

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

SAMUEL HENRIQUE LIMA DA SILVA

**MODELAGEM E SIMULAÇÃO DO MANIPULADOR
ROBÓTICO UR5 NO COPPELIASIM COM OTIMIZAÇÃO
DE POSIÇÃO**

UBERLÂNDIA, MG
2025

Samuel Henrique Lima da Silva

Modelagem e Simulação do Manipulador Robótico UR5 no CoppeliaSim com Otimização de Posição

Trabalho de Conclusão de Curso da Engenharia de Controle e Automação da Universidade Federal de Uberlândia - UFU - Campus Santa Mônica, como requisito para a obtenção do título de Graduação em Engenharia de Controle e Automação.

Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Elétrica - FEELT

Orientador Prof. Me. Éder Alves de Moura

Uberlândia, MG

2025

Samuel Henrique Lima da Silva

Modelagem e Simulação do Manipulador Robótico UR5 no CoppeliaSim com Otimização de Posição

Trabalho de Conclusão de Curso da Engenharia de Controle e Automação da Universidade Federal de Uberlândia - UFU - Campus Santa Mônica, como requisito para a obtenção do título de Graduação em Engenharia de Controle e Automação.

Trabalho aprovado em 05 de Maio de 2025.

COMISSÃO EXAMINADORA

Prof. Me. Éder Alves de Moura
Orientador

Prof. Dr. Fábio Vincenzi Romualdo da Silva
Membro Avaliador

Prof. Dr. Renato Santos Carrijo
Membro Avaliador

Uberlândia, MG
2025

“Tudo parece impossível até que seja feito.”
— *Nelson Mandela*

Agradecimentos

Agradeço primeiramente a Deus, por ter me dado saúde, força e resiliência para concluir esta etapa.

Aos meus familiares, pelo amor, apoio incondicional e incentivo em todos os momentos da minha vida acadêmica.

Ao meu orientador, Prof. Éder Alves de Moura, por sua dedicação, paciência e orientação ao longo de todo o desenvolvimento deste trabalho.

Agradeço também aos colegas e amigos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Por fim, agradeço à (UFU) e aos professores do curso, por todo o conhecimento compartilhado durante essa jornada acadêmica.

Resumo

Este trabalho consiste da utilização do ambiente CoppeliaSim para a simulação de movimentos do manipulador robótico UR5 (Universal Robot 5). O UR5 foi desenvolvido pela empresa *Universal Robots* com a proposta de ser um robô colaborativo (ou *cobot*). Este trabalho tem por objetivo desenvolver um ambiente de simulação integrado entre o CoppeliaSim e o Matlab, para o desenvolvimento de um algoritmo de coordenação da configuração espacial do manipulador robótico, buscando movimentá-lo entre dois pontos do espaço tridimensional definidos, uma configuração inicial e uma final. Para tal, esta tarefa utilizará o CoppeliaSim, que traz a implementação virtual do manipulador UR5, em conjunto com a modelagem matemática da cinemática direta e inversa e da busca de configurações espaciais adequadas, utilizando métodos de otimização, implementados no Matlab. A avaliação se deu pela movimentação entre 7 *waypoints* consecutivos e a avaliação da proposta de otimização apresentou resultados satisfatórios para esta primeira proposta.

Palavras-chaves: manipulador robótico, otimização de posições, UR5, CoppeliaSim.

Abstract

This study employs the CoppeliaSim environment to simulate the movements of the UR5 (Universal Robot 5) robotic manipulator, developed by Universal Robots as a collaborative robot (cobot). The objective is to establish an integrated simulation framework between CoppeliaSim and MATLAB to design a coordination algorithm for the spatial configuration of the manipulator, enabling its movement between two predefined points in three-dimensional space, an initial and a final configuration. To achieve this, the virtual implementation of the UR5 provided by CoppeliaSim is combined with the mathematical modeling of forward and inverse kinematics and the application of optimization methods to determine suitable spatial configurations. The evaluation was carried out through the manipulator's motion across seven consecutive waypoints, and the results of the proposed optimization approach demonstrated satisfactory performance for this preliminary study.

Keywords: robotic manipulator, position optimization, simulation, UR5, CoppeliaSim.

Lista de Figuras

Figura 1.1 – Manipulador robótico Kuka KR 1000 L750 titan.	15
Figura 2.1 – UR5 real.	20
Figura 2.2 – Esquema de juntas e elos por cores do UR5.	23
Figura 2.3 – Sistemas de referência.	24
Figura 2.4 – Sistemas de representações.	25
Figura 2.5 – Eixo ângulo de Euler.	34
Figura 2.6 – Modelagem cinemática direta.	35
Figura 2.7 – Robô com dois graus de liberdade junta rotacional no plano	38
Figura 3.1 – Esquema de comunicação.	43
Figura 3.2 – Robô UR5 estático.	50
Figura A.1 – Cena exemplificando o uso do UR5 no CoppeliaSim.	64
Figura A.2 – Menu superior no CoppeliaSim.	65
Figura A.3 – Continuação do menu de opções do simulador CoppeliaSim.	66
Figura A.4 – Continuação do menu de opções do simulador CoppeliaSim.	67
Figura A.5 – Continuação do menu de opções do simulador CoppeliaSim.	68

Lista de tabelas

Tabela 1	–	Características principais do robô UR5	21
Tabela 2	–	Comparação entre os robôs da série UR	22
Tabela 3	–	Parâmetros do manipulador UR5.	23
Tabela 4	–	Posição e orientação relativa das juntas do robô UR5, ordem ZYX. . .	49
Tabela 5	–	Tabela de <i>Waypoints</i> . Cada linha representa um conjunto de ângulos (em graus) aplicados às juntas do manipulador UR5.	54
Tabela 6	–	Comparação entre posições teóricas e simuladas do efetuador no Cop- peliaSim	55
Tabela 7	–	Comparação entre orientações angulares do sistema teóricas e simuladas do efetuador no CoppeliaSim ordem (ZYX)	55
Tabela 8	–	Erros absolutos entre as posições e orientações calculadas e as fornecidas pelo CoppeliaSim	56
Tabela 9	–	Comparação entre <code>wp_3d_ref</code> e <code>wp_3d_opt</code> obtidos pela otimização . .	59

Lista de Códigos

2.1	Exemplo de uso do lsqnonlin	41
3.1	Código de Conexão do CoppeliaSim via API remota	43
4.1	Função custo de otimização de posição do UR5	57
B.1	Código modelagem com o UR5 estático	69
C.1	Código obtenção matriz homogênea	74
D.1	Código de validação T estática	76
E.1	Código de validação T dinâmica	78
F.1	Código de otimização	83
G.1	Função que calcula a matriz de transformação homogênea dada um vetor de posição p e um vetor de rotação α (ângulos de Euler)	86
G.2	Função que calcula a matriz de transformação homogênea a partir de um vetor de posição, eixo de rotação e ângulo de rotação	87
G.3	Função que extrai o eixo de rotação, o ângulo de rotação e o vetor de posição	88
G.4	Função que extrai o vetor de posição e os ângulos de Euler a partir de uma matriz de transformação homogênea	90
G.5	Função que calcula uma transformação global acumulada	92

Lista de abreviaturas e siglas

API	Interface de Programação de Aplicações ou <i>Application Programming Interface</i>
DOF	Graus de Liberdade ou <i>Degrees of Freedom</i>
IADs	Dispositivos de assistência inteligente ou <i>Intelligent Assistance Devices</i>
PSO	Otimização por enxame de partículas ou <i>Particle Swarm Optimization</i>
ROS	Sistema Operacional de Robô ou <i>Robot Operating System</i>
SO3	Grupo Ortogonal Especial em 3 Dimensões ou <i>Special Orthogonal Group in 3 Dimensions</i>
UR5	Universal robô 5 ou <i>Universal Robot 5</i>
V-REP	Plataforma de Experimentação Virtual para Robôs ou <i>Virtual Robot Experimentation Platform</i>
WP	<i>Waypoint</i>

Lista de símbolos

α	Ângulo Alfa
β	Ângulo Beta
ϕ	Ângulo Phi
θ	Ângulo Theta
ψ	Ângulo Psi
ϑ	Ângulo Theta (variante minúscula)
e	Vetor unitário
c	Cosseno
s	Seno
$\in$	Pertence a

Sumário

1	INTRODUÇÃO	14
1.1	Contextualização	14
1.2	Objetivos	16
1.3	Organização do Trabalho	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	Manipuladores Robóticos	18
2.1.1	Descrição do manipulador robótico UR5	19
2.1.1.1	Divisão dos Elos e Distribuição de Massa do UR5	22
2.1.1.2	Parâmetros Dimensionais do Manipulador UR5	22
2.2	Sistemas de Coordenadas e Matriz de Transformação	23
2.2.1	Matriz de Transformação e Propriedades	28
2.3	Parametrização da Matriz de Rotação	31
2.3.1	Ângulos de Euler	31
2.3.2	Orientação de Euler na aplicação robótica	32
2.4	Cinemática Direta e Inversa	35
2.4.1	Cinemática Direta	35
2.4.2	Cinemática Inversa	36
2.4.3	Cinemática de manipuladores	37
2.5	Planejamento de Movimento e Otimização	38
2.5.1	MATLAB para matemática e otimização	38
2.5.2	MATLAB Optimization Toolbox	39
2.5.3	Algoritmo de otimização lsqnonlin MATLAB funcionamento	39
2.5.4	Modo de operação do lsqnonlin MATLAB	41
3	AMBIENTE DE SIMULAÇÃO E MODELAGEM	42
3.1	Ambiente de simulação: CoppeliaSim	42
3.2	Arquitetura de comunicação do MATLAB com CoppeliaSim	42
3.2.1	Principais funções da Remote API	44
3.3	Descritivo das etapas de desenvolvimento	46
3.3.1	Etapas 1 - Modelagem Estática	47
3.3.2	Etapas 2 - Modelagem incluindo rotações	51
4	RESULTADOS E ANÁLISES	54
4.1	Validação da Modelagem da Cinemática Direta	54
4.2	Implementação Cinemática Inversa por Otimização	56

4.3	Considerações finais do processo de simulação	59
5	CONCLUSÃO	60
	REFERÊNCIAS BIBLIOGRÁFICAS	61
	APÊNDICE A – DESCRIÇÃO DA INTERFACE DO COPPELIASIM	64
	APÊNDICE B – MODELAGEM UR5 ESTÁTICO	69
	APÊNDICE C – CÓDIGO DE OBTENÇÃO DE MATRIZES HOMO- GÊNEAS	74
	APÊNDICE D – CÓDIGO DE VALIDAÇÃO MATRIZ ESTÁTICA . .	76
	APÊNDICE E – CÓDIGO DE VALIDAÇÃO T DINÂMICA	78
	APÊNDICE F – CÓDIGO DE OTIMIZAÇÃO	83
	APÊNDICE G – PRINCIPAIS FUNÇÕES UTILIZADAS	86

1 Introdução

A simulação é algo muito vantajoso quando se trata de robótica; os simuladores atuais conseguem representar leis da física, são customizáveis com vários modelos de robôs, são versáteis quanto à representação de cenários e, em geral, possuem um custo de implementação menor que a criação de um robô e um cenário real, considerando que robôs reais são relativamente caros e frágeis para ambientes de testes (COLLINS et al., 2021). Neste contexto de simuladores dedicados à sistemas robóticos, este trabalho utiliza o CoppeliaSim como alternativa para a simulação de um manipulador robótico em específico, o UR5. Este simulador foi escolhido por apresentar o recurso de implementação do manipulador e integração com aplicações externas, por meio de uma interface de comunicação remota. Assim, a proposta consiste em utilizar o CoppeliaSim conjuntamente com MATLAB para simular a coordenação de movimento do manipulador UR5.

Este capítulo está estruturado da seguinte forma: a Seção 1.1 contextualiza o campo em que este trabalho está inserido e apresenta as motivações para o seu desenvolvimento; a Seção 1.2 sintetiza os objetivos que serão almejados; e a Seção 1.3 descreve a organização do restante deste trabalho.

1.1 Contextualização

O termo robô em si deriva das palavras tchecas *robota* que se define como trabalho obrigatório, e *robotnik* que seria servo, no início a ideia de se construir um robô se baseava em somente efetuar atividades repetitivas e auxiliar humanos, mas com o avançar da tecnologia o conceito de robô está indo muito além possibilitando aplicar essas máquinas em atividades cada vez mais desafiadoras que antes só humanos desempenhavam, robôs atualmente já possuem capacidade de raciocínio e uma autonomia considerável comparado a outras máquinas (MATARIĆ, 2014). Mais precisamente, no início do Século XX o termo robô apareceu com menção de dispositivo automatizado e hoje o conceito moderno de robô é mais ligado ao nível de desenvolvimento da tecnologia do momento, assim podemos explicar o que é um robô e seus verdadeiros objetivos já que podem ir muito além de máquinas automatizadas. A indústria de engenharia moderna está mudando constantemente, com a inserção de sistemas robóticos, e outros avanços tecnológicos, inseridos em diversos estágios das linhas de produção de um contexto de Indústria 4.0 (GALIN et al., 2020). Um dos grandes motivos para o aumento do uso de robôs industriais é que seu custo vem diminuindo com tempo, no decorrer da década de 1990, o preço dos robôs declinou enquanto o da mão de obra humana aumentou, ademais os robôs não estão ficando apenas mais baratos, mas estão se tornando, também, mais eficientes, rápidos, precisos e flexíveis

([CRAIG, 2005](#)).

Os robôs manipuladores são máquinas que integram componentes como sensores que possibilitam leitura de grandezas como força e velocidade de rotação por exemplo e atuadores permitem os movimentos rotativos de juntas se assemelhando ao braço humano com juntas simulando cotovelo e punho no caso de robôs com juntas rotativas, na sua maioria são controlados por Software, essa sofisticação é diretamente ligada a evolução de microcontroladores e microprocessadores ([SIMPLÍCIO; LIMA, 2016](#)). Desta forma robôs são máquinas complexas que demandam a integração de diversas frentes, se destacando a mecânica com conceitos de estática e dinâmica, a matemática para modelagem de movimentos espaciais, a teoria do controle para projetos de algoritmos que realizam movimentos ou aplicação de forças, a elétrica para interface e sensores e por fim a computação contribuindo com a programação destes dispositivos([CRAIG, 2005](#)). Na Figura 1.1 é exemplificado um manipulador robótico.

Figura 1.1 – Manipulador robótico Kuka KR 1000 L750 titan.



Fonte: Adaptado de [KUKA Roboter GmbH \(2024\)](#).

Os robôs desempenham um papel primordial atualmente na redução de custos, uma das grandes utilidades de manipuladores é aumentar a lucratividade de empresas. Estudos comprovam que, devido à alta precisão e velocidade de execução de tarefas os robôs são muito utilizados em setores produtivos devido ao alto rendimento ([KAWAAYE et al., 2022](#)). Como exemplo temos a mineração, em que é altamente recomendado o uso de robôs pois são manuseados enormes quantidades de material de forma economicamente viável e segura, ou seja se só o trabalho humano fosse considerado nesse caso este setor não alcançaria resultados positivos como se obtém atualmente ([COTA et al., 2017](#)).

O simulador utilizado no trabalho é o CoppeliaSim, denominado no passado como

V-REP (Plataforma de Experimentação Virtual para Robôs). Este simulador se mostrou bastante completo por oferecer diversos *scripts* já prontos (Python ou Lua) nos modelos, versatilidade pois possui opção de efetuar desde simulação simples até mesmo incorporar sistemas de controle mais complexos e robustez para simulação de automação industrial (ROHMER; SINGH; FREESE, 2013).

Neste sentido, a robótica aponta como uma grande área de desenvolvimento tecnológico da atualidade e o emprego de sistemas robóticos diversos permite auxiliar diversas áreas da vida humana, indo de tarefas do cotidiano, sistemas médicos à aplicações industriais. Contudo, um manipulador robótico precisa incluir em sua programação mais básica a capacidade de coordenar suas juntas móveis no sentido de mudar sua configuração espacial de forma a sair de um ponto inicial para um ponto final, estipulados. Assim, neste contexto, este trabalho explora um conjunto de estratégias que permitam modelar e implementar a movimentação entre pontos distintos no espaço do UR5 (Universal Robots, 2024; POLLAK et al., 2024).

Desta forma, no presente contexto, este trabalho se propõe a utilizar o simulador CoppeliaSim, por motivos já citados na Seção 1, para compor o ambiente de simulação do manipulador robótico UR5, em conjunto com o MATLAB, para implementar a solução das equações de cinemática inversa e do problema de otimização da configuração espacial ótima, visando coordenar o movimento do UR5 de uma configuração inicial até uma configuração final.

1.2 Objetivos

O objetivo geral desta proposta consiste em utilizar o CoppeliaSim para simular a coordenação de movimento do UR5, com a implementação do cálculo de trajetória por meio da modelagem de cinemática inversa e com o auxílio de algoritmos de otimização no MATLAB.

Como objetivos específicos, temos:

1. Configurar um ambiente de simulação integrado entre o CoppeliaSim e o MATLAB.
2. Derivar a modelagem matemática do manipulador UR5 e de equações auxiliares necessárias à sua simulação.
3. Formular um problema de otimização para ajustar a configuração espacial do manipulador.
4. Utilizar o ambiente de simulação para propor, testar e validar o sistema de otimização de posição do manipulador robótico.

1.3 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica, nesse capítulo contém todo o aparato teórico necessário para o entendimento das ferramentas e métodos utilizados no restante do trabalho, é essencial que se entenda esta parte para a construção de uma base teórica que facilitará o entendimento dos próximos passos;
- O Capítulo 3 é melhor detalhado o processo de simulação, nesse capítulo todos os detalhes são descritos, como a interface do simulador, o descritivo das etapas de desenvolvimento e modelagem.
- O Capítulo 4 são apresentados os resultados e análises como cálculo de erros, comparações com resultados esperados e resultados realmente atingidos, tudo isso com dados de valores de referência e dados vindos do processo de simulação.
- O Capítulo 5 apresenta um apanhado geral de todo o trabalho, o decorrer das etapas resumidamente e recapitulação dos objetivos iniciais, bem como enfatiza os trabalhos futuros com intuito de se continuar o processo aqui defendido.

2 Fundamentação Teórica

Este capítulo tem o intuito de detalhar todo o processo teórico que envolverá o conhecimento necessário para a modelagem cinemática (direta e inversa) do manipulador robótico, a definição de conceitos auxiliares (sistemas de coordenadas e matrizes de rotação), além de uma breve introdução aos conceitos de otimização. Esses conceitos serão importantes para o desenvolvimento do restante deste trabalho e este capítulo está organizado da seguinte forma: a Seção 2.1 descreve a teoria da cinemática juntamente com a descrição do robô escolhido como parâmetros e características construtivas; a Seção 2.2 descreve conceitos fundamentais dos sistemas de coordenadas e matriz de transformação e suas propriedades; a Seção 2.3 utiliza algumas técnicas matemáticas para obter matrizes de rotação e orientação; já a Seção 2.4 descreve o conhecimento teórico acerca de cinemática direta e inversa de manipuladores robóticos; e, ao final, a Seção 2.5 apresenta uma breve introdução à teoria de otimização.

2.1 Manipuladores Robóticos

Em todo o mundo, há uma estimativa de que existem atualmente mais de 800.000 robôs industriais em operação, principalmente no Japão, União Europeia e América do Norte (SPONG; HUTCHINSON; VIDYASAGAR, 2020, pag.3). Após um período de estagnação no final da década de 1980, a venda de robôs industriais começaram a crescer na década de 1990 e o crescimento das vendas provavelmente permanecerá forte (SPONG; HUTCHINSON; VIDYASAGAR, 2020). Dentre esses modelos, os manipuladores representam grande fatia deste mercado (ROSÁRIO, 2012). Por isso, é importante que os profissionais de engenharia envolvidos com essa temática sejam capazes de modelar o comportamento cinemático e dinâmico dessa classe de robôs.

Assim, considerando que este trabalho está dedicado à utilização do manipulador UR5 (Universal robô 5 ou *Universal Robot 5*), este capítulo está dedicado à apresentar os conceitos fundamentais utilizados na modelagem dos manipuladores robóticos e, em específico, nos aspectos relacionados ao UR5.

Para iniciar esta discussão sobre a modelagem de manipuladores, retoma-se que cinemática é uma ciência ligada à mecânica que trata do movimento de algum corpo sem considerar as forças que causam este movimento (CRAIG, 2005, p. 4). É necessário considerar alguns conceitos para compreender a cinemática de forma efetiva, como movimento que é quando um corpo muda de posição no espaço, referencial é a posição em que o observador se encontra no ambiente e por fim a trajetória que se define como sendo sucessões de posições ocupadas pelo corpo no ambiente considerado.

Para posicionar e posteriormente mover um efetuador no espaço tridimensional, é essencial utilizar um sistema de coordenadas pois ele permite definir a posição e orientação da ferramenta, possibilitando o controle preciso para uma trajetória.

A fim de descrever a posição e a orientação de um corpo no espaço, sempre atrelaremos rigidamente ao objeto um sistema de coordenadas de referência (CRAIG, 2005, p. 4), também denominado referencial ou *frame*, em inglês.

Os manipuladores consistem em elos quase rígidos. São conectados por juntas, as quais permitem o movimento relativo dos elos vizinhos. É possível estabelecer que estes são muito semelhantes à braços humanos, em que o braço e antebraço seriam os elos e o cotovelo a junta. No caso das juntas rotacionais (ou de revolução), tais deslocamentos são chamados ângulos de junta. Alguns manipuladores contêm juntas deslizantes (ou prismáticas), nas quais o deslocamento relativo entre os elos é uma translação, às vezes chamada de deslocamento da junta (em inglês, *offset*). O número de graus de liberdade que um manipulador possui é o número de variáveis de posição independentes que teriam de ser especificadas para se localizarem todas as peças do mecanismo (CRAIG, 2005).

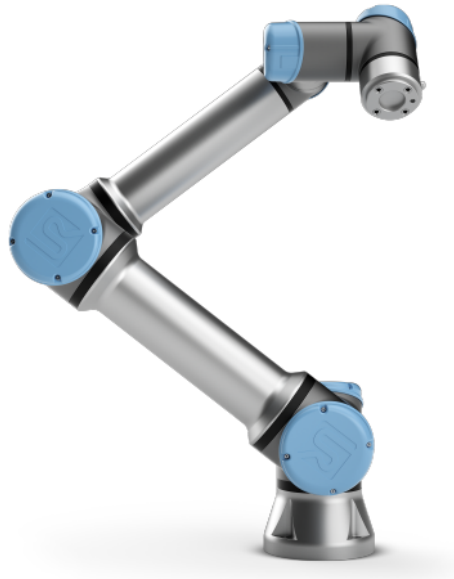
Este trabalho utiliza o robô UR5, ele é um robô colaborativo fabricado pela Universal Robots, o modelo UR5 é um dos vários robôs da família UR sendo intermediário em capacidade de trabalho. A Universal Robots foi fundada por Esben Østergaard, Kasper Støy e Kristian Kassow. Ambos se conheceram na Universidade da Dinamarca, desta forma a empresa foi criada com o objetivo de desenvolver robôs para pequenos e médios negócios, o UR5 foi desenvolvido em 2008, sendo um robô relativamente leve e de fácil programação, sua interface é sofisticada e amigável para vários públicos sendo válida para várias formas de programação e reprogramação e a adaptação das tarefas, em comparação com robôs manipuladores tradicionais (ROBOTS, 2024).

2.1.1 Descrição do manipulador robótico UR5

O UR5, ilustrado na Figura 2.1, é um robô colaborativo ou Cobot e representa o modelo adotado neste trabalho. Os Cobots são projetados para atuar em conjunto com seres humanos em seu espaço de trabalho e, por tal razão, são projetados com uma camada de segurança extra para execução de suas atividades. Assim, podem ser utilizados em vários campos como: indústria, educação, agricultura, serviços de saúde, segurança e exploração espacial. No início essas máquinas eram conhecidas como dispositivos de assistência inteligente, do inglês *Intelligent Assistance Devices* (IADs), mais tarde o termo robô colaborativo se tornou padrão.

Projetados para interagir de forma amigável e eficiente com o operador humano, os manipuladores robóticos do tipo Cobot introduzem a ideia de combinar as habilidades cognitivas dos seres humanos com a precisão dos movimentos dos robôs para realizar tarefas

Figura 2.1 – UR5 real.



Fonte: [Universal Robots \(2024\)](#).

complexas. Ao contrário dos robôs industriais tradicionais, os Cobots são simples, leves, confiáveis, com sensores aprimorados e fáceis de implantar. Graças a essas características, eles podem detectar o movimento de um humano e são equipados com algoritmos de prevenção de colisões em seu espaço de trabalho ([HAMEED et al., 2023](#)). Esses braços, usados individualmente ou em pares, são geralmente equipados com garras de mandíbula, alicates ou, ocasionalmente, garras antropomórficas articuladas com 3, 4 ou 5 dedos. Esse cenário ocorre devido à baixa performance e ao custo ainda elevado das garras antropomórficas ([STARETU, 2021](#)).

A [Universal Robots \(2025\)](#) descreve que o UR5 pode ser utilizado em vários cenários e aplicações, a citar:

- Soldagem a arco, pois requer destreza aguda e profundo entendimento de materiais e aplicação de calor;
- Conjunto pois promove a redução do tempo de montagem, aumenta a velocidade da produção e melhora a qualidade do produto;
- Dispensação pois adiciona maior flexibilidade, eficiência e liberdade às tarefas de colagem, selagem, pintura e outras tarefas de distribuição;
- Manutenção de máquinas possibilitando agilidade para carregamento e descarregamento de materiais sem interrupções e monitoramento consistente do processo de produção;

- Manuseio de materiais pois os Cobots foram implantados na movimentação de materiais para ajudar as empresas a otimizar os fluxos de trabalho e aumentar a produtividade, sem comprometer a qualidade;
- Remoção de material podem executar movimentos altamente precisos e pressão consistente, resultando em remoção de material eficaz e de alta qualidade em formas complexas;
- Paletização executando tarefas repetitivas com consistência e precisão, independentemente das variações nos produtos;
- Inspeção de qualidade por meio de inspeções mais seguras, rápidas e precisas padronizando procedimentos e simplificando fluxos de trabalho com braços robóticos acessíveis e ágeis; e
- Lixamento e polimento braços robóticos colaborativos com controle de força oferecem uma solução flexível que ajusta automaticamente a posição para atingir a força desejada.

O UR5 faz parte de uma classe de outros manipuladores de propósito semelhante e a Tabela 1 apresenta alguns dos principais parâmetros construtivos e operacionais do UR5.

Tabela 1 – Características principais do robô UR5

Característica	Descrição
Capacidade de carga	5 kg (11 lbs)
Alcance	850 mm (33,5 polegadas)
Graus de Liberdade (DoF)	6 juntas (6 DoF), proporcionando alta flexibilidade de movimento
Peso	18,4 kg (40,6 lbs)
Repetibilidade	$\pm 0,03$ mm, garantindo alta precisão para tarefas que exigem exatidão
Design Colaborativo	Projetado para operar de forma segura ao lado de humanos, com sensores de força e detecção de colisão integrados
Programação e Controle	- Programação via interfaces gráficas, linguagens de <i>script</i> e ROS (Robot Operating System). - Suporte a Polyscope (GUI da Universal Robots) e URScript (linguagem de <i>script</i> semelhante ao Python). - Controle via computador externo usando comunicação TCP/IP.
Consumo de Energia	Aproximadamente 250W durante operação, sendo energeticamente eficiente

Fonte: O Autor, adaptado de [Universal Robots \(2023\)](#).

A Tabela 2 apresenta uma comparação das características do UR5 e dos demais modelos da série.

Tabela 2 – Comparação entre os robôs da série UR

Modelo	Carga Útil (kg)	Alcance (mm)	Peso (kg)	Repetibilidade (mm)
UR3	3	500	11,2	$\pm 0,03$
UR5	5	850	18,4	$\pm 0,03$
UR10	12,5	1300	33,5	$\pm 0,05$
UR16	16	900	33,1	$\pm 0,05$
UR20	20	1750	64	$\pm 0,05$

Fonte: O Autor, adaptado de [Universal Robots \(2025\)](#).

Um manipulador consiste em uma série de corpos rígidos (elos) conectados por meio de pares cinemáticos ou juntas. As juntas podem ser essencialmente de dois tipos: revolutas e prismáticas ([BRUNO et al., 1994](#), p. 58) e o UR5 possui seis juntas rotacionais ([Universal Robots, 2021](#)).

É possível notar na Figura 2.1 que o UR5 é um robô que possui 7 elos e 6 juntas rotativas cada junta sendo um grau de liberdade, ou, do inglês, *Degrees of Freedom* (DOF), totalizando seus 6-DOF ([GRAVDAHL, 2014](#)).

2.1.1.1 Divisão dos Elos e Distribuição de Massa do UR5

O fabricante informa que o manipulador UR5 utiliza dois tipos de motores, com tamanhos e massas diferentes ([GRAVDAHL, 2014](#)):

- Motores grandes nas três primeiras juntas.
- Motores pequenos nas três últimas juntas.

De acordo com essas informações, foi observado que:

- **Elo 1:** Contém apenas um motor grande, com massa de 3,7 kg.
- **Elo 2:** Contém dois motores grandes. Como cada motor pesa 3,7 kg, a parte central do elo 2 tem uma massa de: $8,39 - (2 \times 3,7) = 0,99$ kg.
- **Elo 3:** Contém um motor pequeno de 1,219 kg. A massa restante do elo 3 é: $2,275 - 1,219 = 1,056$ kg.
- **Elos 4 e 5:** Contêm um motor pequeno cada, com massa de 1,219 kg.

2.1.1.2 Parâmetros Dimensionais do Manipulador UR5

Os parâmetros dimensionais do manipulador UR5 podem ser visualizados na Tabela 3, condizente com a representação do conjunto de elos e juntas do UR5, apresentada na Figura 2.2.

Esses valores são fundamentais para a modelagem matemática e simulação do UR5.

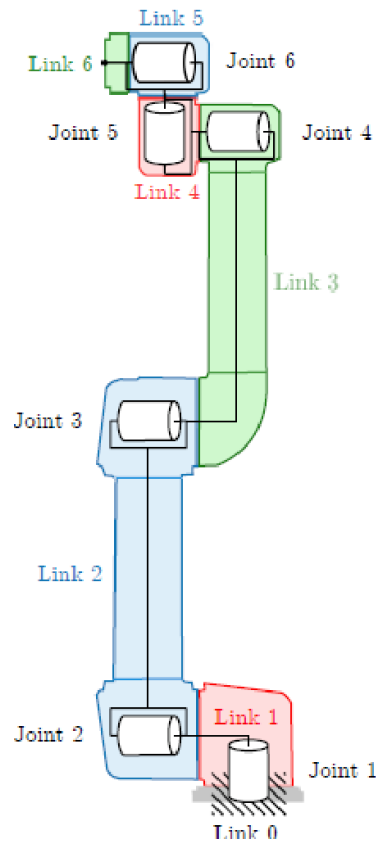
Tabela 3 – Parâmetros do manipulador UR5.

Elo (i)	Comprimento do elo (mm)	Deslocamento ao longo do eixo Z_{i-1} (mm)	θ_i ($^\circ$)
1	0	89.2	θ_1
2	425	0	θ_2
3	392	0	θ_3
4	0	109.3	θ_4
5	0	94.75	θ_5
6	0	82.5	θ_6

Fonte: Adaptado de [Gravdahl \(2014\)](#).

Obs.: θ_i representa o ângulo de rotação em torno de Z_{i-1} .

Figura 2.2 – Esquema de juntas e elos por cores do UR5.



Fonte: [Gravdahl \(2014\)](#).

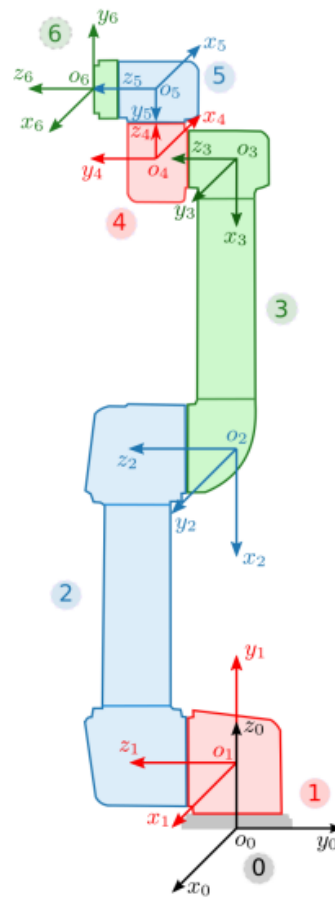
2.2 Sistemas de Coordenadas e Matriz de Transformação

Um sistema de coordenadas pode ser definido como um esquema de representação de um conjunto de vetores e pontos, segundo [Spong, Hutchinson e Vidyasagar \(2004, pg. 34\)](#), esse raciocínio geométrico pode ser definido por dois modos, sendo eles o sintético e o analítico. O modo sintético é definido um raciocínio diretamente com entidades geométricas (pontos e linhas), sem a necessidade de coordenadas numéricas. Já o modo analítico é necessário o uso de coordenadas ou equações, permitindo o raciocínio via manipulações

algébricas.

Em robótica é fundamental que se saiba onde cada componente se encontram no espaço e, para isso, adotamos sistemas de coordenadas de referência (*frame of reference*, do inglês). Na Figura 2.3 é ilustrado o processo, em que cada junta do robô possui seu sistema de referência. Esses sistemas de referências podem ser relativos a qualquer sistema no ambiente, por convenção, o eixo z é o eixo de rotação.

Figura 2.3 – Sistemas de referência.

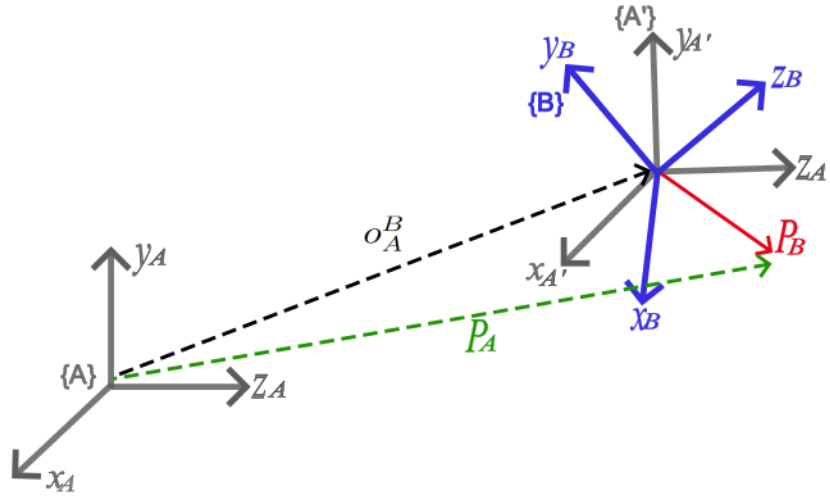


Fonte: Adaptado de [Gravdahl \(2014\)](#).

Com isso surge a necessidade de estabelecer um sistema de coordenada de referência para nortear os demais pois através deste principal que vamos obter a posição e orientação dos demais [Craig \(2005, p. 4\)](#). Em nosso caso será sempre em relação ao sistema base do robô.

Para fazer a transformação de representação de pontos ou vetores entre dois sistemas de coordenadas é necessário um vetor posição relativo entre os sistemas e uma matriz de rotação, representando a atitude relativa entre ambos ([SPONG; HUTCHINSON; VIDYASAGAR, 2004](#))[p. 56], como ilustrado na Figura 2.4. Assim, é necessário conhecer a posição relativa entre os sistemas e a rotação que alinha um sistema em relação ao outro.

Figura 2.4 – Sistemas de representações.



Fonte: O Autor (2025).

Deste modo, se considerarmos dois sistemas de coordenadas de referência $\{A\}$ e $\{B\}$ então, a posição de um ponto $\vec{p}$, representado algebricamente em $\{A\}$ e denominado por $\mathbf{p}_A$, pode ser convertida para o sistema de coordenadas $\{B\}$ (que passa a ser denominado por $\mathbf{p}_B$), por meio da aplicação da equação de transformação:

$$\mathbf{p}_A = \mathbf{R}_A^B \mathbf{p}_B + \mathbf{o}_A^{B/A}, \quad (2.1)$$

Assim:

- $\mathbf{p}_A$ é um vetor de posição de um ponto no sistema de coordenadas A
- $\mathbf{p}_B$ é um vetor de posição de um ponto no sistema de coordenadas B
- $\mathbf{R}_A^B$ é a matriz de rotação que orienta o sistema B em relação ao sistema A ;
- $\mathbf{o}_A^B$ vetor de translação da origem de A para a origem de B .

Expandindo a Equação (2.1), na forma matricial, temos a Equação (2.2), essa transformação é fundamental na robótica, onde se busca representar posições e orientações de diferentes partes de um manipulador (como a garra ou juntas) em relação a um sistema de referência geralmente o sistema da base do robô.

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix} + \begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix} \quad (2.2)$$

Translação de Eixos

A translação de sistemas de coordenadas, é um conceito essencial na robótica uma vez que o robô pode ser modelado como um todo sendo vários sistemas de referência acoplados em suas juntas.

Na Figura 2.4 temos dois sistemas de referência o A pertencente ao sistema $\{A\}$ e B pertencente ao sistema $\{B\}$, desta forma a translação do sistema $\{A\}$ para o $\{B\}$ é possível pelo vetor $\mathbf{o}_A^B$ que liga a origem do sistema A ao sistema B como na Figura 2.4.

Rotação de Eixos

Na Figura 2.4 temos um exemplo que retrata dois sistemas com a mesma posição porém orientados de forma diferente. Na Figura 2.4 é ilustrado um referencial $\{B\}$ expresso no referencial $\{A'\}$ com isso, podemos definir uma matriz de rotação.

A matriz de rotação pode ser interpretada da matriz de cossenos diretores, uma vez que seus elementos descrevem a orientação de um sistema de coordenadas de referência em relação a outro sistema de coordenadas de referência.

Podemos destacar que, embora a matriz de rotação é uma transformação ortogonal com determinante 1, a matriz de cossenos diretores é uma forma mais genérica de descrever a orientação relativa entre dois sistemas de coordenadas sem estar restrita a uma rotação.

Considerando o sistema $\{A'\}$ como referência do sistema $\{B\}$ rotacionado, a matriz de cossenos diretores contém os cossenos dos ângulos entre os eixos do sistema $\{A'\}$ e sistema $\{B\}$ cada elemento da matriz é descrito como (MARKLEY; CRASSIDIS, 2014; CRAIG, 2005):

$$r_{ij} = \cos(\theta_{ij}) \quad (2.3)$$

em que θ_{ij} corresponde ao ângulo entre o eixo i do sistema $\{B\}$ e o eixo j do sistema $\{A'\}$. Neste caso, R é a matriz de cossenos diretores dada por:

$$\mathbf{R} = \begin{bmatrix} \cos(\theta_{11}) & \cos(\theta_{12}) & \cos(\theta_{13}) \\ \cos(\theta_{21}) & \cos(\theta_{22}) & \cos(\theta_{23}) \\ \cos(\theta_{31}) & \cos(\theta_{32}) & \cos(\theta_{33}) \end{bmatrix} \quad (2.4)$$

Considerando a Equação (2.5) que expressa a relação entre as coordenadas de um vetor v nos sistemas de referência a e b ,

$$\mathbf{v}_a = \mathbf{R} \mathbf{v}_b \quad (2.5)$$

ou seja, a relação entre os componentes dos vetores

$$\mathbf{v}_a = \begin{bmatrix} v_{a1} \\ v_{a2} \\ v_{a3} \end{bmatrix} \quad (2.6)$$

$$\mathbf{v}_b = \begin{bmatrix} v_{b1} \\ v_{b2} \\ v_{b3} \end{bmatrix} \quad (2.7)$$

pode ser reescrita como:

$$\begin{bmatrix} v_{a1} \\ v_{a2} \\ v_{a3} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{11}) & \cos(\theta_{12}) & \cos(\theta_{13}) \\ \cos(\theta_{21}) & \cos(\theta_{22}) & \cos(\theta_{23}) \\ \cos(\theta_{31}) & \cos(\theta_{32}) & \cos(\theta_{33}) \end{bmatrix} \begin{bmatrix} v_{b1} \\ v_{b2} \\ v_{b3} \end{bmatrix} \quad (2.8)$$

Propriedades

Como apresentado por [Markley e Crassidis \(2014, p. 21\)](#), [Spong e Vidyasagar \(2008, p. 34\)](#) e [Bruno et al. \(1994, p. 567\)](#), a matriz de rotação $R_{A'}^B$ tem as seguintes propriedades fundamentais:

1. Determinante unitário:

$$\det(\mathbf{R}_{A'}^B) = 1 \quad (2.9)$$

2. A matriz de rotação é ortogonal, ou seja, sua transposta é igual à sua inversa:

$$(\mathbf{R}_{A'}^B)^T = (\mathbf{R}_{A'}^B)^{-1} \quad (2.10)$$

Essa propriedade pode ser demonstrada pela multiplicação da matriz por sua transposta:

$$(\mathbf{R}_{A'}^B)^T (\mathbf{R}_{A'}^B) = (\mathbf{R}_{A'}^B) (\mathbf{R}_{A'}^B)^T = \mathbf{I} \quad (2.11)$$

A matriz de rotação $\mathbf{R}$, descreve a orientação de um sistema de referência $\{B\}$ em relação a outro sistema de referência $\{A'\}$. Os componentes da matriz representam um vetor unitário correspondente a um eixo do sistema $\{B\}$, expresso no referencial $\{A'\}$.

Assim como determinado por [Markley e Crassidis \(2014, p. 24\)](#), os vetores que compõem essa matriz formam um conjunto ortonormal, ou seja, um conjunto de vetores que possuem norma unitária e são mutuamente ortogonais. Em termos matemáticos, esse conjunto satisfaz a seguinte relação de produto interno:

$$x_i \cdot x_j = \delta_{ij} \quad (2.12)$$

onde o delta de *Kronecker* δ_{ij} é definido por:

$$\delta_{ij} = \begin{cases} 0, & \text{se } i \neq j \\ 1, & \text{se } i = j \end{cases} \quad (2.13)$$

Essa propriedade é essencial, pois garante que a matriz de rotação preserve as distâncias e ângulos isto é, trata-se de uma transformação ortogonal com determinante igual a 1. Quando o número de vetores no conjunto é igual à dimensão do espaço, dizemos que eles formam uma **base ortonormal**.

Dessa forma, a matriz de rotação R , além de representar a transformação entre os sistemas $\{A'\}$ e $\{B\}$, também assegura que os vetores da nova base mantêm a estrutura geométrica do espaço vetorial original.

2.2.1 Matriz de Transformação e Propriedades

Existe uma convenção bastante utilizada em robótica para simplificação da análise de cinemática direta do manipulador denominada convenção de Denavit Hartenberg ela é amplamente utilizada para a seleção de referenciais em aplicações robóticas (SPONG; VIDYASAGAR, 2008, pag. 76).

Segundo (SPONG; VIDYASAGAR, 2008), algumas situações específicas podem gerar ambiguidade ou falta de definição na aplicação da convenção de Denavit–Hartenberg (DH), exigindo critérios adicionais de modelagem. Entre as principais estão:

- **Indefinição do eixo z_n no último elo:** Como não há uma junta subsequente ao último elo, a escolha do eixo z_n pode ser arbitrária. Normalmente, ele é alinhado com z_{n-1} para simplificação.
- **Ambiguidade no sistema de coordenadas inicial:** A origem e orientação do frame base (x_0, y_0, z_0) podem ser definidas livremente, o que pode levar a diferentes representações cinemáticas para o mesmo manipulador.
- **Eixos consecutivos paralelos:** Quando z_{i-1} e z_i são paralelos, a normal comum entre eles que define x_i não é única, gerando múltiplas possibilidades válidas nesse caso.
- **Eixos consecutivos se interceptando:** Se os eixos z_{i-1} e z_i se cruzam, a direção de x_i , que deve ser ortogonal a ambos, também se torna ambígua, como não existe uma única normal comum, a orientação do sistema de coordenadas resultante depende da escolha.
- **Juntas prismáticas:** A ausência de rotação em juntas prismáticas torna a definição dos frames menos direta, exigindo critérios adicionais para manter a coerência da modelagem.

Para a modelagem da cinemática direta, optou-se pelo método da matriz de transformação homogênea do manipulador neste trabalho, pois é um método que combina rotação e translação em operação de multiplicação de matrizes, como o robô é uma composição desses movimentos esse método é efetivo em todos os casos (SPONG; VIDYASAGAR, 2008, pag. 35).

De acordo com (LYNCH; PARK, 2017), para as formas de uso das matrizes de transformação podemos formalizar:

1. Elas podem representar um comportamento de um corpo rígido no espaço que seria posição e orientação.
2. É possível alterar o referencial fazendo uma simples transformação de coordenadas como é explicado na Equação (2.5).
3. Transladar um vetor ou quadro de referencia, como na Figura 2.4.

Devido à inúmeras utilidades matemáticas da matriz de transformação, na robótica é essencial se adotar essa ferramenta pois será muito útil na cinemática direta que é melhor explicado na Seção 2.4.1, possibilitando efetuar uma composição de transformações de coordenadas. Não só isso mas também na modelagem matemática do UR5.

Na Equação (2.8) temos o caso da rotação, mas em robótica sempre consideraremos tanto rotação quanto translação, portanto é válido afirmar a relação dada pela expressão da transformação homogênea pela Equação (2.15).

A matriz de transformação $\mathbf{T}_O^D$ permite converter a representação de um vetor de um sistema de referência origem $\{O\}$ para um sistema de referência de destino $\{D\}$:

$$\mathbf{v}_D = \mathbf{T}_O^D \mathbf{v}_O \quad (2.14)$$

onde $\mathbf{T}$ é a matriz de transformação homogênea na referência O com D sendo o destino:

$$\mathbf{T}_O^D = \begin{bmatrix} \mathbf{R}_O^D & \mathbf{t}_O \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \quad (2.15)$$

Assim equivalentemente temos:

$$\begin{bmatrix} \mathbf{v}_D \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_O^D & \mathbf{t}_O \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_O \\ 1 \end{bmatrix} \quad (2.16)$$

Assim a Equação (2.14) define a relação entre os vetores, e a Equação (2.15) representa a matriz de transformação homogênea.

Desta forma a Equação (2.14) é equivalente à Equação (2.16).

A posição de um ponto $\vec{p}$ representado em um Sistema de Coordenadas Cartesiano $\{A\}$, é representado, algebricamente, como $\mathbf{p}_A$. Neste caso, o ponto que representa a origem do frame $\{A\}$ é denominada por $\{O_A\}$.

Considerando dois referenciais, um *frame* de origem $\{O\}$ e um de destino $\{D\}$, podemos representar o vetor posição $\vec{p}$ em ambos os sistemas. Para tal, precisamos determinar o vetor de translação entre ambas as origens dos sistemas de referências $\{O\}$ e $\{D\}$.

$$\mathbf{o}_0^1 = \begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix} \quad (2.17)$$

A orientação do referencial $\{O_1\}$ em relação a $\{O_0\}$ podemos descrever pela matriz de rotação:

$$\mathbf{R}_0^1 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (2.18)$$

Desta forma surge a representação homogênea para combinar de forma mais compacta tanto a rotação quanto translação expressas em uma matriz denominada matriz transformação homogênea.

$$\mathbf{T}_0^1 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & o_x \\ r_{21} & r_{22} & r_{23} & o_y \\ r_{31} & r_{32} & r_{33} & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.19)$$

Considerando, então, essa forma de representação, apresentamos algumas das propriedades essenciais para sua manipulação (BRUNO et al., 1994).

Composição de Transformações: A multiplicação de duas transformações homogêneas resulta em outra transformação homogênea:

$$\mathbf{T}_0^2 = \mathbf{T}_0^1 \mathbf{T}_1^2. \quad (2.20)$$

Essa relação permite combinar múltiplas transformações (rotação e translação) de forma sequencial.

Inversa de uma transformação homogênea: A inversa de uma transformação homogênea é dada por:

$$\mathbf{R}^{-1} = \mathbf{R}^T, \quad (2.21)$$

pois $\mathbf{R}$ é ortogonal, ou seja, $\mathbf{R}^T \mathbf{R} = \mathbf{I}$.

2.3 Parametrização da Matriz de Rotação

Contudo, ainda que a transformação homogênea seja o modo universal de modelagem do robô, existem diversas outras formas de parametrização do manipulador que é interessante mencionar por exemplo (MARKLEY; CRASSIDIS, 2014):

- Matriz de Rotação - $\mathbf{R} \in SO(3)$
- Ângulos de Euler - (ϕ, θ, ψ)
- Eixo-Ângulo de Euler - $(\mathbf{e}, \theta)$
- Quaternions - (q_0, q_1, q_2, q_3) ou $\mathbf{q} \in \mathbb{H}$

2.3.1 Ângulos de Euler

Leonhard Euler propôs que uma forma de descrever a orientação em 3 dimensões com 3 ângulos sucessivos em relação a um sistema de referência pode ser expresso por (MARKLEY; CRASSIDIS, 2014):

- ϕ : rotação em torno do eixo x , conhecida como rolamento (*roll*);
- θ : rotação em torno do eixo y , conhecida como arfagem (*pitch*);
- ψ : rotação em torno do eixo z , conhecida como guinada (*yaw*).

A matriz de rotação total é dada por:

$$\mathbf{R}_{xyz}(\phi, \theta, \psi) = \mathbf{R}_z(\psi) \mathbf{R}_y(\theta) \mathbf{R}_x(\phi). \quad (2.22)$$

Considerando a seguinte notação, onde c e s representam, respectivamente, o cosseno e o seno de um ângulo. Representando as matrizes de rotação individuais temos:

$$\mathbf{R}_z(\psi) = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.23)$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \quad (2.24)$$

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \quad (2.25)$$

Com o produto na ordem: x,y e z temos,

$$\mathbf{R}(\psi, \theta, \phi) = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi + s_\psi c_\phi & c_\psi s_\theta c_\phi - s_\psi s_\phi \\ -s_\psi c_\theta & -s_\psi s_\theta s_\phi + c_\psi c_\phi & -s_\psi s_\theta c_\phi - c_\psi s_\phi \\ s_\theta & -c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \quad (2.26)$$

$$c = \cos, \quad s = \sin$$

Por meio de uma análise simples, é possível obter os ângulos envolvidos,

$$\theta = \arcsin(R_{31}) \quad (2.27)$$

Se $\theta \neq \frac{\pi}{2}$ ou $\theta \neq -\frac{\pi}{2}$, os outros ângulos são dados por:

$$\psi = \arctan 2(-R_{21}, R_{11}) \quad (2.28)$$

$$\phi = \arctan 2(-R_{32}, R_{33}) \quad (2.29)$$

Se $\theta = \frac{\pi}{2}$ ou $\theta = -\frac{\pi}{2}$, ocorre uma singularidade (*gimbal lock*), ou bloqueio de Cardan, ou seja, uma perda de um grau de liberdade. Assim, as equações para ψ e ϕ devem ser ajustadas.

Com base nessa formulação de Euler, pode-se desenvolver um *script* que pode ser criado para gerar a matriz ao passar os ângulos do robô e se passado os ângulos do robô a matriz poderá ser gerada.

2.3.2 Orientação de Euler na aplicação robótica

Nesta seção será detalhado o teorema de rotação de Euler, pois este será muito utilizado na obtenção das matrizes de transformação, e também da equação de Rodrigues pois com ela é possível determinar a matriz de orientação.

Teorema de rotação de Euler

Pelo teorema de Euler através de uma rotação um corpo pode ser levado a uma orientação única. O eixo e o ângulo dessa rotação podem ser usados para parametrizar

as possíveis orientações do corpo com base nos parâmetros de quatérnio ou Rodrigues. Nosso trabalho utilizaremos a formulação de Rodrigues ([BRIOT; KHALIL, 2015](#), pag.23) o módulo desse eixo deve ser 1.

$$\mathbf{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \quad (2.30)$$

Na Equação (2.30) temos o eixo $\mathbf{e}$ em que, e_x, e_y, e_z são as componentes do eixo de rotação, e $\|\mathbf{e}\| = 1$ garante que seja um vetor unitário. O ângulo de rotação ϑ pertence ao conjunto dos números reais positivos:

$$\vartheta \in \mathbb{R}^+ \quad (2.31)$$

Sendo dita a ideia central do Teorema de Euler no início da Seção 2.3.2, apresentada por [Markley e Crassidis \(2014, pag.40\)](#), podemos definir os seguintes componentes:

- $\mathbf{R}$ como sendo as matrizes de atitude, sendo uma rotação num eixo fixo, que são matrizes de rotação 3x3 que pertence ao $\text{SO}(3)$. E $\mathbf{e}$, para fazer uma transformação do sistema de referência $\{F\}$ para o sistema de referência $\{G\}$. $\mathbf{R}_F^G$ como sendo a matriz de atitude que transforma um vetor x_F de um referencial $\{F\}$ para um referencial $\{G\}$

$$\mathbf{x}_G = \mathbf{R}_F^G \mathbf{x}_F \quad (2.32)$$

desta forma:

- $\mathbf{x}_F$ que é o vetor expresso no referencial fixo $\{F\}$;
- $\mathbf{x}_G$ que é o mesmo vetor expresso no referencial rotacionado $\{G\}$;

O Teorema de Euler afirma a existência de um vetor e ao longo da direção do eixo de rotação que tem a mesma representação no referencial $\{G\}$ e no referencial $\{F\}$, assim podemos substituir $\mathbf{x}_F = \mathbf{x}_G = e$ na Equação (2.32) e expressar o Teorema de Euler algebricamente como:

$$\mathbf{R} \mathbf{e} = \mathbf{e}. \quad (2.33)$$

- Um autovetor de uma matriz quadrada geral M é um vetor não nulo x para o qual a multiplicação por M tem o mesmo efeito que a multiplicação por um escalar, ou seja,

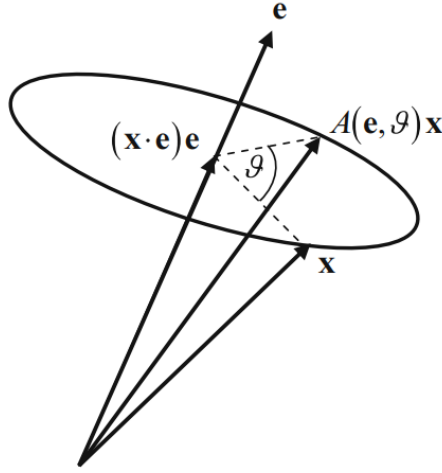
$$\mathbf{M} \mathbf{x} = \lambda \mathbf{x}, \quad (2.34)$$

em que λ é o escalar é o autovalor correspondente ao autovetor $\mathbf{x}$; e $\mathbf{x}$ representa o autovetor. Os autovetores são, quase sempre, expressos como vetores unitários nesse caso.

Para que a Equação (2.34) tenha uma solução não nula para $\mathbf{x}$, a matriz $(\lambda \mathbf{I} - \mathbf{M})$ deve ser singular. Portanto,

$$\det(\lambda \mathbf{I} - \mathbf{M}) = 0. \quad (2.35)$$

Figura 2.5 – Eixo ângulo de Euler.



Fonte: Markley e Crassidis (2014, p. 41)

Esta equação polinomial de grau n é conhecida como a equação característica de M . Os autovalores são as n raízes da equação característica contando as raízes múltiplas de acordo com sua multiplicidade.

O Teorema de Euler diz que um dos autovalores de uma matriz de atitude tem o valor $\lambda = 1$. A equação característica de uma matriz 3×3 $\mathbf{M}$ pode ser facilmente encontrada por meio do cálculo específico de determinante.

Determinação da equação de Rodrigues

Com isso, a matriz de transformação homogênea agora será em função de ϑ , que é o ângulo de rotação, do eixo $\mathbf{e}$, e da posição $\mathbf{p}$.

Desta forma, um script pode ser desenvolvido com o intuito de receber como parâmetros os componentes (ϑ, e, p) , onde ϑ é o ângulo de rotação, $\mathbf{e}$ é o eixo de rotação e $\mathbf{p}$ representa a posição. Dessa forma, o retorno seria a matriz cuja parte de rotação dessa matriz seria em função de $(\vartheta, \mathbf{e})$, que são determinados pela equação de Rodrigues. Portanto, a matriz final seria desta forma:

$$\mathbf{T}(\mathbf{p}, \vartheta, \mathbf{e}) = \begin{bmatrix} \mathbf{R}(\vartheta, \mathbf{e}) & \mathbf{p} \\ 0^T & 1 \end{bmatrix}. \quad (2.36)$$

Com a formulação de Rodrigues, obtemos a matriz de rotação R em função do eixo e e do ângulo ϑ (LYNCH; PARK, 2017, pag.84):

$$\mathbf{R}(\vartheta, \mathbf{e}) = \mathbf{I} + \sin(\vartheta)\mathbf{e} + (1 - \cos(\vartheta))\mathbf{e}^2. \quad (2.37)$$

Podemos concluir então que a fórmula de Rodrigues é uma forma de calcular a matriz de rotação a partir de um eixo de rotação e um ângulo percorrido, como ilustrado na Figura 2.5.

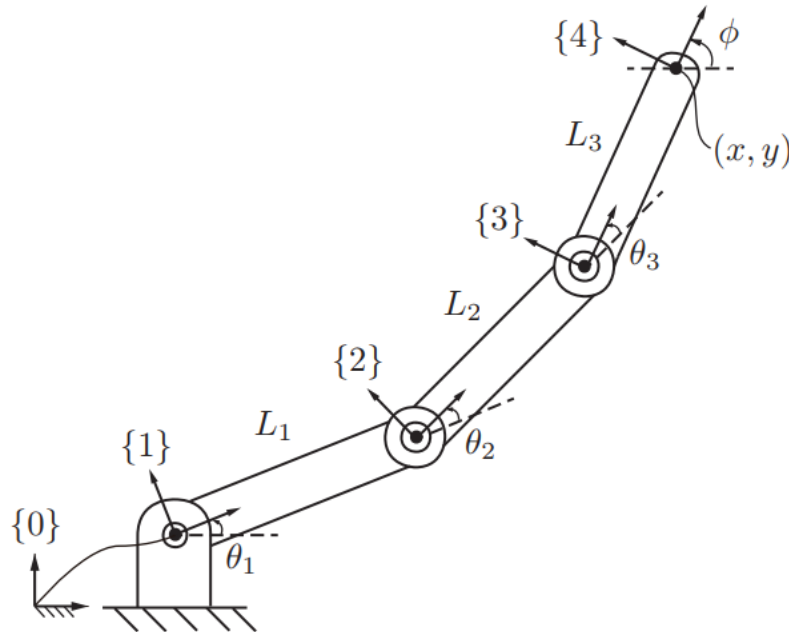
2.4 Cinemática Direta e Inversa

Nesta seção serão abordadas a cinemática direta e a cinemática inversa. Basicamente, a cinemática direta consiste em determinar a posição e a orientação do efetuador final, dadas as configurações articulares do robô. Já a cinemática inversa seria situação dos cálculos das configurações articulares necessárias para que o efetuador atinja uma posição e orientação desejadas.

2.4.1 Cinemática Direta

Podemos então compreender que a cinemática direta no manipulador se define como a composição de transformações homogêneas envolvendo tanto translação como rotação em torno de um eixo específico. Podemos analisar, por exemplo, do manipulador ilustrado na Figura 2.6.

Figura 2.6 – Modelagem cinemática direta.



Fonte: Adaptado de (LYNCH; PARK, 2017) .

De forma geral a cinemática no modelo na Figura 2.6 pode ser dada por

$$T_0^4 = T_0^1 T_1^2 T_2^3 T_3^4 \quad (2.38)$$

Onde, adotando o eixo z como o de rotação e utilizando a seguinte notação abreviada:
 $c\theta = \cos \theta$ e $s\theta = \sin \theta$,

$$\mathbf{T}_0^1 = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.39)$$

$$\mathbf{T}_1^2 = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & L_1 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.40)$$

$$\mathbf{T}_2^3 = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & L_2 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.41)$$

$$\mathbf{T}_3^4 = \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.42)$$

Notemos que $\mathbf{T}_3^4$ é constante e que cada $\mathbf{T}_i^{i-1}$ depende unicamente da variável de junta θ_i .

Nessa abordagem, definimos $\mathbf{M}$ como sendo a posição e orientação do referencial $\{4\}$ quando todos os ângulos das juntas são definidos como zero (a posição “zero” do robô) (LYNCH; PARK, 2017). Pode ser definida como a matriz abaixo:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & L_1 + L_2 + L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.43)$$

2.4.2 Cinemática Inversa

Considerando o problema da cinemática inversa, interpretemos da seguinte forma: dadas a posição e a orientação do efetuador do manipulador, calcule todos os possíveis conjuntos de ângulos de junta que poderiam ser usados para se obter a posição e orientação desejada (CRAIG, 2005).

A cinemática inversa e a otimização são conceitos fundamentais na robótica, sendo necessários para o desenvolvimento e aprimoramento de manipuladores robóticos. A cinemática inversa consiste em determinar as configurações articulares para posicionar o manipulador de acordo com uma posição desejada. Já a otimização busca encontrar a trajetória mais eficiente para o manipulador, considerando objetivos, restrições e critérios de desempenho.

A cinemática inversa enfrenta desafios complexos, como não linearidade, restrições físicas e cinemáticas, singularidades e trajetórias redundantes. Diversas abordagens, como métodos analíticos, numéricos e geométricos, são utilizadas para solucionar esses problemas, cada uma com suas vantagens e limitações.

A integração da cinemática inversa e da otimização em manipuladores robóticos industriais resulta em melhorias significativas, como maior precisão, eficiência, segurança e desempenho. Essas técnicas são amplamente aplicadas em setores industriais, buscando otimizar processos, reduzir custos, aumentar a produtividade e melhorar a qualidade dos produtos.

A cinemática inversa é uma área essencial no estudo e desenvolvimento de manipuladores robóticos, [Craig \(2005, p. 95\)](#) resalta uma vez que permite determinar as configurações articulares necessárias para alcançar uma posição ou orientação desejada no espaço de trabalho ou espaço de trabalho destre, que é um subconjunto do espaço de trabalho.

A cinemática inversa é uma tarefa complexa devido à natureza não linear das equações envolvidas e à presença de restrições de movimento. A cinemática inversa se aplica em diversas áreas, como robótica industrial, cirurgia robótica, animação de personagens virtuais e controle de próteses ([CRAIG, 2005](#)).

2.4.3 Cinemática de manipuladores

Um manipulador pode ser esquematicamente representado do ponto de vista mecânico como uma cadeia cinemática de corpos rígidos (*links*), interligados por meio de revolução ou juntas prismáticas. Uma extremidade da cadeia é restrita a uma base, enquanto uma end-effector é montado na outra extremidade. O movimento resultante da estrutura é obtido pela composição dos movimentos elementares de cada elo com respeito ao anterior. Desta forma para manipular um objeto no espaço, é necessário descrever a posição e a orientação do efetuador final ([BRUNO et al., 1994](#)).

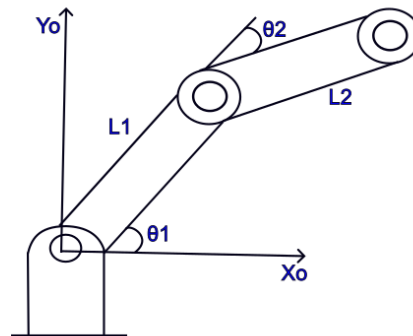
Focaremos nosso trabalho em robôs com juntas rotacionais. Na Figura 2.7 é exemplificado um manipulador com juntas rotacionais no plano.

Considerando um robô manipulador com dois graus de liberdade ou seja 2 DOF, Graus de Liberdade ou *Degrees of Freedom*, e ainda no plano cartesiano, com as medidas

dos elos $L1$ e $L2$ conhecidas podemos representar a posição do efetuador, de duas formas:

- Podemos prever de forma simples e intuitiva a posição final da ponta do robô, ou seja o efetuador final apenas sabendo o valor dos ângulos: θ_1 e θ_2 . Portanto a representação no espaço da junta rotacional ficaria: (θ_1, θ_2) .
- Por outro lado, pelo fato de estar no plano cartesiano podemos também apenas considerar os pontos do efetuador do manipulador, obtendo (x, y) . Portanto temos:
 - Representação no espaço da junta: (θ_1, θ_2) .
 - Representação no espaço cartesiano: (x, y)

Figura 2.7 – Robô com dois graus de liberdade junta rotacional no plano



Fonte: O autor (2025).

2.5 Planejamento de Movimento e Otimização

O planejamento de movimento é um dos problemas mais estudados em robótica, consiste em executar o movimento do robô do ponto inicial até um ponto que se objetiva chegar, porém nesse percurso são considerados as limitações de juntas, valores de torque e obstáculos diversos que poderão estar presentes (REITER, 2020).

2.5.1 MATLAB para matemática e otimização

Na documentação oficial do MATLAB existe a seção *Mathematics and Optimization* reúne ferramentas especializadas para o desenvolvimento, seja matemático ou de otimização no geral, possibilitando análise e visualização de modelos matemáticos. Esse nicho da documentação centraliza o uso de diversos *toolboxes*, oferecendo desde recursos básicos de cálculo numérico e simbólico até métodos avançados de otimização e modelagem física. Os principais *toolboxes* destacados são:

- *Optimization Toolbox*.

- *Global Optimization Toolbox*.
- *Symbolic Math Toolbox*.
- *Mapping Toolbox*.
- *Partial Differential Equation Toolbox*.

O MATLAB demonstra-se uma ferramenta apta e adequada para aplicações de otimização (The MathWorks, Inc., 2025b). Neste trabalho será empregado o *Optimization Toolbox*, o qual disponibiliza um conjunto abrangente de algoritmos numéricos que se mostram especialmente pertinentes para a resolução do problema proposto.

2.5.2 MATLAB Optimization Toolbox

O *Global Optimization Toolbox* do MATLAB oferece uma ampla gama de métodos para solução de situações problemas de otimização para diversas situações, com múltiplos mínimos ou máximos locais, como exemplo existe: algoritmos como busca por padrões, algoritmo genético, enxame de partículas, recozimento simulado, *multistart* e busca global. Estes métodos são solucionadores ideais para funções com alto grau de complexidade, como aquelas descontínuas, estocásticas, sem derivadas ou baseadas em simulações e caixas-pretas (The MathWorks, Inc., 2025a).

O *toolbox* do MATLAB permite também resolver problemas de otimização com múltiplos objetivos, possibilitando a construção da frente de Pareto com métodos como algoritmos genéticos e busca por reconhecimento de padrões. O desempenho dos algoritmos pode ser aprimorado por meio do ajuste de opções ou da personalização de etapas como criação, atualização e busca. E ainda é possível utilizar tipos de dados personalizados para representar situações problemas não tradicionais. Uma funcionalidade adicional é o uso de uma função híbrida, que combina dois algoritmos de otimização sequencialmente para melhorar a qualidade da resolução final (The MathWorks, Inc., 2025a).

Com isso, esta ferramenta do MATLAB se mostrou bastante útil na solução do problema de cinemática inversa por otimização, no caso da simulação do UR5.

2.5.3 Algoritmo de otimização lsqnonlin MATLAB funcionamento

Para a resolução do problema da cinemática inversa, foi adotado o método *lsqnonlin*, pertencente ao conjunto *Optimization Toolbox* do MATLAB. Este solucionador é focado em minimizar problemas de mínimos quadrados não lineares. Consequentemente, ele busca encontrar o vetor de variáveis que minimiza a soma dos quadrados de funções vetoriais (MathWorks, 2025).

Sua formulação é adequada para nosso cenário uma vez que nós desejamos minimizar o erro entre posição e orientação final estimada e alcançada do efetuador, e esse método tem essa finalidade de se reduzir o erro entre um valor observado (ou desejado) e um valor estimado por um modelo não linear ([MathWorks, 2025](#)).

Interpretações

Para que a interpretação seja feita de forma correta é necessário entender alguns aspectos como ([MathWorks, 2025](#)):

- Mínimo local e mínimo global: um mínimo local tem valor menor que os pontos ao redor. Um mínimo global é menor que todos os valores viáveis. O método *lsqnonlin* busca mínimos locais, mas pode achar o global por sorte.

Medida de otimalidade de primeira Ordem:

- Para problemas sem restrições, é o valor máximo absoluto do gradiente.

Com restrições: usa a fórmula

$$|v_i \cdot g_i| \quad (2.44)$$

onde:

- v_i é um fator que depende da posição de x_i em relação ao seu limite;
- g_i é o i -ésimo componente do gradiente.

- Idealmente, essa medida deve ser zero num ponto ótimo.

Tolerâncias:

- *OptimalityTolerance*: para otimalidade de primeira ordem.

- *FunctionTolerance*: para variação mínima no valor da função.

- Tamanho do gradiente: idêntico à medida de otimalidade, usado como critério de parada.

- Tamanho do passo atual: É o tamanho da alteração no vetor de decisão na última iteração. Pode indicar estagnação.

- Localmente singular: O algoritmo Levenberg-Marquardt ajusta o “tamanho da confiança” com um parâmetro que cresce se o modelo quadrático não representar bem a função.

- Cálculo Jacobiano indefinido: Acontece se, ao tentar calcular derivadas numericamente, a função retorna Inf, NaN ou valores complexos. Isso prejudica a otimização.

- **Inf**: Abreviação de *Infinity* (infinito). Significa que a função retornou um valor infinito, geralmente devido a uma divisão por zero ou à geração de um número muito grande.
- **NaN**: Abreviação de *Not a Number*. Indica que o resultado de uma operação matemática não é um número válido, como $\frac{0}{0}$ ou $\sqrt{-1}$ (em cálculo real).
- **Valores complexos**: São números da forma $a + bi$, onde i é a unidade imaginária, tal que $i^2 = -1$. Em problemas de otimização com funções reais, o aparecimento de valores complexos geralmente indica que a função foi avaliada fora de seu domínio.

2.5.4 Modo de operação do lsqnonlin MATLAB

Esse método pode ser aplicado da seguinte forma ([MathWorks, 2025](#)):

Código 2.1 – Exemplo de uso do lsqnonlin

```
1 x = lsqnonlin(fun, x0)
```

Em que, nesse caso:

fun: É uma função que retorna um vetor de erros (resíduos)

x0: É o vetor de valores iniciais para as variáveis a ser ajustadas.

Esse método ajusta os valores de **x** para minimização dos resíduos, podendo também ser considerados limites superiores e inferiores, restrições lineares e não lineares.

É interessante notar algumas mensagens que indicam o *status* da otimização e dão dicas sobre como prosseguir:

- **Mínimo local encontrado**: A função atingiu um ponto onde a medida de otimalidade de primeira ordem é próxima de zero, indicando um mínimo local provável.
- **Mínimo possível local**: A função pode ter parado em um mínimo local, mas sem certeza a medida de otimalidade de primeira ordem não está abaixo da tolerância (*OptimalityTolerance*).
- **Ponto inicial é um mínimo local**: O ponto inicial já satisfaz a condição de mínimo (otimalidade de primeira ordem aproximadamente 0).

3 Ambiente de Simulação e Modelagem

Este capítulo discorre sobre o processo de simulação, trazendo os detalhes de funcionamento do CoppeliaSim, sua interface de comunicação com programas externos — como o Matlab, bem como a modelagem dos manipuladores e o acesso aos parâmetros virtuais. Assim, o restante deste capítulo está organizado da seguinte forma: a Seção 3.1 descreve o funcionamento do ambiente do CoppeliaSim; a Seção 3.2 descreve a arquitetura de comunicação do CoppeliaSim com MATLAB; e a Seção 3.3 descreve as etapas de desenvolvimento das modelagens estática e dinâmica.

3.1 Ambiente de simulação: CoppeliaSim

O CoppeliaSim é o ambiente de simulação no qual será simulado o funcionamento do UR5. Como já apresentado, este ambiente foi desenvolvido para aplicações de robótica. Assim, ele permite virtualizar o comportamento do manipulador de forma semelhante ao comportamento do sistema real.

A interface do CoppeliaSim é organizada em seções que permitem a criação e edição de cenas. A Figura A.1 apresenta uma cena com a simulação do UR5 no ambiente virtual do CoppeliaSim. Neste ambiente é possível visualizar o UR5, editar os comandos de navegação, tempo de simulação dentre outras configurações básicas porém essenciais. Toda a interface é melhor explicada no Apêndice A.

3.2 Arquitetura de comunicação do MATLAB com CoppeliaSim

Para que a conexão seja estabelecida entre o simulador e alguma aplicação de controle externa, como o MATLAB, é necessário utilizar uma das APIs disponibilizada pelos desenvolvedores e pela comunidade do CoppeliaSim. Existem várias alternativas mantidas pela comunidade do CoppeliaSim como: Remote API, Zero MQ Remote API, Legacy Remote API dentre outras, nesse trabalho será utilizado a Remote API (Coppelia Robotics, 2025b).

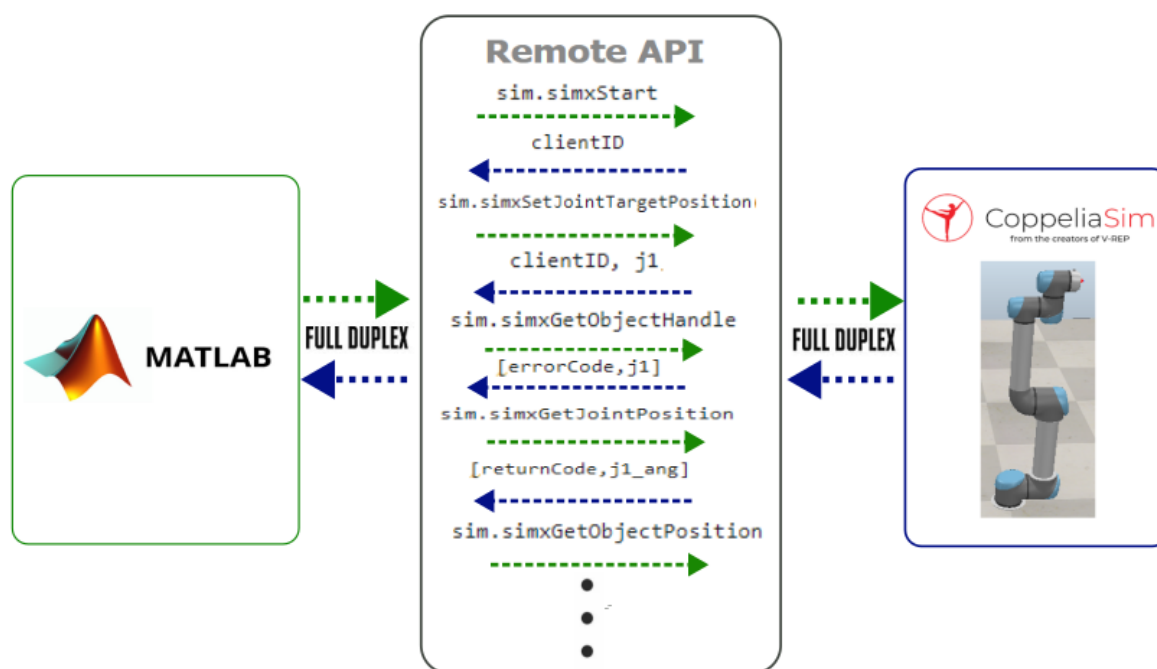
De acordo com a documentação do *Software*, a API permite a comunicação de um aplicativo externo (multiplataforma) com o CoppeliaSim e suporta as mesmas chamadas da função da API de dentro de um *script* CoppeliaSim. Esta comunicação pode ser feita de duas formas distintas: 1) Remote API e ZeroMQ com suporte aos clientes Python, C++,

MATLAB, Octave, Java, Lua e Rust; e 2) Remote API com WebSocket para a conexão de clientes JavaScript (Coppelia Robotics, 2025b).

Desta forma, na Figura 3.1, é possível identificar que o MATLAB atua como cliente requisitando informações específicas de componentes do UR5 enquanto que o CoppeliaSim atua servindo o MATLAB como servidor retornando esses dados solicitados.

A Figura 3.1 apresenta um diagrama representativo do processo de conexão do MATLAB com o CoppeliaSim, utilizando a Remote API. O Código 3.1 apresenta.

Figura 3.1 – Esquema de comunicação.



Fonte: O autor (2025)

Código 3.1 – Código de Conexão do CoppeliaSim via API remota

[illegible]

```
14     if (res == sim.simx_return_ok)
15         fprintf('Numero de objetos na cena: %d\n',length(objs));
16     else
17         fprintf('Chamada de funcao da API Remota retornou com codigo de
18             erro: %d\n',res);
19     end
20 end
21 fprintf('Tempo de Ping: %d\n', sim.simxGetPingTime(clientID))
```

O Código 3.1 apresenta a comunicação do MATLAB com o CoppeliaSim. De forma geral ele segue os seguintes passos:

1. Usa a API `remotemApi('remoteApi')`.
2. Encerra possíveis conexões anteriores com `sim.simxFinish(-1)`.
3. Estabelece uma nova conexão com o servidor CoppeliaSim por meio do comando `sim.simxStart`, utilizando o endereço local e a porta padrão.
4. Verifica se foi bem sucedida a conexão: se `clientID > -1`, então a conexão foi realizada com sucesso.
5. Obtém os objetos presentes na cena com a função `sim.simxGetObjects`, em modo bloqueante.
6. Imprime o número de objetos na cena ou do código de erro retornado.
7. Imprime o tempo de resposta com (*ping*) da conexão, por meio do comando `sim.simxGetPingTime`.

Este código é muito importante para testes e também permitir a comunicação entre o MATLAB e o CoppeliaSim, servindo como base para aplicações mais complexas que foram desenvolvidas mais adiante para modelagem e otimização do UR5.

3.2.1 Principais funções da Remote API

A Remote API do CoppeliaSim permite a comunicação entre o simulador e linguagens externas, como é explicado na Seção 3.2. Algumas funções são frequentemente utilizadas para movimentos essenciais do robô na cena. As principais são:

- `simxStart`: Estabelece uma conexão entre o *script* externo e o CoppeliaSim.
- `simxGetObjectHandle`: Retorna o identificador interno de um objeto da cena, dado seu nome.

- **simxSetJointTargetPosition:** Define a posição alvo (em radianos) de uma junta de revolução.
- **simxGetObjectPosition:** Retorna a posição de um objeto na cena, sempre em relação a um referencial estabelecido.

Modos de Operação da Remote API

As funções da Remote API do CoppeliaSim podem operar sob diferentes modos de transmissão de dados entre o cliente MATLAB e o servidor CoppeliaSim ([Coppelia Robotics, 2025a](#)). Os principais modos de operação são:

- **simx_opmode_blocking:** Esse é o modo bloqueante, possibilita com que o *script* externo espere até que a função seja completamente executada no CoppeliaSim e o valor de retorno esteja pronto.
- **simx_opmode_streaming:** A função é chamada uma vez para iniciar o envio contínuo de dados do CoppeliaSim ao cliente. É acessado com o modo **simx_opmode_buffer**, recuperando os valores armazenados no *buffer* do cliente.
- **simx_opmode_oneshot:** Utilizado para enviar comandos simples e rápidos, sem esperar retorno imediato.

É importante fazer a escolha certa do modo de operação por questões de desempenho e precisão dos valores. Desta forma o robô UR5 recebe os comandos. Dessa forma, o processo foi estruturado da seguinte maneira:

1. CoppeliaSim como servidor:

- Gerencia objetos na cena;
- Mantém a simulação em funcionamento;
- Fornece os dados ao cliente;
- Recebe comandos do cliente;

2. MATLAB como cliente:

- Inicia e finaliza a conexão;
- Solicita dados;
- Processa *scripts* de otimização;

3.3 Descritivo das etapas de desenvolvimento

O projeto foi desenvolvido com o objetivo de otimizar a solução da cinemática inversa do UR5 em configurações em que se objetiva movimentações desejadas na simulação, integrando o desenvolvimento via MATLAB. Para a conexão com o CoppeliaSim o processo baseou-se no Código 3.1 que estabelece a comunicação via API remota, para a conexão.

Para a validação estática

Foi elaborado o *script* que se encontra no Apêndice B para obter os dados de posição e orientação relativa entre os elos do robô diretamente do ambiente de simulação, armazenando essas informações em um arquivo.

Esses dados foram utilizados no *script* do Apêndice C para calcular as matrizes de transformação homogênea de cada elo, esse arquivo se baseia nas funções que se encontra no Apêndice G que calcula a matriz homogênea e para extrair os ângulos, e no Apêndice D se encontra o arquivo que calculará as matrizes finais baseadas no ambiente real de simulação.

Com essas matrizes, foi possível simular e verificar o comportamento do robô com base em sua geometria real (no CoppeliaSim).

Para validação dinâmica

Após a construção da modelagem cinemática direta estática, tornou-se necessário validar essa modelagem frente ao comportamento real do manipulador em movimento, considerando os efeitos dinâmicos introduzidos pelos *waypoints* (WP).

Para isso, foi implementado o *script* no MATLAB, que se encontra no Apêndice E responsável por simular diferentes configurações articulares do robô no ambiente do CoppeliaSim e comparar os resultados obtidos via simulação com os resultados calculados teoricamente a partir da multiplicação das matrizes de transformação homogênea dinâmicas.

Essa modelagem dinâmica foi baseada na inserção dos vetores da Equação (3.2) de eixo de rotação para cada junta, possibilitando a reconstrução da cinemática considerando o movimento efetivo das juntas a partir dos ângulos impostos. O objetivo desse arquivo é garantir que a modelagem cinemática represente fielmente a realidade observada na simulação, funcionando como uma etapa de validação antes da aplicação de algoritmos de otimização.

Otimização de posição

Como última etapa, foi implementado o algoritmo de otimização do MATLAB, que consta no Apêndice F, e cuja função é resolver a cinemática inversa ao encontrar a

melhor configuração de juntas para alcançar a posição desejada com os mesmos *waypoints*, minimizando o erro entre a posição obtida via cinemática direta e a referência.

Assim, para o melhor entendimento, cada uma destas etapas estão detalhadas na sequência.

3.3.1 Etapa 1 - Modelagem Estática

A modelagem estática da cinemática direta do robô UR5 teve como principal objetivo validar matematicamente a estrutura do robô com base em sua geometria real, considerada diretamente do ambiente de simulação no CoppeliaSim. Essa etapa foi fundamental para garantir que qualquer cálculo posterior principalmente os relacionados à cinemática inversa estejam alinhados com a estrutura física do modelo.

A metodologia adotada para a modelagem estática do robô UR5, se deu com os seguintes passos de implementação:

Passo 1: Coleta dos Ângulos das Juntas

Nesta etapa inicial, foram coletados os ângulos atuais de cada junta do robô utilizando a função `simxGetJointPosition`, da API remota do CoppeliaSim. Essa função retorna o valor angular (em radianos) de cada junta articulada do robô UR5 em sua configuração atual.

Esse procedimento é repetido para todas as juntas $j_1, j_2, \dots, j_6$. Os valores retornados são armazenados em variáveis correspondentes a cada junta: $\theta_1, \theta_2, \dots, \theta_6$

Assim, essa coleta representa a base da cinemática direta, que busca determinar a posição e orientação do efetuador final a partir da configuração das juntas.

Passo 2: Coleta da Posição e Orientação Relativa entre os Elos

Para cada elo i , foi obtido:

$$\vec{p}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \quad (3.1)$$

e os ângulos de Euler que é melhor explicado na Seção, 2.3.1.

Esses valores compõem a matriz de transformação homogênea da Equação (2.15). A matriz de rotação é obtida a partir dos ângulos de Euler.

Passo 3: Registro dos Dados

Após coletar as posições e ângulos de Euler de cada elo, esses dados foram organizados em duas matrizes:

- **pos**: Matriz contendo os vetores de posição de cada elo:
- **ang**: Matriz com os ângulos de Euler convertidos de radianos para graus, apenas para facilitar a análise visual:

Essas duas matrizes permitem compor, posteriormente, as matrizes de transformação homogênea T_i^{i-1} para cada elo.

Passo 4: Salvamento das Variáveis

Com um comando do MATLAB, foram salvos os dados, pois isso evita a necessidade de reconectar ao CoppeliaSim para reobter os dados. Os dados são salvos localmente para uso posterior.

Passo 5: Montagem das Matrizes de Transformação Homogênea

Com o comando para carregar os dados, cada transformação homogênea é no formato da Equação (2.15).

Para os ângulos de Euler, são no seguinte modo da Equação (2.3.1), a matriz de rotação composta é dada pela Equação (2.22). Separada por um produto de Equação (2.23), Equação (2.24) e Equação (2.25).

Passo 6: Produto Sucessivo das Transformações

Sejam as transformações homogêneas $T_0^1, T_1^2, \dots, T_6^7$. A posição e orientação final do efetuador em relação à base é dado pelo produto de transformações no formato da Equação (2.38).

Passo 7: Validação com Vetores de Referência

São definidos vetores de referência no sistema da base do robô:

$$\mathbf{P}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{P}_x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{P}_y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{P}_z = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad (3.2)$$

Esses vetores representam a origem e os eixos unitários no sistema local da base do robô.

Cada vetor é transformado para o sistema global por meio da multiplicação pela matriz de transformação acumulada $\mathbf{T}_{\text{acum}}$, ou seja:

$$\mathbf{P}_i^{\text{global}} = \mathbf{T}_{\text{acum}} \cdot \mathbf{P}_i \quad (3.3)$$

Essa operação mostra a nova posição da origem e dos eixos locais de um sistema de coordenadas após todas as transformações até a junta i .

- O vetor $\mathbf{P}_0$ representa a posição final do elo i no espaço global.
- Os vetores $\mathbf{P}_x, \mathbf{P}_y, \mathbf{P}_z$ representam a orientação dos eixos locais do elo i no sistema global.

Essa abordagem permite visualizar e validar se a transformação acumulada está correta, ao comparar com resultados obtidos na simulação. Em resumo, a modelagem estática foi consolidada como:

- Um arquivo de modelagem estática com posições e ângulos das juntas diretamente do CoppeliaSim.
- Um arquivo de transformação homogênea (matriz T) que interpreta os dados do item anterior calcula a matriz de transformação homogênea de cada junta.
- Um arquivo de validação estática que compara os valores da modelagem com os da transformação homogênea para garantir que coincidam.

Resultados

A matriz T_0^7 representa a cinemática direta do robô UR5 ou seja, a posição e orientação de cada elo em relação à base. Com as matrizes calculadas, foi possível simular e verificar o comportamento do robô com base em sua geometria real (no CoppeliaSim). A Tabela 4 enfatiza a posição de cada junta e o ângulo e eixo de rotação:

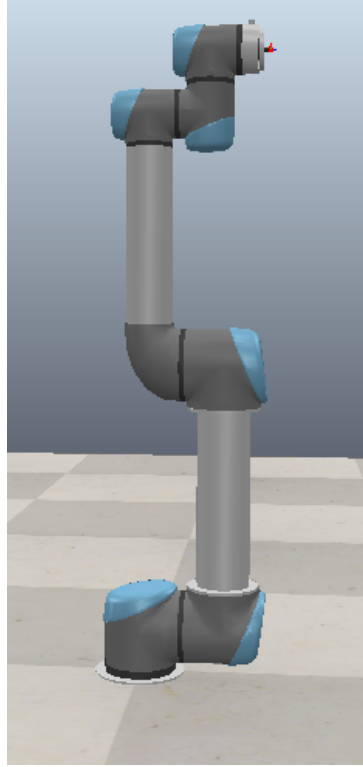
Tabela 4 – Posição e orientação relativa das juntas do robô UR5, ordem ZYX.

Junta	Posição (x, y, z) (m)	Angulação (X, Y, Z) (°)	Eixo de Rotação
T_0^1	(0,0000; -0,0000; 0,0085)	(-0,0; 0,0; -0,0)	Z
T_1^2	(-0,0703; -0,0000; 0,0661)	(0,0; -90,0; 0,0)	Y
T_2^3	(0,4251; 0,0000; 0,0000)	(-0,0; 0,0; 0,0)	Z
T_3^4	(0,3922; 0,0000; 0,0000)	(-0,0; 0,0; -0,0)	Z
T_4^5	(0,0456; -0,0000; 0,0397)	(0,0; 90,0; -0,0)	Y
T_5^6	(0,0144; 0,0000; 0,0492)	(0,0; -90,0; -0,0)	Y
T_6^7	(0,0000; -0,0000; 0,1067)	(-0,0; 0,0; -0,0)	Z

Fonte: O autor (2025).

As matrizes obtidas com o UR5 estático como na Figura 3.2 foram:

Figura 3.2 – Robô UR5 estático.



Fonte: CoppeliaSim

$$\mathbf{T}_0^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.0000 \\ 0 & 0 & 1 & 0.0085 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

$$\mathbf{T}_1^2 = \begin{bmatrix} 0 & 0 & -1 & -0.0703 \\ -0 & 1 & 0 & -0.0000 \\ 1 & 0 & 0 & 0.0661 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

$$\mathbf{T}_2^3 = \begin{bmatrix} 1 & 0 & 0 & 0.4251 \\ 0 & 1 & 0 & 0.0000 \\ 0 & 0 & 1 & 0.0000 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

$$\mathbf{T}_3^4 = \begin{bmatrix} 1 & 0 & 0 & 0.3922 \\ 0 & 1 & 0 & 0.0000 \\ 0 & 0 & 1 & 0.0000 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.7)$$

$$\mathbf{T}_4^5 = \begin{bmatrix} 0 & 0 & 1 & 0.0456 \\ 0 & 1 & 0 & -0.0000 \\ -1 & 0 & 0 & 0.0397 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.8)$$

$$\mathbf{T}_5^6 = \begin{bmatrix} 0 & 0 & -1 & 0.0144 \\ 0 & 1 & 0 & 0.0000 \\ 1 & 0 & 0 & 0.0492 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.9)$$

$$\mathbf{T}_6^7 = \begin{bmatrix} 1 & 0 & 0 & 0.0000 \\ 0 & 1 & 0 & -0.0000 \\ 0 & 0 & 1 & 0.1067 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.10)$$

De acordo com a função expressa no Código [G.5](#) linha 8, a transformação global acumulada pode ser expressa como um produto sucessivo de transformações estáticas e dinâmicas:

$$\mathbf{T}_{\text{global}} = \prod_{i=1}^n \left(\mathbf{T}_{\text{static}}^{(i)} \cdot \mathbf{T}_{\text{dynamic}}^{(i)} \right) \quad (3.11)$$

Ou, de forma expandida:

$$\mathbf{T}_{\text{global}} = \mathbf{T}_{\text{static}}^{(1)} \cdot \mathbf{T}_{\text{dynamic}}^{(n)} \cdot \mathbf{T}_{\text{static}}^{(n)} \cdot \mathbf{T}_{\text{dynamic}}^{(n)} \cdots \mathbf{T}_{\text{static}}^{(n)} \cdot \mathbf{T}_{\text{dynamic}}^{(n)} \quad (3.12)$$

Onde cada parcela possui a estrutura de uma matriz de transformação homogênea dada pela Equação (2.15) estática e Equação (2.36) dinâmica, respectivamente, e que n varia de 1 a 6.

3.3.2 Etapa 2 - Modelagem incluindo rotações

A validação dos movimentos do UR5, tem como principal objetivo assegurar que os resultados almejados teoricamente por meio da estimativa com *waypoints* estejam de acordo com o comportamento observado na simulação. Isso garante que os cálculos de posição e orientação do atuador, derivados da cinemática direta e inversa, sejam precisos e confiáveis.

A validação é essencial tanto para confirmar a consistência do modelo quanto para identificar e corrigir possíveis erros entre a teoria e a simulação. Para isso, um conjunto de passos é executado de forma sequencial, conforme descrito:

Passo 1: Configuração das posições das juntas

Com a função `simxSetJointTargetPosition` a finalidade consistiu em definir a posição alvo das juntas do robô com os valores fornecidos na matriz `WP_q`. Isso move o robô para as posições especificadas.

Tal escolha se justifica porque o controle do movimento do robô possa ser feito manipulando as posições das juntas. O robô se move até essas posições de forma sequencial.

Passo 2: Validação da convergência da posição das juntas

O objetivo nessa etapa foi monitorar a convergência do robô até a posição das juntas desejadas. A cada repetição do laço, as posições reais das juntas são verificadas e comparadas com as posições desejadas (q_i), ajustando o movimento até que a diferença (erro angular) seja suficientemente pequena.

A explicação desse passo é o fato da garantia que o robô atingiu com precisão a posição desejada antes de seguir para o próximo passo. Caso o erro seja maior que o limite especificado, o robô continua ajustando suas posições.

Passo 3: Obtenção da Posição e Orientação do Atuador

Com a função `simxGetObjectPosition` pretendeu-se obter a posição e a orientação do atuador em relação à base do UR5. Isso permite verificar a eficácia do movimento do robô, comparando a posição real do atuador com a posição calculada.

Isso se torna necessário para validar a precisão do movimento do robô e garantir que ele está se movendo corretamente em relação ao espaço de trabalho definido.

Passo 4: Cálculo da cinemática

Essa etapa é muito importante pois usando uma função do Apêndice G com os parâmetros específicos, o objetivo foi calcular a transformação dinâmica teórica do robô com base nas posições das juntas, utilizando outra função do Apêndice G que aplica a cinemática direta para obter a matriz de transformação do sistema de coordenadas do atuador em relação ao sistema base.

Isso se deve ao fato de poder fazer a comparação entre a posição calculada teoricamente e a posição real obtida no passo anterior, permitindo verificar a precisão do modelo de cinemática e a correspondência entre o modelo e a simulação real.

Passo 5: Cálculo e validação da posição e orientação

O intuito foi de calcular a posição e a orientação do atuador com base na matriz de transformação final `T_final`, que inclui o movimento das juntas e o atuador. A ideia é validar se os movimentos do robô, baseados na cinemática, correspondem à realidade

simulada do CoppeliaSim. A posição calculada deve ser próxima à posição obtida na simulação.

Passo 6: Armazenamento dos dados

O objetivo nesse passo foi armazenar os dados de posição e orientação do atuador vindos da simulação e os dados calculados a partir da cinemática, para comparações futuras. Desta forma se permite que seja feita a análise dos dados simulados e calculados e se verifique a precisão do modelo teórico comparado aos resultados da simulação.

Passo 7: Comparação e Análise Final

Com os dados salvos em `WP_3D_sim` e `WP_3D_calc`, a ideia aqui foi exibir as comparações entre as posições simuladas e as calculadas, para verificar se o modelo de cinemática está funcionando corretamente. Pois comparar esses valores é crucial para validar a precisão da cinemática calculada e garantir que o modelo está representando corretamente o comportamento do UR5 no espaço de trabalho.

Uma observação importante é, embora a convenção que adotamos considera sempre o eixo Z como eixo de rotação das juntas, como explicitado nos Apêndices: [E](#) linha 42, [F](#) linha 20, e no Código: [4.1](#) linha 10. O CoppeliaSim pode apresentar rotações aparentes em outros eixos (como Y) como mostra a Tabela: [4](#) devido à forma como os eixos locais são orientados no modelo 3D. Isso é uma característica construtiva do simulador e não altera o fato de que cada junta gira em torno do seu eixo Z local, conforme previsto na modelagem teórica.

4 Resultados e Análises

Este capítulo relata os resultados obtidos das modelagens e suas respectivas análises, sendo interpretados com dados e comparativos com cálculo de erros. O processo de otimização também é analisado e as considerações finais através dos resultados. A organização deste capítulo segue a seguinte ordem: a Seção 4.1 fala da modelagem e validação do UR5 dinâmico; a Seção 4.2 traz os resultados da simulação e otimização aplicadas ao UR5; e a Seção 4.3 apresenta das considerações finais do processo de simulação.

4.1 Validação da Modelagem da Cinemática Direta

O objetivo desta seção foi que dados os conjuntos de ângulos que são os valores da Tabela 5, se espera atingir a posição e os ângulos desejados alcançados no simulador. Posteriormente fazer as devidas análises a respeito dos resultados. Foram comparados os valores calculados via cinemática direta teoricamente e os gerados no simulador. Os passos para a modelagem dinâmica com os *waypoints* da Tabela 5 já foram detalhados na Seção 3.3.2.

Tabela 5 – Tabela de *Waypoints*. Cada linha representa um conjunto de ângulos (em graus) aplicados às juntas do manipulador UR5.

WP	Junta 1 (q_1)	Junta 2 (q_2)	Junta 3 (q_3)	Junta 4 (q_4)	Junta 5 (q_5)	Junta 6 (q_6)
1	0	0	0	0	0	0
2	10	15	-10	-15	-20	10
3	80	30	45	50	20	100
4	-100	-50	-30	0	100	-5
5	-10	30	35	25	15	50
6	150	75	20	10	-5	15
7	0	0	0	0	0	0

Fonte: O autor (2025).

De forma geral os resultados foram bem próximos, tanto posição quanto orientação. Considerando que os ângulos que o sistema foi submetido constam na Tabela 5.

Processo de obtenção de posição e orientação simulada

As medidas obtidas diretamente da simulação tem origem no código que consta no Apêndice E, nas linhas 94 e 95. Em outras palavras, as medidas foram obtidas diretamente do simulador.

Processo de obtenção de posição e orientação teórico

Os valores dos cálculos teóricos tem origem através da Equação (2.36), que foi possível obter as posições e orientações do sistema, no Apêndice E, consta o código que calcula teoricamente estes resultados mais precisamente nas linhas, 106, 109, 114 e 116.

É possível fazer essa análise pela Tabela 6, que se refere apenas aos dados de posição calculada teoricamente e simulada respectivamente do efetuador.

Tabela 6 – Comparação entre posições teóricas e simuladas do efetuador no CoppeliaSim

WP	Ângulos (°)	Calculados efetuador (m)	CoppeliaSim efetuador (m)
1	(0, 0, 0, 0, 0, 0)	(-0,2023, 0,0000, 0,9866)	(-0,2024, 0,0000, 0,9865)
2	(10, 15, -10, -15, -20, 10)	(-0,2213, 0,1223, 0,9746)	(-0,2215, 0,1225, 0,9745)
3	(80, 30, 45, 50, 20, 100)	(-0,7108, -0,0744, 0,5157)	(-0,7113, -0,0743, 0,5148)
4	(-100, -50, -30, 0, 100, -5)	(-0,7921, 0,2351, 0,3429)	(-0,7926, 0,2352, 0,3415)
5	(-10, 30, 35, 25, 15, 50)	(-0,0810, 0,6872, 0,6323)	(-0,0809, 0,6879, 0,6314)
6	(150, 75, 20, 10, -5, 15)	(-0,2705, -0,8723, 0,1181)	(-0,2706, -0,8724, 0,1163)
7	(0, 0, 0, 0, 0, 0)	(-0,2023, 0,0000, 0,9866)	(-0,2024, 0,0000, 0,9865)

Fonte: O autor (2025).

A Tabela 7 apresenta os ângulos do sistema de forma teórica e calculada, seguindo mesmo padrão de posição.

Tabela 7 – Comparação entre orientações angulares do sistema teóricas e simuladas do efetuador no CoppeliaSim ordem (ZYX)

WP	Ângulos	Calculados (ZYX)	CoppeliaSim
1	(0, 0, 0, 0, 0, 0)	(0.00, -90.00, 0.00)	(0.00, -90.00, 0.00)
2	(10, 15, -10, -15, -20, 10)	(-9.83, -86.54, 0.19)	(-9.83, -86.54, 0.19)
3	(80, 30, 45, 50, 20, 100)	(68.14, 41.18, -36.58)	(68.14, 41.18, -36.58)
4	(-100, -50, -30, 0, 100, -5)	(-170.90, -10.82, 175.01)	(-170.90, -10.82, 175.01)
5	(-10, 30, 35, 25, 15, 50)	(-67.37, 47.72, 97.13)	(-67.37, 47.72, 97.13)
6	(150, 75, 20, 10, -5, 15)	(-95.57, 29.93, -121.49)	(-95.57, 29.93, -121.49)
7	(0, 0, 0, 0, 0, 0)	(0.00, -90.00, 0.00)	(0.00, -90.00, 0.00)

Fonte: O autor (2025).

O erro absoluto para cada coordenada pode ser calculado da seguinte forma:

Cálculo dos erros para validação da posição e orientação

Nosso objetivo é validar os valores teóricos com os simulados, foi realizado o cálculo do erro absoluto entre os ângulos de orientação (em ZYX) calculados e os fornecidos pelo ambiente de simulação para cada conjunto de *waypoints*. Neste caso, foi utilizado como norma o cálculo do erro quadrático é definido como a diferença entre os valores teóricos e simulados, servindo como um indicador direto da precisão da implementação da cinemática direta. Valores próximos de zero indicam uma boa correspondência entre o

modelo matemático e a simulação, reforçando a confiabilidade da abordagem adotada.

$$E_{\text{abs}} = |x_{\text{teo}} - x_{\text{sim}}| \quad (4.1)$$

Sendo:

- E_{abs} é o erro absoluto;
- x_{sim} é o valor obtido pela simulação;
- x_{teo} é o valor teórico (modelado).

A seguir, são apresentados os erros absolutos entre as posições teóricas obtidas pelo modelo e as posições simuladas no CoppeliaSim, em metros, para cada *waypoint*, juntamente com os ângulos seguindo mesmo padrão em graus:

Tabela 8 – Erros absolutos entre as posições e orientações calculadas e as fornecidas pelo CoppeliaSim

wp	Ângulos (°)	X (m)	Y (m)	Z (m)	Roll (°)	Pitch (°)	Yaw (°)
1	(0, 0, 0, 0, 0, 0)	0,0001	0,0000	0,0001	0	0	0
2	(10, 15, -10, -15, -20, 10)	0,0002	0,0002	0,0001	0	0	0
3	(80, 30, 45, 50, 20, 100)	0,0005	0,0001	0,0009	0	0	0
4	(-100, -50, -30, 0, 100, -5)	0,0005	0,0001	0,0014	0	0	0
5	(-10, 30, 35, 25, 15, 50)	0,0001	0,0007	0,0009	0	0	0
6	(150, 75, 20, 10, -5, 15)	0,0001	0,0001	0,0018	0	0	0
7	(0, 0, 0, 0, 0, 0)	0,0001	0,0000	0,0001	0	0	0

Fonte: O autor (2025).

Assim, como explicitado na Tabela 8 a validação foi bem-sucedida e o modelo pode ser utilizado para projetos, como a implementação da cinemática inversa otimizando sua posição.

Essa relação entre os eixos da matriz e os ângulos observados valida a consistência da orientação simulada com a matriz de rotação teórica. Com essa interpretação, observa-se que a orientação do efetuador está corretamente representada na matriz teórica, coincidindo com os resultados da simulação.

4.2 Implementação Cinemática Inversa por Otimização

O objetivo central consiste em aplicar a cinemática inversa para chegar nos ângulos dos *waypoints* estimados no código do Apêndice F nas linhas 7 a 14, a partir das posições e orientações.

Código 4.1 – Função custo de otimização de posição do UR5

```

1 function [J] = ur5_J(q)
2
3 q = q * pi/180;
4
5 global T_static_total pos_obj ang_obj
6
7 T_static = T_static_total;
8
9 % 0 eixo em torno de cada angulo que q gira
10 u_i_static = [ 0, 0, 1;
11                0, 0, 1;
12                0, 0, 1;
13                0, 0, 1;
14                0, 0, 1;
15                0, 0, 1];
16
17
18 % Calcular a UTM para cada junta
19 T_juntas = calc_T_dynamic(T_static(:,:,1:6), q, u_i_static);
20
21 % Adiciona o efeito do atuador
22 T_est = T_juntas*T_static(:,:,7);
23
24 % Extrair a posicao e a orientacao do atuador no referencial do robo
25 [u_est, ang_est, pos_est] = calc_T_extract_u_angle(T_est);
26
27 ang_est = ang_est*u_est; % Representacao vetorial do Angulo/Eixo de
    Euler
28
29 err_pos = 100*(pos_est - pos_obj).^2;
30 err_ang = 10*(ang_est - ang_obj).^2;
31
32 err_vec = [ err_pos;
33            err_ang ];
34
35
36 J = 1e3*err_vec;
37 end

```

Considerando a função custo apresentada no Código 4.1, esta pode ser representada matematicamente por:

$$err_{pos} = 100(pos_{est} - pos_{obj})^2 \quad (4.2)$$

$$err_{ang} = 10(ang_{est} - ang_{obj})^2 \quad (4.3)$$

Assim, o vetor de erro é dado por:

$$err_{vec} = \begin{bmatrix} err_{pos} \\ err_{ang} \end{bmatrix} \quad (4.4)$$

E, por fim, a função custo J é expressa por:

$$J = 1 \times 10^3 \left(100(pos_{est} - pos_{obj})^2 + 10(ang_{est} - ang_{obj})^2 \right). \quad (4.5)$$

O objetivo do problema é fazer com que a função *lsqnonlin* do MATLAB procure quais são os valores de $\mathbf{q}$ que fazem com que o erro entre as posições e orientações que se tem e as posições e orientações esperadas sejam mínimos, ou seja, que minimizam o valor da função custo.

Os seguintes passos foram necessários

Uma vez que os carregamento de dados foi consolidado, os *waypoints* de referência no espaço de juntas são definidos, seguido dos eixos de rotação sempre em z .

São inicializadas as variáveis `wp_q_opt`, `wp_3d_calc`, que irá armazenar os *waypoints* otimizados e (posição e orientação) calculadas para cada *waypoint* de referência respectivamente.

Através do laço de repetição as seguintes etapas são executadas:

- Pega a configuração desejada das juntas do robô com `q_ref`, é calculado onde o efetuador do robô deveria estar e informações sobre a estrutura a cinemática direta do robô salvos em `T_static` e `u_static`.
- O código calcula a posição e a orientação exatas com `T_ref` que a extremidade do braço do robô (o efetuador final) deveria ter no espaço 3D.
- Se armazena onde o efetuador deveria estar, em `wp_3d_calc`. Isso cria um registro dos pontos que o robô deveria alcançar no espaço 3D, baseados nos *waypoints* de referência.
- Usa o solucionador do MATLAB `ur5_IK_a_nonlinear` para encontrar uma nova combinação de ângulos das juntas armazenadas em `q_star` que também levará o efetuador do robô à mesma posição e orientação desejadas salva em `T_ref`. Essa é a parte da otimização, onde o código busca uma maneira de alcançar o mesmo ponto.
- A nova combinação de ângulos encontrada `q_star` é armazenada em `wp_q_opt`. Esta matriz conterá as configurações de juntas otimizadas para cada ponto de referência.
- Na comparação final, o código mostra no console os valores originais `q_ref` e os valores otimizados `q_star` que foram encontrados.

Portanto, as saídas dos *waypoints* otimizados, ou seja, as configurações realmente atingidas pelo UR5 na simulação se devem ao método *lsqnonlinear* do MATLAB apresen-

tados na Tabela 9.

Comparação pelo algoritmo lsqnonlinear

Tabela 9 – Comparação entre wp_3d_ref e wp_3d_opt obtidos pela otimização

WP	wp_3d_ref						wp_3d_opt					
	p_x	p_y	p_z	ϕ	θ	ψ	p_x	p_y	p_z	ϕ	θ	ψ
1	-0.2024	-0.0000	0.9865	0.0000	-89.9934	-0.0000	-0.2022	-0.0056	0.9865	0.0000	-89.8419	43.0638
2	-0.2184	0.1055	0.9720	-90.0859	-79.9993	-80.0705	-0.1253	0.1175	0.9704	-68.5057	-81.4110	-68.2089
3	-0.7110	-0.0744	0.5573	70.2068	-6.0328	-92.7215	-0.6732	0.0207	0.4850	72.5749	-20.9025	-49.6310
4	-0.7299	0.2242	0.5901	175.3529	-40.7360	179.6357	-0.7653	0.2460	0.4395	172.0624	-12.2750	-176.5053
5	-0.0922	0.6240	0.7408	-5.8768	-77.6926	98.1219	0.0206	0.6700	0.6039	-33.0008	-72.0213	105.7404
6	0.0598	-0.3399	0.8478	-54.3491	48.8829	-162.4532	-0.1732	0.1506	0.9068	-79.2341	46.3080	-174.8225
7	-0.2735	-0.8775	0.2174	99.7891	59.2033	-178.3477	-0.3458	-0.7878	0.1083	84.0900	61.7059	-142.9979
8	-0.2024	-0.0000	0.9865	0.0000	-89.9932	-0.0000	-0.2024	-0.0000	0.9865	0.0000	-89.9932	-0.0000

Fonte: o Autor (2025).

Análise da Tabela de Erros Absolutos

Os *Waypoints* 1 á 8 apresentam erro nulo ou praticamente nulo em todas as posições. Isso sugere que a posição inicial e final da trajetória foram reproduzidas com alta precisão, o que é esperado para garantir estabilidade e repetibilidade no início e fim do ciclo do robô.

Houve maior erro em juntas intermediárias. Os maiores erros aparecem nos *Waypoints* intermediários (especialmente WP 5 e WP 6).

Isso indica que durante o movimento, o algoritmo pode estar tendo dificuldades em seguir exatamente as orientações previstas, o que pode ser devido a:

- limitações físicas do robô no CoppeliaSim (velocidade, aceleração, limites de junta);
- simplificações no modelo cinemático;
- erros numéricos no cálculo da cinemática inversa.

4.3 Considerações finais do processo de simulação

A combinação entre o Software MATLAB e o simulador CoppeliaSim resultou em um ambiente produtivo e efetivo para a tarefa proposta. O sistema de comunicação foi de fácil configuração e permitiu obter os parâmetros físicos do UR5, simulado virtualmente, de forma interativa. Este cenário permitiu avaliar a qualidade dos resultados da otimização dos movimentos.

5 Conclusão

O objetivo do trabalho foi verificar a possibilidade de otimizar possíveis posições do manipulador robótico UR5 no simulador CoppeliaSim. Foi utilizado o algoritmo de otimização *lsqnonlinear*, implementado via MATLAB, a API de comunicação demonstrou estabilidade e desempenho consistentes nas requisições de chamadas e devolutivas de informações em tempo hábil. Uma análise mais holística dos resultados demonstrou que eles foram positivos, constatando uma boa combinação entre robótica e otimização no ambiente de simulação, sendo uma abordagem que pode ser considerada por setores que queiram testar posições otimizadas de robôs antes de sua implantação real em ambiente de produção.

As dificuldades encontradas no decorrer do projeto existiram. Inicialmente, foram feitos testes com a linguagem Python que também provou ser versátil. Embora testes iniciais tenham sido realizados com Python, optou-se pelo uso do MATLAB em razão de sua estabilidade na integração com o CoppeliaSim e à sua simplificação, fatores cruciais para o cronograma do trabalho. Essa decisão foi tomada ao longo do processo de desenvolvimento de seguir com o MATLAB, para trabalhos futuros o Python poderá ser priorizado novamente devido a versatilidade de desenvolvimento de bibliotecas.

Para trabalhos futuros, será interessante expandir o escopo do projeto para incluir a otimização de trajetórias em operações do tipo *pick and place*, que são amplamente utilizadas na indústria. Nesse contexto, além de otimizar posições estáticas do robô, o manipulador robótico teria sua trajetória entre os pontos de coleta e posicionamento ajustada para minimizar critérios como: tempo, consumo de energia e ainda evitar colisões. Essa abordagem permitiria uma análise bem aplicável em ambientes produtivos, agregando ainda mais valor à integração entre simulação, otimização e robótica.

Referências Bibliográficas

- BRIOT, S.; KHALIL, W. *Dynamics of Parallel Robots*. [S.l.: s.n.], 2015. v. 35. ISBN 978-3-319-19787-6. Citado na página 33.
- BRUNO, S. et al. Robotics: modelling, planning and control, 2010. *Cited on*, v. 1, 1994. Citado 4 vezes nas páginas 22, 27, 30 e 37.
- COLLINS, J. et al. A review of physics simulators for robotic applications. *IEEE Access*, v. 9, p. 51416–51431, 2021. Citado na página 14.
- Coppelia Robotics. *Remote API modus operandi - CoppeliaSim documentation*. 2025. Acesso em: 10 abr. 2025. Disponível em: <<https://manual.coppeliarobotics.com/en/remoteApiModusOperandi.htm>>. Citado na página 45.
- Coppelia Robotics. *Remote API Overview - CoppeliaSim*. 2025. Acessado em: 22 mar. 2025. Disponível em: <<https://manual.coppeliarobotics.com/en/remoteApiOverview.htm>>. Citado 2 vezes nas páginas 42 e 43.
- COPPELIASIM. *CoppeliaSim User Manual*. 2024. Disponível em: <<https://manual.coppeliarobotics.com/index.html>>. Citado na página 68.
- COTA, E. et al. Robótica na mineração. *Anais dos Seminários de Redução, Minério de Ferro e Aglomeração*, v. 47, n. 2, p. 359–370, 2017. Citado na página 15.
- CRAIG, J. Introduction to robotics: mechanics and control, ser. *Reading*, 2005. Citado 7 vezes nas páginas 15, 18, 19, 24, 26, 36 e 37.
- GALIN, R. et al. Cobots and the benefits of their implementation in intelligent manufacturing. In: IOP PUBLISHING. *IOP conference series: Materials science and engineering*. [S.l.], 2020. v. 862, n. 3, p. 032075. Citado na página 14.
- GRAVDAHL, J. T. Force estimation in robotic manipulators: Modeling, simulation and experiments. *MS thesis*, Citeseer, 2014. Citado 3 vezes nas páginas 22, 23 e 24.
- HAMEED, A. et al. Control system design and methods for collaborative robots. *Applied Sciences*, MDPI, v. 13, n. 1, p. 675, 2023. Citado na página 20.
- KAWAAYE, A. G. M. et al. Lucratividade das empresas por meio da mão de obra robótica. 098, 2022. Citado na página 15.
- KUKA Roboter GmbH. *Manual do KR 1000 titan*. [S.l.], 2024. Manual técnico. Acesso em: 24 mar. 2025. Disponível em: <<https://www.kuka.com/pt-br/produtos-servi%C3%A7os/sistemas-de-rob%C3%B4/rob%C3%B4s-industriais/kr-1000-titan>>. Citado na página 15.
- LYNCH, K. M.; PARK, F. C. *Modern Robotics: Mechanics, Planning, and Control*. 1st. ed. USA: Cambridge University Press, 2017. ISBN 1107156300. Citado 4 vezes nas páginas 29, 34, 35 e 36.

MARKLEY, F.; CRASSIDIS, J. *Fundamentals of Spacecraft Attitude Determination and Control*. Springer New York, 2014. (Space Technology Library). ISBN 9781493908011. Disponível em: <<https://books.google.com.br/books?id=HjDCngEACAAJ>>. Citado 5 vezes nas páginas 26, 27, 31, 33 e 34.

MATARIĆ, M. *Introdução à robótica*. BLUCHER, 2014. ISBN 9788521208549. Disponível em: <<https://books.google.com.br/books?id=Nj26DwAAQBAJ>>. Citado na página 14.

MathWorks. *lsqnonlin (Optimization Toolbox)*. Natick, Massachusetts, USA, 2025. <<https://www.mathworks.com/help/optim/ug/lqnonlin.html>>. Citado 3 vezes nas páginas 39, 40 e 41.

POLLAK, M. et al. Suitability of UR5 robot for robotic 3D printing. *Applied Sciences*, v. 14, n. 21, 2024. ISSN 2076-3417. Disponível em: <<https://www.mdpi.com/2076-3417/14/21/9845>>. Citado na página 16.

REITER, A. *Optimal Path and Trajectory Planning for Serial Robots*. [S.l.]: Springer, 2020. Citado na página 38.

ROBOTS, U. *Nossa História*. 2024. Acesso em: 06 mar. 2025. Disponível em: <<https://www.universal-robots.com/br/sobre-a-universal-robots/nossa-hist%C3%B3ria/>>. Citado na página 19.

ROHMER, E.; SINGH, S. P. N.; FREESE, M. V-rep: A versatile and scalable robot simulation framework. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.: s.n.], 2013. p. 1321–1326. Citado na página 16.

ROSÁRIO, J. *Robótica Industrial I Modelagem, Utilização e Programação*. Editora Baraúna, 2012. ISBN 9788579231452. Disponível em: <<https://books.google.com.br/books?id=cbzXOdbJvYoC>>. Citado na página 18.

SIMPLÍCIO, P. V. G.; LIMA, B. R. Manipuladores robóticos industriais. *Caderno de Graduação - Ciências Exatas e Tecnológicas - UNIT - SERGIPE*, v. 3, n. 3, p. 85, out. 2016. Disponível em: <<https://periodicos.set.edu.br/cadernoexatas/article/view/3572>>. Citado na página 15.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. Robot dynamic and control second edition. *Tech. Rep.*, 2004. Citado 2 vezes nas páginas 23 e 24.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. *Robot modeling and control*. [S.l.]: John Wiley & Sons, 2020. Citado na página 18.

SPONG, M. W.; VIDYASAGAR, M. *Robot dynamics and control*. [S.l.]: John Wiley & Sons, 2008. Citado 3 vezes nas páginas 27, 28 e 29.

STARETU, I. Collaborative robots (cobots)-systematization and kinematics. *The Romanian Journal of Technical Sciences. Applied Mechanics*, v. 66, n. 1, p. 81–89, 2021. Citado na página 20.

The MathWorks, Inc. *Global Optimization Toolbox Documentation*. Natick, Massachusetts, USA, 2025. <<https://www.mathworks.com/help/gads/index.html>>. Citado na página 39.

The MathWorks, Inc. *Mathematics and Optimization - MATLAB & Simulink Documentation*. 2025. <<https://www.mathworks.com/help/matlab/mathematics-and-optimization.html>>. Acessado em 28 de abril de 2025. Citado na página 39.

Universal Robots. *UR E-Series Brochure*. 2021. Acesso em: 11 mar. 2025. Disponível em: <https://www.universal-robots.com/media/1821154/ur_e-series_brochure_pt_web.pdf>. Citado na página 22.

Universal Robots. *UR5e Technical Specifications*. [S.l.], 2023. Accessed: 2025-03-16. Disponível em: <https://www.universal-robots.com/media/50588/ur5_en.pdf>. Citado na página 21.

Universal Robots. *UR5e - A flexible and lightweight robotic arm*. 2024. Acessado em: 6 mar. 2025. Disponível em: <<https://www.universal-robots.com/products/ur5e/>>. Citado 2 vezes nas páginas 16 e 20.

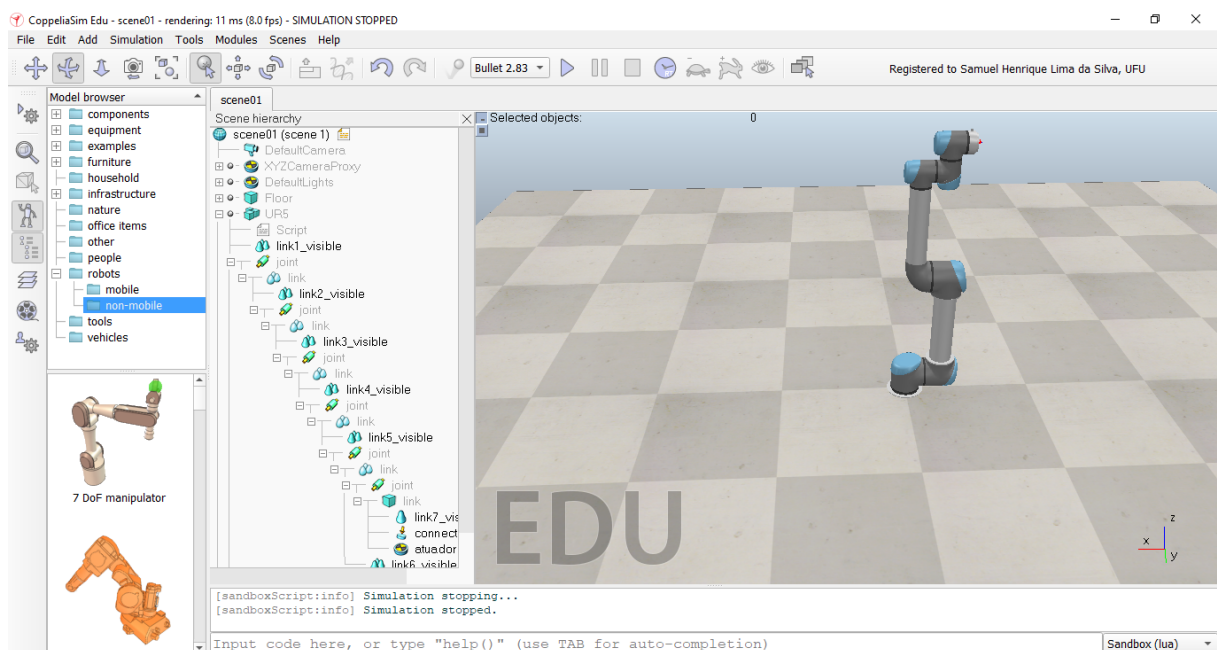
Universal Robots. *Applications*. 2025. Acesso em: 11 mar. 2025. Disponível em: <<https://www.universal-robots.com/applications/>>. Citado 2 vezes nas páginas 20 e 22.

APÊNDICE A – Descrição da Interface do CoppeliaSim

Este apêndice traz o ambiente gráfico de simulação do CoppeliaSim, com ênfase na organização visual das seções com sua devida descrição. Serão descritos os componentes da interface, incluindo a área de visualização 3D, o bloco de hierarquia de objetos e propriedades, bem como os controles de simulação.

A Figura A.1 apresenta uma visão geral da interface de simulação do CoppeliaSim, onde está apresentada uma inserção do robô UR5. Para melhor compreensão das funcionalidades apresentadas neste ambiente, serão detalhadas as principais ferramentas disponibilizadas e seus modos de uso.

Figura A.1 – Cena exemplificando o uso do UR5 no CoppeliaSim.



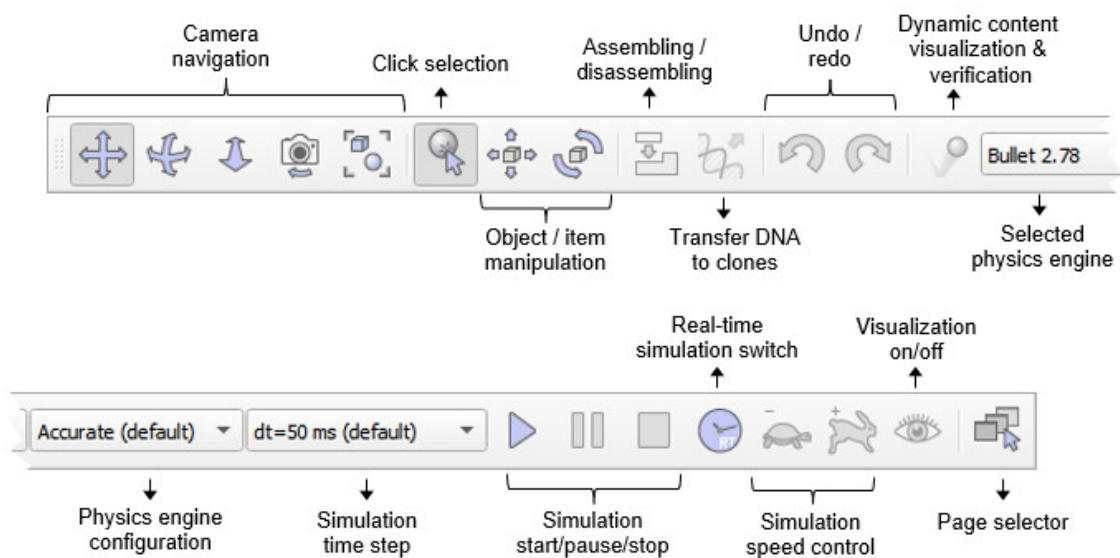
Fonte: O autor (2025)

A Figura A.2 apresenta a barra de navegação, em que consta os principais controles básicos de simulação como movimentos, *play* de simulação, navegação imersiva, pausa parada, com esses comandos é possível movimentar a câmera.

- *Camera navigation*: Navegação da câmera;
- *Click selection*: Seleção por clique;
- *Assembling/disassembling*: Montagem/desmontagem;

- *Undo/redo*: Desfazer/refazer;
- *Dynamic content visualization and verification*: Visualização e verificação de conteúdo dinâmico;
- *Object/item manipulation*: Manipulação de objetos/itens;
- *Transfer DNA to clones*: Transferir DNA para clones;
- *Selected physics engine*: Motor de física selecionado;
- *Physics engine configuration*: Configuração do motor de física;
- *Simulation time step*: Passo de tempo da simulação;
- *Simulation start/pause/stop*: Início/pausa/parada da simulação;
- *Simulation speed control*: Controle de velocidade da simulação;
- *Page selector*: Seletor de página.

Figura A.2 – Menu superior no CoppeliaSim.



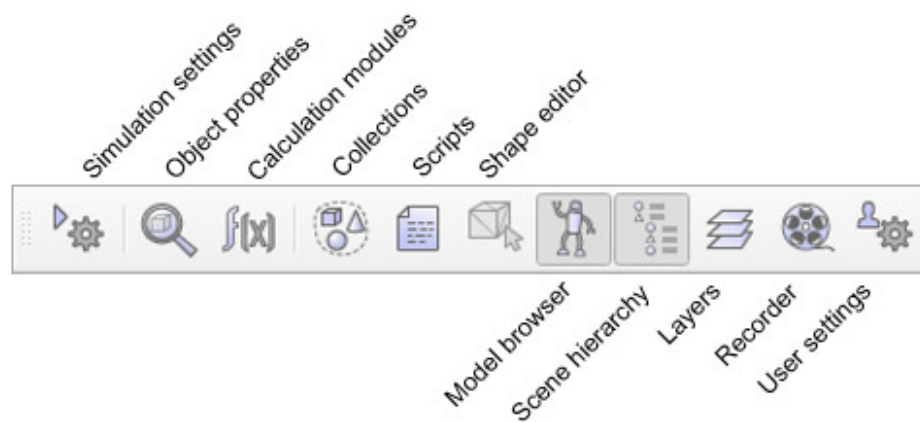
Fonte: O autor (2025).

Na Figura A.3 há uma barra de ferramentas com um menu que permite alguns comandos como: ir nas configurações de cena, propriedade dos objetos, coleções dentre outros.

- *Simulation settings*: Configurações de simulação;

- *Object properties*: Propriedades do objeto;
- *Calculation modules*: Módulos de cálculo;
- *Collections*: Coleções;
- *Scripts*: Scripts;
- *Shape editor*: Editor de formas;
- *Model browser*: Navegador de modelos;
- *Scene hierarchy*: Hierarquia da cena;
- *Layers*: Camadas;
- *Recorder*: Gravador;
- *User settings*: Configurações do usuário.

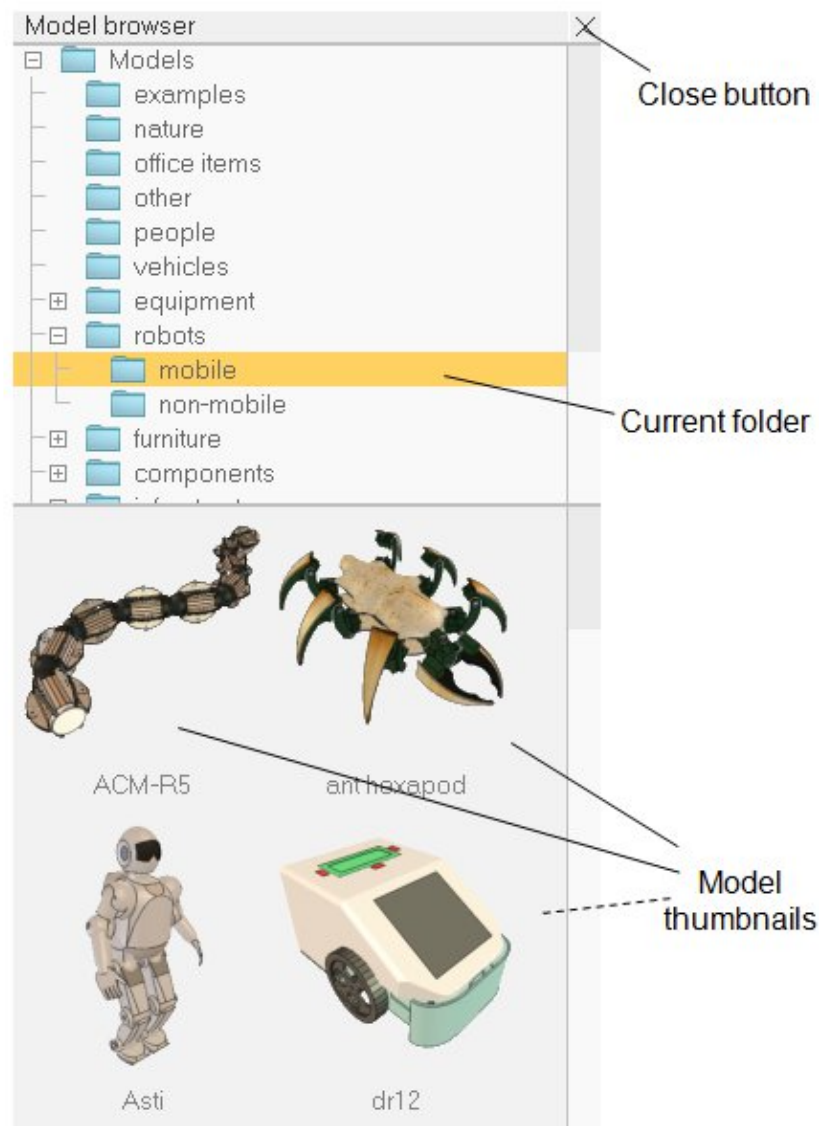
Figura A.3 – Continuação do menu de opções do simulador CoppeliaSim.



A Figura A.4 representa a hierarquia da cena lista todos os objetos em estrutura de árvore. Duplo clique em ícones ou aliases ativa a edição ou visualização de propriedades. A navegação permite múltiplas seleções e organização dos elementos com relação de hierarquia pai-filho.

- *Close button*: Botão de fechar;
- *Current folder*: Pasta atual;
- *Model thumbnails*: Miniatura de modelos.

Figura A.4 – Continuação do menu de opções do simulador CoppeliaSim.

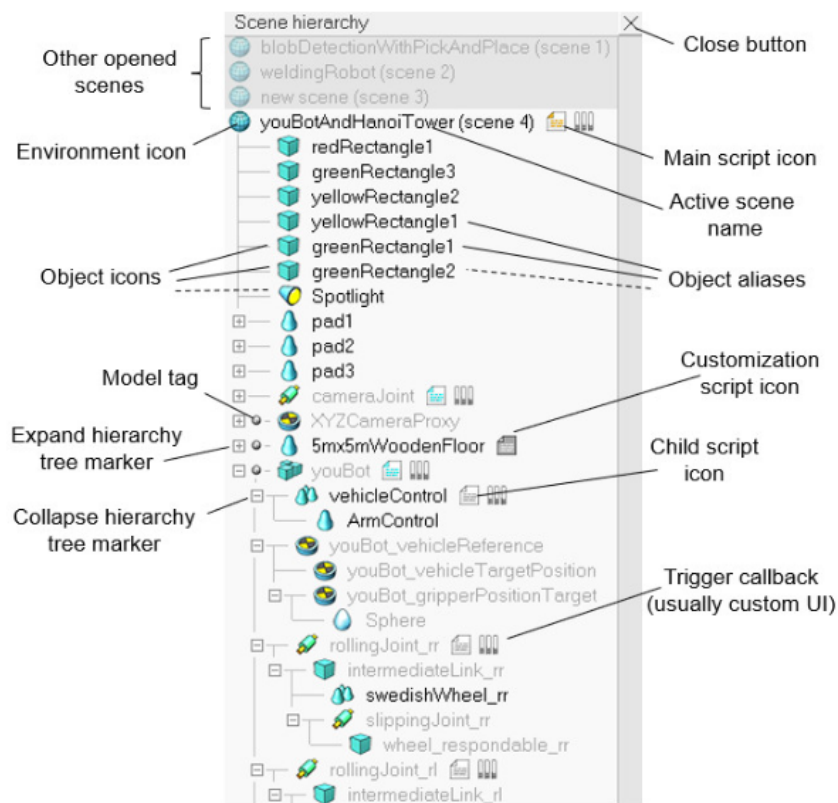


A Figura A.5 consta a hierarquia de objetos como um todo, nela é possível acessar cada componente individualmente e analisar onde ele se encontra na hierarquia.

- *Simulation settings*: Configurações de simulação;
- *Other opened scenes*: Outras cenas abertas;
- *Environment icon*: Ícone de ambiente;
- *Object icons*: Ícones de objetos;
- *Model tag*: Tag de modelo;
- *Expand hierarchy tree marker*: Marcador de expandir árvore de hierarquia;

- *Collapse hierarchy tree marker*: Marcador de recolher árvore de hierarquia;
- *Close button*: Botão de fechar;
- *Main script icon*: Ícone de script principal;
- *Active scene name*: Nome da cena ativa;
- *Object aliases*: Aliases de objetos;
- *Customization script icon*: Ícone de script de personalização;
- *Child script icon*: Ícone de script de filho;
- *Trigger callback (usually custom UI)*: Chamada de retorno de gatilho (UI personalizada).

Figura A.5 – Continuação do menu de opções do simulador CoppeliaSim.



Fonte: Adaptado de [CoppeliaSim \(2024\)](#).

APÊNDICE B – Modelagem UR5 estático

Este apêndice visa mostrar o código de modelagem estática do UR5, onde o robô se conecta com CoppeliaSim via API remota, com o objetivo de ler e exibir as posições e orientações relativas de cada junta do robô UR5, obtendo os parâmetros geométricos da configuração do robô, além da posição final do atuador, ele salva esses dados no arquivo: `.mat` na linha 150.

Código B.1 – Código modelagem com o UR5 estático

```

1 clear, clc, close all
2
3 disp('Programa iniciado');
4 sim = remApi('remoteApi');
5
6 sim.simxFinish(-1);
7 clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5);
8
9 if (clientID > -1)
10     disp('Conectado ao servidor da API remota');
11 else
12     disp('Nao conectado ao servidor da API remota');
13     return;
14 end
15
16 %% Conexao
17
18 [errorCode, ref] = sim.simxGetObjectHandle(clientID,'robotRef',
19     sim.simx_opmode_blocking);
20 [errorCode, j1] = sim.simxGetObjectHandle(clientID,'UR5_joint1',
21     sim.simx_opmode_blocking);
22 [errorCode, j2] = sim.simxGetObjectHandle(clientID,'UR5_joint2',
23     sim.simx_opmode_blocking);
24 [errorCode, j3] = sim.simxGetObjectHandle(clientID,'UR5_joint3',
25     sim.simx_opmode_blocking);
26 [errorCode, j4] = sim.simxGetObjectHandle(clientID,'UR5_joint4',
27     sim.simx_opmode_blocking);
28 [errorCode, j5] = sim.simxGetObjectHandle(clientID,'UR5_joint5',
29     sim.simx_opmode_blocking);
30 [errorCode, j6] = sim.simxGetObjectHandle(clientID,'UR5_joint6',
31     sim.simx_opmode_blocking);
32 [errorCode, atuador] = sim.simxGetObjectHandle(clientID,'atuador',

```

```

sim.simx_opmode_blocking);
26
27
28
29 [returnCode, j1_ang] =
    sim.simxGetJointPosition(clientID, j1, sim.simx_opmode_blocking);
30 [returnCode, j2_ang] =
    sim.simxGetJointPosition(clientID, j2, sim.simx_opmode_blocking);
31 [returnCode, j3_ang] =
    sim.simxGetJointPosition(clientID, j3, sim.simx_opmode_blocking);
32 [returnCode, j4_ang] =
    sim.simxGetJointPosition(clientID, j4, sim.simx_opmode_blocking);
33 [returnCode, j5_ang] =
    sim.simxGetJointPosition(clientID, j5, sim.simx_opmode_blocking);
34 [returnCode, j6_ang] =
    sim.simxGetJointPosition(clientID, j6, sim.simx_opmode_blocking);
35 [returnCode, j7_ang] =
    sim.simxGetJointPosition(clientID, j6, sim.simx_opmode_blocking); %
    verificar
36
37
38 %% Parametros
39
40 ang_med = [j1_ang, j2_ang, j3_ang, j4_ang, j5_ang, j6_ang, j7_ang];
41
42
43 disp(' ')
44 disp(['Angulo das juntas: ', num2str(ang_med*180/pi)]) % converte o
    angulo radianos em graus e string
45 disp(' ')
46
47
48 [returnCode, j1_pos] = sim.simxGetObjectPosition(clientID, j1,
    ref, sim.simx_opmode_blocking);
49 [returnCode, j1_eul] = sim.simxGetObjectOrientation(clientID, j1,
    ref, sim.simx_opmode_blocking);
50 disp('1')
51 disp(['Posicao: ', num2str(j1_pos)])
52 disp(['Angulos de Euler: ', num2str(j1_eul*180/pi)])
53 disp(' ')
54
55
56 [returnCode, j2_pos] = sim.simxGetObjectPosition(clientID, j2, j1,
    sim.simx_opmode_blocking);
57 [returnCode, j2_eul] = sim.simxGetObjectOrientation(clientID, j2, j1,
    sim.simx_opmode_blocking);
58 disp('2')

```

```

59 disp(['Posicao: ', num2str(j2_pos)])
60 disp(['Angulos de Euler: ', num2str(j2_eul*180/pi)])
61 disp(' ')
62
63
64 [returnCode, j3_pos] = sim.simxGetObjectPosition(clientID, j3, j2,
    sim.simx_opmode_blocking);
65 [returnCode, j3_eul] = sim.simxGetObjectOrientation(clientID, j3, j2,
    sim.simx_opmode_blocking);
66 disp('3')
67 disp(['Posicao: ', num2str(j3_pos)])
68 disp(['Angulos de Euler: ', num2str(j3_eul*180/pi)])
69 disp(' ')
70
71
72 [returnCode, j4_pos] = sim.simxGetObjectPosition(clientID, j4, j3,
    sim.simx_opmode_blocking);
73 [returnCode, j4_eul] = sim.simxGetObjectOrientation(clientID, j4, j3,
    sim.simx_opmode_blocking);
74 disp('4')
75 disp(['Posicao: ', num2str(j4_pos)])
76 disp(['Angulos Euler: ', num2str(j4_eul*180/pi)])
77 disp(' ')
78
79 [returnCode, j5_pos] = sim.simxGetObjectPosition(clientID, j5, j4,
    sim.simx_opmode_blocking);
80 [returnCode, j5_eul] = sim.simxGetObjectOrientation(clientID, j5, j4,
    sim.simx_opmode_blocking);
81 disp('5')
82 disp(['Posicao: ', num2str(j5_pos)])
83 disp(['Angulos Euler: ', num2str(j5_eul*180/pi)])
84 disp(' ')
85
86
87 [returnCode, j6_pos] = sim.simxGetObjectPosition(clientID, j6, j5,
    sim.simx_opmode_blocking);
88 [returnCode, j6_eul] = sim.simxGetObjectOrientation(clientID, j6, j5,
    sim.simx_opmode_blocking);
89 disp('6')
90 disp(['Posicao: ', num2str(j6_pos)])
91 disp(['Angulos Euler: ', num2str(j6_eul*180/pi)])
92 disp(' ')
93
94
95 [returnCode, j7_pos] = sim.simxGetObjectPosition(clientID, atuador,
    j6, sim.simx_opmode_blocking);
96 [returnCode, j7_eul] = sim.simxGetObjectOrientation(clientID, atuador,

```

```

        j6, sim.simx_opmode_blocking);
197 disp('Atuador - Relativo')
198 disp(['Posicao: ', num2str(j7_pos)])
199 disp(['Angulos Euler: ', num2str(j7_eul*180/pi)])
200 disp(' ')
201
202
203
204 [returnCode, actuator_pos] = sim.simxGetObjectPosition(clientID,
    atuador, ref, sim.simx_opmode_blocking);
205 [returnCode, actuator_eul] = sim.simxGetObjectOrientation(clientID,
    atuador, ref, sim.simx_opmode_blocking);
206 disp('Atuador - Base')
207 disp(['Posicao: ', num2str(actuator_pos)])
208 disp(['Angulos Euler: ', num2str(actuator_eul*180/pi)])
209 disp(' ')
210
211
212
213 disp('Posicao Relativa')
214 % Posicoes relativas
215 pos = [ j1_pos;
216         j2_pos;
217         j3_pos;
218         j4_pos;
219         j5_pos;
220         j6_pos;
221         j7_pos];
222
223 for i=1:7
224     fprintf("\t%.10f, \t%.10f, \t%.10f\n", pos(i,1), pos(i,2), pos(i,3))
225 end
226 disp(' ')
227
228
229 disp('Angulacao relativa')
230 % Angulacoes relativas
231 ang = [ j1_eul;
232         j2_eul;
233         j3_eul;
234         j4_eul;
235         j5_eul;
236         j6_eul;
237         j7_eul]*180/pi;
238
239
240 pos = double(pos);

```

```
141 ang = double(ang);
142
143 for i=1:7
144     fprintf("\t%.1f, \t%.1f, \t%.1f\n", ang(i,1), ang(i,2), ang(i,3))
145 end
146 disp(' ')
147
148
149 save("UR5_PosAng.mat", "pos","ang")
```

- Da linha 1 á linha 15 é feito o processo de inicialização e conexão do MATLAB com CoppeliaSim.

- O trecho da linha 19 á linha 26 é feita a requisição dos *Handles* dos componentes do UR5, é usado a função `sim.simxGetObjectHandle` e já se obtém o retorno na variável correspondente.

- O trecho da linha 30 á linha 36 é feita a requisição da posição angular das juntas dos componentes do UR5, é usado a função `sim.simxGetJointPosition` e já obtém o retorno na variável correspondente (em radianos).

- Na linha 41 cria um vetor com os ângulos das juntas.

- Posteriormente na linha 45 converte os ângulos de radianos para graus e exhibe.

- Na linha 49 até a linha 54 é retornado a posição da junta em relação ao *frame* de referência e retorna a orientação da junta em ângulos de Euler, depois converte para graus. São usadas as funções: `sim.simxGetJointPosition` e `sim.simxGetObjectOrientation`.

- O processo se repete para o resto do robô sempre em relação ao *frame* anterior.

- No atuador o processo é feito em relação a dois referenciais, que é em relação a junta 6 e em relação ao sistema base.

- Na linha 114 cria a matriz `pos` com as posições relativas das juntas.

- No trecho da linha 124 á 127 usa `fprintf` para imprimir cada posição com 10 casas decimais.

- O processo é semelhante para os ângulos relativos onde são convertidos para graus.

- Os dados são salvos num arquivo de dados por meio do comando `save` do MATLAB.

APÊNDICE C – Código de obtenção de Matrizes Homogêneas

O presente apêndice apresenta o Código C.1 que implementa o procedimento de obtenção da matriz de transformação homogênea do robô UR5, a partir de medições diretas no ambiente simulado do CoppeliaSim.

Este procedimento fundamenta a construção da estrutura cinemática do robô, sendo a base para a análise de movimentos, transformações entre sistemas de coordenadas e implementação de otimização.

Código C.1 – Código obtenção matriz homogênea

```

1 % Obtencao da Matriz T a partir dos parametros de posicao relativa
2 % obtidas com as medidas internas do proprio software
3 clc, clear, close all
4
5 load('UR5_PosAng.mat')
6
7 pos_rel = [pos, ang];
8 % pos_rel = round([pos, ang],6)
9
10
11 T_static = [];
12
13 for i = 1:size(pos_rel,1)
14     i
15     % Calcula a matriz homogenea usando um vetor p e angulos phi, theta.
16     % psi
17     p = pos_rel(i,1:3);
18     ang = pos_rel(i,4:6);
19     T = calc_T_pos_ang(p, ang*pi/180)
20     T_static(:,:,i) = T;
21
22     [p, alpha] = calc_T_extract_pos_ang(T);
23
24     disp(num2str(pos_rel(i,:),6))
25     disp(num2str([p, alpha*180/pi],6))
26     % pos_rel2 = [p, alpha*180/pi]
27 end
28
29 % save("UR5_T_static.mat", "T_static")

```

- Inicializa o ambiente MATLAB com `clc`, `clear` e `close all`.
- Carrega o arquivo de dados que foi salvo no Apêndice B, contendo posições (x, y, z) e orientações (Roll, Pitch, Yaw) relativas entre os elos, em relação ao anterior.
- Os dados carregados são armazenados na matriz `pos_rel`, onde cada linha corresponde a um elo do robô e as colunas contêm os parâmetros da sub matriz de posição e o vetor de orientação.
- Cria a variável `T_static`, responsável por armazenar as matrizes homogêneas 4×4 que representam as transformações entre os elos.
- Para cada linha de `pos_rel`, o código:
 - Separa os vetores de posição e orientação (`p`, `ang`);
 - Converte os ângulos para radianos;
 - Calcula a matriz homogênea com uma função no Apêndice G;
 - Armazena o resultado em `T_static`;
- Realiza a extração da posição e dos ângulos a partir da matriz homogênea com a função de extração do Apêndice G para validação.
- Ao final, é possível salvar as matrizes com o comando `save` (opcional).

APÊNDICE D – Código de validação matriz estática

Esse apêndice apresenta o código cujo objetivo é validar as matrizes de transformação homogênea T , ao longo da cadeia cinemática do robô UR5 utilizando as posições e orientações (ângulos de Euler) relativas entre os elos, obtidas da simulação estática do robô UR5. A função principal usada aqui consta no Apêndice G, que calcula a matriz de transformação homogênea a partir de uma posição e orientação. O código faz isso para cada elo do robô, acumulando as transformações para obter a posição e orientação global do robô em cada configuração fornecida pelos dados da simulação.

Código D.1 – Código de validação T estática

```

1  % Objetiva validar a determinacao das matrizes de transformacao T,
    usando
2  % as coordenadas homogeneas
3  clear, clc, close all
4
5
6  load('UR5_PosAng.mat')
7
8  pos_rel = round([pos, ang],6)
9
10 tam = size(pos_rel,1);
11
12
13 T = zeros(4,4,tam);
14
15 p0 = [0, 0, 0, 1]';
16 px = [1, 0, 0, 1]';
17 py = [0, 1, 0, 1]';
18 pz = [0, 0, 1, 1]';
19
20
21 for i = 1:tam
22     pos_i = pos_rel(i,1:3);
23     ang_i = pos_rel(i,4:6)*pi/180;
24
25     T(:, :, i) = calc_T_pos_ang(pos_i, ang_i);
26
27     Ti = eye(4);
28     for k = 1:i
29         Ti = Ti*T(:, :, k);

```

```

30     end
31
32     i
33
34     Ti
35
36     % Conversao para o sistema global de posicao
37     P0 = (Ti*p0)';
38     Px = (Ti*px)';
39     Py = (Ti*py)';
40     Pz = (Ti*pz)';
41
42     pause
43 end

```

- Os dados são carregados a partir do arquivo de dados no Apêndice B na linha 8, contendo as variáveis `pos` e `ang`, extraídas do CoppeliaSim.

- Na linha 10 a matriz `pos_rel` armazena essas informações com 6 casas decimais, onde cada linha representa uma junta ou o atuador do robô, no formato: $[x, y, z, \psi, \theta, \phi]$.

- Na linha 23 armazena na variável `tam` o número de linhas da matriz `pos_rel`, ou seja, a quantidade total de elementos (juntas e atuador) do robô a serem processados no laço.

- No trecho das linhas 26 á 31 é criado as matrizes 4×4 para cada junta. As demais linhas são pontos com os eixos no espaço global.

- O laço `for` percorre cada uma das juntas para calcular a matriz de transformação homogênea T com base na função de cálculo da T do arquivo que consta no Apêndice G, que converte a posição e orientação em uma matriz 4×4 .

- A matriz $T(:, :, i)$ representa a transformação relativa da i -ésima junta. A matriz acumulada T_i é calculada iterativamente para obter a posição da junta no sistema de coordenadas global.

- Pontos de referência do sistema local `p0`, `px`, `py` e `pz` são transformados para o sistema global por meio da multiplicação com a matriz T_i , possibilitando a visualização da orientação de cada junta.

APÊNDICE E – Código de validação T dinâmica

Esse código MATLAB realiza a validação da matriz de transformação homogênea T de forma dinâmica da cinemática do UR5. Ele verifica se a matriz T calculada com base em modelos teóricos (usando cinemática direta) está coerente com os dados reais de posição e orientação do robô obtidos na simulação. Isso é essencial para garantir que o modelo matemático do robô esteja correto.

Código E.1 – Código de validação T dinâmica

```

1
2 clear, clc, close all
3
4 disp('Programa iniciado');
5 sim = remApi('remoteApi');
6
7 sim.simxFinish(-1);
8 clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5);
9
10 if (clientID > -1)
11     disp('Conectado ao servidor API remota');
12 else
13     disp('Nao conectado ao servidor API remota');
14     return;
15 end
16
17
18 [errorCode, base] = sim.simxGetObjectHandle(clientID,'robotRef' ,
19     sim.simx_opmode_blocking);
20 [errorCode, j1] = sim.simxGetObjectHandle(clientID,'UR5_joint1',
21     sim.simx_opmode_blocking);
22 [errorCode, j2] = sim.simxGetObjectHandle(clientID,'UR5_joint2',
23     sim.simx_opmode_blocking);
24 [errorCode, j3] = sim.simxGetObjectHandle(clientID,'UR5_joint3',
25     sim.simx_opmode_blocking);
26 [errorCode, j4] = sim.simxGetObjectHandle(clientID,'UR5_joint4',
27     sim.simx_opmode_blocking);
28 [errorCode, j5] = sim.simxGetObjectHandle(clientID,'UR5_joint5',
29     sim.simx_opmode_blocking);
30 [errorCode, j6] = sim.simxGetObjectHandle(clientID,'UR5_joint6',
31     sim.simx_opmode_blocking);
32 [errorCode, atuador] = sim.simxGetObjectHandle(clientID,'atuador' ,

```

```

sim.simx_opmode_blocking);
26
27
28 %% Avalia o calculo do teorico e pratico da matriz T
29
30 load("UR5_T_static.mat");
31
32 WP_q      =      [      0,      0,      0,      0,      0,      0;
33                    10,     15,    -10,    -15,    -20,     10;
34                    80,     30,     45,     50,     20,    100;
35                   -100,    -50,    -30,      0,    100,     -5;
36                   -10,     30,     35,     25,     15,     50;
37                   150,     75,     20,     10,     -5,     15;
38                    0,      0,      0,      0,      0,      0];
39
40
41 % Eixo em torno do qual cada angulo de q rotaciona
42 u_i_static = [ 0, 0, 1;
43                0, 0, 1;
44                0, 0, 1;
45                0, 0, 1;
46                0, 0, 1;
47                0, 0, 1];
48
49 tam = size(WP_q,1);
50
51 WP_3D_sim = [];
52 WP_3D_calc = [];
53
54 for i=1:tam
55     disp(i)
56     %% Definindo o espaco de configuracao
57     qi = WP_q(i,:);
58     disp(['Angulos: ', num2str(qi)])
59     qi = qi*pi/180;
60
61     sim.simxSetJointTargetPosition(clientID, j1, qi(1),
62                                     sim.simx_opmode_blocking);
63     sim.simxSetJointTargetPosition(clientID, j2, qi(2),
64                                     sim.simx_opmode_blocking);
65     sim.simxSetJointTargetPosition(clientID, j3, qi(3),
66                                     sim.simx_opmode_blocking);
67     sim.simxSetJointTargetPosition(clientID, j4, qi(4),
68                                     sim.simx_opmode_blocking);
69     sim.simxSetJointTargetPosition(clientID, j5, qi(5),
70                                     sim.simx_opmode_blocking);
71     sim.simxSetJointTargetPosition(clientID, j6, qi(6),

```

```

        sim.simx_opmode_blocking);
67
68     err_ang = 1;
69     ang_med = 0;
70
71     maxIter = 25;
72     kIter = 0;
73     while(norm(err_ang) >= 1e-1)
74         [returnCode, j1_ang] =
            sim.simxGetJointPosition(clientID, j1, sim.simx_opmode_blocking);
75         [returnCode, j2_ang] =
            sim.simxGetJointPosition(clientID, j2, sim.simx_opmode_blocking);
76         [returnCode, j3_ang] =
            sim.simxGetJointPosition(clientID, j3, sim.simx_opmode_blocking);
77         [returnCode, j4_ang] =
            sim.simxGetJointPosition(clientID, j4, sim.simx_opmode_blocking);
78         [returnCode, j5_ang] =
            sim.simxGetJointPosition(clientID, j5, sim.simx_opmode_blocking);
79         [returnCode, j6_ang] =
            sim.simxGetJointPosition(clientID, j6, sim.simx_opmode_blocking);
80
81         ang_med = [j1_ang, j2_ang, j3_ang, j4_ang, j5_ang, j6_ang];
82
83         err_ang = (ang_med - qi)*180/pi;
84
85         kIter = kIter + 1;
86         if kIter >= maxIter
87             disp('Max Iteration limit')
88             break
89         end
90     end
91
92     % % % Obtem a posicao e atitude do atuador em relacao a base
93
94     [returnCode, sim_pos] = sim.simxGetObjectPosition(clientID,
        atuador, base, sim.simx_opmode_blocking);
95     [returnCode, sim_ang] = sim.simxGetObjectOrientation(clientID,
        atuador, base, sim.simx_opmode_blocking);
96     sim_pos = double(sim_pos);
97     sim_ang = double(sim_ang);
98
99     disp(['Sim Posicao: ', num2str(sim_pos,6)])
100     disp(['Sim Angulos de Euler: ', num2str(sim_ang*180/pi)])
101
102
103     % % % Calcula os valores teoricos
104

```

```

105 % Calcula a UTM para cada junta
106 T_juntas = calc_T_dynamic(T_static(:,:,1:6), qi, u_i_static);
107
108 % Adiciona o efeito do atuador
109 T_final = T_juntas*T_static(:,:,7)
110
111 % Extrai a posicao e a orientacao do atuador no referencial do robo
112
113 % [position, eulerAngles] = calc_T_extract_pos_ang(T_final);
114 [u,angle,p] = calc_T_extract_u_angle(T_final);
115
116 T2 = calc_T_u_angle(angle,u,p)
117
118
119
120 pause
121 end
122
123 %%
124
125 WP_3D_sim
126 WP_3D_calc

```

- As linhas 1 até a 15 são para fins de conexão ao servidor.
- As linhas 18 á 25 são para se obter o identificador único de cada componente que é melhor explicado no Apêndice C.
- Na linha 30 é carregado o arquivo de dados da matriz estática no Apêndice C em que contém matrizes de transformação homogênea pré-calculadas que correspondem à configuração estática do robô.
- Na linha 32 é definido uma matriz WP_q, que representa diferentes configurações articulares desejadas os waypoints.
- Na linha 50 esse trecho define a matriz u_i_static, que especifica o eixo de rotação de cada junta do robô UR5 no seu referencial local, no caso todas em z.
- Na linha 57 pega o número de linhas da matriz WP_q e armazena na variável tam.
- Nas linhas 59 e 60 inicializa duas matrizes vazias.
- Da linha 62 até a 98 é criado um laço que percorre todas as configurações de junta, pega a i-ésima linha dos WP_q. Converte os ângulos de graus para radianos.
- Com a função sim.simxSetJointTargetPosition move cada junta do robô para a posição desejada. Inicializa algumas variáveis nas linhas 76 á 80.
- Usa um loop while para aguardar até que as juntas realmente atinjam os ângulos

desejados dentro da margem de erro.

- Solicita as posições atuais das juntas no simulador. Calcula o erro entre o que foi pedido q_i e o que foi realmente alcançado ang_med , o erro é convertido para graus só para fins de leitura/impressão. Se não atingir a posição correta em até 25 iterações, ele interrompe o `loop` e avisa.

- O trecho das linhas 100 a 108 está pegando a posição e orientação real do atuador final do robô em relação à base, diretamente do CoppeliaSim. Ele serve para validar os resultados da cinemática direta e comparar com os valores calculados teoricamente.

- A linha 114 chama a função de cálculo da T dinâmica do Apêndice G, que calcula as matrizes de transformação homogênea de cada junta do robô usando: configuração atual das juntas q_i , os eixos de rotação das juntas u_i_static e as transformações fixas T_static da estrutura do robô.

- Na linha 117 a variável T_juntas representa a transformação homogênea acumulada desde a base até a última junta do UR5, já considerando as rotações dinâmicas das juntas. A matriz $T_static(:, :, 7)$ é a transformação homogênea fixa entre a última junta e o atuador final (como a garra ou ferramenta), incluindo deslocamentos estruturais constantes. Assim, T_final representa a posição e orientação do atuador final em relação à base do robô, e pode ser usada para comparação com a posição simulada no CoppeliaSim.

- Na linha 122 essa função extrai três informações principais da matriz de transformação homogênea T_final : u eixo de rotação (vetor unitário), ângulo de rotação (em rad), vetor posição (translação).

- Na linha 124 reconstrói a matriz de transformação homogênea T_2 a partir dos seguintes parâmetros: u eixo unitário de rotação, ângulo de rotação, e p vetor de translação.

- Nas linhas `WP_3D_sim`: as posições 3D obtidas via simulação no CoppeliaSim e `WP_3D_calc`: as posições 3D calculadas teoricamente com base na cinemática direta.

APÊNDICE F – Código de Otimização

Esse apêndice tem a finalidade de definir um conjunto de *waypoints* (pontos de referência) no espaço de configuração (ângulos das juntas do UR5), a ideia é calcular a posição e orientação correspondente via cinemática direta, e utilizar métodos de cinemática inversa para encontrar os ângulos que levam a essa posição e orientação, finalizando de forma que possa se comparar os resultados calculados com os de referência.

Código F.1 – Código de otimização

```

1
2 clear, clc, close all
3
4 load("UR5_T_static.mat");
5 % load("UR5_WayPoints.mat");
6
7 wp_q_ref      =      [      0,      0,      0,      0,      0,      0;
8                      10,     15,    -10,    -15,    -20,     10;
9                      80,     30,     45,     50,     20,    100;
10                     -100,   -50,   -30,      0,    100,    -5;
11                     -10,     30,     35,     25,     15,     50;
12                      15,      0,     35,   -50,   -150,   -90;
13                     150,     75,     20,     10,     -5,     15;
14                      0,      0,      0,      0,      0,      0];
15
16
17 tam = size(wp_q_ref,1);
18
19 % 0 eixo em torno de cada angulo que q gira
20 u_static = [ 0, 0, 1;
21             0, 0, 1;
22             0, 0, 1;
23             0, 0, 1;
24             0, 0, 1;
25             0, 0, 1];
26
27
28 q0 = zeros(1,6);
29
30 wp_q_opt      = [];
31 wp_3d_calc     = [];
32
33 tic
34 for i = 1:tam
35     % Obtenha a referencia

```

```

36     q_ref = wp_q_ref(i,:);
37     T_ref = calc_T_dynamic(T_static,q_ref*pi/180,u_static);
38     [u_ref, ang_ref, pos_ref] = calc_T_extract_u_angle(T_ref);
39
40     % Armazena os valores calculados dos waypoints em 3D
41     wp_3d_calc = [wp_3d_calc;
42                  pos_ref', rad2deg(ang_ref'), u_ref'];
43
44     q_star = ur5_IK_a_nonlinear(T_ref,T_static,q0);
45     % q_star = ur5_IK_b_fmincon(T_ref,T_static,q0);
46     % q_star = ur5_IK_c_partswarm(T_ref,T_static,q0);
47     % q0 = q_star;
48
49     % Waypoints otimizados no espaco de configuracao do robo
50     wp_q_opt = [wp_q_opt; q_star];
51
52     disp(' ')
53     disp(['q_ref : ',num2str(q_ref)])
54     disp(['q_star: ',num2str(q_star)])
55 end
56 toc
57
58 %%
59
60 wp_q_ref
61 wp_q_opt
62
63 [wp_3d_opt, wp_3d_ref] = follow_waypoints_two(wp_q_opt,wp_q_ref, false);
64
65
66 wp_3d_opt
67 wp_3d_ref

```

- Na linha 4 carrega o arquivo de dados da T estática no Apêndice C carregando a matriz `T_static`, que contém as transformações homogêneas fixas.

- Na linha 19 atribui a matriz `wp_q_ref` um conjunto de waypoints no espaço de configuração. Cada linha é uma posição que o robô deve alcançar.

- Na linha 29 define a quantidade de waypoints a serem processados na variável `tam`.

- Na linha 32 é criado vetores de rotação fixos para cada junta no eixo z .

- Na linha 40 q_0 é a configuração inicial (todos os ângulos em zero).

- Na linha 42 `wp_q_opt` contém os ângulos otimizados.

- Na linha 43 `wp_3d_calc` são posições calculadas a partir da cinemática direta

serão usadas para comparar com a posição e orientação obtida via cinemática inversa.

- Dentro do laço a linha 48 pega a i -ésima linha da matriz `wp_q_ref`, que contém os ângulos das juntas do robô em graus, a linha 49 aplica cinemática direta com a função de cálculo da T dinâmica no Apêndice G, na linha 50 se extrai da matriz `T_ref` três componentes, o vetor posição do atuador, o ângulo de rotação e o eixo de rotação que define a orientação.

- As linhas 53 e 54 armazena em graus e coordenadas cartesianas, a (posição e orientação) do atuador calculada via cinemática direta a estrutura será salva da seguinte forma `[x, y, z, ângulo-em-graus, ux, uy, uz]`.

- Na linha 58 resolve a cinemática inversa do robô UR5 usando um algoritmo de otimização *lsqnonlinear*, sendo `T_ref` a posição desejada (matriz de transformação 4x4) do atuador, calculada com cinemática direta;

- `T_static` contém as transformações fixas entre os elos;

- `q0` é a configuração inicial usada como palpite para iniciar a busca;

- `q_star` é o resultado da otimização, ou seja, os ângulos das juntas que fazem o robô alcançar (ou chegar bem próximo) da posição desejada `T_ref`.

- Na linha 62 armazena os valores de juntas calculados pela cinemática inversa `q_star` em uma matriz acumuladora chamada `wp_q_opt`.

- Exibi nas linhas 65 e 66 os valores das juntas de referência e os calculados através da cinemática inversa.

- Nas linhas 72 a 79 é feita comparação compara o que se queria com o que foi realmente alcançado no espaço cartesiano.

APÊNDICE G – Principais funções utilizadas

Esse apêndice tem a finalidade de detalhar as principais funções utilizadas nos arquivos, devido a importância na sua utilização no projeto.

Código G.1 – Função que calcula a matriz de transformação homogênea dada um vetor de posição p e um vetor de rotação α (ângulos de Euler)

```

1
2 function [T] = calc_T_pos_ang(p, alpha)
3
4     px = p(1);
5     py = p(2);
6     pz = p(3);
7
8     % Extrai os angulos de Euler (roll, pitch, yaw)
9     phi = alpha(1);    % Roll (rotacao ao redor do eixo x)
10    theta = alpha(2);  % Pitch (rotacao ao redor do eixo y)
11    psi = alpha(3);    % Yaw (rotacao ao redor do eixo z)
12
13    % % Matriz de rotacao em torno do eixo x (roll)
14    Rx = [1, 0, 0;
15          0, cos(phi), -sin(phi);
16          0, sin(phi), cos(phi)];
17
18    % % Matriz de rotacao em torno do eixo y (pitch)
19    Ry = [cos(theta), 0, sin(theta);
20          0, 1, 0;
21          -sin(theta), 0, cos(theta)];
22
23    % % Matriz de rotacao em torno do eixo z (yaw)
24    Rz = [cos(psi), -sin(psi), 0;
25          sin(psi), cos(psi), 0;
26          0, 0, 1];
27
28    % % Matriz de rotacao combinada R = Rz * Ry * Rx
29    R = Rz * Ry * Rx;
30
31    % % Constroi a matriz de transformacao homogenea
32
33    T = [R, [px; py; pz];
34          0, 0, 0, 1];
35 end

```

Explicação da função G.1.

- As linhas 4 a 6 extraem os componentes do vetor de posição p , armazenando-os nas variáveis px , py e pz , correspondendo às coordenadas x, y e z.
 - As linhas 9 a 11 extraem os ângulos de Euler **phi**, **theta** e **psi** do vetor **alpha**, que representam as rotações em torno dos eixos x (roll), y (pitch) e z (yaw), respectivamente.
 - As linhas 14 a 16 constroem a matriz de rotação **Rx**, referente à rotação em torno do eixo x.
 - As linhas 20 a 22 constroem a matriz de rotação **Ry**, referente à rotação em torno do eixo y.
 - As linhas 24 a 27 constroem a matriz de rotação **Rz**, referente à rotação em torno do eixo z.
 - A linha 30 calcula a matriz de rotação composta **R**, através da multiplicação das matrizes na ordem **Rz * Ry * Rx**, seguindo a convenção de ângulos de Euler ZYX (yaw-pitch-roll).
 - As linhas 34 e 35 constroem a matriz de transformação homogênea **T**, combinando a matriz de rotação **R** com o vetor de posição [**px**; **py**; **pz**] e a última linha [0 0 0 1].
- Código G.2 – Função que calcula a matriz de transformação homogênea a partir de um vetor de posição, eixo de rotação e ângulo de rotação

```
1
2 function T = calc_T_u_angle(angle, axis, p)
3
4     if ~exist('p','var')
5         p = zeros(3,1);
6     elseif length(p) ~= 3
7         error('0 vetor de posicao p deve ser um vetor 3x1.');
```

```
8     end
9
10    if length(axis) ~= 3
11        error('0 eixo de rotacao deve ser um vetor 3x1.');
```

```
12    end
13
14    %% Garantir que o eixo seja um vetor unitario
15
16    axis = axis / norm(axis);
17
18    %% Extrair componentes do eixo
19
20    ux = axis(1);
21    uy = axis(2);
```

```

22     uz = axis(3);
23
24     %% Definir a matriz assimetrica (skew-symmetric) de u
25
26     U_cross = [    0, -uz,  uy;
27                 uz,    0, -ux;
28                 -uy,  ux,    0 ];
29
30     %% Calcula a matriz de rotacao usando a formula de rotacao de
31         Rodrigues
32
33     R = eye(3) + sin(angle) * U_cross + (1 - cos(angle)) * (U_cross *
34         U_cross);
35
36     %% Construir a matriz de transformacao homogenea
37
38     T = [R, p(:); % p(:) % Garantir que p seja um vetor coluna
39
40         0, 0, 0, 1];
41
42 end

```

Explicação da função G.2.

- A linha 2 define a função `calc_T_u_angle`, que recebe como entrada um ângulo de rotação (`angle`), um eixo de rotação (`axis`) e um vetor de posição (`p`), e retorna a matriz de transformação homogênea `T`.

- As linhas 4 a 8 verificam se o vetor de posição `p` foi fornecido. Se não, ele é definido como o vetor nulo `[0; 0; 0]`. Também é verificado se o vetor tem tamanho 3.

- As linhas 10 a 12 verificam se o vetor `axis` é um vetor 3x1. Se não for, a função gera um erro.

- A linha 16 normaliza o vetor `axis`, garantindo que ele seja unitário, essencial para a correta aplicação da fórmula de Rodrigues.

- As linhas 20 a 22 extraem os componentes `ux`, `uy` e `uz` do vetor de rotação unitário.

- As linhas 26 a 28 definem a matriz assimétrica *skew-symmetric* `U_cross`, construída a partir dos componentes de `axis`. Essa matriz representa o produto vetorial.

- A linha 32 aplica a fórmula de Rodrigues para calcular a matriz de rotação `R`, utilizando o ângulo fornecido e a matriz `U_cross`.

- As linhas 36 e 38 constroem a matriz de transformação homogênea `T`, combinando a matriz de rotação `R` com o vetor de translação `p`, garantindo que `p` seja coluna com `p(:)` e completando com a última linha homogênea `[0 0 0 1]`.

Código G.3 – Função que extrai o eixo de rotação, o ângulo de rotação e o vetor de posição

```
1
2 function [axis, angle, p] = calc_T_extract_u_angle(T)
3
4     if size(T, 1) ~= 4 || size(T, 2) ~= 4
5         error('A matriz de entrada T deve ser uma matriz de
6             transformacao homogenea 4x4.');
```

6

```
7     end
8
9     %% Extrair o vetor de posicao da ultima coluna de T
10
11     p = T(1:3, 4);
12
13     %% Extrair a matriz de rotacao do bloco 3x3 no canto superior
14     %% esquerdo de T
15
16     R = T(1:3, 1:3);
17
18     %% Calcular o angulo de rotacao a partir do traco de R
19
20     angle = acos((trace(R) - 1) / 2);
21
22     %% Tratar casos especiais para o angulo
23
24     if abs(angle) < 1e-6
25         %% Angulo aproximadamente zero: sem rotacao, eixo arbitrario
26
27         axis = [1; 0; 0];
28     elseif abs(angle - pi) < 1e-6
29         %% Angulo aproximadamente pi: caso especial para o eixo
30         %% Extrair os elementos da diagonal
31
32         diagR = diag(R);
33         axis = sqrt((diagR + 1) / 2);
34
35         %% Ajustar os sinais com base nos elementos fora da diagonal
36
37         if R(1, 2) < 0, axis(2) = -axis(2); end
38         if R(1, 3) < 0, axis(3) = -axis(3); end
39     else
40         %% Caso geral: calcular o eixo a partir dos elementos fora da
41         %% diagonal
42
43         axis = [R(3, 2) - R(2, 3);
44             R(1, 3) - R(3, 1);
45             R(2, 1) - R(1, 2)] / (2 * sin(angle));
46     end
47 end
```

```

45 % Garantir que o eixo seja um vetor unitario
46
47 axis = axis / norm(axis);
48 end

```

Explicação da função G.3.

- A linha 2 define a função `calc_T_extract_u_angle`, que recebe uma matriz de transformação homogênea T (4x4) e retorna o eixo de rotação `axis`, o ângulo de rotação `angle` e o vetor de posição `p`.

- As linhas 3 a 6 validam se a matriz T tem tamanho 4x4. Caso contrário, a função gera um erro.

- A linha 10 extrai o vetor de posição `p` da última coluna da matriz T (primeiras três linhas da quarta coluna).

- A linha 14 extrai a submatriz 3x3 de rotação R do canto superior esquerdo da matriz T .

- A linha 18 calcula o ângulo de rotação `angle` a partir do traço (soma dos elementos diagonais) da matriz R , usando a fórmula inversa de Rodrigues.

- As linhas 22 a 25 tratam o caso especial em que o ângulo de rotação é aproximadamente zero. Nesse caso, não há rotação e o eixo é arbitrariamente definido como `[1; 0; 0]`.

- As linhas 26 a 31 tratam o caso em que o ângulo de rotação é aproximadamente π . A direção do eixo é calculada a partir dos elementos da diagonal de R , e os sinais são ajustados com base nos elementos fora da diagonal.

- As linhas 33 a 36 tratam o caso geral, calculando o eixo de rotação a partir dos elementos fora da diagonal da matriz R com base na fórmula de Rodrigues.

A linha 47 normaliza o vetor `axis` para garantir que ele seja unitário, o que é necessário para representar corretamente o eixo de rotação.

Código G.4 – Função que extrai o vetor de posição e os ângulos de Euler a partir de uma matriz de transformação homogênea

```

1
2 function [p, alpha] = calc_T_extract_pos_ang(T)
3
4     if size(T,1) ~= 4 || size(T,2) ~= 4
5         error('A matriz de entrada T deve ser uma matriz de
6             transformacao homogenea 4x4.');
```

```

9
10     p = T(1:3, 4);
11
12     % % Extrair a matriz de rotacao do bloco 3x3 no canto superior
        esquerdo de T
13
14     R = T(1:3, 1:3);
15
16     % alpha = compute_euler_angles_optimization(R);
17
18     % Calcular os angulos de Euler a partir da matriz de rotacao
19     % Usando a convencao ZYX: Rz * Ry * Rx
20
21     theta = real(-asin(R(3,1)));
22
23     % Verificar se ha bloqueio de cardan (singularidades)
24
25     if abs(cos(theta)) > 1e-6
26         % % Caso geral: sem bloqueio de cardan
27
28         phi = atan2(R(3,2), R(3,3)); % Roll
29         psi = atan2(R(2,1), R(1,1)); % Yaw
30     else
31         % % Caso de bloqueio de cardan
32         theta = +-pi/2
33         phi = atan2(-R(2,3), R(2,2)); % Roll
34         psi = 0; % 0 angulo de guinada (yaw) nao pode ser determinado,
            definido como 0
35
36     end
37
38     % Combina os angulos de Euler em um vetor de rotacao
39     alpha = [phi; theta; psi];
40 end

```

Explicação da função G.4.

- A linha 2 define a função `calc_T_extract_pos_ang`, que extrai o vetor de posição e os ângulos de Euler de uma matriz de transformação homogênea 4×4 .
- As linhas 4 a 6 verifica se a matriz de entrada `T` possui dimensão 4×4 . Caso contrário, exibe um erro.
- A linha 10 extrai o vetor de posição `p` a partir da última coluna da matriz `T`.
- A linha 14 extrai a matriz de rotação 3×3 do canto superior esquerdo de `T`.
- Os ângulos de Euler usando a convenção ZYX ($R_z * R_y * R_x$):

- Primeiro calcula-se o ângulo θ (pitch) a partir do elemento $R_{3,1}$.
- Em seguida, se verifica se ocorre bloqueio de cardan (singularidade), ou seja, se $\cos(\theta) \approx 0$.
- No caso geral (sem bloqueio), calcula-se:
 - ϕ (roll) a partir de $R_{3,2}$ e $R_{3,3}$,
 - ψ (yaw) a partir de $R_{2,1}$ e $R_{1,1}$.
- Em caso de bloqueio de cardan ($\theta = \pm \frac{\pi}{2}$), define-se:
 - ϕ (roll) a partir de $R_{2,3}$ e $R_{2,2}$,
 - $\psi = 0$ (o ângulo yaw não pode ser determinado com precisão).
- A linha 39 agrupa os ângulos ϕ, θ, ψ no vetor **alpha**, representando a orientação em ângulos de Euler.

Código G.5 – Função que calcula uma transformação global acumulada

```

1
2 function T_global = calc_T_dynamic(T_static, angles_of_rotation,
   rotation_axis)
3
4   T_global = eye(4);
5
6   for i = 1:length(angles_of_rotation)
7       T_dynamic = calc_T_u_angle(angles_of_rotation(i),
           rotation_axis(i,:));
8       T_global = T_global * T_static(:,:,i) * T_dynamic;
9   end
10
11 end

```

Explicação da função G.5.

- A linha 1 define a função **calc_T_dynamic**, que recebe uma matriz tridimensional **T_static** contendo transformações homogêneas estáticas, um vetor **angles_of_rotation** com os ângulos de rotação (um para cada junta) e uma matriz **rotation_axis**, onde cada linha representa o eixo de rotação de uma junta.
- A linha 4 inicializa a matriz identidade 4x4 **T_global**, que irá acumular as transformações aplicadas sucessivamente.
- A linha 6 contém um laço **for** que percorre cada ângulo de rotação fornecido.
- Em cada iteração, a função **calc_T_u_angle** é chamada para calcular a transformação dinâmica **T_dynamic** correspondente ao ângulo e eixo da junta atual.
- Em seguida, essa transformação dinâmica é multiplicada pela transformação

estática correspondente `T_static(:, :, i)`.

- O resultado é acumulado na variável `T_global`, multiplicando sequencialmente todas as transformações para obter a posição e orientação final.