricos e analíticos para melhorar a robustez e o desempenho em sistemas dinâmicos mais complexos.

Referências Bibliográficas

[1] FUJITA, M.; DOMAE, Y.; NODA, A.; GARCIA RICARDEZ, G. A.; NAGATANI, T.; ZENG, A.; et al. Quais são as tecnologias importantes para o bin picking? Análise tecnológica de robôs em competições com base em um conjunto de métricas de desempenho. Advanced Robotics, v. 34, n. 7, p. 560-574, 2020.

[2] YANG, J. et al. Survey on pick-and-place robotic systems for electronics manufacturing. IEEE Transactions on Automation Science and Engineering, v. 14, n. 4, p. 1458-1474, 2017.

[3] TSAI, Chi-Yi; WONG, Ching-Chang; YU, Chia-Jun; LIU, Chih-Cheng; LIU, Tsung-Yen. A Hybrid Switched Reactive-Based Visual Servo Control of 5-DOF Robot Manipulators for Pick-and-Place Tasks. IEEE Systems Journal, v. 9, n. 1, p. 1-10, mar. 2015.

[4] FEI, Tao; QINGLIN, Qi. Service-oriented Smart Manufacturing. Journal of Mechanical, v. 54, n. 16, p. 11-23, 2018.

[5] CHOWDHURY, Muhammed; PAUL, Shuvo Kumar; NICOLESCU, Monica; NICOLESCU, Mircea; FEIL-SEIFER, David; DASCALU, Sergiu. Computation of Suitable Grasp Pose for Usage of Objects Based on Predefined Training and Real-time Pose Estimation. In: The Sixteenth International Conference on Autonomic and Autonomous, 2020.

[6] ACACCIA, G.; BRUZZONE, L.; RAZZOLI, R. A modular robotic system for industrial applications. Assembly Automation, Emerald Journal, v. 28, n. 2, p. 151-162, 2008.

[7] PETERS, Lloyd S.; BRACKMANN, Elizabeth J.; PARK, William T. Future Directions In Automation and Robotics For Manufacturing. International Electronic Robotics Into Manufacturing and Suggests Development. IEEE, 1987.

[8] SUCAN, A.; CHITTA, S. MoveIt! Disponível em: http://moveit.ros.org. Acesso em: 22 nov. 2022.

[9] UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION (UNIDO). Industrial Development Report 2016: The Role of Technology and Innovation in Inclusive and Sustainable Industrial Development. 2015. ISBN: 978-92-1-106454-4.

[10] NAOHIKO, I.; HIROTO, N.; HIKARU, K. Utilization of AI in the Manufacturing Sector - Case Studies and Outlook for Linked Factories. Hitachi Review, v. 65, n. 6, p. 145-149, 2016.

[11] KOLBEINSSON, A.; LAGERSTEDT, E.; LINDBLOM, J. Foundation for a classification of collaboration levels for human-robot cooperation in manufacturing. Production and Manufacturing Research, v. 7, n. 1, p. 448-471, 2019.

[12] ROSER, Christoph. Publications and Presentations. Offenbach, Germany. Disponível em: https://AllAboutLean.com. Acesso em: 10 fev. 2023.

[13] A. Weiss, A. -K. Wortmeier and B. Kubicek, "Cobots in Industry 4.0: A Roadmap for Future Practice Studies on Human–Robot Collaboration," in IEEE Transactions on Human-Machine Systems, vol. 51, no. 4, pp. 335-345, Aug. 2021, doi: 10.1109/THMS.2021.3092684.

[14] A. Tellaeche, I. Maurtua and A. Ibarguren, "Use of machine vision in collaborative robotics: An industrial case," *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Berlin, Germany, 2016, pp. 1-6, doi: 10.1109/ETFA.2016.7733689.

[15] J. -J. Kim, W. Lee and S. Kang, "Pick-and-place task with manipulator by modular approach," 2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, Korea (South), 2017, pp. 202-203, doi: 10.1109/URAI.2017.7992712.

[16] P. -C. Huang and A. K. Mok, "A Case Study of Cyber-Physical System Design: Autonomous Pick-and-Place Robot," 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hakodate, Japan, 2018, pp. 22-31, doi: 10.1109/RTCSA.2018.00012.

[17] Y. Fu, G. Zhu, M. Zhu, et al., Digital twin for integration of design-manufacturing maintenance: An overview, Chin. J. Mech. Eng. 35 (80) (2022) http://dx.doi.org/ 10.1186/s10033-022-00760-x, 2022.

[18] REZENDE, Luiz Gustavo; ROSSINI, Flávio Luiz. “Simulação da cinemática inversa de um robô SCARA através da implementação de algoritmos por métodos numéricos. In: XIII Seminário de Extensão e Inovação e XXVIII Seminário de Iniciação Científica e Tecnológica da UTFPR”, 20 a 23 de novembro de 2023, Ponta Grossa, PR. Ponta Grossa: UTFPR, 2023. Disponível em: https://seisicite.com.br. Acesso em: 3 nov. 2024.

[19] DIAS, João Vitor. Evolução diferencial híbrida com o método BFGS para a resolução do problema de despacho econômico com o efeito de ponto de carregamento de válvula. 2021. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Estadual Paulista, Faculdade de Engenharia de Bauru, Bauru, SP, 2021.

[20] P. K. Chemelil, J. G. Njiri and J. K. Kimotho, "Real time object detection using single shot multibox detector network for autonomous robotic arm", J. Sustain. Res. Eng., vol. 6, no. 1, pp. 11-23, 2020.

[21] M. Schwarz and S. Behnke, "Data-efficient deep learning for RGB-D object perception in cluttered bin picking", *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, pp. 2-4, May 2017.

[22] C. -C. Wong et al ., "Generic Development of Bin Pick-and-Place System Based on Robot Operating System", em IEEE Access , vol. 10, pp. 65257-65270, 2022, doi: 10.1109/ACCESS.2022.3182114.

[23] ROYER, Clément W.; O'NEILL, Michael; WRIGHT, Stephen J. A Newton-CG Algorithm with Complexity Guarantees for Smooth Unconstrained Optimization. Wisconsin Institute for Discovery and Computer Sciences Department, University of Wisconsin. Disponível em: https://optimization-online.org/wp-content/uploads/2018/03/6513.pdf Acesso em: 06 nov. 2024.

[24] MENDONÇA, Melissa Weber. O Método L-BFGS com Fatoração Incompleta para a Resolução de Problemas de Minimização. 2005. Dissertação (Mestrado em Matemática Aplicada) – Universidade Federal de Santa Catarina, Florianópolis, 2005.

[25] MÜLLER, R.; VETTE, M.; GEENEN, A. Skill-based Dynamic Task Allocation in HumanRobot-Cooperation with the Example of Welding Application. Procedia Manufacturing, v. 11, n. June, p. 13–21, 2017.

[26] What is Industrie 4.0?. Plataform Industrie 4.0. 2023. Disponível em: <https://www.plattform-i40.de/IP/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.html>. Acesso em 02 de jan. de 2023.

[27] Müller, Christopher: World Robotics 2022 – Industrial Robots, IFR Statistical Department, VDMA Services GmbH, Frankfurt am Main, Germany, 2022.

[28] ROBOTS, U. e-Series. Built to do more. Disponível em: <https://www.universal-robots.com/pt/produtos/ur5-robot/>. Acesso em: 2 jun. 2023.

[29] GUTTA, S. Object Detection Algorithm — YOLO v5 Architecture. Disponível em: <https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-89e0a35472ef>. Acesso em: 14 dez. 2022.

[30] NVIDIA, Vingelmann, P., & Fitzek, F. H. P. (2020). Convolutional Neural Network . Disponível em <https://developer.nvidia.com/discover/convolutional-neural-network>. Acesso em: 20 jan. 2022.

[31] Open-Source Robotics Foundation. Robotic Operating System. Disponível em: https://www.ros.org. Acesso em: 20 jul. 2023.

[32] SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. Robot Modeling and Control. [S.l.]: John Wiley & Sons, Inc., 2005. ISBN 0471649902.

[33] Low KH, Dubey RN (1987) A comparative study of generalized coordinates for solving the inverse-kinematics problem of a 6r robot manipulator. International Journal of Robotics Research 5(4):69-88

[34] A. Balestrino, G. De Maria, and L. Sciavicco. “Robust Control of Robotic Manipulators”. In: *IFAC Proceedings Volumes* 17.2 (1984). 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984, pp. 2435–2440. issn: 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)61347-8. url: https://www.sciencedirect.com/science/article/pii/S147466701761347.

[35] W. A. Wolovich and H. Elliott. A computational technique for inverse kinematics. 1984. doi:10.1109/CDC.1984.272258.

[36] Kritzinger, W., Karner, M., Traar, G., Henjes, J., and Sihn, W. (2018). Digital Twin in Manufacturing: A Categorical Literature Review and Classification. IFAC-PapersOnLine 51, 1016–1022. doi:10.1016/j.ifacol.2018.08.474

[37] Carrara, V. “Apostila de Robótica”, 1999. Universidade de Braz Cubas. Mogi das Cruzes. Disponível em: <http://www.valcar.net/cursos/rb_apostila.pdf>. Acesso em junho de 2017.

[38] LAVALLE, S. M. Planning Algorithms. Disponível em: <http://planning.cs.uiuc.edu/node102.html>. Acesso em: 17 fev. 2024.

[39] Bouteille, D., Bouteille, N., Chantreul, S., et al., 1997, “Les Automatismes Programables. Cépaduès-éditions”, 2 ed., Toulouse.

[40] ISO**,** Industrial robots. Manipulators. Disponível em: <https://www.iso.org/ics/25.040.30/x/>. Acesso em: 12 jul. 2024.

[41] SICILIANO, B. et al. Robotics: Modelling, Planning and Control. [S.l.]: Springer, 2009. ISBN 978-1-84628-641-4.

[42] SPONG, M. W.; VIDYASAGAR, M. Robot Dynamics and Control. [S.l.]: John Wiley & Sons, Inc., 1989. ISBN 0-471-61243-X.

[43] ROKBANI, N.; CASALS, A.; ALIMI, A. M. Ik-fa, a new heuristic inverse kinematics solver using firefly algorithm. In: Computational Intelligence Applications in Modeling and Control. [S.l.]: Springer, 2015. p. 369–395.

[44] R. Mattone, G. Campagiorni, and F. Galati, “Sorting of items on a moving conveyor belt. part i. a technique for detecting and classifying objects,” Robotics and Computer-Integrated Manufacturing, vol. 16, no. 2-3, pp. 73–80, 2000.

[45] T. Gecks, D. Henrich, and Ieee, Human-robot cooperation: Safe pick-and-place operations, ser. 2005 IEEE International Workshop on Robot and Human Interactive Communication. New York: Ieee, 2005.

[46] L. D. Wang, L. Ren, J. K. Mills, and W. L. Cleghorn, “Automated 3-d micrograsping tasks performed by vision-based control,” Ieee Transactions on Automation Science and Engineering, vol. 7, no. 3, pp. 417–426, 2010.

[47] LAVALLE, S. M. Planning Algorithms. Disponível em: < https://lavalle.pl/planning/ >. Acesso em: 17 fev. 2024.

[48] Wikipédia. Parâmetros de Denavit-Hartenberg — Wikipedia, a enciclopédia livre. https://pt.wikipedia.org/wiki/Par%C3%A2metros_de_Denavit-Hartenberg, 2024. [Online]. Acessado em 21 de outubro de 2024.

[49] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, “Inverse kinematics techniques in computer graphics: A survey”, Comput. Graph. Forum, vol. 37, no. 6, pp. 35–58, Nov. 2017. Accessed: Jul. 9, 2024. [Online]. Available: https://doi.org/10.1111/cgf.13310.

[50] Knapp and Cobet, 2000. Wolfgang Knapp, Matthias Cobet, “The IWF Hexaglide: A new concept for high-speed machining”, IWF, Zurich, November, 2000.

[51] STARKE, S.; HENDRICH, N.; ZHANG, J. Memetic evolution for generic full-body inverse kinematics in robotics and animation. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 23, n. 3, p. 406–420, 2018. Citado na página 22.

[52] MECACADEMIC. *What are Singularities in a 6-axis Robot Arm?* Disponível em: https://mecademic.com/insights/academic-tutorials/what-are-singularities-6-axis-robot-arm/. Acesso em: 3 nov. 2024.

[53] NOCEDAL, J.; WRIGHT, S. *Numerical optimization*. [S.l.]: Springer, 2006. Acesso em: 6 out. 2024.

[54] BAZARAA, M. S.; SHERALI, H. D.; SHETTY, C. M. *Nonlinear programming: theory and algorithms*. [S.l.]: New York: John Wiley & Sons, 1979. Citado 6 vezes nas páginas 19, 32, 34, 35, 40 e 111.

[55] NOCEDAL, J., “Updating quasi-Newton matrices with limited storage", Mathematics of Computation 35, 773-782, 1980.

[56] REMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. *You only look once: Unified, real-time object detection*. University of Washington; Allen Institute for AI; Facebook AI Research, 2016. Disponível em: https://arxiv.org/abs/1506.02640. Acesso em: 3 nov. 2024.

[57] RAJPUT, M. YOLO V5 — Explained and Demystified. Disponível em: <https://towardsai.net/p/computer-vision/yolo-v5 - explained-and-demystified>. Acesso em: 10 out. 2024.

[58] WANG, C. et al. CSPNET: A NEW BACKBONE THAT CAN ENHANCE LEARNING CAPABILITY OF CNN. 2019.

[59] XU, R. et al. A forest fire detection system based on ensemble learning. **Forests**, v. 12, n. 2, p. 1–17, 2021.

[60] ULTRALYTICS. YOLOv5: Versão oficial do YOLOv5 no GitHub. Disponível em: https://github.com/ultralytics/yolov5. Acesso em: 3 nov. 2024.

[61] ROS-UNITN. Legoblocks: Lego blocks YOLOv5 dataset. Disponível em: https://github.com/ros-unitn/legoblocks. Acesso em: 21 nov. 2024.

[62] UNIVERSAL ROBOTS. DH parameters for calculations of kinematics and dynamics. Disponível em: https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/. Acesso em: 1 nov. 2024.

[63] Gao, F. and Han, L. Implementing the Nelder-Mead simplex algorithm with adaptive parameters. 2012. Computational Optimization and Applications. 51:1, pp. 259-277.

[64] FONTINELE, H. I. P. Modelos locais para aproximação da cinemática inversa de robôs redundantes: um estudo comparativo. 2015. Dissertação (Mestrado em Engenharia de Teleinformática) – Universidade Federal do Ceará, Fortaleza, 2015.

[65] ERTHAL, J. L. Estudo de métodos para a solução da cinemática inversa de robôs industriais para implementação computacional. 1992. Dissertação (Mestrado em Engenharia Mecânica) – Universidade Federal de Santa Catarina, Florianópolis, 1992.

[66] RUPPEL, P. Performance optimization and implementation of evolutionary inverse kinematics in ROS. 2018. Master's Thesis (Master of Science in Automation Engineering) – Hochschule Offenburg, Offenburg, 2017.

[67] TAVARES, Pedro Miguel Santos. Planeamento de trajetórias em manipuladores em ambientes industriais. 2015. Dissertação (Mestrado Integrado em Engenharia Eletrotécnica e de Computadores) – Faculdade de Engenharia, Universidade do Porto, 2015.

[68] T. Sugihara, "Solvability-unconcerned inverse kinematics based on Levenberg-Marquardt method with robust damping," *2009 9th IEEE-RAS International Conference on Humanoid Robots*, Paris, France, 2009, pp. 555-560, doi: 10.1109/ICHR.2009.5379515.

APÊNDICE A – Código para a resolução da cinemática direta pelo DH

```python
#UR5 parametros
d    = [0.089159,  0.00000,  0.00000,  0.10915,  0.09465,  0.0903 + 0.1628]
a    = [0.00000,  -0.42500, -0.39225,  0.00000,  0.00000,  0.0000]
alpha = [pi/2,      0,        0,        pi/2,     -pi/2,    0]

# ************************************************ Calculo da cinematica direta

def HT(n, th):
    T_a = np.identity(4)
    T_a[0, 3] = a[n - 1]

    T_d = np.identity(4)
    T_d[2, 3] = d[n - 1]

    Rzt = arr(
        [[cos(th[n - 1]), -sin(th[n - 1]), 0, 0],
         [sin(th[n - 1]), cos(th[n - 1]),  0, 0],
         [0,                 0,                  1, 0],
         [0,                 0,                  0, 1]])

    Rxa = arr(
        [[1, 0,                0,                 0],
         [0, cos(alpha[n - 1]), -sin(alpha[n - 1]), 0],
         [0, sin(alpha[n - 1]), cos(alpha[n - 1]),  0],
         [0, 0,                0,                 1]])

    return T_d @ Rzt @ T_a @ Rxa


def forward(th):
    A_1 = HT(1, th)
    A_2 = HT(2, th)
    A_3 = HT(3, th)
    A_4 = HT(4, th)
    A_5 = HT(5, th)
    A_6 = HT(6, th)

    T_06 = A_1 @ A_2 @ A_3 @ A_4 @ A_5 @ A_6

    return T_06
```

APÊNDICE B – Código para a resolução da cinemática inversa pelo método BFGS

```python
def optimize_joints(initial_joints, target_position, target_rotation):
    """
    Otimiza os ângulos das juntas do robô para alcançar uma posição e orientação alvo.

    Parâmetros:
    - initial_joints (list ou np.array): Ângulos iniciais das juntas do robô.
    - target_position (list ou np.array): Posição desejada do end-effector no espaço cartesiano (x, y, z).
    - target_rotation (np.array): Matriz de rotação 3x3 representando a orientação desejada do end-effector.

    Retorna:
    - result.x (np.array): Ângulos das juntas otimizados para alcançar a pose desejada.
    - joint_trajectory (list): Histórico dos ângulos das juntas ao longo das iterações de otimização.
    - time_trajectory (np.array): Tempo em que cada iteração foi concluída, relativo ao início da otimização.
    """

    # Lista para armazenar a trajetória dos ângulos das juntas durante a otimização
    joint_trajectory = []

    # Lista para armazenar os tempos de cada iteração
    time_trajectory = []

    def cost_function(th, target_pose):
        """
        Calcula a função de custo para a otimização.
        A função de custo é baseada no erro entre a posição e a orientação atuais e desejadas.

        Parâmetros:
        - th (np.array): Ângulos das juntas atuais.
        - target_pose (np.array): Vetor combinado da posição desejada (3 valores) e rotação desejada (9 valores).

        Retorna:
        - float: Soma do erro posicional e rotacional.
        """
        # Calcula a matriz de transformação do end-effector com os ângulos fornecidos
        T_06 = forward(th)

        # Extração da posição e orientação atuais do end-effector
        current_pose = T_06[:3, 3]  # Posição atual
        current_rot = T_06[:3, :3]  # Orientação atual (matriz de rotação)

        # Separação da posição e orientação desejadas
        target_pos = target_pose[:3]
        target_rot = target_pose[3:].reshape((3, 3))  # Reconstrói a matriz de rotação desejada

        # Calcula o erro posicional e o erro rotacional
        pos_error = np.linalg.norm(current_pose - target_pos)  # Distância Euclidiana
        rot_error = np.linalg.norm(current_rot - target_rot)  # Norma da diferença das matrizes de rotação

        return pos_error + rot_error  # Soma dos erros como custo total

    def callback(xk):
        """
        Função de callback chamada após cada iteração da otimização.
        Armazena os ângulos das juntas e o tempo de execução.

        Parâmetros:
        - xk (np.array): Ângulos das juntas na iteração atual.
        """
        joint_trajectory.append(np.copy(xk))  # Salva o estado atual das juntas
        time_trajectory.append(time.time())  # Registra o tempo da iteração

    # Chute inicial para a otimização (ângulos iniciais das juntas)
    initial_guess = np.array(initial_joints)

    # Combina a posição e a orientação alvo em um único vetor
    target_pose = np.hstack([target_position, target_rotation.flatten()])

    # Chama o otimizador BFGS para minimizar a função de custo
    result = minimize(
        cost_function,  # Função de custo
        initial_guess,  # Ângulos iniciais das juntas
        args=(target_pose,),  # Argumentos adicionais para a função de custo
        method='BFGS',  # Método de otimização
        callback=callback,  # Função de callback
        options={'disp': False, 'gtol': 1e-6}  # Configurações do otimizador
    )

    # Ajusta os tempos de iteração para serem relativos ao início
    if time_trajectory:
        start_time = time_trajectory[0]
        time_trajectory = np.array(time_trajectory) - time_trajectory[0]  # Normaliza os tempos em relação ao início

    # Retorna os resultados: ângulos otimizados, trajetória das juntas e tempos
    return result.x, joint_trajectory, time_trajectory
```

APÊNDICE C – Código para a resolução da cinemática inversa pelo método L-BFGS.

```python
def cost_function(th, target_pose):
    """
    Calcula a função de custo para a otimização dos ângulos das juntas do robô.
    A função avalia o erro total, combinando erros de posição e orientação.

    Parâmetros:
    - th (np.array): Ângulos atuais das juntas do robô.
    - target_pose (np.array): Vetor combinado contendo a posição desejada (3 valores)
      e a rotação desejada achatada (9 valores).

    Retorna:
    - float: Valor da função de custo, ponderando erros posicional e rotacional.
    """

    # Calcula a matriz de transformação do end-effector com os ângulos fornecidos
    T_06 = forward(th)

    # Extrai a posição e orientação atuais do end-effector
    current_pose = T_06[:3, 3]  # Posição atual do end-effector
    current_rot = T_06[:3, :3]  # Matriz de rotação atual do end-effector

    # Extrai a posição e orientação desejadas
    target_pos = target_pose[:3]  # Posição desejada
    target_rot = target_pose[3:].reshape((3, 3))  # Reconstrói a matriz de rotação desejada

    # Calcula os erros posicional e rotacional
    pos_error = np.linalg.norm(current_pose - target_pos)  # Erro de posição
    rot_error = np.linalg.norm(current_rot - target_rot)  # Erro de rotação

    # Ponderações para os erros
    pos_weight = 1.0  # Peso do erro de posição
    rot_weight = 0.1  # Peso do erro de rotação

    # Retorna o custo total ponderado
    return pos_weight * pos_error + rot_weight * rot_error


def optimize_joints(initial_joints, target_position, target_rotation):
    """
    Otimiza os ângulos das juntas do robô para alcançar uma posição e orientação alvo,
    respeitando os limites das juntas.

    Parâmetros:
    - initial_joints (list ou np.array): Ângulos iniciais das juntas do robô.
    - target_position (list ou np.array): Posição desejada do end-effector no espaço cartesiano (x, y, z).
    - target_rotation (np.array): Matriz de rotação 3x3 representando a orientação desejada do end-effector.

    Retorna:
    - result (OptimizeResult): Resultado da otimização contendo os ângulos das juntas otimizados.
    - errors[-1] (float): Valor final da função de custo após a otimização.
    - joint_trajectory (list): Histórico dos ângulos das juntas ao longo das iterações.
    """

    # Converte os ângulos iniciais para um array numpy
    initial_pose = np.array(initial_joints)

    # Lista para armazenar os erros durante a otimização
    errors = []

    # Lista para armazenar a trajetória dos ângulos das juntas durante a otimização
    joint_trajectory = []

    # Combina a posição e orientação alvo em um único vetor
    target_pose = np.hstack([target_position, target_rotation.flatten()])

    def callback(params):
        """
        Função de callback chamada após cada iteração da otimização.
        Armazena os erros e os ângulos das juntas para análise posterior.

        Parâmetros:
        - params (np.array): Ângulos das juntas na iteração atual.
        """
        errors.append(cost_function(params, target_pose))  # Calcula e armazena o erro atual
        joint_trajectory.append(params.copy())  # Armazena os ângulos das juntas

    # Limites das juntas (definidos com base nas especificações do robô)
    joint_limits = [
        (-np.pi, np.pi),        # Junta 1
        (-np.pi / 2, np.pi / 2),  # Junta 2
        (-np.pi / 2, np.pi / 2),  # Junta 3
        (-np.pi, np.pi),        # Junta 4
        (-np.pi / 2, np.pi / 2),  # Junta 5
        (-np.pi, np.pi)         # Junta 6
    ]

    # Executa a otimização utilizando o método L-BFGS-B com limites
    result = minimize(
        cost_function,          # Função de custo
        initial_pose,           # Chute inicial para os ângulos das juntas
        args=(target_pose,),    # Argumentos adicionais para a função de custo
        method='L-BFGS-B',      # Método de otimização
        bounds=joint_limits,    # Limites das juntas
        callback=callback,      # Função de callback
        options={
            'disp': True,       # Exibe o progresso no console
            'maxiter': 5000,    # Número máximo de iterações
            'gtol': 0.0,        # Tolerância para o gradiente
            'ftol': 1e-6        # Tolerância para a função de custo
        }
    )

    # Exibe o resumo da otimização no console
    print("Quantidade de iterações:", result.nit)
    print("Número de avaliações da função de custo:", result.nfev)

    # Retorna os resultados: ângulos otimizados, último erro e trajetória das juntas
    return result, errors[-1], joint_trajectory
```

APÊNDICE D – Código XML para criação da cena principal no ambiente gazebo.

```xml
<?xml version="1.0" ?>

<sdf version="1.6">

  <world name="ur5_world">

    <plugin name="ros_link_attacher_plugin" filename="libgazebo_ros_link_attacher.so"/>
    <gui>
      <camera name="user_camera">
        <pose>1.4 -2.3 1.4 0.0 0.25 1.9</pose>
      </camera>
    </gui>

    <gravity>0 0 -9.81</gravity>

    <physics name="default_physics" default="0" type="ode">
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1</real_time_factor>
      <real_time_update_rate>1000</real_time_update_rate>
    </physics>

    <scene>
      <ambient>0.4 0.4 0.4 1</ambient>
      <background>0.7 0.7 0.7 1</background>
      <shadows>false</shadows>
    </scene>

    <!-- Light Source -->
    <include>
      <uri>model://sun</uri>
    </include>

    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
      <pose>0 0 0 0 0 0</pose>
    </include>


    <model name="ur5_base">
      <static>true</static>
      <include>
        <uri>model://ur5_base</uri>
         <pose>0 0.14 0.02 0 0 0</pose>
      </include>
    </model>

    <model name="kinect">
      <static>true</static>
      <include>
        <uri>model://kinect</uri>
        <pose>-0.44 -0.50 1.58 1.58 1.57079 0</pose>

      </include>
    </model>
  </world>
</sdf>

```

APÊNDICE E – Código para a resolução da cinemática inversa usando pseuda-inversa do jacobiano e gradiente descendente.

```python
def compute_error(current_pose, target_pose):
    """Calcula o vetor de erro entre a pose atual e a pose alvo."""
    current_pos = current_pose[:3, 3]
    target_pos = target_pose[:3]
    pos_error = target_pos - current_pos

    current_rot = current_pose[:3, :3]
    target_rot = target_pose[3:].reshape((3, 3))
    rot_error = 0.5 * (np.cross(current_rot[:, 0], target_rot[:, 0]) +
                       np.cross(current_rot[:, 1], target_rot[:, 1]) +
                       np.cross(current_rot[:, 2], target_rot[:, 2]))

    return np.hstack([pos_error, rot_error])

def pseudo_inverse_jacobian(jacobian):
    """Calcula a pseudo-inversa do Jacobiano usando SVD."""
    u, s, vh = np.linalg.svd(jacobian, full_matrices=False)
    s_inv = np.diag(1.0 / s)
    return vh.T @ s_inv @ u.T

def optimize_joints(initial_joints, target_position, target_rotation, max_iters=5000, tolerance=1e-6, learning_rate=0.05):
    """Resolve a cinemática inversa usando pseudo-inversa do Jacobiano e gradiente descendente."""
    th = np.array(initial_joints)
    target_pose = np.hstack([target_position, target_rotation.flatten()])
    errors = []
    joint_trajectory = []
    UR5 = rtb.models.DH.UR5()

    for i in range(max_iters):
        # Calcula a pose atual usando a cinemática direta
        current_pose = forward(th)

        # Calcula o erro atual
        error = compute_error(current_pose, target_pose)
        errors.append(np.linalg.norm(error))
        joint_trajectory.append(th.copy())

        # Verifica a convergência
        if np.linalg.norm(error) < tolerance:
            print(f"Convergiu após {i + 1} iterações.")
            break

        # Calcul o Jacobiano
        jacobian =  UR5.jacob0(th)

        # Calcula a pseudo-inversa do Jacobiano
        jacobian_pseudo_inv = pseudo_inverse_jacobian(jacobian)

        # Atualiza os ângulos das juntas
        delta_th = learning_rate * jacobian_pseudo_inv @ error
        th += delta_th

    print("Quantidade de iterações:", i + 1)
    print("Erro final:", errors[-1] if errors else None)
    return {'x': th}, errors, joint_trajectory
```

APÊNDICE F – Código para a resolução da cinemática inversa usando gradiente descendente.

```python

def cost_function(th, target_pose):
    """Calcula o custo com base no erro entre a pose atual e a pose alvo."""
    T_06 = forward(th)
    current_pos = T_06[:3, 3]
    current_rot = T_06[:3, :3]
    target_pos = target_pose[:3]
    target_rot = target_pose[3:].reshape((3, 3))

    # Erro de posição
    pos_error = np.linalg.norm(current_pos - target_pos)

    # Erro de rotação (norma de Frobenius)
    rot_error = np.linalg.norm(current_rot - target_rot)

    # Pesos para posição e rotação
    pos_weight = 1.0
    rot_weight = 0.1

    # Custo total
    return pos_weight * pos_error + rot_weight * rot_error

def compute_gradient(th, target_pose, epsilon=1e-5):
    """Calcula o gradiente numericamente usando diferenças centrais."""
    grad = np.zeros_like(th)
    for i in range(len(th)):
        th_plus = th.copy()
        th_minus = th.copy()
        th_plus[i] += epsilon
        th_minus[i] -= epsilon
        grad[i] = (cost_function(th_plus, target_pose) - cost_function(th_minus, target_pose)) / (2 * epsilon)
    return grad

def optimize_joints(initial_joints, target_position, target_rotation, max_iters=5000, tolerance=1e-6, learning_rate=0.01):
    """Resolve a cinemática inversa usando gradiente descendente."""
    th = np.array(initial_joints)
    target_pose = np.hstack([target_position, target_rotation.flatten()])
    errors = []
    joint_trajectory = []

    for i in range(max_iters):
        # Calcula o custo atual
        cost = cost_function(th, target_pose)
        errors.append(cost)
        joint_trajectory.append(th.copy())

        # Verifica a convergência
        if cost < tolerance:
            print(f"Convergiu após {i + 1} iterações.")
            break

        # Calcula o gradiente da função de custo
        grad = compute_gradient(th, target_pose)

        # Atualiza os ângulos das juntas
        th -= learning_rate * grad

    print("Quantidade de iterações:", i + 1)
    print("Erro final da função de custo:", cost)
    return th, errors, joint_trajectory

```

APÊNDICE E – Código para plotar a estabilidade das juntas.

```python
def plot_joint_stability(joint_trajectory, method_name):
    """
    Plota a estabilidade de cada junta ao longo da trajetória otimizada.

    Parâmetros:
    - joint_trajectory (list ou np.array): Trajetória das juntas (NxM), onde N é o número de pontos na trajetória
      e M é o número de juntas. Cada linha representa as posições das juntas em um ponto específico da trajetória.
    - method_name (str): Nome do método de otimização utilizado (usado no título e legenda do gráfico).

    Saída:
    - Um gráfico mostrando a posição de cada junta em função dos pontos na trajetória.
    - O gráfico é salvo como uma imagem com o nome `joint_stability_<method_name>.png`.
    """
    # Converter joint_trajectory para numpy array, se necessário
    joint_trajectory = np.array(joint_trajectory)

    # Verifica se joint_trajectory tem 2 dimensões (NxM)
    if joint_trajectory.ndim != 2:
        raise ValueError(
            f"joint_trajectory deve ser uma matriz 2D (NxM), mas tem formato {joint_trajectory.shape}. "
            "Verifique os dados de entrada para garantir que representam posições das juntas ao longo da trajetória."
        )

    # Número de pontos na trajetória (N) e número de juntas (M)
    num_points, num_joints = joint_trajectory.shape

    # Iterações correspondentes a cada ponto da trajetória
    iterations = np.arange(num_points)

    # Configuração da figura do gráfico
    plt.figure(figsize=(10, 6))

    # Loop para plotar a trajetória de cada junta
    for i in range(num_joints):
        plt.plot(
            iterations,               # Eixo X: Ponto na trajetória
            joint_trajectory[:, i],   # Eixo Y: Posição da junta i
            label=f'Joint {i+1}',     # Rótulo para a legenda
            marker='o'                # Marcadores para maior clareza
        )

    # Configurações gerais do gráfico
    plt.xlabel('Ponto na Trajetória')  # Rótulo do eixo X
    plt.ylabel('Posição da Junta (radianos)')  # Rótulo do eixo Y
    plt.title(f'Estabilidade das Juntas ao longo da Trajetória ({method_name})')  # Título do gráfico
    plt.grid(True)  # Adiciona uma grade para facilitar a leitura
    plt.legend(title='Juntas')  # Adiciona a legenda, identificando cada junta

    # Salvar o gráfico como uma imagem (PNG) com base no nome do método
    plt.savefig(f'joint_stability_{method_name}.png')

    # Opcional: descomente plt.show() se quiser exibir o gráfico interativamente
    # plt.show()
```