UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

PROJETO FINAL

Desenvolvimento da Modelagem, Simulação e Projeto Mecânico de um manipulador robótico

Amanda Hellen Venâncio Reis – 13679287

Gabriel Calori Badini – 12676063

Gabrielle de Moura Pereira – 13828178

João Augusto Costa Perino – 13678967

DOCENTE:

Prof. Dr. Marcelo Becker

São Carlos - SP

Dezembro de 2025

SUMÁRIO

1 INTRODUÇÃO GERAL **4**

2 FUNCIONAMENTO DO MANIPULADOR **5**

3 DESENVOLVIMENTO DO PROJETO **8**

3.1 Modelagem matemática 8

3.1.1 Cinemática direta 8

3.1.2 Cinemática inversa 9

3.1.3 Jacobiano 10

3.1.4 Dinâmica 13

3.2 Projeto mecânico (CAD) 16

3.3 Simulação computacional 17

4 CONCLUSÃO **21**

5 REFERÊNCIAS BIBLIOGRÁFICAS **22**

LISTA DE FIGURAS

Figura 1: Identidade visual do manipulador. 5
Figura 2: Croqui do robô. 6
Figura 3: Exemplo de robô que realiza limpeza de painéis solares. 7
Figura 4: Cinemática direta final obtida para o manipulador. 9
Figura 5: Cinemática inversa final obtida para o manipulador. 10
Figura 6: Encontrando o Jacobiano. 12
Figura 7: Jacobiano final obtido para o manipulador. 12
Figura 8: Posições e velocidades do link 1. 14
Figura 9: Posições e velocidades do link 2. 15
Figura 10: Posições e velocidades do link 3. 15
Figura 11: Encontrando a matriz de inércia. 15
Figura 12: Matriz de inércia. 16
Figura 13: Forças e torques finais. 16
Figura 14: Imagem 1 do CAD do manipulador. 17
Figura 15: Imagem 2 do CAD do manipulador. 17
Figura 16: Perfis de posição, velocidade, aceleração e torque/força das juntas da cinemática direta até a posição [20° 20° 0.4]. 19
Figura 17: Perfis de posição, velocidade, aceleração e torque/força das juntas da cinemática inversa até a posição [0.4 0.4 -0,4]. 19
Figura 18: Perfis de posição, velocidade, aceleração e torque/força das juntas da trajetória de escovação. 20

1 INTRODUÇÃO GERAL

Este relatório apresenta o desenvolvimento de um manipulador robótico, abrangendo seu funcionamento, modelagem matemática, simulação computacional e projeto mecânico. O objetivo central é aplicar os conceitos estudados na disciplina de Dinâmica de Sistemas Robóticos.

O processo de desenvolvimento iniciou-se pela definição da configuração do manipulador, incluindo seus graus de liberdade (previamente estabelecidos pela orientação da disciplina) e o tipo de movimento desejado. A partir dessas decisões, foi elaborada a modelagem cinemática e dinâmica capaz de descrever o comportamento do robô e prever sua resposta frente a diferentes comandos. As simulações computacionais, por sua vez, possibilitaram validar esse comportamento, identificar ajustes necessários e analisar o desempenho do sistema em distintos cenários de operação.

Paralelamente, desenvolveu-se o projeto mecânico em ambiente CAD, assegurando que a estrutura concebida refletisse as características determinadas pela modelagem e atendesse às exigências funcionais do manipulador.

Ao longo do relatório, são detalhados os principais aspectos envolvidos no desenvolvimento do manipulador robótico, desde sua concepção até a validação de seu comportamento.

2 FUNCIONAMENTO DO MANIPULADOR

O manipulador desenvolvido recebeu o nome SOLVER, uma combinação de “Solar” e “Rover”, representando a integração entre o sistema de limpeza e o veículo móvel responsável pela locomoção. O nome também remete ao verbo em inglês “to solve”, reforçando a ideia de solução. A identidade visual do projeto inclui um logotipo que combina elementos que representam o robô utilizado e painéis solares.

Figura 1: Identidade visual do manipulador.

O manipulador SOLVER atua em conjunto com o rover no processo de limpeza de painéis solares, sendo responsável por posicionar e movimentar a escova que remove poeira, areia e outros resíduos da superfície. Sua função principal é manter a escova na posição adequada durante a limpeza, ajustando continuamente a orientação e a distância em relação aos módulos solares. Dessa forma, o manipulador realiza os ajustes finos necessários ao longo do percurso do robô.

A configuração adotada para o manipulador consiste em três graus de liberdade, distribuídos em duas juntas rotacionais e uma junta prismática. A primeira junta, na base, gera a rotação horizontal do conjunto, orientando o manipulador em direção ao painel. A segunda junta, também rotacional, faz o mesmo movimento que a primeira, porém permitindo ajustes mais finos. A junta prismática, localizada no segmento final, controla o movimento linear de aproximação e afastamento da escova, permitindo ajustar a distância entre o end-effector e o painel conforme necessário. A movimentação dessas três juntas é coordenada para posicionar a escova antes e durante o processo de limpeza.

O end-effector consiste em uma escova rotativa montada na extremidade do braço. Essa escova possui cerdas macias para evitar danos à superfície dos módulos solares, que são suscetíveis a arranhões e impactos. A função do end-effector é realizar a limpeza

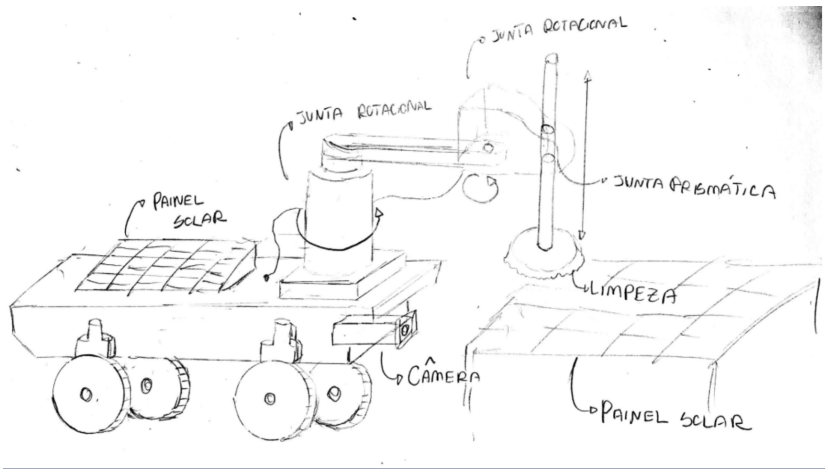

Figura 2: Croqui do robô.

ativa da superfície por meio da rotação contínua da escova, acionada por um motor independente instalado no conjunto final.

A velocidade de rotação da escova pode ser ajustada pelo sistema de controle, permitindo adaptar o processo de limpeza a diferentes condições de acúmulo de sujeira. A posição do end-effector, alinhada com a junta prismática, permite controlar a força de contato com o painel por meio da variação da distância, o que é essencial para evitar desgaste desnecessário tanto da escova quanto da superfície do painel.

A operação tem início com a aproximação do rover à fileira de painéis solares. O alinhamento é realizado com auxílio de GPS, sensores LIDAR ou ultrassônicos, que garantem o posicionamento correto em relação ao painel.

Durante o deslocamento, o manipulador realiza apenas pequenos ajustes verticais e de profundidade para manter o contato e compensar variações de altura ou inclinação entre os módulos. Quando o rover completa uma fileira, pode realizar um reposicionamento lateral para iniciar a limpeza da faixa seguinte, repetindo o procedimento até cobrir toda a área planejada. Sensores e câmeras auxiliam na detecção de obstáculos, no alinhamento e na avaliação da sujeira ao longo da operação.

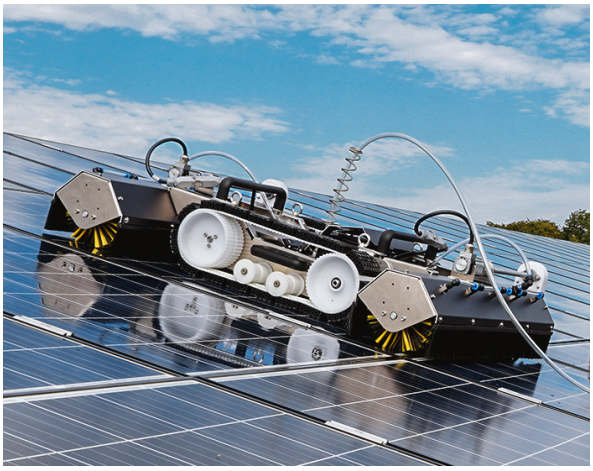

Figura 3: Exemplo de robô que realiza limpeza de painéis solares.

A integração com os demais sistemas do rover ocorre por meio da eletrônica embarcada e do software de controle. O controlador central processa as informações dos sensores, calcula os ajustes necessários nas juntas e coordena o acionamento da escova. Os drivers enviam comandos para cada motor, enquanto o sistema de navegação gerencia a locomoção do robô. Essa integração garante que o processo de limpeza seja executado de forma contínua e sincronizada.

3 DESENVOLVIMENTO DO PROJETO

3.1 Modelagem matemática

3.1.1 Cinemática direta

A cinemática direta descreve a posição e a orientação do end-effector em função das variáveis articulares do manipulador. Dado um conjunto de ângulos e deslocamentos das juntas, a cinemática direta permite determinar a configuração do end-effector no espaço cartesiano. Esse procedimento, posteriormente, será utilizado na determinação do Jacobiano, da cinemática inversa e na formulação da dinâmica do manipulador.

Para a formulação da cinemática direta é aplicado o método de Denavit–Hartenberg (DH). Nesse método, cada elo é representado por uma matriz de transformação homogênea ${}^{i-1}A_i$, que relaciona o sistema de coordenadas do elo i com o do elo anterior. Cada matriz é descrita por quatro parâmetros: a_i, α_i, d_i e θ_i.

A matriz de transformação homogênea geral é dada por:

$$ {}^{i-1}A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Para um manipulador com n juntas, a transformação do sistema da base até o end-effector é obtida pela multiplicação sequencial das matrizes:

$$ {}^0A_n = {}^0A_1 \oplus {}^1A_2 \oplus \cdots \oplus {}^{n-1}A_n $$

A matriz completa contém simultaneamente a posição e a orientação do end-effector:

$$ P_n = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}, \qquad R_n = \begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix} $$

No manipulador SOLVER, os parâmetros de Denavit–Hartenberg são definidos de forma a representar sua geometria, permitindo construir as matrizes ${}^{i-1}A_i$ correspondentes a cada elo. A multiplicação dessas matrizes fornece a transformação completa ${}^0A_3$.

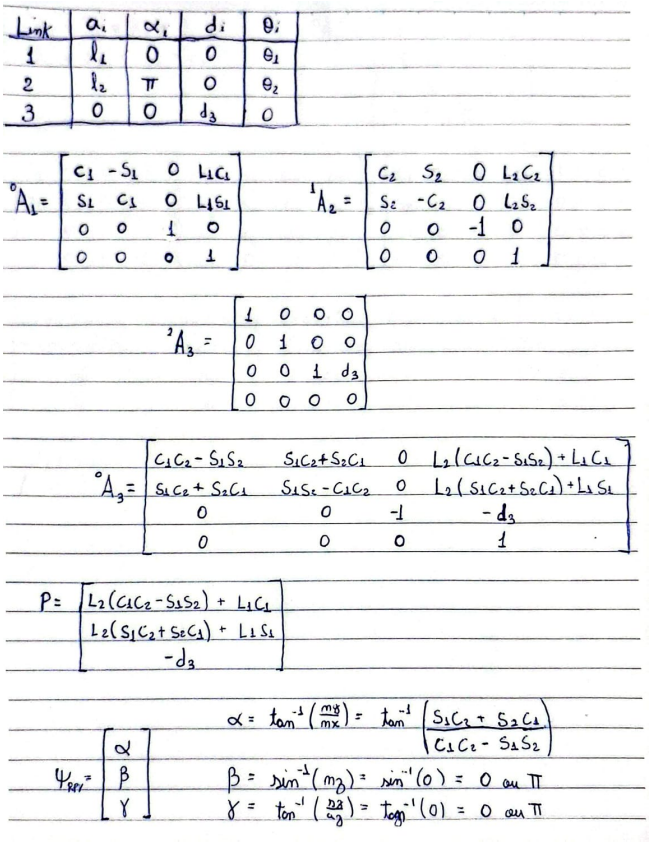

Figura 4: Cinemática direta final obtida para o manipulador.

3.1.2 Cinemática inversa

A cinemática inversa tem como objetivo determinar as variáveis articulares $q = [\theta_1,\ \theta_2,\ d_3]^T$ necessárias para que o manipulador alcance uma posição cartesiana desejada $P_n = [P_x,\ P_y,\ P_z]^T$. Para isso, parte-se da expressão já conhecida da cinemática direta, construída como a composição das transformações homogêneas das juntas:

$$^0A_3 = {}^0A_1 \ \oplus\ {}^1A_2 \ \oplus\ {}^2A_3$$

A estratégia consiste em inverter essa relação de forma sequencial, isolando cada matriz correspondente a uma junta. Esse procedimento é realizado multiplicando-se a expressão por inversas apropriadas, de modo a eliminar gradualmente as transformações correspondentes às juntas internas.

Por exemplo, ao multiplicar a expressão à esquerda por $(^0A_1)^{-1}$, remove-se a contribuição do primeiro elo:

$$(^0A_1)^{-1}\,{}^0A_3 = {}^1A_2 \ \oplus\ {}^2A_3$$

Isso ocorre porque:

$$({}^0A_1)^{-1}\,{}^0A_1 = I$$

O mesmo raciocínio pode ser aplicado novamente, utilizando a inversa de ${}^1A_2$, e assim sucessivamente. Esse processo de cancelamento progressivo permite isolar cada transformação ${}^iA_{i+1}$ e, portanto, cada variável articular associada. Dessa forma, obtêm-se as expressões que relacionam diretamente a pose desejada ${}^0A_3$ aos valores das juntas que devem ser ajustadas para alcançá-la.

Implementando esse procedimento no manipulador, chega-se aos seguintes resultados:

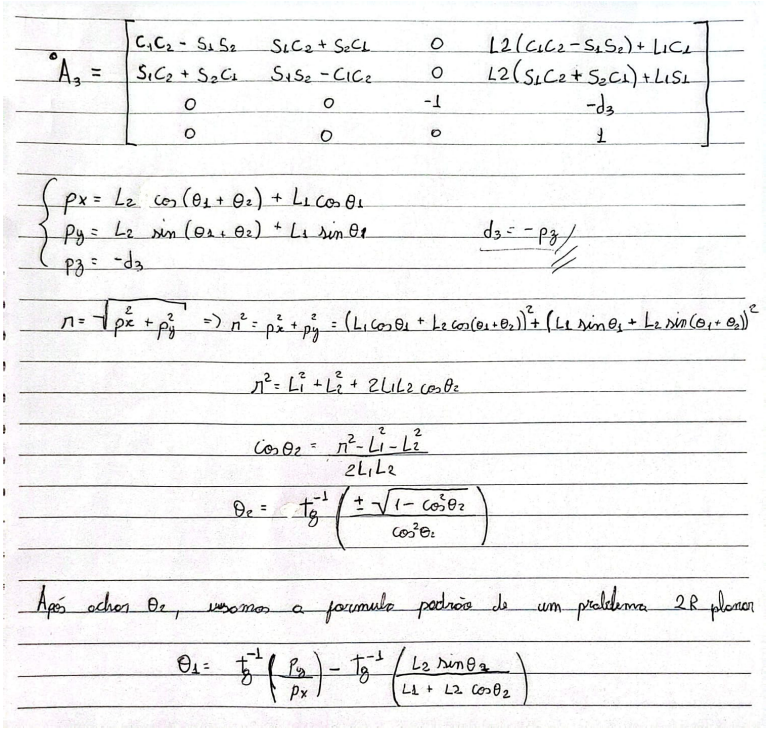

Figura 5: Cinemática inversa final obtida para o manipulador.

3.1.3 Jacobiano

O Jacobiano é uma matriz utilizada na modelagem de manipuladores que relaciona diretamente as velocidades articulares com as velocidades lineares e angulares do end-effector. Enquanto a cinemática direta determina a posição do end-effector a partir das variáveis articulares, o Jacobiano descreve como variações instantâneas nos ângulos e deslocamentos das juntas afetam o movimento no espaço cartesiano. Essa relação é utilizada principalmente no controle de velocidade, na análise de singularidades e no planejamento de trajetórias.

Para um manipulador com n graus de liberdade, a relação entre as velocidades articulares $\dot{q}$ e a velocidade cartesiana do end-effector $\dot{x}$ é dada por:

$$\dot{x} = J(q) \oplus \dot{q}$$

onde:

- $\dot{x}$ é o vetor de velocidades linear e angular do end-effector;
- $\dot{q}$ é o vetor de velocidades das juntas;
- $J(q)$ é a matriz Jacobiana.

Para juntas rotacionais e prismáticas, utilizam-se expressões específicas. Se a junta i for rotacional:

$$J_{Li} = b_{i-1} \times (r_n - r_{i-1})$$

$$J_{Ai} = b_{i-1}$$

Se a junta i for prismática:

$$J_{Li} = b_{i-1}$$

$$J_{Ai} = 0$$

Assim, o Jacobiano completo assume a forma:

$$\mathrm{J} = \begin{bmatrix} J_{L1} & J_{L2} & J_{L3} \\ J_{A1} & J_{A2} & J_{A3} \end{bmatrix}$$

Aplicando o Jacobiano no SOLVER, a sua construção segue diretamente as expressões apresentadas. As duas primeiras colunas da matriz correspondem às juntas rotacionais, enquanto a terceira representa a contribuição da junta prismática.

Cada vetor r_i e b_i é extraído das matrizes de transformação obtidas na cinemática direta. Para manter o padrão utilizado na cinemática direta e inversa, considere que as variáveis l_1, l_2, l_3 utilizadas na resolução seguinte equivalem a, respectivamente, 0, l_1 e l_2.

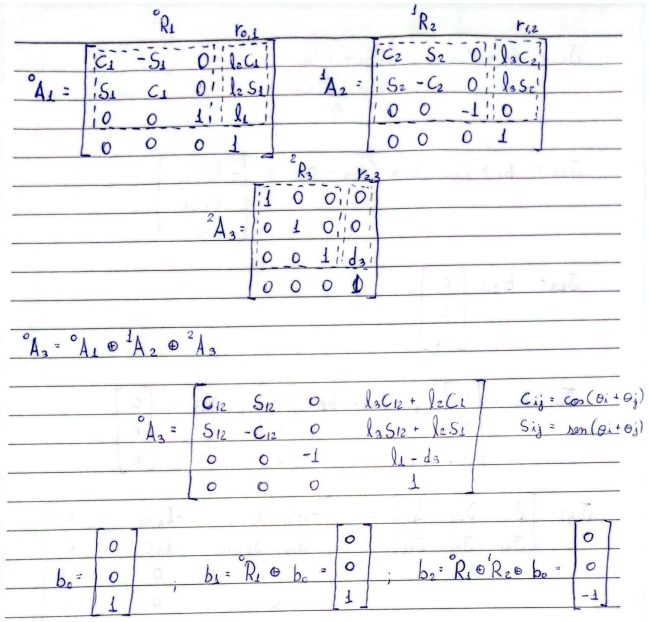

Figura 6: Encontrando o Jacobiano.

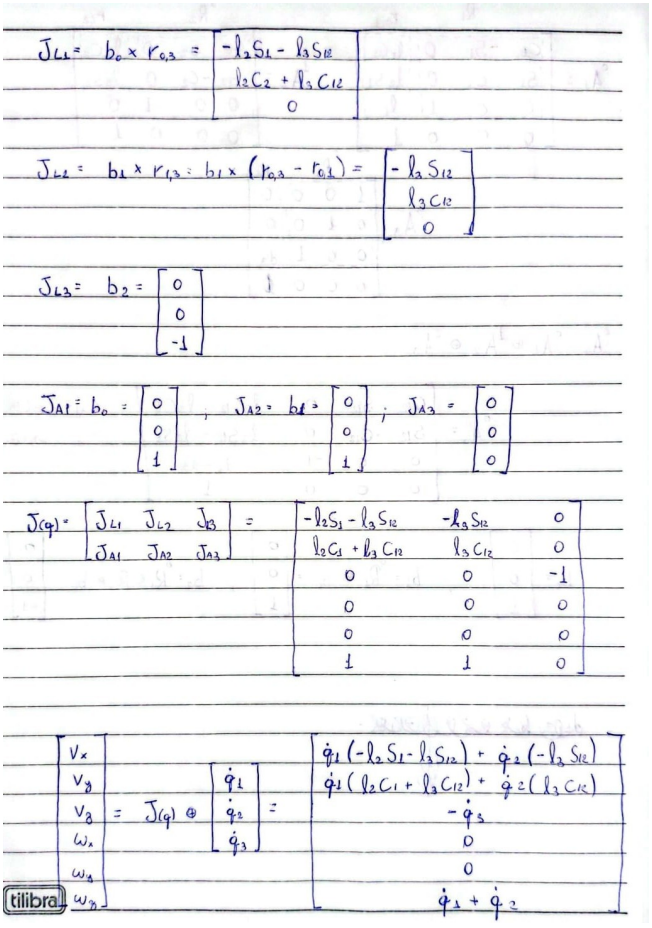

Figura 7: Jacobiano final obtido para o manipulador.

3.1.4 Dinâmica

O modelo dinâmico descreve as forças e torques necessários para produzir um movimento desejado do manipulador, levando em conta suas massas, geometrias, velocidades e acelerações. A formulação dinâmica é expressa pela equação geral de movimento abaixo:

$$M(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + G(q) = \tau$$

onde:

- $\dot{q}$ é o vetor das velocidades articulares;
- $\ddot{q}$ é o vetor das acelerações articulares;
- $M(q)$ é a matriz de inércia, que relaciona as acelerações articulares às forças e torques necessários;
- $C(q,\dot{q})\,\dot{q}$ são os termos de Coriolis e centrífugos, que surgem devido à movimentação dos elos e às interações dinâmicas entre eles;
- $G(q)$ é o vetor de forças e torques gravitacionais;
- τ é o vetor de torques (ou forças, no caso de juntas prismáticas) aplicados nas juntas.

Para se obter esse modelo, inicia-se descrevendo a energia cinética e potencial de cada elo. Em geral, para um manipulador com n juntas, a energia cinética total é dada por:

$$K = \sum_{i=1}^{n} \left(\frac{1}{2} m_i\, v_i^T v_i + \frac{1}{2}\, \omega_i^T I_i \omega_i \right),$$

onde:

- m_i é a massa do elo i;
- v_i é a velocidade linear do centro de massa do elo i;
- $v_i^T v_i$ é o produto interno da velocidade linear, que representa o módulo ao quadrado dessa velocidade;
- ω_i é a velocidade angular do elo i;
- I_i é o tensor de inércia do elo i em relação ao seu centro de massa;

- $\omega_i^T I_i \omega_i$ é o termo da energia cinética associado à rotação do elo;
- $\sum_{i=1}^{n}$ indica que a energia cinética total é obtida somando a energia de todos os elos do manipulador.

A energia potencial gravitacional é:

$$P = \sum_{i=1}^{n} m_i g h_i,$$

sendo h_i a altura do centro de massa. A dinâmica é então obtida aplicando as equações de Euler-Lagrange:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = \tau, \qquad L = K - P.$$

A partir dessas derivadas obtêm-se explicitamente as expressões para $M(q)$, $C(q, \dot{q})$ e $G(q)$. O termo inercial $M(q)$ relaciona as acelerações articulares às forças necessárias; o termo $C(q, \dot{q})$ capta interações dinâmicas entre os elos; e o vetor $G(q)$ fornece as forças requeridas apenas para compensar o efeito da gravidade. Dessa forma, o modelo completo descreve as exigências dinâmicas do manipulador em qualquer trajetória.

Uma vez obtido o modelo dinâmico, ele foi implementado no SOLVER para calcular as forças e torques necessários durante a execução das trajetórias de limpeza. Os resultados incluem as matrizes de inércia, as velocidades e posições de cada elo e as forças e torques finais aplicados nas juntas. As imagens apresentadas a seguir mostram esses resultados:

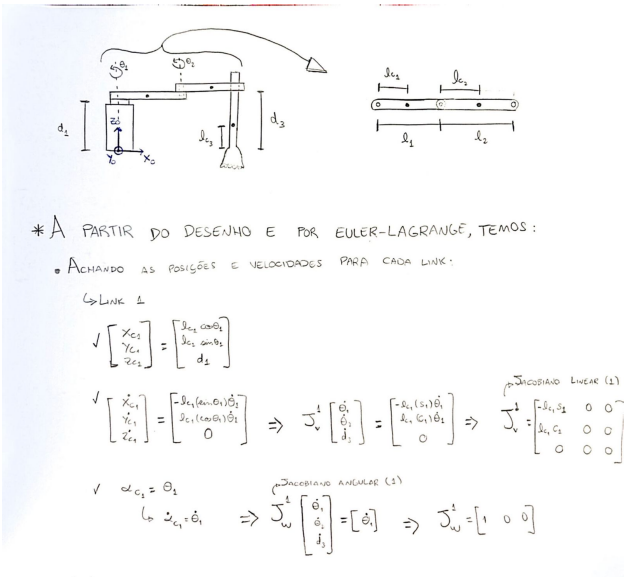

Figura 8: Posições e velocidades do link 1.

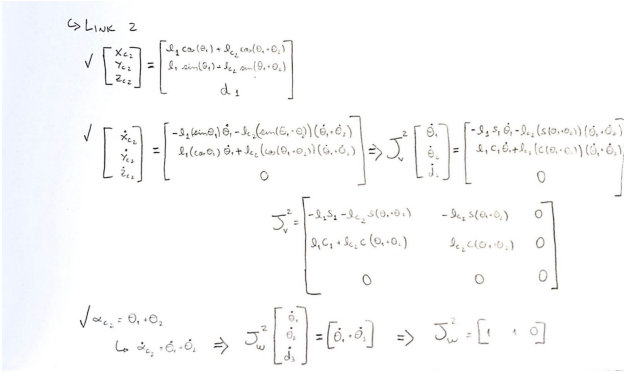

Figura 9: Posições e velocidades do link 2.

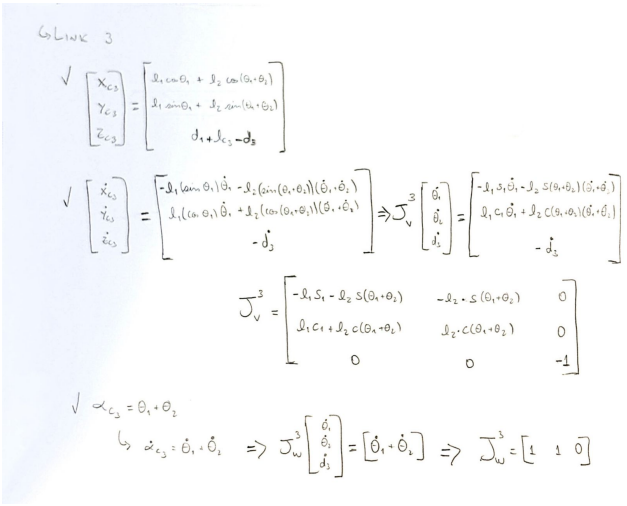

Figura 10: Posições e velocidades do link 3.

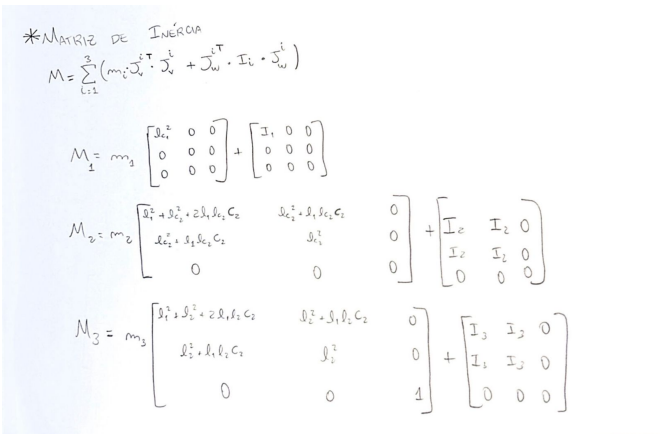

Figura 11: Encontrando a matriz de inércia.

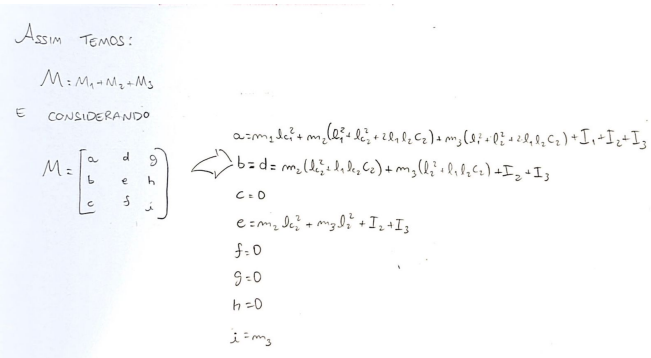

Figura 12: Matriz de inércia.

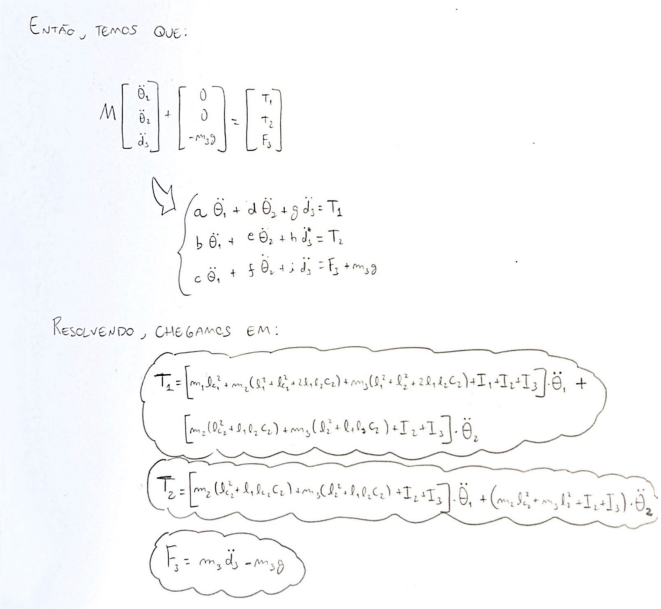

Figura 13: Forças e torques finais.

3.2 Projeto mecânico (CAD)

O projeto mecânico do SOLVER foi desenvolvido em ambiente CAD com o objetivo de representar sua estrutura física e verificar a compatibilidade entre os elementos definidos na modelagem matemática. O modelo tridimensional permitiu avaliar dimensões, movimentos das juntas e o espaço necessário para acoplamento ao rover.

A base do manipulador foi modelada para fixação ao chassi do veículo. As duas juntas rotacionais foram posicionadas conforme os graus de liberdade definidos, garantindo o alcance necessário para atuar sobre a superfície dos painéis solares. O primeiro elo foi dimensionado para permitir o deslocamento horizontal inicial, enquanto o segundo elo complementa esse alcance. A junta prismática foi representada por um trilho linear alinhado ao eixo da escova, responsável pelo movimento de aproximação e afastamento do end-effector, que foi modelado com um suporte destinado à escova rotativa.

As Figuras 14 e 15 apresentam o manipulador completo no ambiente CAD.

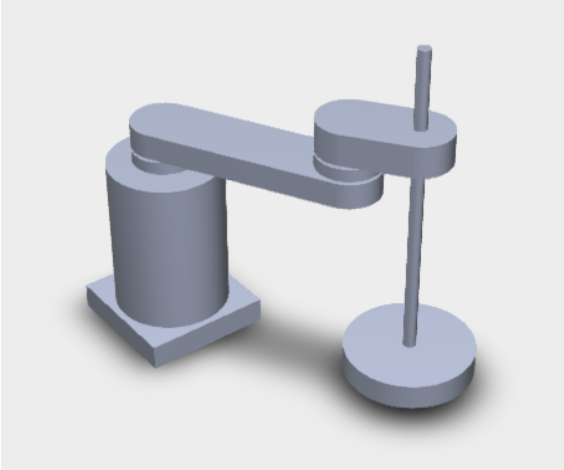

Figura 14: Imagem 1 do CAD do manipulador.

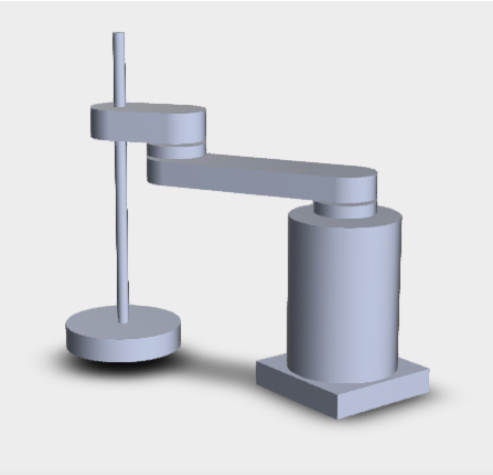

Figura 15: Imagem 2 do CAD do manipulador.

3.3 Simulação computacional

A simulação computacional do manipulador SOLVER (Anexo 1) foi desenvolvida em ambiente Python com o objetivo de validar os modelos cinemático e dinâmico apresentados anteriormente, bem como analisar o comportamento do manipulador durante a execução de trajetórias representativas da tarefa de limpeza de painéis solares. Para isso, foi utilizada a *Robotics Toolbox for Python*, em conjunto com as bibliotecas *NumPy* e *Matplotlib*, permitindo a integração entre modelagem matemática, cálculo dinâmico e visualização gráfica.

A caracterização computacional do manipulador é realizada a partir da definição de seus parâmetros geométricos e físicos. O robô possui uma arquitetura do tipo RRP e os parâmetros de Denavit–Hartenberg são definidos no código conforme a geometria estabelecida no projeto mecânico, permitindo a construção do modelo cinemático consistente com o manipulador real.

Além dos parâmetros DH, são atribuídas propriedades físicas a cada elo, como massa, posição do centro de massa e tensor de inércia. As massas são estimadas considerando densidade uniforme do alumínio e os elos são aproximados como paralelepípedos, o que possibilita o cálculo analítico dos momentos e produtos de inércia.

A cinemática direta é implementada computacionalmente a partir da multiplicação das matrizes de transformação homogênea correspondentes a cada junta do manipulador. No código, essa implementação permite calcular a pose final do end-effector para um dado conjunto de variáveis articulares, retornando explicitamente a matriz de transformação do sistema da base até a extremidade do robô.

Para fins de simulação, valores iniciais e finais das juntas são definidos, e o movimento entre essas configurações é interpolado por meio de uma trajetória suave no espaço das juntas. A simulação permite observar a evolução temporal da posição do end-effector, bem como a animação do manipulador executando o movimento correspondente à cinemática direta.

A cinemática inversa é utilizada para determinar os valores das variáveis articulares necessários para que o end-effector alcance uma posição cartesiana desejada. No código, essa etapa é implementada de forma analítica, explorando a geometria do manipulador RRP e resultando em expressões fechadas para o cálculo das juntas rotacionais e da junta prismática.

Na simulação, o usuário fornece um ponto no espaço cartesiano, e o algoritmo calcula automaticamente os valores correspondentes das juntas. Em seguida, é realizada uma simulação temporal que leva o manipulador da configuração atual até a nova configuração calculada, novamente utilizando interpolação suave no espaço das juntas.

Para representar a aplicação real do manipulador, foi implementado o planejamento de uma trajetória de escovação diretamente no espaço cartesiano. Nessa abordagem, são definidos pontos estratégicos que representam a posição inicial, a aproximação ao painel e o movimento repetitivo da escova ao longo da superfície a ser limpa.

A interpolação entre esses pontos é realizada por meio de um polinômio de quinto grau, garantindo continuidade de posição, velocidade e aceleração ao longo da trajetória. A cada ponto da trajetória cartesiana, a cinemática inversa é utilizada para obter as variáveis articulares correspondentes, permitindo que o manipulador execute o movimento desejado.

A partir das trajetórias geradas, tanto no espaço das juntas quanto no espaço cartesiano, são calculadas as grandezas dinâmicas associadas ao movimento do manipulador. Utilizando as posições, velocidades e acelerações articulares, o código aplica o algoritmo de dinâmica inversa de Newton–Euler para determinar os torques nas juntas rotacionais e a força na junta prismática ao longo do tempo.

Os gráficos gerados sintetizam essas informações e permitem uma análise das exigências dinâmicas durante a execução das trajetórias simuladas.

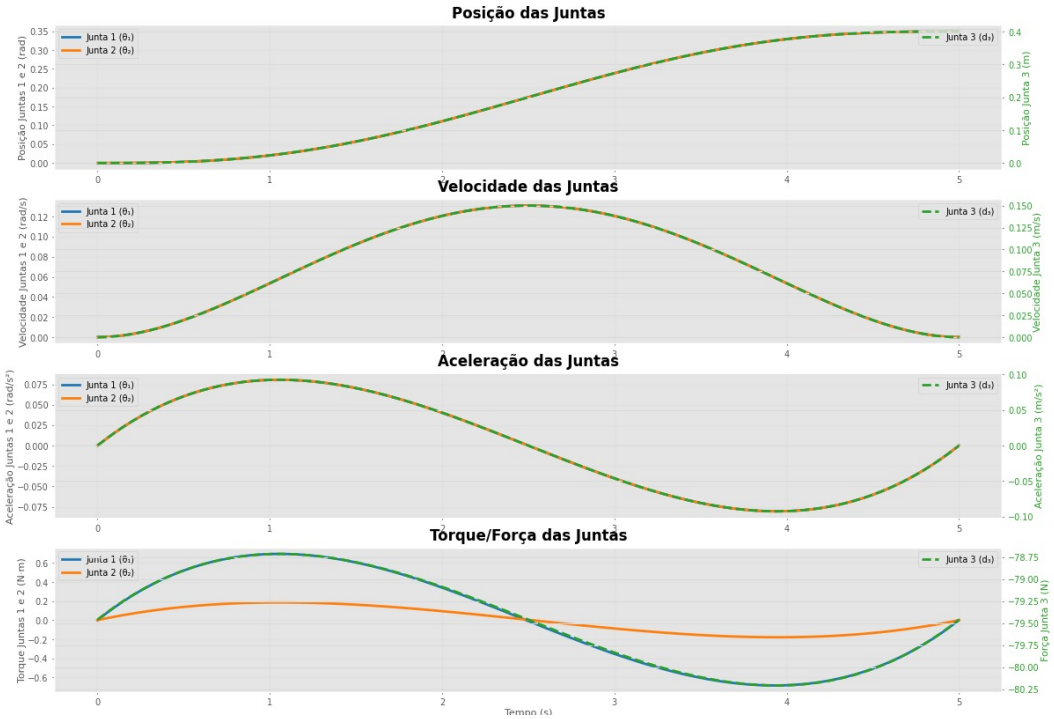

Figura 16: Perfis de posição, velocidade, aceleração e torque/força das juntas da cinemática direta até a posição [20° 20° 0.4].

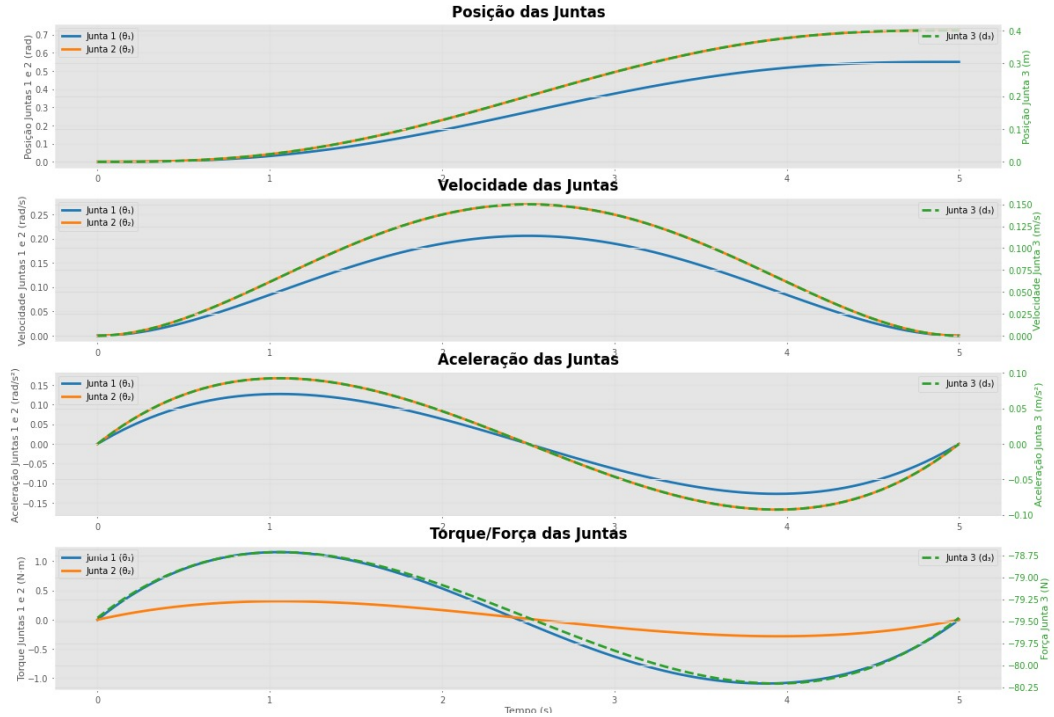

Figura 17: Perfis de posição, velocidade, aceleração e torque/força das juntas da cinemática inversa até a posição [0.4 0.4 -0,4].

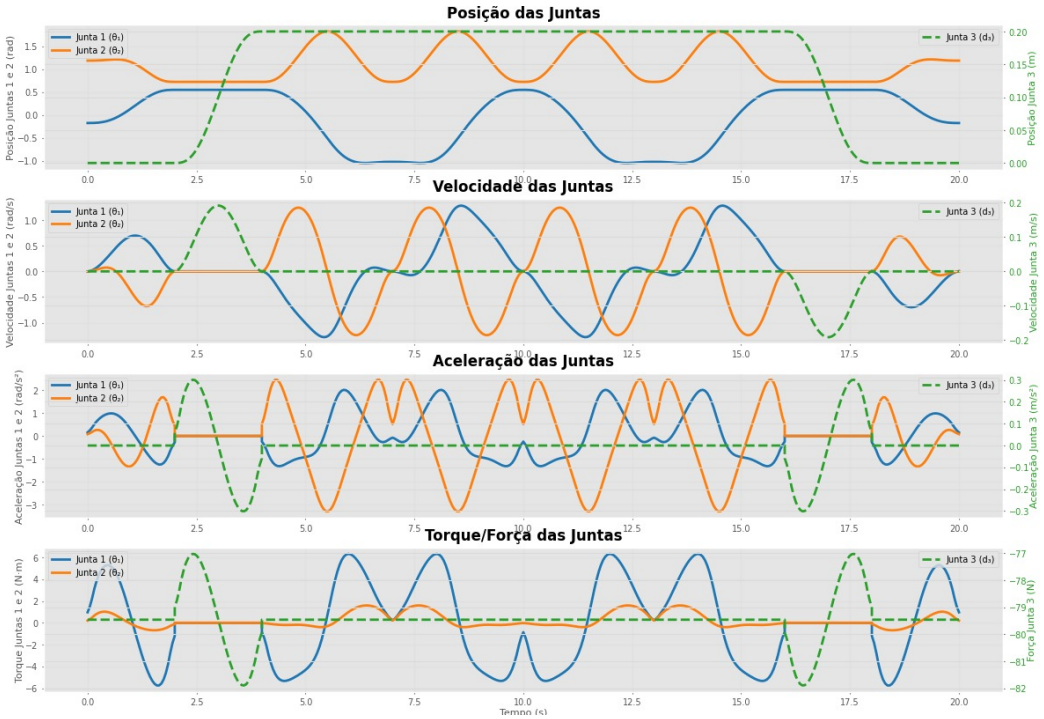

Figura 18: Perfis de posição, velocidade, aceleração e torque/força das juntas da trajetória de escovação.

O código e os arquivos utilizados neste trabalho estão disponíveis no repositório a seguir: `https://github.com/AmandaHellen84/Manipulador-SOLVER`

4 CONCLUSÃO

O desenvolvimento do manipulador SOLVER permitiu integrar modelagem matemática, projeto mecânico e simulação em um único sistema aplicado à limpeza de painéis solares. A modelagem cinemática e dinâmica forneceu as equações necessárias para prever o comportamento do manipulador e orientar as decisões de projeto. O modelo CAD complementou essa etapa ao representar a estrutura física e verificar a viabilidade dos movimentos definidos.

As simulações possibilitaram analisar o desempenho do manipulador em diferentes condições e confirmar a coerência entre os modelos teóricos e o projeto mecânico. O conjunto final apresenta uma solução funcional para posicionamento e acionamento da escova utilizada no processo de limpeza. O trabalho estabeleceu uma base consistente para futuras melhorias, como o detalhamento de componentes e a implementação do controle, seguidas posteriormente pela construção e montagem física do manipulador.

5 REFERÊNCIAS BIBLIOGRÁFICAS

[1] Almeida, Rynaldo. *Modelagem Dinâmica e Controle de Robô Manipulador de Arquitetura Paralela Assimétrica de Três Graus de Liberdade*. Tese (Doutorado) — Universidade de São Paulo, 2014. Disponível em: `https://www.teses.usp.br/teses/disponiveis/3/3151/tde-19092014-113652/publico/Tese_Rynaldo_Almeida.pdf`. Acesso em: nov. 2025.

[2] Souza Júnior, Valdir de. *Modelagem Dinâmica e Estática do Braço Robótico UR5*. Universidade Federal de Alagoas. Disponível em: `http://www.repositorio.ufal.br/jspui/handle/123456789/9735`. Acesso em: nov. 2025.

[3] Peter Corke – Robotics Toolbox. *Documentação traduzida*. Disponível em: `https://translate.google.com/translate?u=https://petercorke.com/toolboxes/robotics-toolbox/&hl=pt&sl=en&tl=pt&client=srp`. Acesso em: nov. 2025.

[4] Fetter. *Jacobian — notas de aula*. Universidade Federal do Rio Grande do Sul. Disponível em: `http://www.ece.ufrgs.br/~fetter/eng10026/jacobian.pdf`. Acesso em: nov. 2025.

[5] Lima, Élton Franklin Silva de; Nascimento, Ian Ribeiro; Soares, Fábio Guilherme de Andrade; Souza, Valdenio João Francisco José Lima de; Sena, Alexander Patrick Chaves de. *Modelagem Cinemática Inversa e Simulação de Controle de Manipuladores Robóticos com 3 a 5 Graus de Liberdade*. Disponível em: `https://smart.institutoidv.org/2024/pdvg/uploads/1008.pdf`. Acesso em: nov. 2025.

ANEXO 1 - Código-fonte do manipulador SOLVER

```
import roboticstoolbox as rtb
import numpy as np
from math import cos, sin, atan2, acos, sqrt, pi
import matplotlib.pyplot as plt
import time

class RRP:
    def __init__(self):
        l1 = 0.4 #400mm
        l2 = 0.5*l1
        l3 = 1.25*l1

        self.a1 = l1; self.a2 = l2; self.a3 = 0
        self.alpha1 = 0; self.alpha2 = np.pi; self.alpha3 = 0
        self.d1 = 0; self.d2 = 0; self.d3 = 0
        self.theta1 = 0; self.theta2 = 0; self.theta3 = 0

        w = l1*0.25
        h = l1*0.15

        w3 = l1/20
        h3 = l2/20

        #considerando densidade do al minio
        m1 = 2700*l1*w*h
        m2 = 2700*l2*w*h
        m3 = 2700*l3*w*h

        #Aproximando para paralelep pedos
        Ixx1 = (1/12)*m1*(pow(h,2)+pow(w,2))
        Iyy1 = (1/12)*m1*(pow(h,2)+pow(l1,2))
        Izz1 = (1/12)*m1*(pow(l1,2)+pow(w,2))

        Ixx2 = (1/12)*m2*(pow(h,2)+pow(w,2))
        Iyy2 = (1/12)*m2*(pow(h,2)+pow(l2,2))
        Izz2 = (1/12)*m2*(pow(l2,2)+pow(w,2))

        Ixx3 = (1/12)*m2*(pow(l3,2)+pow(w3,2))
        Iyy3 = (1/12)*m2*(pow(l3,2)+pow(h3,2))
        Izz3 = (1/12)*m2*(pow(h3,2)+pow(w3,2))

        self.robot = rtb.DHRobot([
            rtb.RevoluteDH(d=self.d1, a=self.a1, alpha=self.alpha1,
   offset=self.theta1, m=m1, r=[l1/2,0,0], I=np.diag([Ixx1, Iyy1,
   Izz1])),
            rtb.RevoluteDH(d=self.d2, a=self.a2, alpha=self.alpha2,
   offset=self.theta2, m=m2, r=[l2/2,0,0], I=np.diag([Ixx2, Iyy2,
   Izz2])),
            rtb.PrismaticDH(theta=self.theta3, a=self.a3, alpha=self.
   alpha3, offset=self.d3, m=m3, r=[0,0,l3/2], I=np.diag([Ixx3, Iyy3,
    Izz3]))
        ])

        # para visualizar a declara  o do manipulador
        print(self.robot)

```

```
51      def cinematica_direta(self, q):
52
53          theta1, theta2, d3 = q
54
55          T01 = np.array([
56              [cos(theta1), -sin(theta1)*cos(self.alpha1), sin(theta1)*
    sin(self.alpha1), self.a1*cos(theta1)],
57              [sin(theta1), cos(theta1)*cos(self.alpha1), -cos(theta1)*
    sin(self.alpha1), self.a1*sin(theta1)],
58              [0, sin(self.alpha1), cos(self.alpha1), self.d1],
59              [0, 0, 0, 1]
60          ])
61
62          T12 = np.array([
63              [cos(theta2), -sin(theta2)*cos(self.alpha2), sin(theta2)*
    sin(self.alpha2), self.a2*cos(theta2)],
64              [sin(theta2), cos(theta2)*cos(self.alpha2), -cos(theta2)*
    sin(self.alpha2), self.a2*sin(theta2)],
65              [0, sin(self.alpha2), cos(self.alpha2), self.d2],
66              [0, 0, 0, 1]
67          ])
68
69          T23 = np.array([
70              [cos(self.theta3), -sin(self.theta3)*cos(self.alpha3), sin
    (self.theta3)*sin(self.alpha3), self.a3*cos(self.theta3)],
71              [sin(self.theta3), cos(self.theta3)*cos(self.alpha3), -cos
    (self.theta3)*sin(self.alpha3), self.a3*sin(self.theta3)],
72              [0, sin(self.alpha3), cos(self.alpha3), d3],
73              [0, 0, 0, 1]
74          ])
75
76          T03 = T01 @ T12 @ T23
77
78          return T03
79
80      def cinematica_inversa(self, p):
81
82          x, y, z = p
83
84          # junta 3 (prism tica)
85          q3 = -z
86
87          # junta 2
88          c2 = (x**2 + y**2 - self.a1**2 - self.a2**2) / (2*self.a1*self
    .a2)
89          c2 = np.clip(c2, -1, 1)
90
91          s2 = sqrt(1 - c2**2)
92          theta2 = atan2(s2, c2)   # solu   o cotovelo para cima
93          # theta2 = atan2(-s2, c2)  # alternativa cotovelo para baixo
94
95          # junta 1
96          k1 = self.a1 + self.a2*cos(theta2)
97          k2 = self.a2*sin(theta2)
98
99          theta1 = atan2(y, x) - atan2(k2, k1)
100
101         return np.array([theta1, theta2, q3])
```

```
102
103
104     #Calculo da trajet ria no espa o das juntas (Para a simula  o
    da cinem tica direta e inversa)
105     def trajetoria_simulacao(self,q_i, q_f):
106         # Posicoes inicial e final
107         q_start = q_i
108         q_end = q_f
109
110         # Tempo de simulacao e dt
111         time_s = 5.0
112         dt = 0.05
113         steps = int(time_s / dt)
114
115         # Vetor de tempo
116         t = np.linspace(0, time_s, steps)
117
118         # Trajet ria polinomial de 5   grau
119         traj = rtb.jtraj(q_start, q_end, t)
120
121         # Calcular torques usando din mica inversa
122         tau = self._calcular_torques(traj.q, traj.qd, traj.qdd)
123
124         # Plotar gr ficos
125         self._plotar_graficos(t, traj.q, traj.qd, traj.qdd, tau)
126
127         # Animar trajet ria
128         self.robot.plot(traj.q, backend="pyplot", dt=dt, block=False)
129
130         return 0
131
132     #Calculo da trajet ria no espa o cartesiano (Para a simula  o
    da trajet ria de escova  o)
133     def trajetoria_cartesiana(self, p_start, p_end, time_segment, dt):
134         steps = int(time_segment / dt)
135         t = np.linspace(0, time_segment, steps)
136
137         tau = t / time_segment
138         s = 10*tau**3 - 15*tau**4 + 6*tau**5
139
140         p_traj = np.zeros((steps, 3))
141         for i in range(3):
142             p_traj[:, i] = p_start[i] + s * (p_end[i] - p_start[i])
143
144         q = np.zeros((steps, 3))
145         for k in range(steps):
146             q[k, :] = self.cinematica_inversa(p_traj[k, :])
147
148         qd = np.gradient(q, dt, axis=0, edge_order=2)
149         qdd = np.gradient(qd, dt, axis=0, edge_order=2)
150
151         self.q_all.append(q)
152         self.qd_all.append(qd)
153         self.qdd_all.append(qdd)
154         self.t_all.append(t + self.t_acc)
155         self.t_acc += t[-1]
156
157
```

```
158     def trajetoria_escovacao(self):
159         print("\n" + "="*60)
160         print("INICIANDO TRAJET RIA")
161         print("="*60)
162 
163         # Definir pontos da trajet ria (x, y, z)
164         p_home = np.array([0.5, 0.1, 0.0])
165         p_above = np.array([0.4, 0.4, 0.0])
166         p_brush1 = np.array([0.4, 0.4, -0.2])
167         p_brush2 = np.array([0.4, -0.4, -0.2])
168 
169 
170         # Par metros de tempo
171         dt = 0.05
172         time_segment = 2
173 
174         # Armazenar trajet ria completa
175         self.q_all = []
176         self.qd_all = []
177         self.qdd_all = []
178         self.t_all = []
179         self.t_acc = 0
180 
181         # Montando a trajet ria
182         self.trajetoria_cartesiana(p_home, p_above, time_segment, dt)
183         self.trajetoria_cartesiana(p_above, p_brush1, time_segment, dt
    )
184         self.trajetoria_cartesiana(p_brush1, p_brush2, 3, dt)
185         self.trajetoria_cartesiana(p_brush2, p_brush1, 3, dt)
186         self.trajetoria_cartesiana(p_brush1, p_brush2, 3, dt)
187         self.trajetoria_cartesiana(p_brush2, p_brush1, 3, dt)
188         self.trajetoria_cartesiana(p_brush1, p_above, time_segment, dt
    )
189         self.trajetoria_cartesiana(p_above, p_home, time_segment, dt)
190 
191         # Concatenar todos os segmentos
192         q_total = np.vstack(self.q_all)
193         qd_total = np.vstack(self.qd_all)
194         qdd_total = np.vstack(self.qdd_all)
195         t_total = np.concatenate(self.t_all)
196 
197         # Calcular torques usando din mica inversa
198         tau = self._calcular_torques(q_total, qd_total, qdd_total)
199 
200         # Plotar gr ficos
201         self._plotar_graficos(t_total, q_total, qd_total, qdd_total,
    tau)
202 
203         # Animar trajet ria
204         print("\nAnimando trajet ria...")
205         self.robot.plot(q_total, backend="pyplot", dt=dt, block=False)
206 
207         return q_total, qd_total, qdd_total, t_total
208 
209     def _calcular_torques(self, q, qd, qdd):
210         n_points = q.shape[0]
211         tau = np.zeros((n_points, 3))
212 
```

```
213         for i in range(n_points):
214             tau[i, :] = self.robot.rne(q[i, :], qd[i, :], qdd[i, :])
215         return tau
216 
217     def _plotar_graficos(self, t, q, qd, qdd, tau):
218         fig, axes = plt.subplots(4, 1, figsize=(12, 12))
219 
220         joint_names = ['Junta 1 (     )', 'Junta 2 (     )', 'Junta 3
    ( d  )']
221         colors = ['#1f77b4', '#ff7f0e', '#2ca02c']
222 
223         # Gr fico de Posi   o
224         for i in range(3):
225             if i < 2:
226                 axes[0].plot(t, q[:, i], color=colors[i], linewidth=2,
     label=joint_names[i])
227             else:
228                 ax0_twin = axes[0].twinx()
229                 ax0_twin.plot(t, q[:, i], color=colors[i], linewidth
    =2, label=joint_names[i], linestyle='--')
230                 ax0_twin.set_ylabel('Posi   o Junta 3 (m)', fontsize
    =8, color=colors[i])
231                 ax0_twin.tick_params(axis='y', labelcolor=colors[i])
232                 ax0_twin.legend(loc='upper right')
233 
234         axes[0].set_ylabel('Posi   o Juntas 1 e 2 (rad)', fontsize=8)
235         axes[0].set_title('Posi   o das Juntas', fontsize=12,
    fontweight='bold')
236         axes[0].grid(True, alpha=0.3)
237         axes[0].legend(loc='upper left')
238 
239         # Gr fico de Velocidade
240         for i in range(3):
241             if i < 2:
242                 axes[1].plot(t, qd[:, i], color=colors[i], linewidth
    =2, label=joint_names[i])
243             else:
244                 ax1_twin = axes[1].twinx()
245                 ax1_twin.plot(t, qd[:, i], color=colors[i], linewidth
    =2, label=joint_names[i], linestyle='--')
246                 ax1_twin.set_ylabel('Velocidade Junta 3 (m/s)',
    fontsize=8, color=colors[i])
247                 ax1_twin.tick_params(axis='y', labelcolor=colors[i])
248                 ax1_twin.legend(loc='upper right')
249 
250         axes[1].set_ylabel('Velocidade Juntas 1 e 2 (rad/s)', fontsize
    =8)
251         axes[1].set_title('Velocidade das Juntas', fontsize=12,
    fontweight='bold')
252         axes[1].grid(True, alpha=0.3)
253         axes[1].legend(loc='upper left')
254 
255         # Gr fico de Acelera   o
256         for i in range(3):
257             if i < 2:
258                 axes[2].plot(t, qdd[:, i], color=colors[i], linewidth
    =2, label=joint_names[i])
259             else:
```

```
260                 ax2_twin = axes[2].twinx()
261                 ax2_twin.plot(t, qdd[:, i], color=colors[i], linewidth
    =2, label=joint_names[i], linestyle='--')
262                 ax2_twin.set_ylabel('Acelera  o Junta 3 (m/s )',
    fontsize=8, color=colors[i])
263                 ax2_twin.tick_params(axis='y', labelcolor=colors[i])
264                 ax2_twin.legend(loc='upper right')
265
266         axes[2].set_ylabel('Acelera  o Juntas 1 e 2 (rad/s )',
    fontsize=8)
267         axes[2].set_title('Acelera  o das Juntas', fontsize=12,
    fontweight='bold')
268         axes[2].grid(True, alpha=0.3)
269         axes[2].legend(loc='upper left')
270
271         # Gr fico de Torque/For a
272         for i in range(3):
273             if i < 2:
274                 axes[3].plot(t, tau[:, i], color=colors[i], linewidth
    =2, label=joint_names[i])
275             else:
276                 ax3_twin = axes[3].twinx()
277                 ax3_twin.plot(t, tau[:, i], color=colors[i], linewidth
    =2, label=joint_names[i], linestyle='--')
278                 ax3_twin.set_ylabel('For a Junta 3 (N)', fontsize=8,
    color=colors[i])
279                 ax3_twin.tick_params(axis='y', labelcolor=colors[i])
280                 ax3_twin.legend(loc='upper right')
281
282         axes[3].set_xlabel('Tempo (s)', fontsize=8)
283         axes[3].set_ylabel('Torque Juntas 1 e 2 (N m)', fontsize=8)
284         axes[3].set_title('Torque/For a das Juntas', fontsize=12,
    fontweight='bold')
285         axes[3].grid(True, alpha=0.3)
286         axes[3].legend(loc='upper left')
287         plt.tight_layout()
288         plt.show()
289
290
291
292 Solver = RRP()
293 qi = np.array([0, 0, 0])
294 while True:
295     print("Escolha qual fun  o deseja executar:")
296     escolha = input("cd (Cinem tica direta) / ci (Cinem tica inversa
    ) / te (Trajet ria de escova  o) / limpar (voltar a posi  o
    [0 0 0]) / s (Sair) \n")
297
298     if escolha == "cd":
299         valores = input("Insira os valores das 3 juntas sepadados por
    espa o (rotacionais em graus):")
300         q = [float(x) for x in valores.split()]
301         for i in range(2):
302             q[i] = q[i]*np.pi/180
303         Matriz = Solver.cinematica_direta(q)
304         print(Matriz)
305         Solver.trajetoria_simulacao(qi,q)
306         print(f"Posi  o final: {Matriz[:3, 3]}")
```

```
307         qi = q
308 
309     elif escolha == "ci":
310         valores = input("Insira os valores as coordenadas x y z
    separadas por espa o:")
311         p = [float(x) for x in valores.split()]
312         q = Solver.cinematica_inversa(p)
313         Solver.trajetoria_simulacao(qi,q)
314         q_graus = [q[0]*180/np.pi, q[1]*180/np.pi, q[2]]
315         print(f"Posi  o das juntas: theta1={q_graus[0]:.2f}  ,
    theta2={q_graus[1]:.2f}  , d3={q[2]:.2f}")
316         for i in range(2):
317             q[i] = q_graus[i]*np.pi/180
318         Matriz = Solver.cinematica_direta(q)
319         print(f"Posi  o final: {Matriz[:3, 3]}")
320         qi = q
321 
322     elif escolha == "te":
323         q_traj, qd_traj, qdd_traj, t_traj = Solver.
    trajetoria_escovacao()
324         qi = q_traj[-1]
325 
326     elif escolha == "s":
327         break
328 
329     elif escolha == "limpar":
330         qi = np.array([0, 0, 0])
331         Solver.trajetoria_simulacao(qi,qi)
332 
333     else:
334         print("ERRO!")
335         print("Digite cd, ci, te ou s")
```