Universidade de São Paulo

Manipulação Robótica

Relatório de Atividades

Alunos
Lucas F. R. Mazzetto — 17013253
Matheus S. Soares — 9773322

São Carlos
2025

1 Objetivos

Este relatório descreve as atividades realizadas na disciplina Manipulação Robótica com o robô FRANK. Os objetivos principais foram:

- Descrever o manipulador utilizando a notação de Denavit–Hartenberg e *Screw Theory*.
- Determinar e aplicar a cinemática direta e inversa.
- Aplicar conceitos de cinemática em tarefas de manipulação (pick and place).
- Explorar métodos modernos e técnicas de aprendizagem de máquina aplicadas à manipulação robótica.

2 Introdução

2.1 Notação de Denavit-Hartenberg

A notação de Denavit–Hartenberg (DH) é uma convenção amplamente utilizada para descrever a geometria de manipuladores robóticos com apenas quatro parâmetros por junta. Seu objetivo é padronizar a forma de posicionar e orientar sistemas de coordenadas ao longo da cadeia cinemática com um número mínimo de parâmetros. A partir dessa convenção, a obtenção da cinemática direta torna-se simples e permite a compatibilidade com algoritmos computacionais já implementados.

O procedimento consiste em associar um referencial a cada junta e definir quatro parâmetros que descrevem de maneira mínima a transformação entre dois elos consecutivos. Esses parâmetros, conhecidos como comprimento de elo (a_i), torção de elo (α_i), distância entre elo (d_i) e ângulo de junta (θ_i), caracterizam as operações de rotação e translação necessárias para ir do frame i ao frame $i+1$. O método estabelece regras específicas para posicionar os eixos z_i e x_i.

A definição dos parâmetros segue os passos a seguir:

- O eixo z_i é sempre escolhido como o eixo da junta i, seja ela revoluta ou prismática.
- O eixo x_i é definido ao longo da interseção ou perpendicular comum entre z_i e z_{i+1}, apontando do elo i para o elo $i+1$.
- O eixo y_i é determinado pela regra da mão direita, completando o sistema coordenado.

Uma vez definidos os referenciais, os quatro parâmetros DH são estabelecidos:

- a_i: distância entre z_i e z_{i+1} medida ao longo de x_i; representa o comprimento do elo.
- α_i: ângulo entre z_i e z_{i+1} medido ao redor de x_i; representa a torção entre os eixos.
- d_i: distância entre a origem do frame i e a interseção com x_i, medida ao longo de z_i; variável em juntas prismáticas.

- θ_i: ângulo entre x_i e x_{i+1} ao redor de z_i; variável em juntas revolutas.

Dados esses parâmetros, a matriz de transformação entre o elo i e i+1 é definida por:

$$ {}^{i}T_{i+1} = \begin{bmatrix} cos(\theta_i) & -sen(\theta_i)cos(\alpha_i) & sen(\theta_i)sen(\alpha_i) & a_i cos(\theta_i) \\ sen(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sen(\alpha_i) & a_i sin(\theta_i) \\ 0 & sen(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Multiplicando-se sequencialmente as transformações ${}^{0}T_1$, ${}^{1}T_2$, ..., ${}^{n-1}T_n$ é possível obter a transformação completa ${}^{0}T_n$ que relaciona a base ao último elo.

2.2 Cinemática Direta

A cinemática direta trata do problema de determinar a posição e a orientação da extremidade operacional (end-effector) ou punho de um manipulador robótico a partir da posição das juntas. O que é equivalente ao mapeamento entre o espaço das juntas e o espaço cartesiano.

Os manipuladores robóticos são compostos por elos conectados por juntas, formando uma cadeia cinemática aberta. Para descrever matematicamente essa cadeia, utiliza-se uma sequência de sistemas de coordenadas ligados a cada elo, relacionados uns aos outros por transformações homogêneas. Um método amplamente utilizado para essa parametrização é o modelo de Denavit–Hartenberg (DH), como mostrado anteriormente. A transformação homogênea de cada elo para o próximo pode ser escrita genericamente como

$$ {}^{i-1}T_i(q_i) = \begin{bmatrix} {}^{i-1}R_i(q_i) & {}^{i-1}p_i(q_i) \\ 0 & 1 \end{bmatrix}, $$

onde R_i representa a rotação entre os sistemas de coordenadas das juntas, enquanto p_i representa a translação.

A composição dessas transformações sucessivas leva à matriz de transformação total entre a base do robô e o end-effector,

$$ {}^{0}T_n(\mathbf{q}) = {}^{0}T_1(q_1)\ {}^{1}T_2(q_2) \cdots {}^{n-1}T_n(q_n), $$

que encapsula toda a contribuição geométrica das juntas. A partir dessa matriz final, extrai-se a pose do efetuador. A cinemática direta também permite avaliar propriedades importantes, como alcance, zonas de operação e possíveis colisões entre elos.

2.3 Cinemática Inversa

A cinemática inversa aborda o problema oposto ao da cinemática direta: dada uma pose desejada para o punho, determinar quais valores de junta o manipulador deve assumir para alcançá-la. Esse problema é central em qualquer tarefa de manipulação, pois normalmente os comandos humanos ou de alto nível são especificados no espaço cartesiano. Transformar essas metas espaciais em ângulos ou deslocamentos de junta é, portanto, essencial para a execução prática do movimento.

A cinemática inversa, ao contrário da cinemática direta, pode apresentar múltiplas soluções, nenhuma solução ou soluções não triviais, especialmente em robôs com redundâncias ou com restrições geométricas. Todos os sistemas com até 6 graus de liberdade compostos por juntas rotacionais ou prismáticas em uma única cadeia em série são solucionáveis.

Para manipuladores com até 4 graus de liberdade ou estrutura geométrica particular é possível derivar soluções analíticas fechadas. Se 3 eixos consecutivos se cruzam em um ponto, ou são paralelos, é possível aplicar a solução de Pieper para obter uma solução fechada para manipuladores de 6 graus de liberdade. Porém, para manipuladores com mais de 6 graus de liberdade vão existir múltiplas soluções e soluções analíticas tornam-se impraticáveis ou inexistentes devido ao alto acoplamento não linear. Nesses casos, métodos numéricos são utilizados.

2.4 Jacobiano e Cinemática Diferencial

O Jacobiano é uma ferramenta central no estudo e controle de manipuladores robóticos, pois estabelece a relação entre velocidade no espaço cartesiano e no espaço das juntas dada por:

$$\dot{\mathbf{x}} = J(\mathbf{q})\,\dot{\mathbf{q}},$$

Onde $\dot{\mathbf{x}}$ representa a velocidade cartesiana, $\dot{\mathbf{q}}$ o vetor de velocidades das juntas e $J(\mathbf{q})$ o Jacobiano avaliada na configuração atual. Cada coluna de J indica como a velocidade de uma junta influencia diretamente o movimento do punho. O Jacobiano pode ser escrita como:

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix},$$

Onde J_v contém as contribuições de velocidade linear e J_ω as contribuições de velocidade angular.

Assim, para juntas de revolução, temos:

$$J_v^i = z_{i-1} \times (p_e - p_{i-1}), \qquad J_\omega^i = z_{i-1},$$

Enquanto para juntas prismáticas vale:

$$J_v^i = z_{i-1}, \qquad J_\omega^i = 0,$$

Em que z_{i-1} é o vetor que define o eixo da junta i-1, p_e é a posição da junta i e p_{i-1} é a posição do eixo i-1.

Para a encontrar a velocidade das juntas dada uma velocidade no espaço cartesiano, é preciso utilizar a relação inversa:

$$\dot{\mathbf{q}} = J^{-1}(\mathbf{q})\dot{\mathbf{x}}$$

Contudo, o Jacobiano nem sempre será invertível e pode não ser uma matriz quadrada.

2.4.1 Manipuladores de 6 Juntas de Revolução

Manipuladores com seis juntas podem alcançar posições e a orientações genéricas dentro do espaço de trabalho. Descrevendo a orientação em termos dos ângulos

de euler temos uma relação de um espaço cartesiano com 6 dimensões, 3 para posição (x, y, z) e 3 para orientação (α, β, γ), e um espaço das juntas com 6 graus de liberdade.

Para esse tipo de manipulador temos:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \\ \alpha \\ \beta \\ \gamma \end{bmatrix} \; com \; \dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} \; e \; \mathbf{q} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix} \; com \; \dot{\mathbf{q}} = \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \\ \dot{\theta_3} \\ \dot{\theta_4} \\ \dot{\theta_5} \\ \dot{\theta_6} \end{bmatrix}$$

Nesse caso, o Jacobiano é uma matriz quadrada com dimensões 6×6 e, quando longe de singularidades, é invertível.

2.4.2 Manipuladores com 5 Juntas de Revolução

Para manipuladores com 5 graus de liberdade, a matriz jacobiana não é quadrada e portanto não é possível calcular diretamente a sua inversa.

$$J(\mathbf{q}) \in \mathbb{R}^{6 \times 5},$$

A pseudoinversa pode ser utilizada como uma aproximação para a inversa do Jacobiano e nos permite relacionar a velocidade no espaço cartesiano com a velocidade no espaço das juntas.

$$\dot{\mathbf{q}} = J^{\dagger} \, \dot{\mathbf{x}}.$$

2.4.3 Cinemática Diferencial

A cinemática diferencial é abordagem interessante para a resolução do problema da cinemática inversa para manipuladores em que a solução analítica não é possível ou não é conveniente. Dado que a relação entre velocidade no espaço cartesiano e velocidade das juntas é conhecida, é possível determinar a solução da cinemática inversa por meio da integração numérica. Podemos aproximar a posição da junta no passo k+1 de integração por meio da equação:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \dot{\mathbf{q}} \, dt.$$

Substituindo a relação inversa do Jacobiano é possível obter:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + J^{\dagger}(\mathbf{q}_k) \, \dot{\mathbf{v}}_k \, dt$$

.

Dado um valor inicial para a posição das juntas, um intervalo de tempo de integração arbitrário e uma velocidade no espaço cartesiano em direção a posição desejada, é possível obter uma nova posição das juntas que é mais próxima do alvo. A cada iteração é calculada a cinemática direta para que se possa verificar se o objetivo final foi atingido. Ao escolher a velocidade no espaço cartesiano e o valor de tempo de integração é importante notar que para valores muito altos a solução pode não convergir e para valores muito pequenos a solução pode levar um grande número de iteraçãoes.

2.5 Screw Theory

A teoria de *screws* fornece uma estrutura geométrica unificada para descrever movimentos de corpos rígidos no espaço tridimensional. Seu elemento fundamental é o *twist*, que representa o movimento instantâneo de um corpo, combinando rotação e translação em uma única entidade matemática.

2.5.1 Twist de uma Junta de Revolução

Um *twist* é um vetor de seis dimensões que caracteriza a velocidade instantânea de um corpo rígido. No caso de uma junta de revolução, o twist espacial assume a forma:

$$S = \begin{bmatrix} \omega \\ \omega \times q \end{bmatrix},$$

onde ω é um vetor unitário que define a direção do eixo de rotação e q é um ponto pertencente a esse eixo. Essa construção garante que o movimento descrito corresponda a uma rotação em torno da linha determinada por (ω, q), preservando a geometria do eixo físico da junta.

2.5.2 Exponencial de um Twist

A transformação associada ao movimento de uma junta é obtida pela exponencial matricial do twist:

$$e^{[S]\theta}.$$

Para uma junta de revolução, essa exponencial descreve uma rotação de ângulo θ em torno do eixo ω, acompanhada de uma translação que mantém a coerência com a linha do parafuso (*screw axis*).

2.5.3 Produto das Exponenciais

A cinemática direta de um manipulador com n juntas pode ser expressa pelo modelo do Produto das Exponenciais:

$$T(\theta_1, \theta_2, \ldots, \theta_n) = e^{[S_1]\theta_1} e^{[S_2]\theta_2} \cdots e^{[S_n]\theta_n} M_0,$$

onde cada termo $e^{[S_i]\theta_i}$ representa o movimento helicoidal da junta i e M_0 é a pose do end-effector quando todas as juntas estão em sua configuração inicial.

2.6 Aprendizagem de Máquina

Atualmente, métodos de aprendizagem de máquina tem sido bastante usados em manipulação robótica, complementando e, em alguns casos, até superando métodos clássicos em cenários complexos e pouco estruturados. Enquanto abordagens tradicionais dependem de modelos precisos, técnicas baseadas em dados permitem lidar melhor com incertezas de percepção, contatos, ruídos e propriedades físicas difíceis de modelar.

Redes neurais profundas podem, por exemplo, fornecer estimativas de forma e pose, facilitando a identificação e caracterização de objetos. Por outro lado, algoritmos de aprendizagem por reforço possibilitam que robôs adquiram habilidades diretamente por interação, aprendendo a lidar com fenômenos que podem ser difíceis de modelar por métodos tradicionais. Embora ainda existam desafios

de eficiência e estabilidade, a combinação de princípios clássicos com componentes aprendidos tem se mostrado uma boa estratégia para aumentar a autonomia e a adaptabilidade dos manipuladores.

3 Materiais

Para este trabalho foi utilizado o Robotics Toolbox for Python, desenvolvido por Peter Corke, como ferramenta principal de simulação do manipulador. Esse toolbox fornece recursos para modelagem de robôs, permitindo criar modelos via parâmetros DH, calcular cinemática direta e inversa, gerar trajetórias e visualizar os movimentos em ambiente virtual.

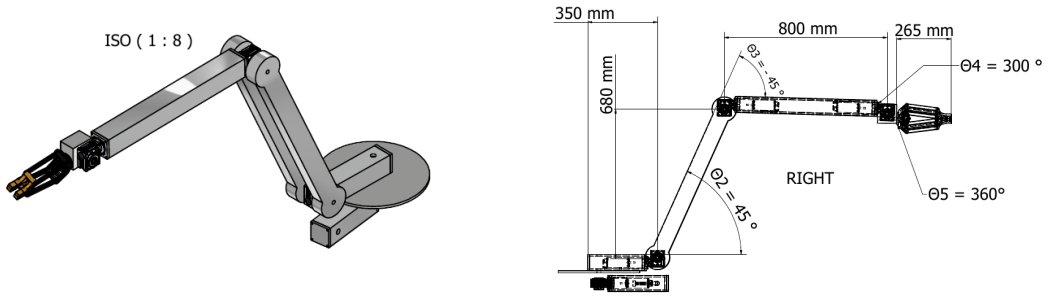
4 Metodologia e Resultados

Para este trabalho, foi utilizado o robô manipulador composto por cinco juntas de revolução, denominado FRANK. O modelo computacional do robô pode ser observado na Figura 1a, enquanto a Figura 1b apresenta uma vista complementar contendo as principais dimensões utilizadas no desenvolvimento do modelo.

Figura 1: Robô manipulador proposto para este trabalho.

(a) FRANK.

(b) Vista com medidas.



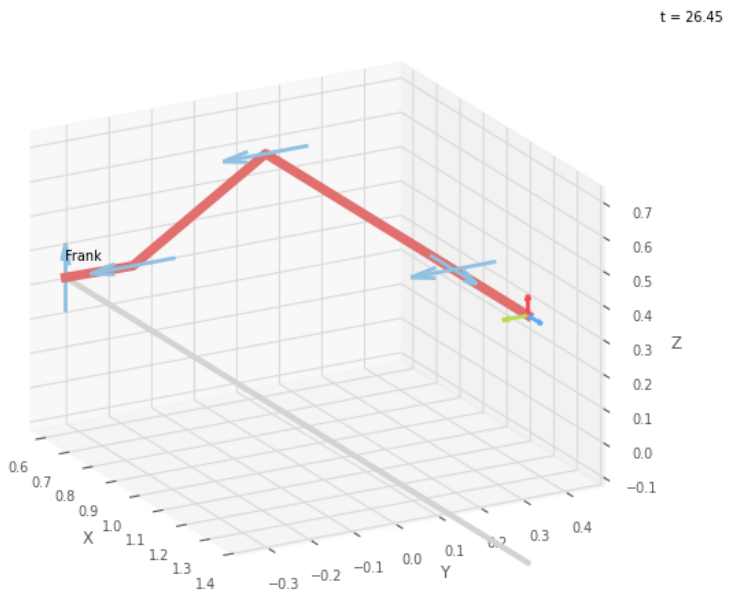
Fonte: elaboração dos autores.

A metodologia adotada incluiu a determinação dos parâmetros de Denavit–Hartenberg do manipulador por meio da definição dos sistemas de coordenadas e identificação das medidas necessárias, seguida da implementação do modelo completo do robô na Robotics Toolbox de Peter Corke. Com o modelo definido, foram obtidos os cálculos da Jacobiana e aplicada a cinemática inversa diferencial para relacionar velocidades articulares e do *end-effector*, além da realização da cinemática direta por Screw Theory como abordagem complementar. Também, foi desenvolvido uma simulação de Pick and Place na toolbox, permitindo verificar a coerência entre a modelagem teórica e o desempenho do robô em tarefas de manipulação. Por fim, a Robotics Toolbox foi utilizada para encontrar a cinemática inversa para um conjunto de poses do end-effector e interpolar trajetórias de modo a construir um dataset que relaciona posição das juntas com posição do punho. Esses dados poderão ser utilizados futuramente no treinamento de modelos de aprendizagem de máquina.

4.1 Parâmetros de Denavit-Hartenberg

Primeiramente, foram determinados os parâmetros de Denavit–Hartenberg correspondentes a cada elo e junta do robô. Para isso, adotou-se o procedimento de definição dos sistemas de coordenadas, identificação dos eixos das juntas e medição das distâncias e ângulos necessários. Na figura 4 é possível observar os eixos definidos para cada uma das 5 juntas.

Figura 2: Modelo do robô FRANK no Robotics Toolbox.



Fonte: elaboração dos autores.

A partir dessas informações, foi construída a Tabela 1 com os parâmetros de Denavit–Hartenberg, que serviu como base para a modelagem do manipulador.

Tabela 1: Parâmetros de Denavit-Hartenberg para o robô FRANK.

θ_j	d_j	a_j	α_j
θ_1	0,155	0,287	$\frac{\pi}{2}$
$\theta_2 + \frac{\pi}{2}$	0	0,8	0
θ_3	0	0,8	0
$\theta_4 + \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
θ_5	0,317	0	0

Fonte: elaboração dos autores.

4.2 Implementação do Robô em Simulação

Após a definição dos parâmetros de Denavit-Hartenberg, o modelo completo do robô FRANK foi implementado utilizando a Robotics Toolbox de Peter Corke. Essa ferramenta permitiu representar o manipulador por meio da sequência de

elos definidos pela tabela DH, facilitando a validação dos parâmetros obtidos e a realização de simulações.

O trecho de código do Código 1 cria cada elo do robô FRANK diretamente a partir dos valores numéricos dos parâmetros de Denavit–Hartenberg da Tabela 1, utilizando as classes DHLink e DHRobot da Robotics Toolbox. As variáveis l_i definem as características geométricas de um elo, e, ao final, todos são reunidos em um único objeto DHRobot chamado de robot, que representa o manipulador no ambiente Python e permite executar operações como cálculo de cinemática e manipulação do modelo.

Código 1: Definição do robô FRANK

```
import numpy as np
import roboticstoolbox as rtb

l1 = rtb.DHLink(a=0.287, alpha=np.pi/2, d=0.155, offset=0)
l2 = rtb.DHLink(a=0.8, alpha=0, d=0, offset=np.pi/2)
l3 = rtb.DHLink(a=0.8, alpha=0, d=0, offset=0)
l4 = rtb.DHLink(a=0, alpha=np.pi/2, d=0.0, offset=np.pi/2)
l5 = rtb.DHLink(a=0, alpha=0, d=0.317, offset=0)
robot = rtb.DHRobot([l1, l2, l3, l4, l5], name="Frank")
print(robot)
q = [0, -np.pi/4, -np.pi/4, 0, 0]
robot.plot(q, backend='pyplot', block=True)
```

Esse código constroi a classe do robô FRANK e mostra na tela a Tabela 1 por meio do comando 'print(robot)' e gera a figura 4 com o comando 'robot.plot(q, backend=\'pyplot\', block=True)'.

4.3 Cálculos da Jacobiana

Com o modelo estabelecido, foram desenvolvidos os cálculos da Jacobiana do robô FRANK. A matriz Jacobiana foi posteriormente utilizada nos cálculo da cinemática inversa diferencial, permitindo relacionar velocidades do end-effector com velocidades das juntas.

O cálculo da Jacobiana no Código 2 é realizado construindo-se primeiramente as matrizes de transformação homogênea de cada elo utilizando a biblioteca *SymPy*, definidas a partir dos parâmetros das juntas.

Código 2: Definição das variáveis para cálculo de Jacobiana do manipulador.

```
from sympy import *
import numpy as np

t1, t2, t3, t4, t5 = symbols('theta1 theta2 theta3 theta4
   theta5')
d1, a1, a2, a3, d5 = symbols('d1 a1 a2 a3 d5')
```

Em seguida são definidas as tranformação entre as juntas:

Código 3: Definição das transformações.

```
T01 = Matrix([
    [cos(t1),    0,  sin(t1), cos(t1)*a1],
    [sin(t1),    0, -cos(t1), sin(t1)*a1],
    [      0,    1,         0,         d1],
```

```
    [      0,   0,         0,           1]])

T12 = Matrix([
    [-sin(t2), -cos(t2),  0, -sin(t2)*a2],
    [ cos(t2), -sin(t2),  0,  cos(t2)*a2],
    [       0,        0,  1,           0],
    [       0,        0,  0,           1]])

T23 = Matrix([
    [cos(t3), -sin(t3),  0, cos(t3)*a3],
    [sin(t3),  cos(t3),  0, sin(t3)*a3],
    [      0,        0,  1,          0],
    [      0,        0,  0,          1]])

T34 = Matrix([
    [-sin(t4),  0, cos(t4),  0],
    [ cos(t4),  0, sin(t4),  0],
    [       0,  1,       0,  0],
    [       0,  0,       0,  1]])

T45 = Matrix([
    [cos(t5), -sin(t5),  0,  0],
    [sin(t5),  cos(t5),  0,  0],
    [      0,        0,  1, d5],
    [      0,        0,  0,  1]])

T05 = T01*T12*T23*T34*T45
T05 = simplify(T05)
```

Então, essas transformações são multiplicadas para obter a pose final do end-effector e também para determinar a posição da origem dos sistema de coordenadas de cada junta no espaço, O_1, O_2, O_3, O_4, O_5, em relação ao sistema de coordenadas da base.

Código 4: Cálculo do centro das juntas.

```
Origin = Matrix ([[0], [0], [0], [1]])
O1 = T01 * Origin
O1 = O1[0:3, :]
O2 = T01* T12 * Origin
O2 = simplify(O2[0:3, :])
O3 = T01* T12 * T23 * Origin
O3 = simplify(O3[0:3, :])
O4 = T01* T12 * T23 * T34 * Origin
O4 = simplify(O4[0:3, :])
O5 = T01* T12 * T23 * T34 * T45 * Origin
O5 = simplify(O5[0:3, :])
```

A figura 3 mostra a origem dos eixos em sequencia da esqueda para a direita e de cima para baixo:

Para cada elo, extrai-se também o vetor de direção do eixo da junta de Z_1 até Z_5, obtidos aplicando as matrizes de rotação nos eixos subsequentes. Seguindo a notação de Denavit Hartenberg, todos os eixos são posicionados na direção Z de seus respectivos sistemas de coordenadas.

Figura 3: Origem dos eixos das juntas no sistema de coordenadas da base.

$$\left(\begin{bmatrix} a_1\cos(\theta_1) \\ a_1\sin(\theta_1) \\ d_1 \end{bmatrix}, \begin{bmatrix} (a_1 - a_2\sin(\theta_2))\cos(\theta_1) \\ (a_1 - a_2\sin(\theta_2))\sin(\theta_1) \\ a_2\cos(\theta_2) + d_1 \end{bmatrix}, \begin{bmatrix} (a_1 - a_2\sin(\theta_2) - a_3\sin(\theta_2+\theta_3))\cos(\theta_1) \\ (a_1 - a_2\sin(\theta_2) - a_3\sin(\theta_2+\theta_3))\sin(\theta_1) \\ a_2\cos(\theta_2) + a_3\cos(\theta_2+\theta_3) + d_1 \end{bmatrix}\right)$$

$$\left(\begin{bmatrix} (a_1 - a_2\sin(\theta_2) - a_3\sin(\theta_2+\theta_3))\cos(\theta_1) \\ (a_1 - a_2\sin(\theta_2) - a_3\sin(\theta_2+\theta_3))\sin(\theta_1) \\ a_2\cos(\theta_2) + a_3\cos(\theta_2+\theta_3) + d_1 \end{bmatrix}, \begin{bmatrix} (a_1 - a_2\sin(\theta_2) - a_3\sin(\theta_2+\theta_3) - d_5\sin(\theta_2+\theta_3+\theta_4))\cos(\theta_1) \\ (a_1 - a_2\sin(\theta_2) - a_3\sin(\theta_2+\theta_3) - d_5\sin(\theta_2+\theta_3+\theta_4))\sin(\theta_1) \\ a_2\cos(\theta_2) + a_3\cos(\theta_2+\theta_3) + d_1 + d_5\cos(\theta_2+\theta_3+\theta_4) \end{bmatrix}\right)$$

Fonte: elaboração dos autores.

Código 5: Cálculo do eixo das juntas.

```
ZAxis = Matrix ([[0], [0], [1]])
Z1 = ZAxis
Z2 = T01[0:3, 0:3] * ZAxis
Z3 = T01[0:3, 0:3] * T12[0:3, 0:3] * ZAxis
Z4 = T01[0:3, 0:3] * T12[0:3, 0:3] * T23[0:3, 0:3] * ZAxis
Z5 = T01[0:3, 0:3] * T12[0:3, 0:3] * T23[0:3, 0:3] * T34
   [0:3, 0:3] * ZAxis
Z5 = simplify(Z5)
```

A Figura 4 mostra os eixos de cada junta em relação ao sistema de coordenadas da base:

Figura 4: Eixos das juntas em relação ao sistema de coordenadas da base.

$$\left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} \sin(\theta_1) \\ -\cos(\theta_1) \\ 0 \end{bmatrix}, \begin{bmatrix} \sin(\theta_1) \\ -\cos(\theta_1) \\ 0 \end{bmatrix}, \begin{bmatrix} \sin(\theta_1) \\ -\cos(\theta_1) \\ 0 \end{bmatrix}, \begin{bmatrix} -\sin(\theta_2+\theta_3+\theta_4)\cos(\theta_1) \\ -\sin(\theta_1)\sin(\theta_2+\theta_3+\theta_4) \\ \cos(\theta_2+\theta_3+\theta_4) \end{bmatrix}\right)$$

Fonte: elaboração dos autores.

Com as posições O_i e os vetores Z_i, o código calcula as colunas da Jacobiana: a parte linear é dada pelo produto vetorial entre o eixo da junta e o vetor que liga a junta ao end-effector, enquanto a parte angular corresponde diretamente aos vetores Z_i. Por fim, todas as colunas são reunidas em uma única matriz J, que é simplificada simbolicamente e posteriormente avaliada numericamente por meio da substituição dos valores reais. O resultado foi comparado com o que é obtido numericamente pela Robotics Toolbox.

Código 6: Calculo das componentes do Jacobiano.

```
JV1 = Z1.cross(O5)
JW1 = Z1

JV2 = simplify(Z2.cross(O5 - O1))
JW2 = Z2

JV3 = simplify(Z3.cross(O5 - O2))
JW3 = Z3

JV4 = simplify(Z4.cross(O5 - O3))
JW4 = Z4

JV5 = simplify(Z5.cross(O5 - O4))
JW5 = Z5
```

```
J = simplify(Matrix([
    [JV1, JV2, JV3, JV4, JV5],
    [JW1, JW2, JW3, JW4, JW5]]))

# Calculando a matriz Jacobiana para comparar com a
   Robotics Toolbox
J.subs({t1: np.pi/6, t2: np.pi/6, t3: np.pi/6, t4: np.pi/6,
    t5: np.pi/6, d1: 0.155, a1: 0.287, a2: 0.8, a3: 0.8, d5
   : 0.317})
```

A matriz Jacobiana obtida a partir do desenvolvimento simbólico das transformações homogêneas e dos vetores de posição e orientação das juntas resulta na seguinte expressão:

$$
\begin{aligned}
J_{1,1} &= \left(-a_1 + a_2 S_2 + a_3 S_{23} + d_5 S_{234}\right) S_1 \\
J_{1,2} &= -\left(a_2 C_2 + a_3 C_{23} + d_5 C_{234}\right) C_1 \\
J_{1,3} &= -\left(a_3 C_{23} + d_5 C_{234}\right) C_1 \\
J_{1,4} &= -d_5 C_1 C_{234} \\
J_{1,5} &= 0
\end{aligned}
$$

$$
\begin{aligned}
J_{2,1} &= \left(a_1 - a_2 S_2 - a_3 S_{23} - d_5 S_{234}\right) C_1 \\
J_{2,2} &= -\left(a_2 C_2 + a_3 C_{23} + d_5 C_{234}\right) S_1 \\
J_{2,3} &= -\left(a_3 C_{23} + d_5 C_{234}\right) S_1 \\
J_{2,4} &= -d_5 S_1 C_{234} \\
J_{2,5} &= 0
\end{aligned}
$$

$$
\begin{aligned}
J_{3,1} &= 0 \\
J_{3,2} &= -a_2 S_2 - a_3 S_{23} - d_5 S_{234} \\
J_{3,3} &= -a_3 S_{23} - d_5 S_{234} \\
J_{3,4} &= -d_5 S_{234} \\
J_{3,5} &= 0
\end{aligned}
$$

$$
\begin{aligned}
J_{4,1} &= 0 \\
J_{4,2} &= S_1 \\
J_{4,3} &= S_1 \\
J_{4,4} &= S_1 \\
J_{4,5} &= -S_{234} C_1
\end{aligned}
$$

$$
\begin{aligned}
J_{5,1} &= 0 \\
J_{5,2} &= -C_1 \\
J_{5,3} &= -C_1 \\
J_{5,4} &= -C_1 \\
J_{5,5} &= -S_1 S_{234}
\end{aligned}
$$

$$
\begin{aligned}
J_{6,1} &= 1 \\
J_{6,2} &= 0 \\
J_{6,3} &= 0 \\
J_{6,4} &= 0 \\
J_{6,5} &= C_{234}
\end{aligned}
$$

$$
\mathbf{J} = \begin{bmatrix}
J_{1,1} & J_{1,2} & J_{1,3} & J_{1,4} & J_{1,5} \\
J_{2,1} & J_{2,2} & J_{2,3} & J_{2,4} & J_{2,5} \\
J_{3,1} & J_{3,2} & J_{3,3} & J_{3,4} & J_{3,5} \\
J_{4,1} & J_{4,2} & J_{4,3} & J_{4,4} & J_{4,5} \\
J_{5,1} & J_{5,2} & J_{5,3} & J_{5,4} & J_{5,5} \\
J_{6,1} & J_{6,2} & J_{6,3} & J_{6,4} & J_{6,5}
\end{bmatrix}
$$

Na matriz acima, **J**, para simplificar a escrita, adotou-se a seguinte convenção de notação:

- $S_i = \sin(\theta_i)$, ou seja, o seno do ângulo da junta i;
- $C_i = \cos(\theta_i)$, ou seja, o cosseno do ângulo da junta i;
- $S_{ij} = \sin(\theta_i + \theta_j)$ e $C_{ij} = \cos(\theta_i + \theta_j)$, representando a soma de ângulos de duas juntas consecutivas; de forma análoga para S_{ijk}, C_{ijk}, etc.;
- Os parâmetros a_k representam os comprimentos dos elos, d_k as distâncias ao longo do eixo das juntas prismáticas ou deslocamentos fixos, e θ_k os ângulos das juntas.

Essa notação foi usada para tornar equações de cinemática mais legíveis, evitando expressões longas com múltiplos senos e cossenos.

4.4 Cinemática Inversa Diferencial

A cinemática inversa diferencial utiliza a relação entre as velocidades cartesianas e das juntas, através da matriz Jacobiana, para conduzir o manipulador até uma pose desejada de forma incremental. Em vez de resolver diretamente as posições das juntas, o método ajusta a posição das juntas passo a passo a partir do erro entre a pose atual e a pose final.

O Código 7 implementa a cinemática inversa diferencial ao iterativamente ajustar as juntas de acordo com o erro entre a pose atual e a pose desejada do end-effector. O procedimento inicia com o cálculo da posição e orientação associadas à configuração inicial, a partir da qual são definidas velocidades lineares proporcionais ao erro de posição. Essas velocidades servem como referência para o processo iterativo.

Em cada iteração, a função `jacobian_ik` é chamada para obter a Jacobiana na configuração corrente e montar o vetor de velocidades cartesianas, combinando componentes lineares e angulares. Em seguida, calcula-se a pseudoinversa da Jacobiana para determinar as velocidades articulares que melhor realizam o movimento desejado no espaço cartesiano. As juntas são então atualizadas pela integração dessas velocidades ao longo de um pequeno intervalo de tempo.

Após cada atualização, o código recalcula a pose do robô e verifica se a posição e a orientação estão suficientemente próximas dos valores desejados. Caso o erro permaneça acima do limite admissível, uma nova iteração é realizada com o erro de posição atualizado, refinando progressivamente o movimento. O processo continua até que se atinja a convergência ou que o número máximo de iterações seja alcançado, resultando na configuração de juntas que melhor aproxima a pose final especificada.

Código 7: Cinemática inversa diferencial

```
import numpy as np
import roboticstoolbox as rtb

def differential_ik(robot: rtb.DHRobot, qi, final_pos,
   final_ori):
    initial_fk = robot.fkine(qi)
    initial_pos = initial_fk.t
    initial_ori = initial_fk.rpy()
    max_iterations = 100
    lin_gain = 2.0
    ang_gain = 0.2
    lin_vel = [lin_gain * (x[0] - x[1]) for x in zip(
       final_pos, initial_pos)]
    ang_vel = [0,0,0]

    for i in range(0, max_iterations):
        qn = jacobian_ik(robot, qi, lin_vel=lin_vel,
           ang_vel=ang_vel, dt=0.1)
        curr_pos = robot.fkine(qn)
        print(f"Position on iteration {i}: {[format(i, '.3f
           ') for i in curr_pos.t]}")

        reached_pos = np.allclose(final_pos, curr_pos.t,
           atol=1e-3)
        reached_angle = np.allclose(final_ori, curr_pos.rpy
           (),atol=1e-3)

        if reached_pos and reached_angle: break

        qi = qn
        lin_vel = [lin_gain * (x[0] - x[1]) for x in zip(
           final_pos, curr_pos.t)]

    return qn

def jacobian_ik(robot: rtb.DHRobot, q, lin_vel, ang_vel, dt
   ):
    J = robot.jacob0(q)
```

```
    v = np.zeros(6)
    v[:3] = lin_vel
    v[3:] = ang_vel
    dq = np.linalg.pinv(J) @ v
    nq = q + dq * dt
    return nq
```

Para posições próximas de singularidades a solução não apresenta uma boa convergência e para um pequeno deslocamento linear um grande deslocamento das juntas é produzido. Técnicas de otimização podem ser aplicadas para solucionar esse problema como adicionar restrições de velocidade, posição e aceleração. Na implementação proposta no Código 7 o termo da velocidade angular foi omitido para melhorar a convergência do método.

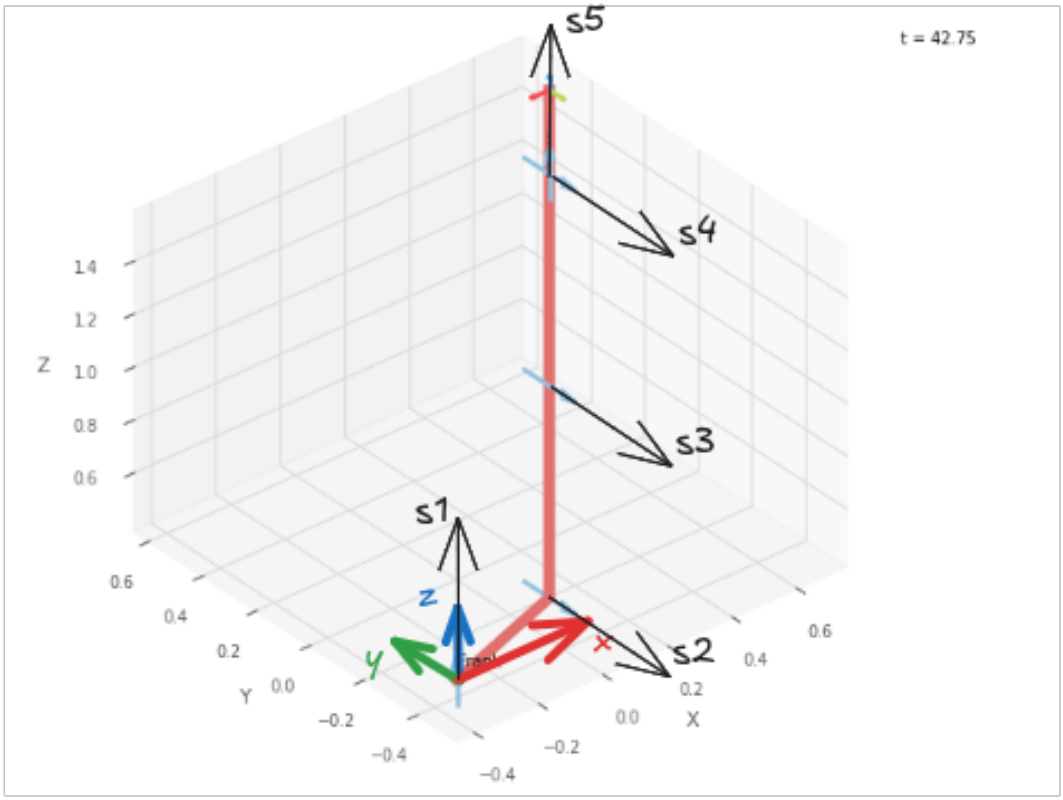
Para uma movimentação da posição (1.6527, 0, 0.4037) para a posição (1.2, 0.4, 0.5) a cinemática inversa converge após 27 iterações como mostrado nos logs abaixo:

```
Initial position: [1.6527 -0.0000 0.4037]
Initial orientation: [3.1416 -0.0000 -0.0000]
Final position: [1.2, 0.4, 0.5]
Final orientation: [3.141592653589793, 0, 0]
Linear Velocity: [-0.9053708498984765, 0.8,
   0.19262915010152393]
Angular Velocity: [0, 0, 0]
Position on iteration 0: ['1.550', '0.075', '0.415']
Position on iteration 1: ['1.476', '0.137', '0.429']
Position on iteration 2: ['1.419', '0.187', '0.442']
Position on iteration 3: ['1.374', '0.228', '0.452']
Position on iteration 4: ['1.339', '0.262', '0.461']
Position on iteration 5: ['1.311', '0.289', '0.469']
Position on iteration 6: ['1.288', '0.311', '0.475']
Position on iteration 7: ['1.270', '0.328', '0.480']
Position on iteration 8: ['1.256', '0.342', '0.484']
Position on iteration 9: ['1.245', '0.354', '0.487']
Position on iteration 10: ['1.236', '0.363', '0.490']
Position on iteration 11: ['1.229', '0.370', '0.492']
Position on iteration 12: ['1.223', '0.376', '0.493']
Position on iteration 13: ['1.218', '0.381', '0.495']
Position on iteration 14: ['1.215', '0.385', '0.496']
Position on iteration 15: ['1.212', '0.388', '0.497']
Position on iteration 16: ['1.209', '0.390', '0.497']
Position on iteration 17: ['1.208', '0.392', '0.498']
Position on iteration 18: ['1.206', '0.394', '0.498']
Position on iteration 19: ['1.205', '0.395', '0.499']
Position on iteration 20: ['1.204', '0.396', '0.499']
Position on iteration 21: ['1.203', '0.397', '0.499']
Position on iteration 22: ['1.202', '0.397', '0.499']
Position on iteration 23: ['1.202', '0.398', '0.499']
Position on iteration 24: ['1.202', '0.398', '0.500']
Position on iteration 25: ['1.201', '0.399', '0.500']
Position on iteration 26: ['1.201', '0.399', '0.500']
Position on iteration 27: ['1.201', '0.399', '0.500']
Final IK position is:  [0.3209 -0.2358 -1.4807 -1.4252
   0.3209]
```

4.5 Cinemática com Screw Theory

Foi calculada a cinemática direta utilizando a Screw Theory, proporcionando uma abordagem alternativa à formulação clássica baseada em DH. Para isso, primeiramente o robô foi movido para a posição em que todas as juntas são 0 conforme a figura 5. Nessa posição foram definidos os sistemas de coordenadas da base e da ferramenta além dos eixos de screw s_n.

Figura 5: FRANK na posição zero com eixos de screw.



Fonte: elaboração dos autores.

Nessa posição foi definida a matriz M que tem a transformação da base até a origem do sistema de coordenadas da ferramenta. Essa transformação foi calculada usando as dimensões conhecidas do manipulador.

$$M = \begin{bmatrix} -1 & 0 & 0 & 0.287 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 2.072 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Em seguida foram calculadas a posição de pontos arbitrários a_n no eixo das juntas na posição zero. Foram escolhidos os pontos definidos como o centro do sistema de coordenadas utilizados na notação de Denavit Hartenberg por conveniencia.

$$a_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \; a_2 = \begin{bmatrix} 0.287 \\ 0 \\ 0.155 \end{bmatrix}, \; a_3 = \begin{bmatrix} 0.287 \\ 0 \\ 0.955 \end{bmatrix}, \; a_4 = \begin{bmatrix} 0.287 \\ 0 \\ 1.755 \end{bmatrix}, \; a_5 = \begin{bmatrix} 0.287 \\ 0 \\ 2.072 \end{bmatrix}.$$

Em seguida foram calculados os eixos de screw s_n expressos no sistema de coordenadas da base do robô. Primeiramente é calculada a parte angular do screw, $s_{\omega n}$. Como se tratam de juntas rotacionais, basta apenas escrever o eixo da junta no sistema de coordenadas da base.

$$s_{\omega 1} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \; s_{\omega 2} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \; s_{\omega 3} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \; s_{\omega 4} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \; s_{\omega 5} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

A parte linear do screw pode ser encontrada usando a equação $S_{vn} = -S_{\omega n} \times a_n$:

$$s_{v1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \; s_{v2} = \begin{bmatrix} 0.155 \\ 0 \\ -0.287 \end{bmatrix}, \; s_{v3} = \begin{bmatrix} 0.955 \\ 0 \\ -0.287 \end{bmatrix}, \; s_{v4} = \begin{bmatrix} 1.755 \\ 0 \\ -0.287 \end{bmatrix}, \; s_{v5} = \begin{bmatrix} 0 \\ -0.287 \\ 0 \end{bmatrix}.$$

O eixo de screw é obtido concatenando a porção angular e linear:

$$s_n = \begin{bmatrix} s_{\omega n} \\ s_{vn} \end{bmatrix}.$$

A tranformação é dado então por:

$$T(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5) = e^{[s_1]\theta_1} e^{[s_2]\theta_2} e^{[s_3]\theta_3} e^{[s_4]\theta_4} e^{[s_5]\theta_5}$$

Para efetuar essa operação é preciso converter o vetor 6D para uma matriz de transforamação homogênea. A porção rotacional da matriz é dada pela matriz antissimétrica da porção angular do screw e a tranlação é dada pela porção linear no screw conforme inplementado na função '*twist_to_matrix*'.

O Código 8 abaixo fornece a implementação completa seguindo os passos descritos anteriormente.

Código 8: Cinemática com Screw Theory

```
import numpy as np
from scipy.linalg import expm

def to_skew_symmetric(omega):
    return np.array([
        [0, -omega[2], omega[1]],
        [omega[2], 0, -omega[0]],
        [-omega[1], omega[0], 0]
    ])

def twist_to_matrix(S):
    # Converte o vetor de Twist S (6x1) para a matriz [S]
       (4x4)
    S_flat = np.ravel(S)

    omega = S_flat[:3]
    v = S_flat[3:]

    omega_skew = to_skew_symmetric(omega)
```

```
    # Cria a matriz 4x4
    S_matrix = np.zeros((4, 4))
    S_matrix[:3, :3] = omega_skew
    S_matrix[:3, 3] = v

    return S_matrix

M = np.matrix([
    [-1,  0, 0, 0.287],
    [ 0, -1, 0,     0],
    [ 0,  0, 1, 2.072],
    [ 0,  0, 0,     1]])

a1 = np.matrix([0, 0, 0])
a2 = np.matrix([0.287, 0, 0.155])
a3 = np.matrix([0.287, 0, 0.955])
a4 = np.matrix([0.287, 0, 1.755])
a5 = np.matrix([0.287, 0, 2.072])

Sw1 = np.matrix([0, 0, 1])
Sw2 = np.matrix([0, -1, 0])
Sw3 = np.matrix([0, -1, 0])
Sw4 = np.matrix([0, -1, 0])
Sw5 = np.matrix([0, 0, 1])

Sv1 = -np.cross(Sw1, a1)
Sv2 = -np.cross(Sw2, a2)
Sv3 = -np.cross(Sw3, a3)
Sv4 = -np.cross(Sw4, a4)
Sv5 = -np.cross(Sw5, a5)

S1 = np.concatenate((Sw1, Sv1), axis=1)
S2 = np.concatenate((Sw2, Sv2), axis=1)
S3 = np.concatenate((Sw3, Sv3), axis=1)
S4 = np.concatenate((Sw4, Sv4), axis=1)
S5 = np.concatenate((Sw5, Sv5), axis=1)

theta1 = 0
theta2 = -np.pi/4
theta3 = -np.pi/4
theta4 = 0
theta5 = 0

T1 = expm(twist_to_matrix(S1) * theta1)
T2 = expm(twist_to_matrix(S2) * theta2)
T3 = expm(twist_to_matrix(S3) * theta3)
T4 = expm(twist_to_matrix(S4) * theta4)
T5 = expm(twist_to_matrix(S5) * theta5)

T = T1 @ T2 @ T3 @ T4 @ T5 @ M

float_formatter = "{:.3f}".format
np.set_printoptions(formatter={'float_kind':
```

```
   float_formatter})
print(T)
```

A transformação obtida T é identica ao resultado utilizando a Robotics Toolbox para os ângulos de (0, -π/4, -π/4, 0 , 0) que correspondem a posição inicial do FRANK.

$$T = \begin{bmatrix} 0 & 0 & 1 & 1.970 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0.721 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

4.6 Algoritmo de Pick and Place

Também, foi desenvolvido um experimento de Pick and Place utilizando a Robotics Toolbox de Peter Corke, no qual o modelo do FRANK foi empregado para executar trajetórias e manipular objetos dentro do ambiente simulado. A cinemática inversa foi calculada utilizando a cinemática inversa diferencial. A geração da trajetória foi feita utilizando a função 'jtraj' da Robotics Toolbox.

Código 9: Algoritmo de Pick and Place

```
import roboticstoolbox as rtb
import numpy as np
from ik import differential_ik

# Print do numpy
float_formatter = "{:.4f}".format
np.set_printoptions(formatter={'float_kind':
   float_formatter})

L1 = rtb.RevoluteDH(a=0.287, alpha=np.pi/2, d=0.155, offset
   =0)
L2 = rtb.RevoluteDH(a=0.8, alpha=0, d=0, offset=np.pi/2)
L3 = rtb.RevoluteDH(a=0.8, alpha=0, d=0.0, offset=0)
L4 = rtb.RevoluteDH(a=0.0, alpha=np.pi/2, d=0.0, offset=np.
   pi/2)
L5 = rtb.RevoluteDH(a=0.0, alpha=0, d=0.317, offset=0)
# Create the robot model
robot = rtb.DHRobot([L1, L2, L3, L4, L5], name="Frank")
# Simula pick and place
qi = [0, -np.pi/4, -np.pi/4, -np.pi/2, 0]
initial_fk = robot.fkine(qi)

task = [
    {"pos": initial_fk.t, "ori": [np.pi, 0, 0]}, # Apenas
       rodando
    {"pos": [1.2, 0.4, 0.5], "ori": [np.pi, 0, 0]}, # Acima
        do objeto
    {"pos": [1.2, 0.4, 0.2], "ori": [np.pi, 0, 0]}, #
       Descendo para pegar
    {"pos": [1.2, 0.4, 0.5], "ori": [np.pi, 0, 0]}, #
       Retornando
    {"pos": [1.5, -0.4, 0.5], "ori": [np.pi, 0, 0]}, # Pose
        de deposito
```

```
    {"pos": [1.5, -0.4, 0.2], "ori": [np.pi, 0, 0]}, #
       Descendo para colocar
    {"pos": [1.5, -0.4, 0.5], "ori": [np.pi, 0, 0]}, #
       Subindo depois de colocar
    {"pos": initial_fk.t, "ori": initial_fk.rpy()}, #
       Retornando
]

task_counter = 0
for point in task:
    print(f"\nExecuting task {task_counter}")

    final_pos = point["pos"]
    final_ori = point["ori"]
    qn = differential_ik(robot, qi, final_pos, final_ori)

    print("Final IK position is: ", qn)
    traj = rtb.jtraj(qi, qn, 50)
    robot.plot(traj.q)
    qi = qn
    task_counter += 1
```

Foi gerado um vídeo com a execução da trajetória que está no repositório desse projeto: https://github.com/Mat198/frank/tree/main/videos

4.7 Coleta de Dados para Treinamento

Neste trabalho, também foi gerado um conjunto de dados contendo poses desejadas e suas respectivas soluções de cinemática inversa com o objetivo de futuramente treinar uma rede neural capaz de imitar o solver de cinemática inversa da *Robotics Toolbox* de Peter Corke.

O Código 10 implementa um procedimento para gerar amostras destinadas, futuramente, ao treinamento de um modelo de aprendizagem de máquina. Com o robô definido, o código entra em um laço de iterações, dentro do qual uma configuração aleatória de juntas é amostrada a partir dos limites do manipulador. Essa configuração é utilizada para calcular, via cinemática direta, a pose correspondente do end-effector, que passa a ser a pose desejada do problema de cinemática inversa.

Em seguida, o script gera uma configuração inicial também aleatória, que serve como estimativa para o método numérico `ikine_LM` da biblioteca. A partir da pose desejada, uma transformação homogênea é construida usando as classes `SE3`, combinando posição e orientação. O solver é então executado e, quando converge, retorna uma configuração de juntas compatível com a pose desejada.

Quando uma solução válida é encontrada, o código gera uma trajetória interpolada entre *q_initial* e *q_solution* utilizando `jtraj`. Em seguida, a função `save_to_csv` armazena em arquivo a pose desejada do end-effector, a configuração inicial, a configuração final do solver e a trajetória gerada. A repetição desse processo ao longo das iterações produz um conjunto de dados que documenta o comportamento do solver de cinemática inversa, formando a base de dados necessária para o posterior treinamento do modelo de aprendizagem de máquina.

Código 10: Script para Coleta de Dados

```
import numpy as np
import matplotlib.pyplot as plt
import csv
import os
import argparse
import roboticstoolbox as rtb
from spatialmath import SE3

def robot():
    l1 = rtb.DHLink(a=0.287, alpha=np.pi/2, d=0.155, offset
       =0)
    l2 = rtb.DHLink(a=0.8, alpha=0, d=0, offset=np.pi/2)
    l3 = rtb.DHLink(a=0.8, alpha=0, d=0.0, offset=0)
    l4 = rtb.DHLink(a=0.0, alpha=np.pi/2, d=0.0, offset=np.
       pi/2)
    l5 = rtb.DHLink(a=0.0, alpha=0, d=0.317, offset=0)

    return rtb.DHRobot(
        [l1, l2, l3, l4, l5],
        name="Frank"
    )


def save_to_csv(target_pose, trajectory, dataset_path):
    os.makedirs(dataset_path, exist_ok=True)

    existing_files = [f for f in os.listdir(dataset_path)
       if f.endswith('.csv')]

    if not existing_files:
        file_index = 0
    else:
        indices = [int(f.split('.')[0]) for f in
           existing_files]
        file_index = max(indices) + 1

    file_path = os.path.join(dataset_path, f"{file_index}.
       csv")

    q_initial = trajectory.q[0]
    q_final = trajectory.q[-1]

    row = np.concatenate((target_pose, q_initial, q_final))

    header = [
        'target_x', 'target_y', 'target_z', 'target_roll',
           'target_pitch', 'target_yaw',
        'q0_init', 'q1_init', 'q2_init', 'q3_init', '
           q4_init',
        'q0_final', 'q1_final', 'q2_final', 'q3_final', '
           q4_final'
    ]
```

```
    with open(file_path, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(header)
        writer.writerow(row)


def execute(robot, target_pose, q_initial, q_solution,
   dataset_path):
    trajectory = rtb.jtraj(q_initial, q_solution, 100)
    save_to_csv(target_pose, trajectory, dataset_path)


dataset_path = "./data"
num_iterations = 1000
robot = robot()

for i in range(num_iterations):

    qlim = robot.qlim.T

    q_random = np.array([np.random.uniform(low, high) for
       low, high in qlim])
    T = robot.fkine(q_random)

    pose = {
        "x": T.t[0],
        "y": T.t[1],
        "z": T.t[2],
        "roll": T.rpy()[0],
        "pitch": T.rpy()[1],
        "yaw": T.rpy()[2]
    }

    q_initial = np.random.uniform(-np.pi, np.pi, robot.n)

    T_goal = SE3.Trans(pose['x'], pose['y'], pose['z']) * \
             SE3.RPY(pose['roll'], pose['pitch'], pose['yaw
                ], order='zyx')

    sol = robot.ikine_LM(T_goal, q0=q_initial)

    q_solution, success = sol.q, sol.success

    if success:
        target_pose = np.array([
            pose["x"],
            pose["y"],
            pose["z"],
            pose["roll"],
            pose["pitch"],
            pose["yaw"]
        ])
```

```
            print(f"Initial joint configuration (q_initial): {
                np.round(q_initial, 4)}")
            print(f"Target Pose: {np.round(target_pose, 4)}")

            execute(robot, target_pose, q_initial, q_solution,
                dataset_path)
        else:
            continue
```

5 Conclusão

O trabalho realizado permitiu modelar e analisar a cinemática do manipulador FRANK, desde a definição dos parâmetros de Denavit-Hartenberg até a implementação de métodos numéricos de cinemática inversa baseados na Jacobiana. As simulações confirmaram a capacidade do robô em executar trajetórias e tarefas básicas. Além disso, a geração automática de dados mostrou-se eficaz para gerar dados de forma rápida para apoiar o treinamento de uma rede neural destinada a aproximar o comportamento do solver de cinemática inversa da *Robotics Toolbox*.

Adicionalmente, foi empregado um modelo de rede neural artificial com camadas totalmente conectadas para realizar a regressão das trajetórias geradas pelo script de coleta de dados, buscando aproximar o comportamento do solver de cinemática inversa. Entretanto, o modelo não apresentou convergência satisfatória: as trajetórias previstas não alcançavam a pose desejada do end-effector, indicando limitação da arquitetura adotada para essa tarefa. Assim, trabalhos futuros incluem a investigação de outros modelos, como arquiteturas baseadas em *Transformers* e métodos generativos, especialmente modelos de difusão, que podem oferecer maior capacidade de generalização e precisão na estimação das configurações das juntas.