Universidade de São Paulo - USP
Escola de Engenharia de São Carlos - EESC
Engenharia Mecatrônica

SEM0591 - Controle de Sistemas Robóticos

Projeto Final - Controle de Manipulador Robótico

Fael Juraszek Pareja - 13730023

João Lucas de Felício Pereira da Silva - 13678904

Pedro Augusto Codognhoto dos Santos - 13679120

Victor Martinez Lechuga Dutra - 13678950

Vinicius Mori Sartor - 13678550

Prof. Marcelo Becker

Prof. Adriano Almeida Gonçalves Siqueira

05 de julho de 2025

Conteúdo

1 Introdução

Manipuladores robóticos têm se destacado como ferramentas essenciais na automação de tarefas complexas e repetitivas, especialmente em ambientes que demandam precisão e eficiência, como o setor alimentício. Com o avanço das tecnologias de controle e simulação, é possível projetar sistemas robóticos cada vez mais otimizados para aplicações específicas, levando em consideração aspectos como trajetória, singularidades e desempenho dinâmico.

Este relatório apresenta o desenvolvimento de um manipulador robótico com configuração PRR (Prismática–Rotacional–Rotacional), projetado para operar em ambientes de preparo automatizado de alimentos, como cozinhas industriais ou estabelecimentos de fast-food. O trabalho, realizado no contexto da disciplina, tem como foco principal a definição de uma trajetória no espaço operacional, a análise das singularidades associadas a essa trajetória e o projeto de um controlador linear para garantir o seguimento adequado do movimento planejado.

A análise teórica foi conduzida a partir da modelagem do manipulador, considerando sua cinemática direta e inversa, além da identificação de possíveis regiões de singularidade que limitam o desempenho do sistema. Em seguida, foi implementado um controlador PID para cada junta, com ajuste e validação dos ganhos por meio de simulações. Os resultados foram analisados por meio de gráficos de seguimento de trajetória, resposta temporal e atuação dos torques de controle.

1.1 Projeto Anterior

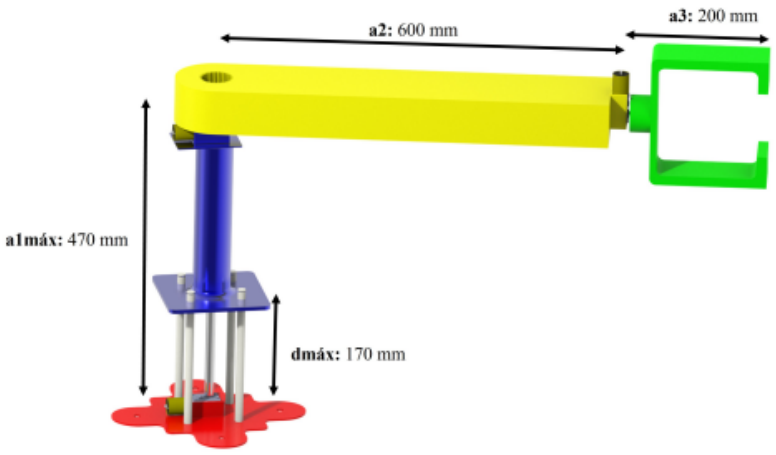

Figura 1: Manipulador original

O manipulador desenvolvido originalmente pelo grupo foi concebido para atuar no preparo automatizado de frituras em ambientes de fast-food. A escolha da configuração PRR teve como objetivo minimizar o número de graus de liberdade sem comprometer a execução da tarefa. As três juntas foram projetadas para operar da seguinte forma:

- Junta 1 (prismática): Responsável por mover o manipulador na direção vertical, permitindo submergir e elevar alimentos em óleo quente e realizar ajuste de altura para servir as batatas;

- Junta 2 (rotacional): Realiza o deslocamento horizontal do efetuador entre a fritadeira e o recipiente de descarte;
- Junta 3 (rotacional): executa o giro do cesto de fritura para despejar os alimentos no compartimento de armazenamento.

A construção mecânica foi feita utilizando materiais leves e resistentes, com partes produzidas por manufatura aditiva (impressão 3D), enquanto a parte eletrônica foi composta por motores JGY-370 com encoder, microcontrolador ESP32 e ponte H L298N.

Para o novo projeto, foram feitas algumas alterações no CAD original, com o intuito de eliminar singularidades que comprometeriam o funcionamento do manipulador. Detalhes sobre os motivos da troca serão apresentados nas seções seguintes.

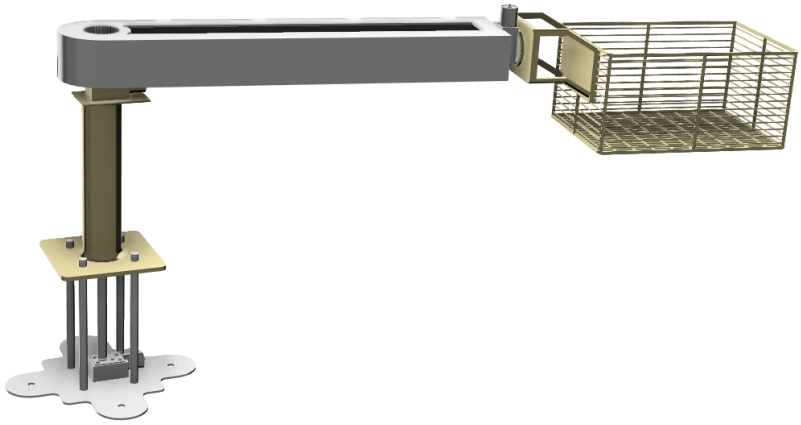

Figura 2: Manipulador atual

1.2 Objetivos

- Obter a trajetória do robô ao realizar sua tarefa;
- Verificar se existem singularidades durante a trajetória encontrada;
- Encontrar a equação do controlador linear (PID);
- Calcular os ganhos do controlador (Kp, Kd, Ki);
- Avaliar de a trajetória obtida é condizente com a desejada.

2 Projeto Dinâmico

A parte dinâmica do projeto tem como objetivo fundamental a determinação do espaço de trabalho do manipulador, para então definir a trajetória a ser seguida e analisar o comportamento dinâmico do robô nela. Para isso, a modelagem matemática do manipulador é abordada, considerando os aspectos cinemáticos e dinâmicos que influenciam diretamente o movimento do robô. Além disso, serão explorados os conceitos de trajetórias e singularidades, que são fundamentais para a compreensão das limitações operacionais e do impacto que essas singularidades podem ter no desempenho do manipulador.

A análise dinâmica envolve a descrição das forças e torques envolvidos no movimento das juntas e o cálculo da cinemática direta e inversa, que são essenciais para definir as relações entre as coordenadas articulares e as coordenadas do espaço cartesiano. A partir dessa modelagem, é possível calcular as trajetórias ideais do manipulador e identificar configurações críticas, como as singularidades, que podem comprometer o desempenho do sistema.

A modelagem matemática simbólica, a definição de trajetórias e a análise de singularidades foram realizadas em código *Python*, utilizando a biblioteca *Robotics Toolbox* de *Peter Corke*. A seguir, será apresentado o código utilizado para definir os parâmetros do robô e sua estrutura, bem como a explicação teórica de cada passo.

2.1 Modelagem do Manipulador

Para compreender a trajetória do manipulador, é necessário primeiro conhecer sua estrutura. O manipulador desenvolvido possui uma configuração PRR (prismática-rotacional-rotacional), onde a primeira junta é prismática, permitindo movimento linear, e as outras duas juntas são rotacionais. A cinemática direta é usada para calcular a posição do end-effector do manipulador a partir dos ângulos das juntas e suas posições relativas.

```
#BIBLIOTECAS
import numpy as np
from roboticstoolbox import ET, ERobot, Link, mstraj
from spatialmath import SE3
import sympy
import matplotlib.pyplot as plt
import numpy as np

# Definindo os parametros do robo Cookbot
a2 = 0.6
a3 = 0.2
m0 = 0.816
m1 = 1.094
m2 = 1.958
m3 = 0.3
g = 9.81

I0 = np.diag([0.004653, 0.004743, 0.003209])
I1 = np.diag([0.01206, 0.01217, 0.001835])
I2 = np.diag([0.005, 0.099, 0.101])
I3 = np.diag([0.01702, 0.01727, 0.005741])


```

```
# Definindo o manipulador robotico
link0 = Link(ET.tz(qlim=[0, 0.170]), r = [3.222e-3, 0.155e-3, 46.056e-3], m = m0, I = I0, Tc = 0.0107, Jm = 2.4e-07, B = 2e-04)
link1 = Link(ET.tz(0.300) * ET.Rz(qlim=[-np.pi, np.pi]), r = [0.002836, 0.000116, 0.110482], m = m1, I = I1, Tc = 0.0107, Jm = 2.4e-07, B = 2e-04)
link2 = Link(ET.tx(a2) * ET.tx(a3) * ET.Rx(qlim=[-np.pi, np.pi]), r = [0.266, -6.265e-5, 0.034], m = m2, I = I2, Tc = 0.0107, Jm = 2.4e-07, B = 2e-04)
link3 = Link(ET.Ry(np.pi/6) * ET.tz(-0.100) * ET.tx(0.100) * ET.tz(0.100), r = [0.266, -6.265e-5, 0.034], m = m2, I = I2, Tc = 0.0107, Jm = 2.4e-07, B = 2e-04)

cookbot = ERobot([link0, link1, link2, link3], name="Cookbot")
print(cookbot)
print(cookbot.dynamics())
```

No código acima, são definidos os parâmetros do manipulador, como os tamanhos das juntas, as massas e os momentos de inércia de cada uma das partes do manipulador. Os links do robô são configurados para refletir a configuração PRR, da seguinte maneira:

- **Junta 1 (prismática)**: realiza o movimento linear no eixo Z, com um limite de movimento de até 170 mm.
- **Junta 2 (rotacional)**: realiza a rotação ao redor do eixo Z, com um limite de $-180°$ a $180°$, além de possuir uma translação de 300mm.
- **Junta 3 (rotacional)**: realiza rotação no eixo X e tem uma translação de 800 mm.
- **End-effector (junta 4)**: possui uma inclinação de 30° para evitar singularidades, e possui translações de 100mm em X e Z, formando uma cesta.

Para o modelo dinâmico, foram definidos o centro de massa (r), o momento de inércia(I), o coeficiente de fricção de Coulomb (Tc), a inércia do motor (Jm) e o coeficiente de amortecimento (B) de cada junta, que são necessários para calcular as forças e os torques atuantes em cada parte do manipulador. O cálculo da fricção de Coulomb do motor está apresentada no *Apêndice A*.1

link	**link**	**joint**	**parent**	**ETS: parent to link**
0	link-0	0	BASE	tz(q0)
1	link-1	1	link-0	tz(0.3) ⊕ Rz(q1)
2	link-2	2	link-1	tx(0.6) ⊕ tx(0.2) ⊕ Rx(q2)
3	@link-3		link-2	Ry(30°) ⊕ tz(-0.1) ⊕ tx(0.1) ⊕ tz(0.1)

Tabela 1: Estrutura Cinemática do Manipulador Cookbot

2.2 Jacobiano e Análise de Singularidades

A análise das singularidades exige a definição do Jacobiano, matriz das derivadas parciais da posição e do end effector em função das variáveis de junta, que relaciona a velocidade das juntas com a da extremidade do manipulador. Por consequência, também é necessário encontrar a cinemática direta, para obter a relação entre a posição das variáveis de junta e do end effector.

Com essas ferramentas é possível encontrar as configurações de Singularidade do manipulador. Nelas não é possível gerar todos os movimentos desejados no espaço operacional, ou seja, o manipulador perde graus de liberdade, o que pode afetar a precisão e o controle do sistema.

No código abaixo encontra-se a matriz de transformação da cinemática direta, bem como o jacobiano simbólico:

```
# ANALISE SINGULARIDADES
from roboticstoolbox import jsingu
import sympy as sp

#variaveis simbolicas
q0, q1, q2 = sp.symbols('q0 q1 q2')
pos_sim = (q0, q1, q2)

#definindo a cinematica direta simbolica
TE = cookbot.fkine([q0, q1, q2])
print("A cinematica direta simbolica eh:")
print(TE)

#componente translacional do end-effector
p = TE.t

#definindo as variaveis simbolicas para o jacobiano
q = pos_sim

#definindo o jacobiano simbolico
J = sp.Matrix(p).jacobian(q)
print("O Jacobiano simbolico eh:")
print(J)
print(sp.Matrix.rank(J))
detJ = J.det()
print("O Determinante do Jacobiano eh:", detJ)
# Valores de q1 que geram singularidades

solq1 = sp.solve(detJ, q1)
print("Valores de q1 que geram singularidades:", solq1) # disso vemos
    que q1 nao gera singularidades!
solq2 = sp.solve(detJ, q2)
```

As tabelas abaixo mostram a cinemática direta e o jacobiano simbólico do manipulador Cookbot, que foram utilizados para determinar as singularidades:

Cinemática Direta do Manipulador Cookbot

$$\begin{bmatrix} 0.5\sin(q_1)\sin(q_2) + 0.866\cos(q_1) & -1.0\sin(q_1)\cos(q_2) & 0.866\sin(q_1)\sin(q_2) - 0.5\cos(q_1) & 0.0687\sin(q_1)\sin(q_2) + 0.676995646560476\cos(q_1) \\ 0.866\sin(q_1) - 0.5\sin(q_2)\cos(q_1) & 1.0\cos(q_1)\cos(q_2) & -0.5\sin(q_1) - 0.866\sin(q_2)\cos(q_1) & 0.677\sin(q_1) - 0.0687\sin(q_2)\cos(q_1) \\ 0.5\cos(q_2) & 1.0\sin(q_2) & 0.866\cos(q_2) & 1.0(q_0) + 0.0687\cos(q_2) + 0.3 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

Jacobiano Simbólico do Manipulador Cookbot

$$\begin{bmatrix} 0 & -0.677\sin(q1) + 0.0687\sin(q2)\cos(q1) & 0.0687\sin(q1)\cos(q2) \\ 0 & 0.0687\sin(q1)\sin(q2) + 0.677\cos(q1) & 0.0687\cos(q1)\cos(q2) \\ 1 & 0 & 0.0687\sin(q2) \end{bmatrix}$$

Como cada linha do jacobiano representa um vetor das derivadas parciais da posição do end-effector, eles representam como ela muda quando apenas uma junta se move. Portanto se há independência linear entre os vetores, a extremidade pode se mover em todas as direções. Caso contrário, o manipulador perde graus de liberdade e há uma singularidade.

Aplicando princípios de algebra linear, os vetores são linearmente dependentes quando o determinante da matriz jacobiana é igual a zero.

Determinante do Jacobiano

$$\det(J) = \sin(q2)\cos(q2)(-0.00472\sin^2(q1) - 0.00472\cos^2(q1))$$

$$\det(J) = \sin(q2)\cos(q2)(-0.00472)$$

Analisando a equação podemos ver que os valores que geram singularidade são exclusivos de q_2 e são

$$q_2 = \{-90°, 0°, 90°\ 180°\}$$

Nesses pontos o último link se alinha ao plano dos outros links, gerando a singularidade, como pode ser visto abaixo:

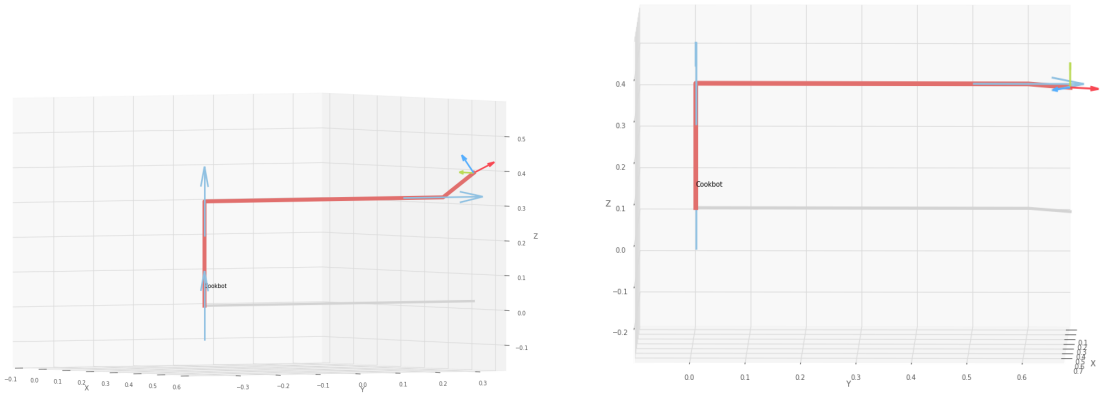

Figura 3: Link 3 alinhado ao Link 1 (à esquerda) e ao Link 2 (à direita).

No projeto original, o robô encontrava-se em singularidade em todas as posições da junta 3, pois os últimos 2 links estavam alinhados. Dessa forma, o ponto do end-effector não mudaria de posição no espaço, apesar de a junta manter sua função. A configuração gerava a chamada *Singularidade de Punho*, na qual do ponto de vista cinemático, o robô tem apenas 2 graus de liberdade, apesar de ter 3 juntas.

Alterando o end-effector, alteramos o ponto de interesse do último link, que passa a ser inclinado em relação ao link interior. Isso previne a configuração de singularidade permanente.

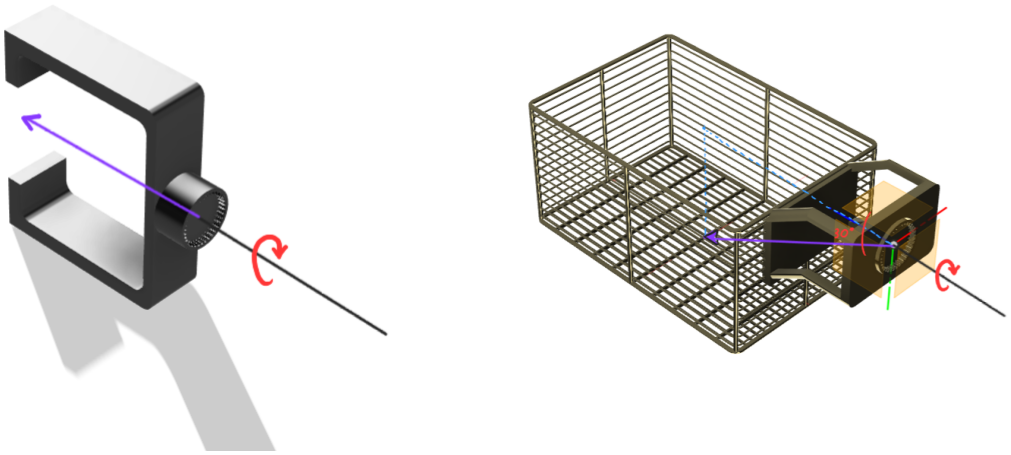

Figura 4: Comparação entre o ponto de interesse do end-effector original e do atual.

2.3 Trajetória e Espaço das juntas

A trajetória foi pensada de acordo com a função do robô: auxiliar estabelecimentos alimentícios a prepararem alimentos. Para isso, definiu-se uma sequência de movimentos que simula o processo real de fritura e despejo do alimento.

A trajetória foi definida no espaço das juntas, diretamente em termos das variáveis articulares do manipulador. Essa abordagem garante controle direto sobre os atuadores do robô e facilita a aplicação de estratégias de controle (como PID) para cada junta individual. Além disso, torna possível respeitar restrições físicas, como limites de deslocamento, velocidade e aceleração, o que é essencial para uma operação segura e realista em ambientes industriais.

A seguir, apresenta-se a sequência de movimento do manipulador, com as respectivas posições do end-effector (pontos que foram definidos como via points):

1. Manipulador parado na posição inicial, com seu end-effector inclinado $[0, 0, 10]$.
2. Manipulador levantado em 170mm e rotacionado em 90°, posicionando as batatas acima da fritadeira $[0.17, 90°, 0]$.
3. Manipulador abaixado em 170mm para que o alimento seja frito $[0, 90°, 0]$.
4. Manipulador é levantado em 170mm para retirar a cesta da fritadeira $[0.17, 90°, 0]$.
5. Manipulador abaixado em 120 mm e rotacionado em mais 90°, ficando acima do depósito $[0.05, 180°, 45°]$.
6. End-Effector inclinado em 270° para despejar as batatas no depósito $[0, 180°, 270°]$.

O código para encontrar as trajetórias também foi feito utilizando *Robotics toolbox*, e está descrito a seguir:

```
#TRAJETORIA COOKBOT

#importar parametros do robo
```

```
from parametros_cookbot import cookbot, g # type: ignore
import numpy as np
from roboticstoolbox import mstraj

frit = np.deg2rad(8)
frit_desp = np.deg2rad(280)

#TRAJETORIAS - analises do end-effector
#batatas serao despejadas na bandeja
POS0 = cookbot.fkine([0, 0, frit])
#posicao 1: batatas acima da fritadeira
POS1 = cookbot.fkine([0.17, np.pi/2, frit])
#posicao 2: batatas  dentro da fritadeira
POS2 = cookbot.fkine([0, np.pi/2, frit])
#posicao 3: retirar as batatas da fritadeira
POS3 = POS1
#posicao 4: levar as batatas ate a bandeja de despejo
POS4 = cookbot.fkine([0.17, np.pi, frit])
#posicao 5: despejar as batatas na bandeja
POS5 = cookbot.fkine([0.17, np.pi, frit_desp])

#Crinado um vetor com as posicoes
viapoints = np.array([
    [0, 0, np.deg2rad(10)],                         # POS0
    [0.17, np.pi/2, 0],                             # POS1
    [0, np.pi/2, 0],                                # POS2
    [0.17, np.pi/2, 0],                             # POS3
    [0.05, np.pi, np.deg2rad(45)],                  # POS4
    [0.05, np.pi, np.deg2rad(270)],                 # POS5
])

#vetor com os tempos de cada segmento
tsegment = np.array([5,          # Tempo para ir de POS0 a POS1
                     8,          # Tempo para ir de POS1 a POS2
                     8,          # Tempo para ir de POS2 a POS3
                     6,          # Tempo para ir de POS3 a POS4
                     5])         # Tempo para ir de POS4 a POS5

traj_des = mstraj(
    viapoints=viapoints,
    dt=0.04,
    tacc=2,
    tsegment = tsegment
)

q_des = traj_des.q
t_des = traj_des.t

# Derivada numerica para obter velocidade
qd_des = np.gradient(q_des, t_des, axis=0)

# Derivada numerica para obter aceleracao
qdd_des = np.gradient(qd_des, t_des, axis=0)

#TORQUE NA TRAJETORIA

# Calculo do torque ideal usando a funcao rne (Robotic Newton-Euler)
Torque_traj = cookbot.rne(q_des, qd_des, qdd_des, gravity=[0, 0, g],
```

```
    symbolic=False)
```

A função *mstraj()* gera trajetórias suaves entre os pontos com aceleração contínua. A interpolação da trajetória é gerenciada internamente pela função, e se se baseia no conceito de polinômios de quinto grau, que garantem continuidade em posição, velocidade e aceleração. A forma geral da interpolação é:

$$q_i(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

Os coeficientes a_k são ajustados para garantir continuidade da posição velocidade e aceleração entre os *viapoints*.

Após isso, as velocidades e acelerações das juntas foram obtidas numericamente com derivadas centradas e, com essas informações, foi possível calcular o Torque Ideal por junta ao longo da trajetória, por meio do algoritmo de *Newton-Euler*.

Com esse método é possível encontrar a modelagem dinâmica do manipulador, encontrando o torque necessário de cada junta para o movimento. A equação geral do torque resultante em pode ser representada como:

$$\boldsymbol{\tau}(t) = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q})$$

onde:

- $\mathbf{M}(\mathbf{q})$ é a matriz de inércia;
- $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ representa os efeitos centrífugos e coriolis;
- $\mathbf{G}(\mathbf{q})$ é o vetor de torques gravitacionais;
- $\boldsymbol{\tau}(t)$ é o vetor de torques articulares resultantes.

A implementação foi realizada com o comando `rne()` da *Robotics Toolbox*, que calcula o vetor de torque em cada instante da trajetória utilizando os perfis de posição, velocidade e aceleração das juntas, além do vetor gravidade.

2.3.1 Gráficos das Juntas

A partir da definição da trajetória, foi possível plotar os gráficos de posição, velocidade e aceleração de cada uma das juntas, sem aplicação de controle:

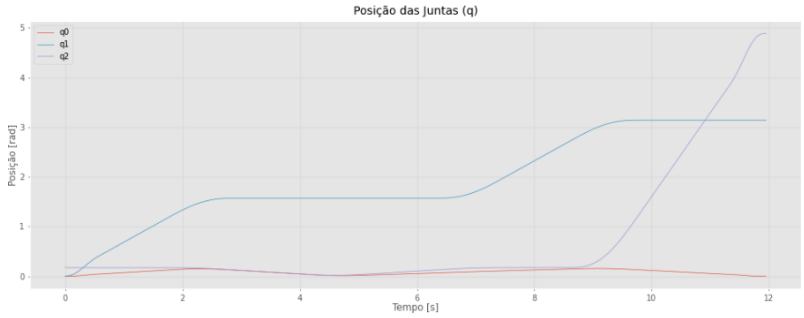

Figura 5: Gráfico da posição das juntas

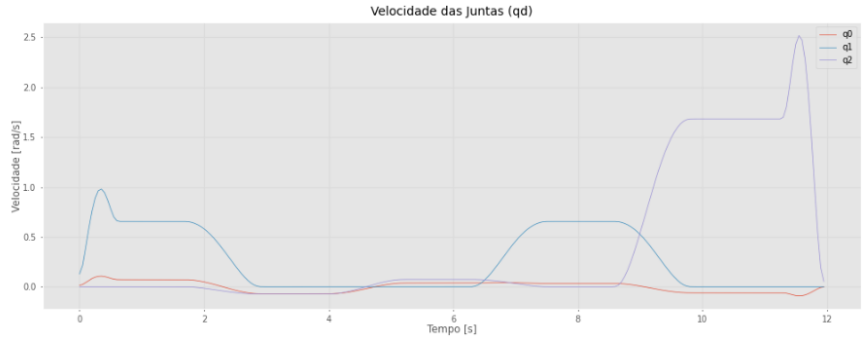

Figura 6: Gráfico da velocidade das juntas

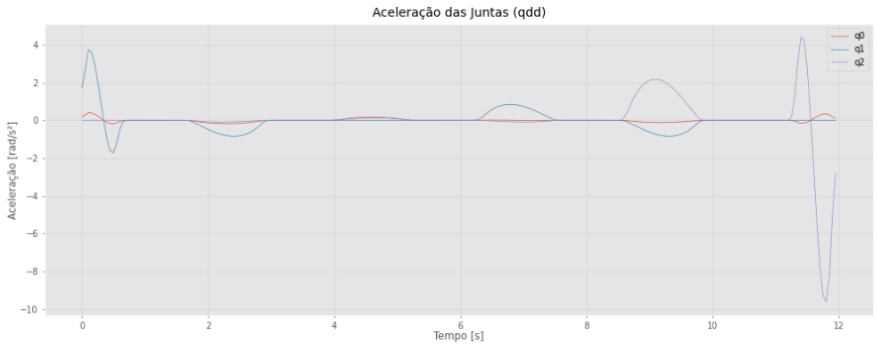

Figura 7: Gráfico da aceleração das juntas

2.3.2 Passagem por Singularidades

Durante a execução da trajetória planejada, o manipulador passa por configurações singulares, ao rotacionar a junta q_2, que foram detalhadas nas seções anteriores. Idealmente, evitaria-se a passagem por esses pontos, pois neles ocorre uma perda de posto do Jacobiano (determinante se aproxima de zero).

No entanto, devido à natureza da tarefa — que envolve abaixar a cesta de batatas diretamente na vertical dentro da fritadeira e posteriormente despejá-las com inclinação definida — não há grau de liberdade suficiente no ambiente nem na estrutura cinemática para evitar essas configurações. A trajetória precisa passar, por exemplo, por configurações onde $q_2 \approx \pi$, o que causa alinhamentos entre elos.a.

Além disso, o manipulador utilizado possui apenas 3 graus de liberdade, o que limita significativamente a redundância para contornar essas regiões problemáticas. Com mais graus de liberdade há mais flexibilidade para "desviar"de singularidades mantendo a posição do efetuador constante, o que não é possível neste caso.

Portanto, optou-se por tratar a trajetória de cruzando as singularidades prezando pela suavidade e controlando as velocidades e acelerações das juntas e, assim, minimizar instabilidades dinâmicas. O uso de interpolação suave (como polinômios de quinto grau) e o planejamento no espaço das juntas contribuíram para atravessar essas regiões sem comprometimento crítico do movimento.

Mesmo sendo necessário atravesá-las, as singularidades também geram problemas ao aplicar PID controle no manipulador. Também foram tomadas medidas para mitigar os efeitos prejudiciais disso, que serão explicadas nas seções subsequentes.

3 Projeto do Controlador Linear

Nesta seção, será desenvolvido o modelo dinâmico completo do manipulador, com base na formulação de Euler-Lagrange, e posteriormente o projeto do controlador PID para cada uma das juntas. O objetivo é garantir que o manipulador siga a trajetória definida de forma estável, precisa e eficiente, considerando as características dinâmicas do sistema.

3.1 Modelo Dinâmico do Manipulador

A modelagem dinâmica parte da equação geral de Euler-Lagrange:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i \tag{1}$$

Onde $\mathcal{L} = T - V$ é o Lagrangiano, com:

- T = energia cinética total do sistema
- V = energia potencial total do sistema
- q_i = coordenada generalizada (posição da junta i)
- τ_i = torque ou força aplicada na junta i

A partir disso, a equação geral do sistema dinâmico é expressa como:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{2}$$

Onde:

- $M(q)$: matriz de inércia (n×n)
- $C(q,\dot{q})$: matriz de Coriolis e centrífuga
- $G(q)$: vetor de gravidade
- τ: vetor de torques/forças aplicadas

Este modelo foi extraído diretamente utilizando a função `cookbot.dynamics()` da biblioteca `Robotics Toolbox`, permitindo obter automaticamente as matrizes $M(q)$, $C(q,\dot{q})$ e $G(q)$ com base nos parâmetros definidos no modelo.

3.2 Equações do Controlador PID

Para cada junta i, é implementado um controlador do tipo PID com a seguinte equação de controle:

$$\tau_i = K_{p_i} e_i + K_{d_i} \dot{e}i + Ki_i \int e_i, dt \tag{3}$$

Onde:

- $e_i = q_{i,desejado} - q_i$: erro de posição da junta i

- K_p, K_d, K_i: ganhos proporcional, derivativo e integral do PID

O controle PID é aplicado diretamente no espaço das juntas, sendo robusto frente às singularidades do Jacobiano e mais simples de implementar em sistemas com poucos graus de liberdade.

3.3 Cálculo Otimizado dos Ganhos

Com o modelo dinâmico já definido e a equação do controlador PID formalizada, o grupo optou por uma abordagem de otimização para determinar os ganhos K_p, K_d e K_i de cada junta do manipulador.

Foi desenvolvido um código em Python utilizando Regressão Recursiva (Recursive Least Squares – RLS) para otimizar individualmente os ganhos de cada junta. A lógica do método baseou-se em minimizar o erro quadrático entre a trajetória desejada $qd(t)$ e trajetoria realizada $q(t)$ ao longo do tempo adotando um criterio de custo J

$$J = \sum_{t=0}^{T} \left(q_d(t) - q(t)\right)^2 \tag{4}$$

Esse processo iterativo permitiu ajustar automaticamente os ganhos do controlador com base no desempenho observado em simulação, levando em conta o modelo dinâmico completo do sistema.

3.4 Teste Prático e Ajuste dos Ganhos

Com os ganhos iniciais estimados teoricamente, foi realizada uma simulação completa no *Python*, considerando as trajetórias planejadas e o modelo dinâmico do manipulador.

Os sinais de controle, erros e trajetórias obtidas foram analisados para refinar os valores de K_p, K_d e K_i de cada junta.

Os critérios observados incluíram:

- Aderência à trajetória desejada (erro máximo e médio)
- Estabilidade e ausência de oscilações
- Suavidade do torque aplicado, especialmente em transições rápidas (como o despejo de batatas)

Esse processo permitiu chegar automaticamente aos seguintes ganhos do controlador com base no desempenho observado em simulação, levando em conta o modelo dinâmico completo do sistema.

Os ganhos finais utilizados para cada junta foram:

$$K_p = \begin{bmatrix} 20 & 40 & 30 \end{bmatrix} \qquad K_d = \begin{bmatrix} 5 & 10 & 7.5 \end{bmatrix} \qquad K_i = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

Durante esse processo, foi observado que, para algumas juntas, o ganho integral K_i não contribuía significativamente para a melhoria do erro em regime. Pelo contrário, sua presença induzia *overshoot* e instabilidades indesejadas.

Esse comportamento está diretamente relacionado às características geométricas e dinâmicas do manipulador, cuja configuração PRR apresenta baixa resistência a

acúmulo de erro integrado em certos movimentos, especialmente em trajetórias com mudanças bruscas ou curtas variações em regime estacionário.

Por esse motivo, optou-se por manter $Ki = 0$ implementando um controle PD (Proporcional-Derivativo), o que resultou em um comportamento mais suave e robusto para o tipo de movimento envolvido.

3.5 Código para Cálculo da trajetória com PID

O código implementa um controlador PID no espaço das juntas para o manipulador `cookbot`, com base na trajetória gerada previamente com a função `mstraj`. A simulação considera a dinâmica do robô a partir da matriz de inércia $\mathbf{M}(\mathbf{q})$ e dos termos de Coriolis $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ calculados dinamicamente pela *Robotics Toolbox*, com a gravidade desprezada nesta implementação.

3.5.1 Definição do Controle PID

A função `dynamics()` define a equação do movimento:

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

com o torque calculado pelo controlador PID clássico:

$$\boldsymbol{\tau} = \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{K}_i \int (\mathbf{q}_d - \mathbf{q})\, dt$$

No código, esse cálculo aparece como:

```
e = qd_t - q
de = dqd_t - dq
tau = Kp * e + Kd * de + Ki * integral_e
```

Os valores desejados de posição e velocidade das juntas são obtidos por interpolação ponto a ponto:

```
qd_t = np.array([np.interp(t_clamped, t_des, q_des[:, i]) for i in
    range(q_des.shape[1])])
dqd_t = np.array([np.interp(t_clamped, t_des, qd_des[:, i]) for i in
    range(qd_des.shape[1])])
```

A simulação é realizada com a função `solve_ivp`, utilizando o método `'Radau'` por ser mais adequado para sistemas rígidos. O vetor de estado da simulação contém posições, velocidades e os termos de erro integral:

```
y0 = np.concatenate([q0, dq0, integral_e0])
sol = solve_ivp(dynamics, t_span_sim, y0, t_eval=t_eval_sim, args
    =(...), method='Radau')
```

3.5.2 Adaptações por conta das singularidades

Como a trajetória passa por singularidades e não é possível evitá-las, como mencionado anteriormente, foram aplicados mecanismos de prevenção para que essas posições problemáticas não influenciem na simulação (visto que elas podem fazer os valores irem ao infinito).

Foi implementado um mecanismo de *anti-windup* para evitar que o erro integral cresça indefinidamente, o que pode prejudicar a estabilidade do controlador:

```
integral_e_new = integral_e + de_integral * (t - y[-1])
integral_e_new = np.clip(integral_e_new, -integral_e_limit,
    integral_e_limit)
```

Para garantir estabilidade numérica, o código verifica a condição da matriz de inércia $\mathbf{M}(\mathbf{q})$ e interrompe a simulação caso ela esteja próxima da singularidade:

```
cond_M = np.linalg.cond(M)
if cond_M > 1e10:
    print("Singularity detected...")
    return np.full_like(y, np.nan)
```

3.5.3 Plots finais

Ao final da simulação, são plotadas as curvas de posição, velocidade e erro de rastreamento das juntas. Os dados também são salvos com `np.save` para posterior análise:

```
np.save('trajetoria_q_sim.npy', q_sim)
np.save('tempos_sim.npy', t_sim)
```

O código completo está disponível na página do github apresentada no final do relatório.

4 Simulações e Resultados

Para teste e avaliação da modelagem do sistema dinâmico criado e do controle PID posteriormente aplicado foi plotada a posição e velocidade de cada uma das juntas, tanto considerando a trajetória desejada, quando a real.

Além disso, buscando uma análise visual e prática da estabilidade do sistema, também foram plotados os erros de posição das 3 juntas.

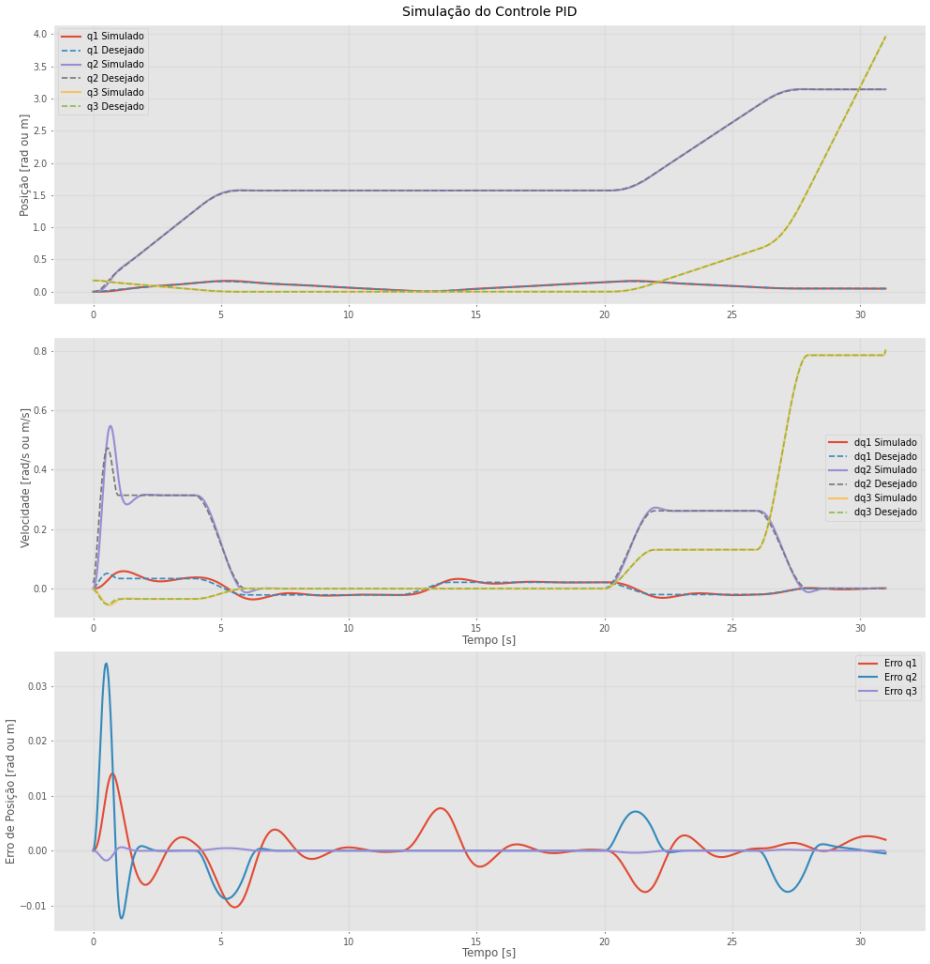

Figura 8: Em ordem: Posição desejada e real das juntas, Velocidade desejada e real das juntas e Erro de posição das juntas

Percebe-se, por análise gráfica, que os resultados da aplicação do controle PID foram extremamente satifatórios, resultando em uma trajetória semelhante a ideal, sem overshoot e com baixo tempo de acomodação.

Analisa-se também, por meio do gráfico do erro, que a ordem de grandeza deste é muito baixa para todas as juntas, tendo picos principalmente nas áreas onde há aceleração relevante (coincidem com as regiões de maior variação do gráfico das velocidades).

Conclui-se que precisão do movimento real,em comparação com o ideal, se deve não apenas ao projeto de controle PID completo, mas também à simplicidade do movimento, que não envolve alterações bruscas na posição, e do manipulador, com apenas 3 graus de liberdade

4.1 Gráficos de Torque

Para análise do projeto do robô, cabe também a avaliação do torque dinâmico exijido dos motores em cada junta. Dessa forma, foi realizado o plot dos 3 torques:

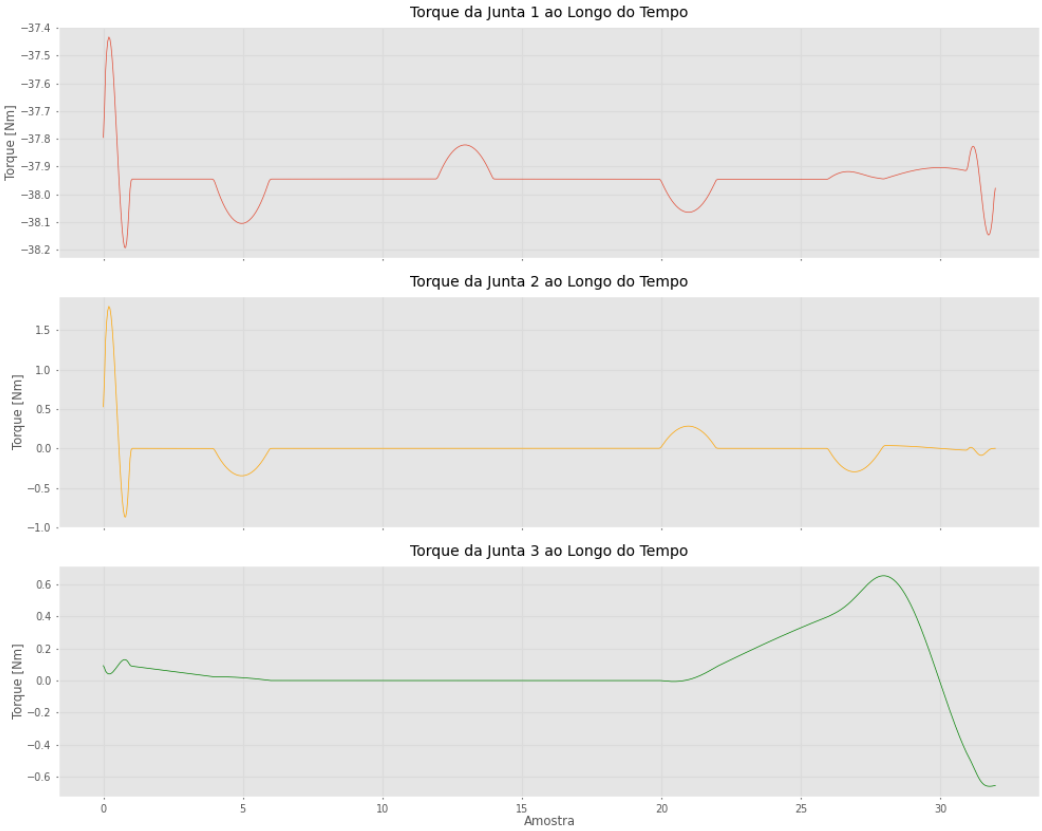

Figura 9: Em ordem: torque da junta q1, toque da junta q2 e toque da junta q3

Analisando o gráfico, percebe-se que as juntas possuem picos de torque nos momentos em que saem da posição travada (stall), porém, com o início do movimento esse pico diminui. Percebe-se que o torque da primeira junta é sempre negativo, pois ela precisa compensar o peço do robô como um todo. Quanto às outras juntas, nota-se que elas passam a maior parte do tempo com torque próximo de 0, pois seu movimento só é exigido pela trajetória em determinados momentos (como a junta 3 no final do movimento em que a cesta é girada para depositar as batatas no prato).

5 Conclusão

O desenvolvimento deste projeto proporcionou uma aplicação prática dos principais conceitos abordados na disciplina de Controle de Sistemas Robóticos, passando por modelagem dinâmica, planejamento de trajetória e implementação de controladores para manipuladores com múltiplos graus de liberdade.

A configuração PRR (Prismática–Rotacional–Rotacional) mostrou-se adequada para a tarefa proposta, permitindo que o manipulador execute todas as etapas do processo automatizado de fritura e descarte de alimentos. A modelagem simbólica em Python, com o uso da biblioteca `Robotics Toolbox`, permitiu descrever com precisão o comportamento dinâmico do sistema.

O planejamento de trajetória foi realizado diretamente no espaço das juntas, utilizando interpolação com polinômios de quinto grau para garantir continuidade em posição, velocidade e aceleração. Apesar da necessidade de atravessar configurações singulares, o sistema se manteve estável devido à suavização da trajetória.

O controlador PID por junta, com ganhos ajustados via regressão recursiva (RLS), demonstrou ótimo desempenho em simulação. Os resultados apresentaram baixo erro de seguimento, resposta rápida e ausência de overshoot. Considerando a estrutura do manipulador, o termo integral (K_i) foi desativado, evitando instabilidades e contribuindo para a suavidade do movimento.

Implementações futuras incluem:

- Validação experimental com protótipo físico.
- Implementação de controle em malha fechada com sensores de posição e corrente.
- Uso de controle adaptativo ou backstepping para robustez em ambientes não modelados.
- Inclusão de estratégias de desvio automático de singularidades.
- Interface gráfica para planejamento e visualização em tempo real.

Dessa forma, o projeto não apenas atingiu seus objetivos, como também estabelece uma base sólida para avanços mais complexos na área de robótica aplicada.

6 Repositório GitHub

`https://github.com/moo0ri/cookbot2.0`

7 Referências Bibliográficas

Referências

[1] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB for Scientists and Engineers*, 2nd ed., Springer, 2017.

[2] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, 2nd ed., Wiley, 2020.

[3] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, Springer, 2010.

8 Apêndice A

Estimativa da Fricção de Coulomb do Motor ASLONG JGY-370 12V

Com base nos dados fornecidos pelo datasheet do motor:

- Corrente sob carga: $I = 180\,\text{mA} = 0{,}18\,\text{A}$
- Torque sob carga: $\tau = 0{,}55\,\text{kg·cm} = 0{,}055\,\text{Nm}$
- Corrente sem carga: $I_0 = 35\,\text{mA} = 0{,}035\,\text{A}$

A constante de torque do motor pode ser estimada por:

$$K_t = \frac{\tau}{I} = \frac{0{,}055}{0{,}18} \approx 0{,}3056\,\text{Nm/A}$$

Assumindo que o torque gerado pela corrente sem carga é utilizado para vencer apenas a fricção de Coulomb, temos:

$$\tau_c \approx K_t \cdot I_0 = 0{,}3056 \cdot 0{,}035 \approx 0{,}0107\,\text{Nm}$$

Portanto, a fricção de Coulomb estimada para o motor é:

$$\boxed{\tau_c \approx 0{,}0107\,\text{Nm}}$$

O datasheet do motor está disponível no site: `https://media.digikey.com/pdf/data%20sheets/seeed%20technology/108990007_web.pdf`

**Os valores do datasheet foram apenas utilizados para obter uma estimativa da fricção de coulomb, não sendo considerado na análise do torque.