



Universidade Federal de Minas Gerais
Escola de Engenharia

Manipuladores Robóticos Trabalho Prático da Disciplina

Discentes: Giovanni Kurotaki
João Flávio de Freitas Cruz
Leonardo Hemerly Menezes Collaço dos Santos
Vitor Holler Teixeira de Barros
Docente: Gustavo Medeiros Freitas

Belo Horizonte
2025

Sumário

1	Introdução	2
2	Metodologia	3
2.1	Modelagem do Manipulador Robótico	3
2.2	Controle de Regulação	6
2.3	Controle de Trajetória	7
2.3.1	Prova de estabilidade assintótica	7
2.4	Funções de Movimentação	8
3	Resultados	10
3.1	Programação do manipulador industrial Comau Smart Six utilizando o PDL2	10
3.1.1	Implementação do tutorial de programação em PDL2	11
3.1.2	Análise dos comandos de movimentação	11
3.2	Programação do manipulador industrial Comau Smart Six utilizando o Matlab com o Robotic Toolbox	11
3.2.1	Resultados quantitativos	12
4	Conclusão	18

1 Introdução

Para a demonstração do conhecimento teórico e prático de disciplinas de graduação, são utilizados projetos de aplicação do conteúdo de forma a exercitar as capacidades de desenvolvimento e relato de projetos de maneira formal. O presente relatório tem por objetivo demonstrar a aplicação prática de metodologias de controle de manipuladores robóticos. Dessa forma, um modelo físico e outro computacional do manipulador COMAU Smartsix foram os alvos do controle em questão.

Na primeira parte do trabalho relatado neste documento, é realizado o desenvolvimento de códigos na linguagem PDL2 para o controle do manipulador industrial presente no laboratório de manipuladores robóticos da UFMG. São apresentados também links para vídeos que demonstrem a prática e a correta implementação dos códigos de controle.

Na segunda parte do trabalho, há um foco no uso da ferramenta MATLAB e da plataforma de simulação robótica CoppeliaSim para implementação e testes de algoritmos de controle em ambiente virtual. Também são analisados gráficos gerados pela movimentação, bem como a correlação correta entre os movimentos desejados e obtidos tanto no modelo feito no Coppelia Sim quanto no Robotic Toolbox.

2 Metodologia

O objetivo principal do projeto é realizar o controle e monitoramento de diversos parâmetros ao controlar o manipulador robótico COMAU Smart Six, possibilitando a análise do comportamento em diferentes trajetórias e pontos de controle.

2.1 Modelagem do Manipulador Robótico

Para realizar as simulações do manipulador no MATLAB, é necessário montar a matriz de Denavit-Hartenberg [1] e utilizar as ferramentas do Robotics Toolbox para manipular e visualizar o robô. Os parâmetros das juntas 1 a 6 são obtidos a partir do arquivo fornecido no *moodle* 'Comau SmartSix - 2 - Mateus Rodrigues Martins.pdf'.

Ao criar o manipulador no MATLAB, utilizando a ferramenta *Robotics Toolbox*, o manipulador se encontra entrando no plano XY, em outras palavras, ele está de cabeça para baixo Figura 1 . Isso ocorre por conta da modelagem utilizada, uma vez que para atender os requisitos de Denavit-Hartenberg a direção do eixo Z da junta 1 está para baixo, devido ao fato da direção de rotação da mesma Figura 2.

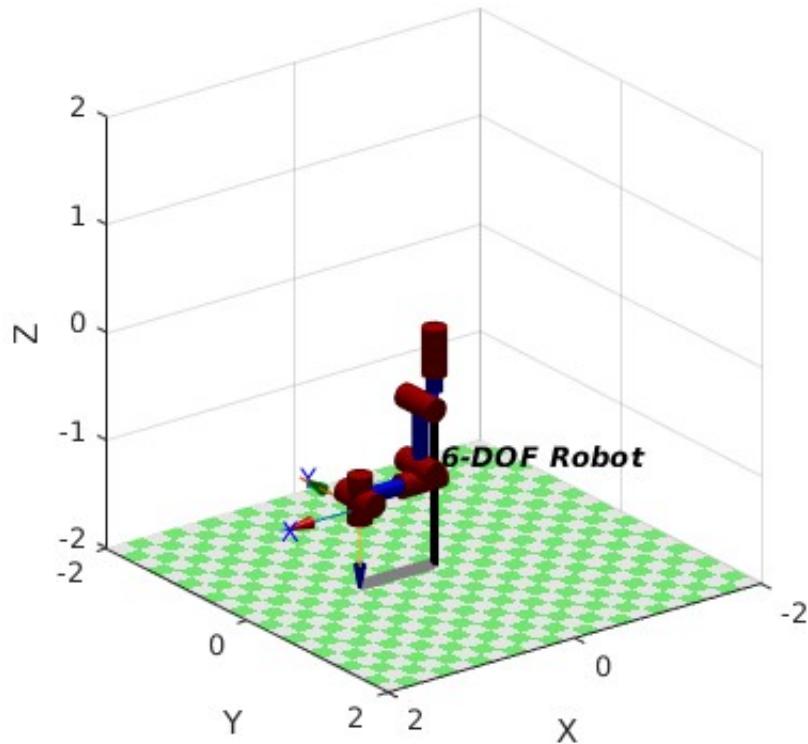


Figura 1: Manipulador COMAU SmartSix - parâmetros de Denavit-Hartenberg sem adicionar junta virtual

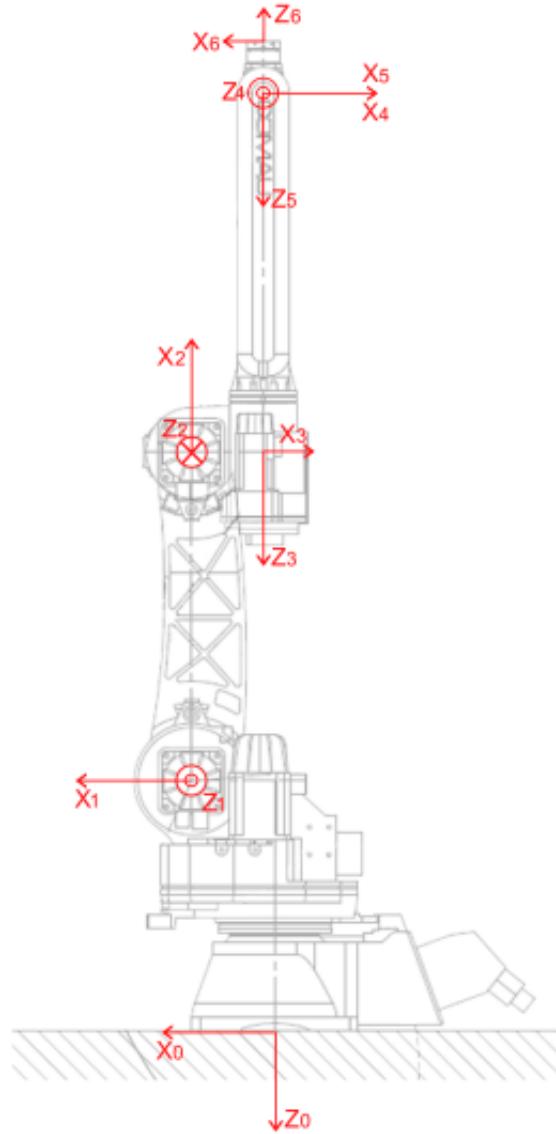


Figura 2: Sistema de coordenadas do manipulador COMAU Smartsix sem a junta virtual. Fonte: 'Comau SmartSix - 1 - João Carlos Oliveira Pena.pdf'

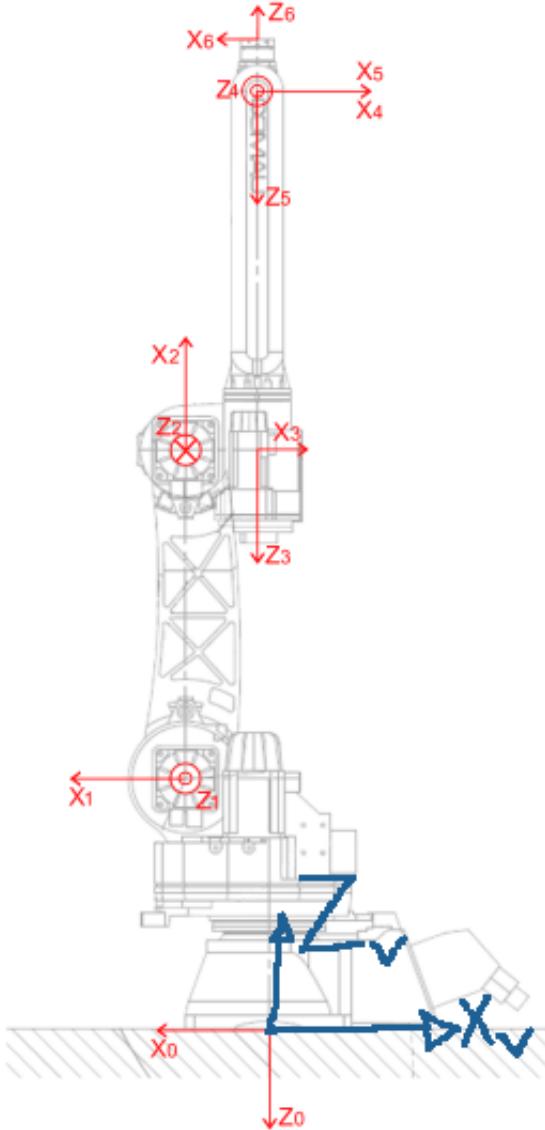


Figura 3: Sistema de coordenadas do manipulador COMAU Smartsix com a junta virtual

Além disso, a direção do eixo Z da junta 1 do manipulador robótico simulado no Coppelia está para cima, assim devemos corrigir isso na matriz de Denavit-Hartenberg para ter resultados consistentes. Desse modo, foi adicionado uma junta virtual na primeira coluna da Tabela 1, esta que tem o intuito de ajustar a rotação do robô em relação ao plano XY, Figuras (4, 3).

Parâmetro	eixo virtual	eixo 1	eixo 2	eixo 3	eixo 4	eixo 5	eixo 6
θ	0	0	$-\pi/2$	$\pi/2$	0	0	0
d	0	-0.45	0	0	-0.64707	0	-0.095
a	0	0	0.150	0.590	0.13	0	0
α	$-\pi$	$\pi/2$	π	$-\pi/2$	$-\pi/2$	$\pi/2$	π
juntas virtuais	1	0	0	0	0	0	0

Tabela 1: Parâmetros de Denavit-Hartenberg para o robô COMAU Smart SiX

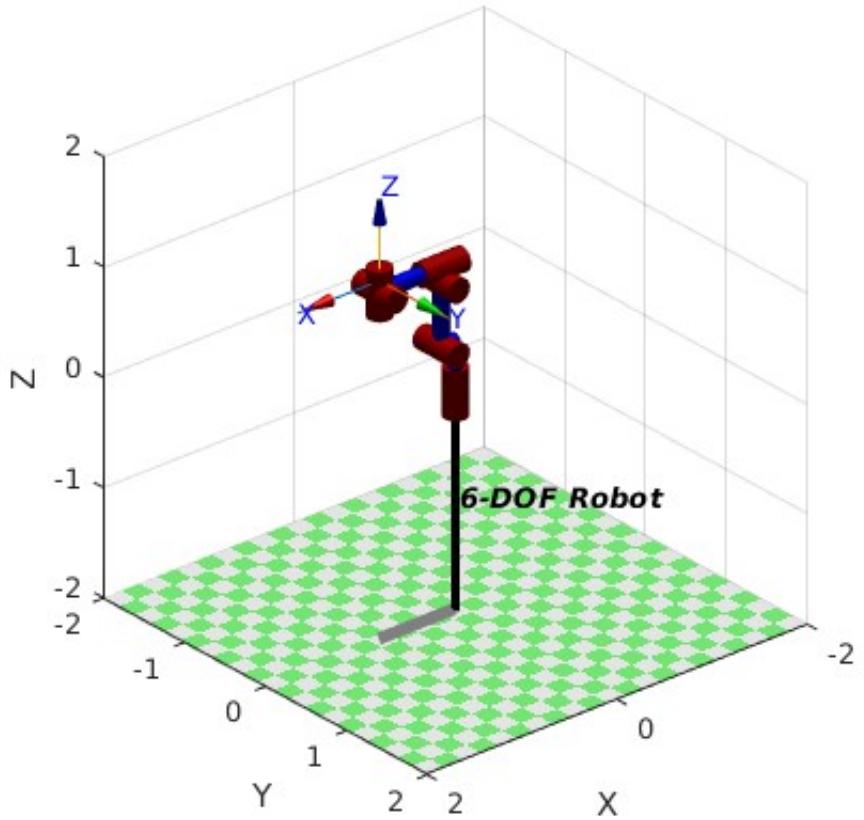


Figura 4: Manipulador COMAU SmartSix - parâmetros de Denavit-Hartenberg após adicionar junta virtual

2.2 Controle de Regulação

O controle de regulação implementa uma lógica baseada em erro para ajustar a posição e a orientação de um robô manipulador. Ele utiliza um critério de parada baseado na norma da diferença entre os erros atuais e anteriores. Além disso, o loop ajusta iterativamente os ângulos das juntas do robô para minimizar o erro de posição e orientação.

Ademais, a modelagem por Denavit-Hartenberg tem como saída um manipulador com 7 *DOF*, assim, precisamos reduzir a Jacobiana para poder inverte-la. Desse modo, como a primeira coluna representa uma junta virtual, ou seja, uma junta que não aplica, fisicamente, velocidade no efetuador, eliminamos ela para encontrar a Jacobina reduzida. Em segundo plano, vale ressaltar que a ação de controle atua apenas nas juntas 1 a 6, assim, a junta virtual permanece sempre com valor de 0.

Para encontrar o erro de posição, é realizada a cinemática direta da configuração atual das juntas e é subtraída pela posição desejada. Já para o erro de orientação, é realizada a cinemática direta da configuração atual das juntas, é multiplicado este valor à matriz de rotação desejada e depois convertido para o formato eixo-ângulo para se obter o erro de orientação.

Além disso, a ação de controle é dada por uma ação proporcional multiplicada pelo erro da pose que multiplica a jacobiana inversa reduzida.

$$\mathbf{u}_{\text{reduced}} = \mathbf{J}_{\text{reduced}}^+ (\mathbf{K} \cdot \mathbf{e})$$

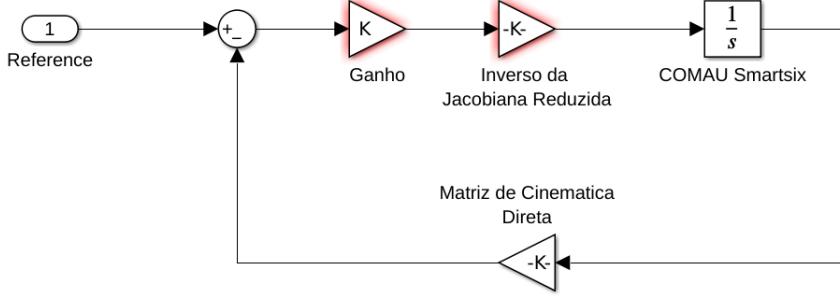


Figura 5: Diagrama de Blocos - Controle de Regulação

2.3 Controle de Trajetória

Os erros de posição e orientação são obtidos da mesma forma que no controle por regulação, como explicado na Seção 2.2. Porém, o que diferencia o controle de trajetória e o controle de regulação são dois fatores principais.

O primeiro está na ação de controle, onde existe um feedforward da velocidade do efetuador no cálculo da ação de controle. Já a segunda diferença está na forma em que o loop de controle funciona. Não existe um critério de parada baseado no erro da pose, mas existe o término da trajetória do efetuador. O critério de parada do loop é o efetuador percorrer todos os pontos da trajetória calculada, assim os valores de referência mudam a cada iteração do loop.

$$\mathbf{u}_{\text{reduced}} = \mathbf{J}_{\text{reduced}}^+ (\mathbf{K} \cdot \mathbf{e} + v_{\text{efetuador}})$$

2.3.1 Prova de estabilidade assintótica

Primeiramente, define-se a entrada de controle como as velocidades das juntas

$$u := \dot{q}. \quad (1)$$

Então, define-se o erro como

$$\tilde{x} = x_d - x, \quad (2)$$

onde x_d é a configuração alvo e x é a configuração atual. Portanto,

$$\dot{\tilde{x}} = \dot{x}_d - \dot{x}. \quad (3)$$

Usando um controlador proporcional com ação *feedforward*, que tem a dinâmica de erro

$$\dot{\tilde{x}} = -\lambda \tilde{x}, \quad (4)$$

que é uma EDO de primeiro grau, com solução

$$\tilde{x}(t) = \tilde{x}(0) \exp^{-\lambda t}, \quad (5)$$

o erro converge assintoticamente para 0 para qualquer ganho $\lambda > 0$.

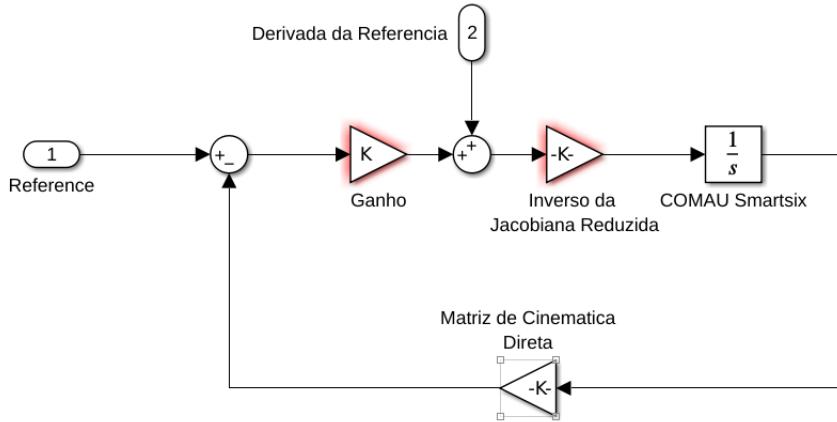


Figura 6: Diagrama de Blocos - Controle de Trajetória

2.4 Funções de Movimentação

No início do código, são chamadas funções responsáveis por movimentar o robô para pontos específicos e definir trajetórias:

- `go_to_P1()`, `go_to_P2()`, `go_to_P3()`, `go_to_P4()`: Movimentam o robô para pontos fixos no espaço, esses pontos foram especificados no arquivo 'Controle Comau - Robotics Toolbox - 2024_2.pdf'. Vale ressaltar que a função `go_to_P1()` inicia o movimento partindo de P_0 e vai até P_1 . Já os outros arquivos fazem o movimento da configuração atual das juntas até os respectivos pontos.

$$P_0 = \begin{bmatrix} 700 \\ 0 \\ 700 \end{bmatrix} \quad P_1 = \begin{bmatrix} 1000 \\ -600 \\ 1000 \end{bmatrix} \quad P_2 = \begin{bmatrix} 1000 \\ -600 \\ 300 \end{bmatrix}$$

$$P_3 = \begin{bmatrix} 1000 \\ 600 \\ 300 \end{bmatrix} \quad P_4 = \begin{bmatrix} 1000 \\ 600 \\ 1000 \end{bmatrix} \quad P_C = \begin{bmatrix} 1000 \\ 0 \\ 650 \end{bmatrix}$$

- `move_trajectory_to_P1()`, `move_trajectory_to_P2()`, `move_trajectory_to_PC()`: Configuram trajetórias entre pontos específicos. Essas trajetórias são todas retilíneas, para encontrar a equação da reta de um ponto P_a até um ponto P_b , utilizaremos a equação paramétrica da reta, dada por:

$$\begin{aligned} x(t) &= P_{x_a} + t \cdot \frac{P_{x_b} - P_{x_a}}{T}, \\ y(t) &= P_{y_a} + t \cdot \frac{P_{y_b} - P_{y_a}}{T}, \\ z(t) &= P_{z_a} + t \cdot \frac{P_{z_b} - P_{z_a}}{T} \end{aligned} \tag{6}$$

Já as derivadas dessas retas são dadas por:

$$\dot{x}(t) = \frac{P_{x_b} - P_{x_a}}{T}, \quad \dot{y}(t) = \frac{P_{y_b} - P_{y_a}}{T}, \quad \dot{z}(t) = \frac{P_{z_b} - P_{z_a}}{T} \tag{7}$$

- `move_trajectory_circle()`: Define uma trajetória circular, esta trajetória é calculada utilizando a equação paramétrica da circunferência, além disso, os pontos são constantes em X:

$$x(t) = 1000 \quad (8)$$

$$y(t) = y_{\text{centro}} + r \cdot \cos(\omega t + \phi) \quad (9)$$

$$z(t) = z_{\text{centro}} + r \cdot \sin(\omega t + \phi) \quad (10)$$

Já as derivadas são dadas por:

$$\dot{x}(t) = 0 \quad (11)$$

$$\dot{y}(t) = -r \cdot \omega \cdot \sin(\omega t + \phi) \quad (12)$$

$$\dot{z}(t) = r \cdot \omega \cdot \cos(\omega t + \phi) \quad (13)$$

A utilização destas funções permite configurar e avaliar o comportamento do robô em diferentes cenários, garantindo que o sistema responda adequadamente a comandos pré-programados.

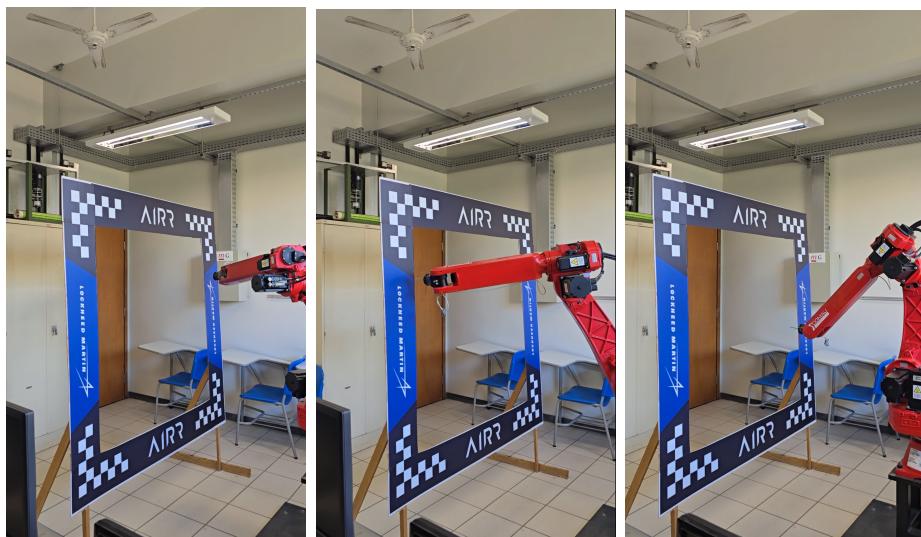
3 Resultados

3.1 Programação do manipulador industrial Comau Smart Six utilizando o PDL2

A programação do manipulador robótico Comau Smart Six foi feita com linguagem PDL2. PDL2 é uma linguagem de programação derivada do Pascal usada para programar diretamente os robôs Comau. É possível executar programas em PDL2 diretamente pelo controle remoto do robô. Neste trabalho, programou-se o robô para posicionar, sequencialmente, o efetuador em 5 pontos pré-definidos, conforme resultado mostrado na Figura 7 e em vídeo¹. O código desenvolvido para a execução dessa tarefa está mostrado na Seção 4.



(a) Configuração base. (b) Primeira configuração (c) Segunda configuração (d) Terceira configuração
da tarefa. da tarefa. da tarefa. da tarefa.



(e) Quarta configuração (f) Quinta configuração (g) Configuração base.
da tarefa. da tarefa. da tarefa.

Figura 7: Sequência de configurações visitadas pelo robô durante a execução da tarefa.

¹https://youtube.com/shorts/CeHLx_tLeDQ

3.1.1 Implementação do tutorial de programação em PDL2

A solução proposta é baseada em um *loop while* infinito - enquanto o robô estiver ligado, o *loop* estará sendo executado. Após entrar no *loop*, o robô move-se para a posição central no modo JOINT e espera o comando do usuário. Se o usuário pressionar o botão para SAIR, o *loop* é quebrado e o programa, finalizado. Se o usuário pressionar o botão para CONTINUAR, o programa move o robô para o primeiro ponto desejado, em seguida para o segundo após o primeiro ser alcançado, e assim sucessivamente. Após visitar todos os pontos, o robô volta para a posição central e o *loop* é reiniciado.

Primeiramente, em laboratório, o robô foi configurado na pose padrão (\$CAL_SYS). A partir dela, movendo as juntas 3 e 5, definiu-se o ponto P0 de modo que o efetuador esteja com o seu eixo z apontado horizontalmente para o quadro de referência, e a altura deslocado para o centro desse quadro, de modo a evitar o alinhamento dos eixos das juntas 4 e 6 - o que acarretaria numa configuração de singularidade do robô, do tipo singularidade do punho. Para capturar, utilizou-se a tecla REC do *Teach Pendant*.

O código da solução proposta é baseado em um *loop while* que, enquanto a variável auxiliar booleana “v_ativo” seja verdadeira, continua a ser executado. Após entrar no *loop*, o robô move-se para a posição central (ponto P0) no modo JOINT e espera algum comando do usuário a partir das entradas digitais. Se o usuário pressionar a tecla U2 (FDIN[22]), referente ao comando SAIR, robô sai do *loop*, configura-se na pose original (\$CAL_SYS) e o programa é finalizado. Se o usuário pressionar a tecla U1, referente ao comando CONTINUAR, o robô move-se do ponto P0 para o ponto P1, em seguida para o P2 após o primeiro ser alcançado, e assim sucessivamente. Após visitar todos os pontos, o robô volta para a posição P0 e o *loop* é reiniciado, aguardando o comando do usuário.

3.1.2 Análise dos comandos de movimentação

O Comau oferece dois modos de movimentação: JOINT e LINEAR. Em ambos, o resultado final é o mesmo: alcançar a configuração desejada. Eles diferem na trajetória executada. Enquanto o modo LINEAR garante que o efetuador seguirá uma trajetória reta no espaço de trabalho - o que é importante para robôs soldadores, por exemplo, não danificarem a peça sendo soldada -, o modo JOINT não oferece nenhuma garantia quanto ao formato da trajetória. Neste trabalho, foram testados os dois modos, sendo o JOINT usado para três primeiras configurações e o modo LINEAR para as três últimas.

Para o modo de movimentação do tipo JOINT, foi possível observar que o caminho percorrido assemelhou-se mais a um arco suave da posição de origem à posição de destino. Esse efeito foi possível ser observado com mais clareza ao realizar o movimento do ponto P1 ao ponto P2 no modo JOINT, em que o efetuador do robô acabou encostando no quadrado de referência durante boa parte da trajetória - o que não ocorreu no movimento do tipo LINEAR, do ponto P3 ao P4.

Essa diferença na trajetória entre a interpolação JOINT ou LINEAR - apesar de se parecer pouco relevante nesta prática - em outras aplicações em automação industrial poderia ter um impacto maior, ocasionando, por exemplo, colisões com outros objetos ou superfícies que poderiam ser danificadas ou até mesmo danificar o efetuador.

3.2 Programação do manipulador industrial Comau Smart Six utilizando o Matlab com o Robotic Toolbox

A mesma tarefa executada no robô físico Comau e mostrada na Seção 3.1 foi feita no simulador CoppeliaSim, utilizando o MATLAB e o pacote Robotics Toolbox [2], o código para realizar todos os passos está tanto no Github² quanto na Seção 4. O Robotics Toolbox fornece um conjunto

²https://github.com/vitorHoller/TP_Manipuladores.git

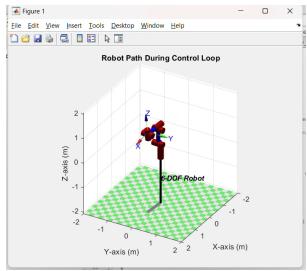
grande de ferramentas úteis para o estudo de manipuladores no MATLAB. Especificamente para esse trabalho, foram utilizadas ferramentas de modelagem cinemática, como parametrização por Denavit-Hartenberg, e técnicas de controle de regulação e seguimento de trajetória.

Para demonstração qualitativa dos resultados, simulou-se a movimentação no CoppeliaSim. Utilizando-se o MATLAB, conforme modelagem e estratégia de controle implementadas. Executou-se as ações de controle e salvou-se a configuração do robô em cada instante de tempo durante a movimentação. Essa sequência de configurações foi replicada no CoppeliaSim, conforme mostrado em vídeo³.

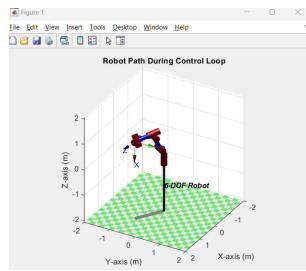
3.2.1 Resultados quantitativos

Para demonstração quantitativa dos resultados, salvou-se os valores de: posição das juntas; ação de controle; erro de posição (calculado por um escalar equivalente à distância) e orientação (erros de roll, pitch e yaw); e caminho (3D) percorrido pelo efetuador. A posição das juntas ao longo do número de iterações está mostrada na Figura 8. Estão mostradas no mesmo gráfico as posições de todas as juntas durante o movimento completo. A primeira parte do movimento corresponde ao controle de regulação, no qual o robô visita diferentes pontos sequencialmente. A segunda parte da movimentação, que começa por volta da iteração 1050 na Figura 8, corresponde ao movimento circular do seguimento de trajetória. Nota-se claramente a maior suavidade dos ângulos das juntas quando é usado o seguimento de trajetória. Os sinais de controle em cada junta ao longo das iterações estão mostrados na Figura 10. O erro de posição ao longo do tempo, mostrado na Figura 11, convergiu para 0 em todos os trechos. O erro de orientação em *roll*, *pitch*, *yaw* ao longo do tempo, mostrado na Figura 12, também convergiu para 0 em todos os trechos. Já o caminho 3D percorrido pelo efetuador está na Figura 9, em azul é o percurso utilizando controle de regulação e em vermelho o percurso usando controle de trajetória.

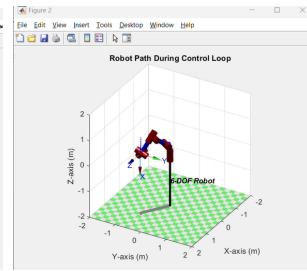
³<https://www.youtube.com/watch?v=mNXHPPc1q2g>



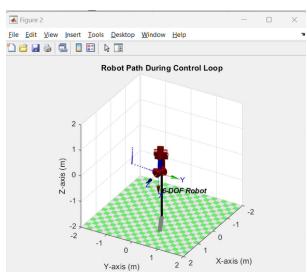
(a) Simulação MATLAB P0.



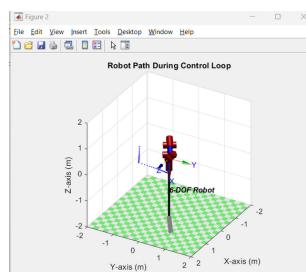
(b) Simulação MATLAB P1.



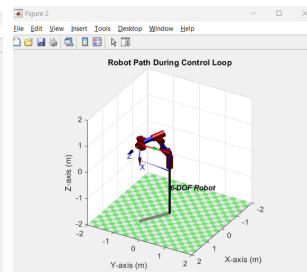
(c) Simulação MATLAB P2.



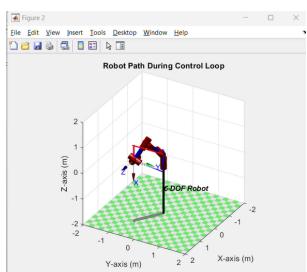
(d) Simulação MATLAB P3.



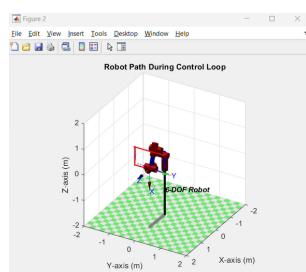
(e) Simulação MATLAB P4.



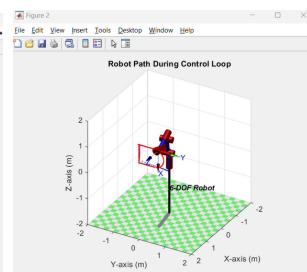
(f) Simulação MATLAB P1 - Controle de Trajetória.



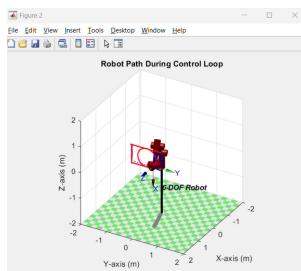
(g) Simulação MATLAB P2 - Controle de Trajetória.



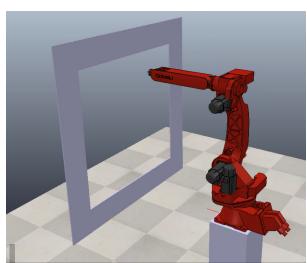
(h) Simulação MATLAB PC - Controle de Trajetória.



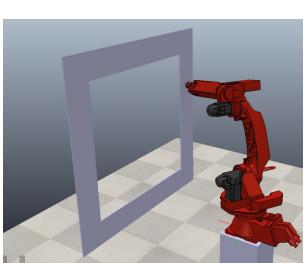
(i) Simulação MATLAB Inicio Circunferencia - Controle de Trajetória.



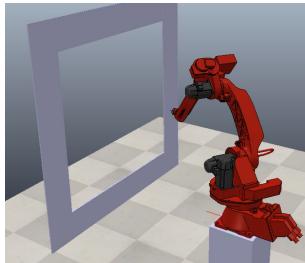
(j) Simulação MATLAB Final da Circunferência - Controle de Trajetória.



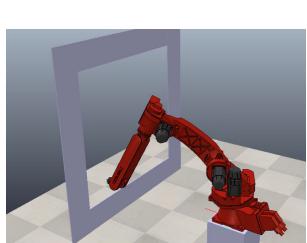
(k) Simulação COMAU -
Coppelia - P0.



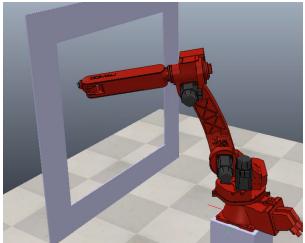
(l) Simulação COMAU -
Coppelia - P1.



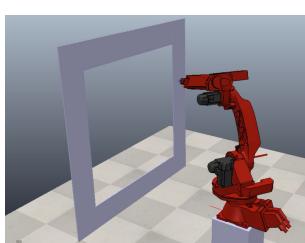
(m) Simulação COMAU -
Coppelia - P2.



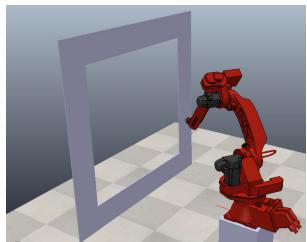
(n) Simulação COMAU -
Coppelia - P3.



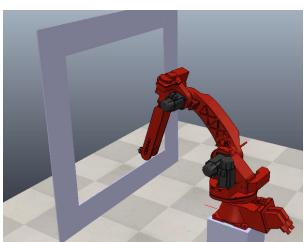
(o) Simulação COMAU -
Coppelia - P4.



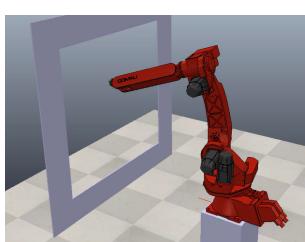
(p) Simulação COMAU -
Coppelia - P1 - Controle
de Trajetória.



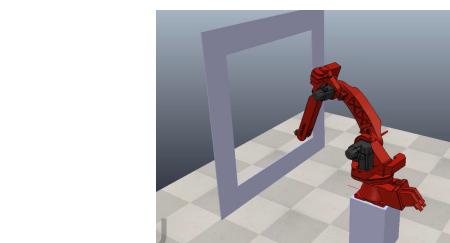
(q) Simulação COMAU -
Coppelia - P2 - Controle



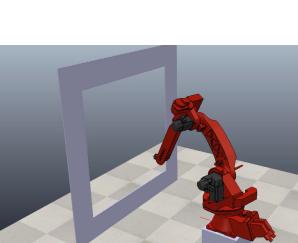
Coppelia - PC - Controle
Coppelia - Inicio Circun-
ferencia - Controle de Tra-
jetória.



(r) Simulação COMAU -
Coppelia - P2 - Controle
Coppelia - Controle
Coppelia - Inicio Circun-
ferencia - Controle de Tra-
jetória.



(s) Simulação COMAU -
Coppelia - P2 - Controle
Coppelia - Controle
Coppelia - Inicio Circun-
ferencia - Controle de Tra-
jetória.



(t) Simulação COMAU -
Coppelia - Metade da Cir-
cunferência - Controle de
Trajetória.

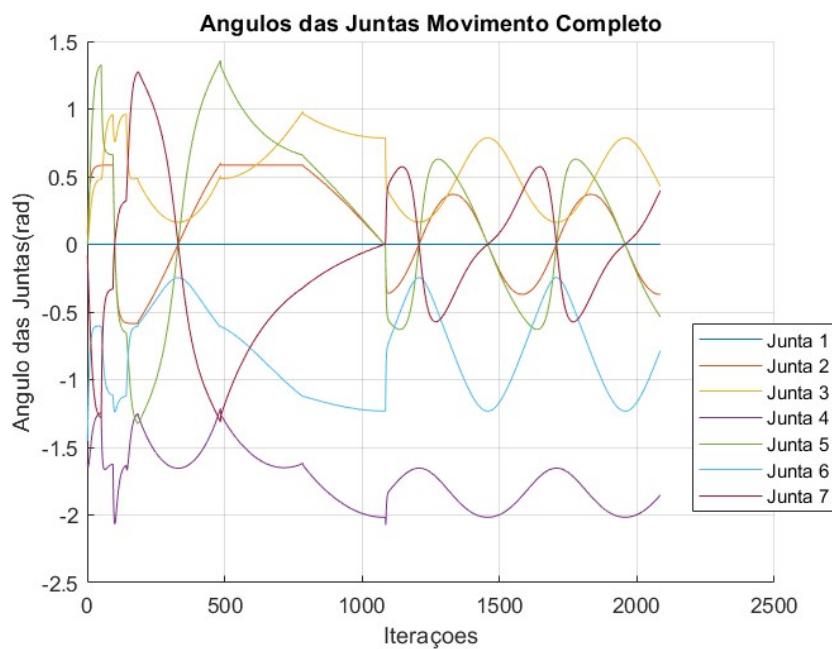


Figura 8: Erro de posição ao longo do número de iterações durante a execução do movimento.

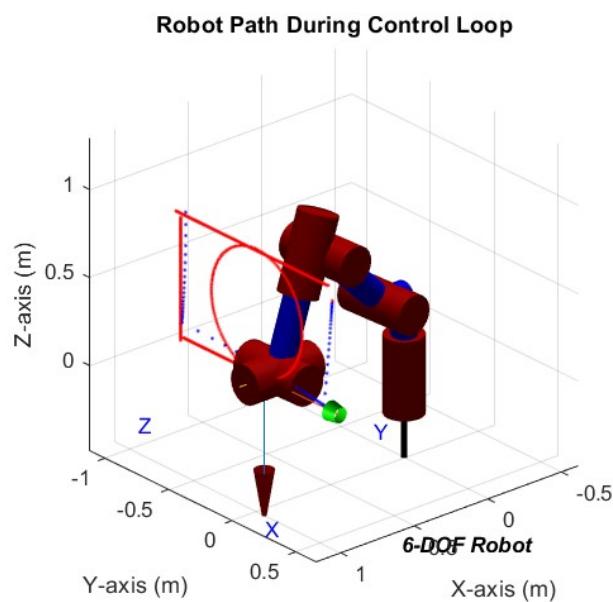


Figura 9: Caminho 3D percorrido pelo efetuador

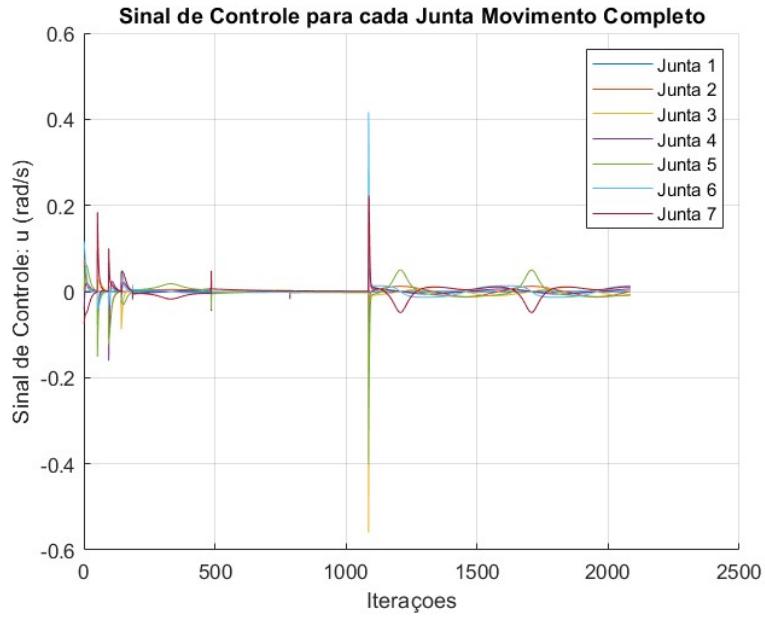


Figura 10: Sinal de controle ao longo do número de iterações durante a execução do movimento.

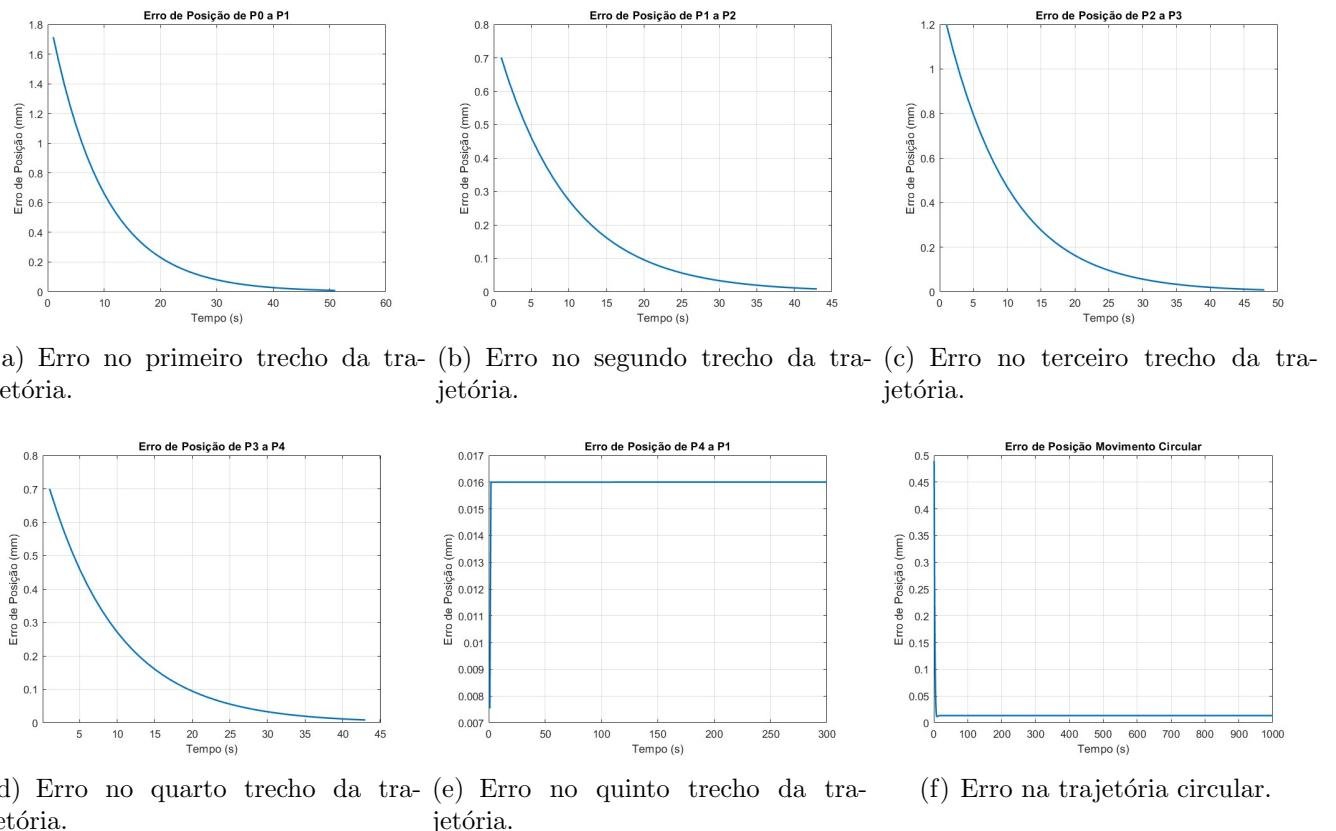
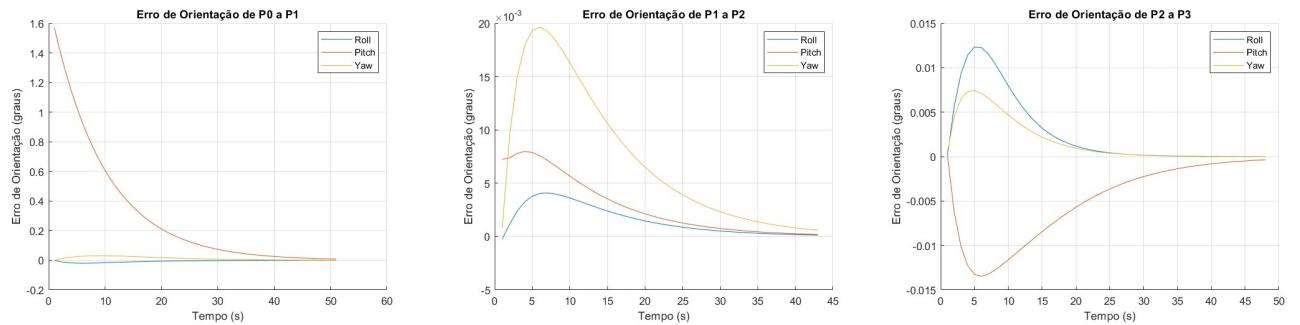
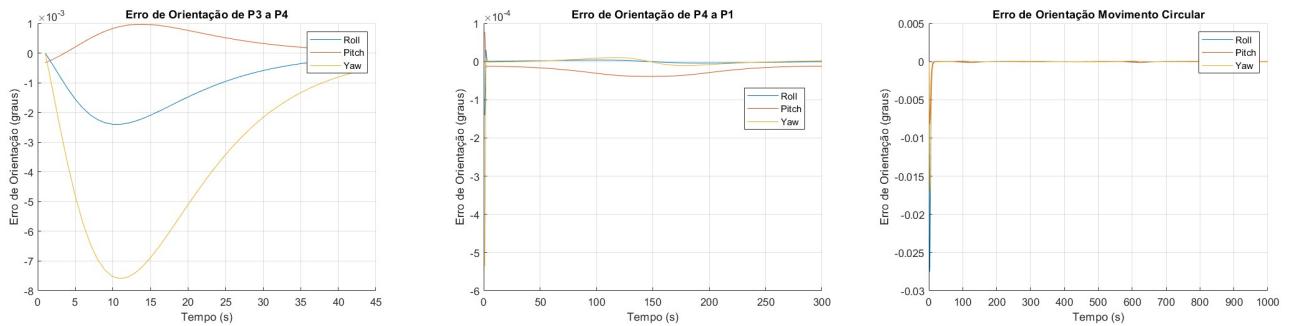


Figura 11: Erro de posição ao longo do tempo durante a execução de cada trecho da trajetória.



(a) Erro no primeiro trecho da trajetória. (b) Erro no segundo trecho da trajetória. (c) Erro no terceiro trecho da trajetória.



(d) Erro no quarto trecho da trajetória. (e) Erro no quinto trecho da trajetória. (f) Erro na trajetória circular.

Figura 12: Erro de orientação ao longo do tempo durante a execução de cada trecho da trajetória.

4 Conclusão

Neste trabalho, foi realizada a aplicação prática de conceitos de controle e modelagem de manipuladores robóticos. A programação do manipulador industrial Comau Smart Six demonstrou o uso da linguagem PDL2 para controle em plataforma real, enquanto o uso do MATLAB com o Robotics Toolbox, e o simulador CoppeliaSim, permitiu a implementação e validação de estratégias de controle em um ambiente virtual.

Os resultados obtidos mostraram que tanto o controle de regulação quanto o de trajetória alcançaram os objetivos de minimizar os erros de posição e orientação, garantindo a precisão e suavidade dos movimentos. A introdução de parâmetros ajustados, como o uso de uma junta virtual na modelagem Denavit-Hartenberg, foi essencial para a compatibilidade entre as ferramentas utilizadas e o comportamento esperado do manipulador.

Além disso, a análise qualitativa e quantitativa dos dados evidenciou a diferença entre os modos de movimentação (JOINT ou LINEAR), ressaltando a importância de escolher o modo de operação adequado à aplicação, especialmente em cenários onde trajetórias precisas são críticas para evitar colisões ou danos ao ambiente e ao próprio robô.

Por fim, o trabalho reforça a relevância acadêmica e prática do estudo de manipuladores robóticos, com práticas que podem ser aplicados em cenários reais de automação industrial e pesquisa em robótica. As ferramentas e métodos explorados neste relatório constituem uma base sólida para o desenvolvimento de aplicações mais avançadas no futuro.

Anexos

Código em PDL2 para movimentação do robô Comau

```
PROGRAM prog_ORION EZ, PROG_ARM = 1, STACK = 2048
VAR pnt0001P, pnt0002P, pnt0003P, pnt0004P, pnt0005P : POSITION
VAR v_ativo : BOOLEAN
ROUTINE call_prog_ORION EXPORTED FROM prog_ORION
ROUTINE call_prog_ORION
BEGIN

MOVE JOINT TO $CAL_SYS

v_ativo := TRUE

WHILE v_ativo = TRUE DO

MOVE JOINT TO pnt0001P

WAIT FOR $FDIN[21] OR $FDIN[22]

IF $FDIN[21] = ON THEN
MOVE JOINT TO pnt0002P
$FDOUT[21] := ON
MOVE JOINT TO pnt0003P
$FDOUT[21] := OFF
$FDOUT[22] := ON
MOVE JOINT TO pnt0004P
$FDOUT[22] := OFF
$FDOUT[23] := ON
MOVE LINEAR TO pnt0005P
$FDOUT[23] := OFF
$FDOUT[24] := ON
MOVE LINEAR TO pnt0002P
$FDOUT[23] := OFF
MOVE LINEAR TO pnt0001P
ENDIF

IF $FDIN[22] = ON THEN
v_ativo := FALSE
ENDIF
ENDWHILE

MOVE JOINT TO $CAL_SYS

END call_prog_ORION

BEGIN
call_prog_ORION
END prog_ORION
```

Código no MATLAB - RUN_tp.m

```
go_to_P1()
go_to_P2()
go_to_P3()
go_to_P4()
move_trajectory_to_P1()
move_trajectory_to_P2()
move_trajectory_to_PC()
move_trajectory_circle()

% Open a new figure for plotting control signals
figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, 1:j);

% Plot control signals for each joint
hold on;
for n = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(n, :), 'DisplayName', ['Junta ', num2str(n)]);
end
hold off;

% Add labels, title, and legend
xlabel('Iterações');
ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta Movimento Completo');
legend('show'); % Display joint labels in the legend
grid on;

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, 1:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(n, :), 'DisplayName', ['Junta ', num2str(n)]);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas Movimento Completo');
legend('show'); % Exibe a legenda
```

```

grid on;

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
plot(err(1:i), 'LineWidth', 1.5);
xlabel('Tempo (s)');
ylabel('Erro de Posição (mm)');
title('Erro de Posição Movimento Completo');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, 1:i);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');
ylabel('Erro de Orientação (graus)');
title('Erro de Orientação Movimento Completo');
legend('Roll', 'Pitch', 'Yaw');
grid on;

```

Código no MATLAB - go_to_P1.m

```

% Import the Robotics Toolbox
import robotics.*;

% Define the robot's DH parameters based on the provided table
L(1) = Revolute('d', 0, 'a', 0, 'alpha', -pi);      % Joint virtual
L(2) = Revolute('d', -0.45, 'a', 0.15, 'alpha', pi/2); % Joint 1
L(3) = Revolute('d', 0, 'a', 0.59, 'alpha', pi, 'offset', -pi/2); % Joint 2
L(4) = Revolute('d', 0, 'a', 0.13, 'alpha', -pi/2, 'offset', pi/2); % Joint 3
L(5) = Revolute('d', -0.6471, 'a', 0, 'alpha', -pi/2); % Joint 4
L(6) = Revolute('d', 0, 'a', 0, 'alpha', pi/2);        % Joint 5
L(7) = Revolute('d', -0.095, 'a', 0, 'alpha', pi, 'offset', pi); % Joint 6

% Combine the links into a SerialLink robot
robot = SerialLink(L, 'name', '6-DOF Robot');

```

```

% Display the robot's structure
robot.display();

q = [0, 0, 0, -pi/2, 0, -pi/2, 0]; % Define configuração inicial do robô
figure(1)

% Compute the Jacobian in the base frame
J = robot.jacob0(q);

% Display the Jacobian
disp('Jacobian Matrix (Base Frame):');
disp(J);

% Extrair apenas as 6 ultimas colunas (jacobiana reduzida para posição)
J_reduced = J(:, 2:7);

disp('Jacobiana Reduzida (para posição):');
disp(J_reduced);

%% Posições alvo (em mm convertidas para metros)
P0 = [0.7; 0; 0.7]; % [700; 0; 700]
P1 = [1; -0.6; 1]; % [1000; -600; 1000]
P2 = [1; -0.6; 0.3]; % [1000; -600; 300]
P3 = [1; 0.6; 0.3]; % [1000; 600; 300]
P4 = [1; 0.6; 1]; % [1000; 600; 1000]
%
% % Initial robot plot with workspace configuration
% figure(1);
% robot.plotopt = {'workspace', [-2 2 -2 2 -2 2], 'view', [120, 30]};
% robot.plot(q);
% title('Initial Configuration of the Robot');
% xlabel('X-axis (m)');
% ylabel('Y-axis (m)');
% zlabel('Z-axis (m)');
% grid on;
%
% pause(3);
% % Target points and orientation
% hold on;
% plot3(P0(1), P0(2), P0(3), 'ro', 'MarkerSize', 10, 'LineWidth', 2);
% plot3(P1(1), P1(2), P1(3), 'ro', 'MarkerSize', 10, 'LineWidth', 2, 'Color', "blue");
% plot3(P2(1), P2(2), P2(3), 'ro', 'MarkerSize', 10, 'LineWidth', 2, 'Color', "green");
% plot3(P3(1), P3(2), P3(3), 'ro', 'MarkerSize', 10, 'LineWidth', 2, 'Color', "yellow");
% plot3(P4(1), P4(2), P4(3), 'ro', 'MarkerSize', 10, 'LineWidth', 2, 'Color', "black");
% %legend('P0', 'P1', 'P2', 'P3', 'P4');
% hold off;

% Matriz de rotação constante

```

```

Rd = [0 0 1;
      0 1 0;
      -1 0 0];

i = 0;
pd = P1;

K = 0.1; % Define ganho
epsilon = 10e-3; % Define critério de parada
e_ant = 1;
e = 0;

control_sig = zeros(7, 1000); % 7 joints, assume up to 1000 iterations
joint_angles = zeros(length(q), 1000); % 1000 é o número máximo de iterações
err_rot = zeros(3, 1000); % 1000 é o número máximo de iterações
theta = [0 0 0 -pi/2 0 -pi/2 0]'; % Define configuração inicial do robô
T1 = [Rd, P1; 0 0 0 1];
% Resolver a cinemática inversa
q_solucao = robot.ikine(T1, q, [0, 1, 1, 1 ,1 ,1, 1]); % Ajuste as restrições conforme
disp('Ângulos das juntas (cinemática inversa):');
disp(q_solucao);

T1 = robot.fkine(q_solucao);

% Control loop visualization
figure(1);
robot.plot(theta');
hold on;
%T1.plot('rgb');
title('Robot Path During Control Loop');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
robot.plotopt = {'workspace', [-2 2 -2 2 -2 2], 'view', [120, 30]};
grid on;
view(3);

% Redundancy resolution factor (null space control)
lambda = 0.01; % Tuning parameter for redundancy resolution

% Control loop
while (norm(e - e_ant) > epsilon) % Stopping criterion
    i = i + 1; % Counter

    % Calcula a jacobiana completa e cinemática direta
    J_full = robot.jacob0(theta); % Jacobiana completa (6x7)
    T = robot.fkine(theta); % Pose atual do efetuador

    % Reduz a jacobiana (remove a contribuição da junta 1)
    J_reduced = J_full(:, 2:end); % Jacobiana reduzida (6x6)

```

```

% Erros de posição e orientação
p = transl(T); % Extração da posição
R = S03(T).R; % Extração da rotação
p_err = pd' - p; % Erro de posição
nphi = rotm2axang2(Rd * R'); % Erro de rotação (em eixo-ângulo)
nphi_err = nphi(1:3) * nphi(4); % Vetor n * phi (parte do erro angular)

% Vetor de erro combinado
e_ant = e; % Atualizar erro anterior
e = [p_err'; nphi_err'];

% Resolve o controle com a jacobiana reduzida
u_reduced = pinv(J_reduced) * (K * e); % Movimento das juntas 2 a 7

% Atualiza apenas as juntas 2 a 7
theta(2:end) = theta(2:end) + u_reduced;

% Junta 1 permanece fixa
theta(1) = 0;

% Visualização e armazenamento de dados
robot.plot(theta');
hold on;
plot3(p(1), p(2), p(3), 'b.', 'MarkerSize', 3);
hold off;
control_sig(:, i) = [0; u_reduced]; % Adiciona 0 como movimento da junta 1
err(i) = norm(e); % Armazena a norma do erro
err_rot(:, i) = [nphi_err(1) nphi_err(2) nphi_err(3)];
joint_angles(:, i) = theta; % Armazena os ângulos das juntas para cada iteração
q_seq(:, i) = theta(2:7);
disp('Ângulos finais das juntas:');
disp(theta);
end
%
% Open a new figure for plotting control signals
figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, 1:i);

% Plot control signals for each joint
hold on;
for n = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(n, :), 'DisplayName', ['Junta ', num2str(n)]);
end
hold off;

% Add labels, title, and legend
xlabel('Iterações');

```

```

ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta de P0 a P1');
legend('show'); % Display joint labels in the legend
grid on;

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, 1:i);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(n, :), 'DisplayName', ['Junta ', num2str(n)]);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas de P0 a P1');
legend('show'); % Exibe a legenda
grid on;

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm from P0 to P1', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
plot(err(1:i), 'LineWidth', 1.5);
xlabel('Tempo (s)');
ylabel('Erro de Posição (mm)');
title('Erro de Posição de P0 a P1');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, 1:i);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw from P0 to P1', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end

```

```

hold off;

% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');
ylabel('Erro de Orientação (graus)');
title('Erro de Orientação de P0 a P1');
legend('Roll', 'Pitch', 'Yaw');
grid on;

```

Código no MATLAB - go_to_P2.m

```

K = 0.1; % Define ganho
epsilon = 10e-3; % Define critério de parada
e_ant = 1;
e = 0;

%theta = [0 0 0 -pi/2 0 -pi/2 0]'; % Define configuração inicial do robô
T2 = [Rd, P2; 0 0 0 1];
pd = P2;
% Resolver a cinemática inversa
q_solucao = robot.ikine(T2, theta, [0, 1, 1, 1, 1, 1]); % Ajuste as restrições conf
disp('Ângulos das juntas (cinemática inversa):');
disp(q_solucao);

T1 = robot.fkine(q_solucao);

%Control loop visualization
figure(2);
robot.plot(theta');
hold on;
%T1.plot('rgb');
title('Robot Path During Control Loop');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
grid on;
view(3);

% Redundancy resolution factor (null space control)
lambda = 0.01; % Tuning parameter for redundancy resolution
j_ant = i + 1;
j = i;
% Control loop
while (norm(e - e_ant) > epsilon) % Stopping criterion
    i = i + 1; % Counter
    j = j + 1; % Counter

    % Calcula a jacobiana completa e cinemática direta
    J_full = robot.jacob0(theta); % Jacobiana completa (6x7)

```

```

T = robot.fkine(theta); % Pose atual do efetuador

% Reduz a jacobiana (remove a contribuição da junta 1)
J_reduced = J_full(:, 2:end); % Jacobiana reduzida (6x6)

% Erros de posição e orientação
p = transl(T); % Extração da posição
R = SO3(T).R; % Extração da rotação
p_err = pd' - p; % Erro de posição
nphi = rotm2axang2(Rd * R'); % Erro de rotação (em eixo-ângulo)
nphi_err = nphi(1:3) * nphi(4); % Vetor n * phi (parte do erro angular)

% Vetor de erro combinado
e_ant = e; % Atualizar erro anterior
e = [p_err'; nphi_err'];

% Resolve o controle com a jacobiana reduzida
u_reduced = pinv(J_reduced) * (K * e); % Movimento das juntas 2 a 7

% Atualiza apenas as juntas 2 a 7
theta(2:end) = theta(2:end) + u_reduced;

% Junta 1 permanece fixa
theta(1) = 0;

% Visualização e armazenamento de dados
robot.plot(theta');
plot3(p(1), p(2), p(3), 'b.', 'MarkerSize', 3);
control_sig(:, j) = [0; u_reduced]; % Adiciona 0 como movimento da junta 1
err(j) = norm(e); % Armazena a norma do erro
err_rot(:, j) = [nphi_err(1) nphi_err(2) nphi_err(3)];
joint_angles(:, j) = theta; % Armazena os ângulos das juntas para cada iteração
q_seq(:, j) = theta(2:7);
disp('Ângulos finais das juntas:');
disp(theta);
end

% Open a new figure for plotting control signals
figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, j_ant:j);

% Plot control signals for each joint
hold on;
for m = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

```

```

% Add labels, title, and legend
xlabel('Iterações');
ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta de P1 a P2');
legend('show'); % Display joint labels in the legend
grid on;

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for m = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas de P1 a P2');
legend('show'); % Exibe a legenda
grid on;
grid on;
% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm from P2 to P3', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
plot(err(j_ant:j), 'LineWidth', 1.5);
xlabel('Tempo (s)');
ylabel('Erro de Posição (mm)');
title('Erro de Posição de P1 a P2');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw from P1 to P2', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end
hold off;

```

```
% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');
ylabel('Erro de Orientação (graus)');
title('Erro de Orientação de P1 a P2');
legend('Roll', 'Pitch', 'Yaw');
grid on;
```

Código no MATLAB - go_to_P4.m

```
K = 0.1; % Define ganho
epsilon = 10e-3; % Define critério de parada
e_ant = 1;
e = 0;

%theta = [0 0 0 -pi/2 0 -pi/2 0]'; % Define configuração inicial do robô
T3 = [Rd, P3; 0 0 0 1];
pd = P3;
% Resolver a cinemática inversa
q_solucao = robot.ikine(T3, theta, [0, 1, 1, 1, 1, 1]); % Ajuste as restrições conf
disp('Ângulos das juntas (cinemática inversa):');
disp(q_solucao);

T1 = robot.fkine(q_solucao);

% % Control loop visualization
figure(2);
robot.plot(theta');
hold on;
%T1.plot('rgb');
title('Robot Path During Control Loop');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
grid on;
view(3);

% Redundancy resolution factor (null space control)
lambda = 0.01; % Tuning parameter for redundancy resolution
j_ant = j + 1;
% Control loop
while (norm(e - e_ant) > epsilon) % Stopping criterion
    i = i + 1; % Counter
    j = j + 1; % Counter

    % Calcula a jacobiana completa e cinemática direta
    J_full = robot.jacob0(theta); % Jacobiana completa (6x7)
    T = robot.fkine(theta); % Pose atual do efetuador
```

```

% Reduz a jacobiana (remove a contribuição da junta 1)
J_reduced = J_full(:, 2:end); % Jacobiana reduzida (6x6)

% Erros de posição e orientação
p = transl(T); % Extração da posição
R = S03(T).R; % Extração da rotação
p_err = pd' - p; % Erro de posição
nphi = rotm2axang2(Rd * R'); % Erro de rotação (em eixo-ângulo)
nphi_err = nphi(1:3) * nphi(4); % Vetor n * phi (parte do erro angular)

% Vetor de erro combinado
e_ant = e; % Atualizar erro anterior
e = [p_err'; nphi_err'];

% Resolve o controle com a jacobiana reduzida
u_reduced = pinv(J_reduced) * (K * e); % Movimento das juntas 2 a 7

% Atualiza apenas as juntas 2 a 7
theta(2:end) = theta(2:end) + u_reduced;

% Junta 1 permanece fixa
theta(1) = 0;

% Visualização e armazenamento de dados
robot.plot(theta');
plot3(p(1), p(2), p(3), 'b.', 'MarkerSize', 3);
control_sig(:, j) = [0; u_reduced]; % Adiciona 0 como movimento da junta 1
err(j) = norm(e); % Armazena a norma do erro
joint_angles(:, j) = theta; % Armazena os ângulos das juntas para cada iteração
q_seq(:, j) = theta(2:7);
err_rot(:, j) = [nphi_err(1) nphi_err(2) nphi_err(3)];
disp('Ângulos finais das juntas:');
disp(theta);
end
%
% Open a new figure for plotting control signals
figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, j_ant:j);

% Plot control signals for each joint
hold on;
for m = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Add labels, title, and legend

```

```

xlabel('Iterações');
ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta de P2 a P3');
legend('show'); % Display joint labels in the legend
grid on;

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for m = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas de P2 a P3');
legend('show'); % Exibe a legenda
grid on;

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm from P2 to P3', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
plot(err(j_ant:j), 'LineWidth', 1.5);
xlabel('Tempo (s)');
ylabel('Erro de Posição (mm)');
title('Erro de Posição de P2 a P3');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw from P1 to P2', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;

```

```

for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');
ylabel('Erro de Orientação (graus)');
title('Erro de Orientação de P2 a P3');
legend('Roll', 'Pitch', 'Yaw');
grid on;

```

Código no MATLAB - go_to_P4.m

```

K = 0.1; % Define ganh
epsilon = 10e-3; % Define critério de parada
e_ant = 1;
e = 0;

%theta = [0 0 0 -pi/2 0 -pi/2 0]'; % Define configuração inicial do robô
T4 = [Rd, P4; 0 0 0 1];
pd = P4;
% Resolver a cinemática inversa
q_solucao = robot.ikine(T4, theta, [0, 1, 1, 1, 1, 1]); % Ajuste as restrições conf
disp('Ângulos das juntas (cinemática inversa):');
disp(q_solucao);

T1 = robot.fkine(q_solucao);

% % Control loop visualization
figure(2);
robot.plot(theta');
hold on;
%T1.plot('rgb');
title('Robot Path During Control Loop');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
grid on;
view(3);

% Redundancy resolution factor (null space control)
lambda = 0.1; % Tuning parameter for redundancy resolution
j_ant = j + 1;
% Control loop
while (norm(e - e_ant) > epsilon) % Stopping criterion
    i = i + 1; % Counter
    j = j + 1; % Counter

```

```

% Calcula a jacobiana completa e cinemática direta
J_full = robot.jacob0(theta); % Jacobiana completa (6x7)
T = robot.fkine(theta); % Pose atual do efetuador

% Reduz a jacobiana (remove a contribuição da junta 1)
J_reduced = J_full(:, 2:end); % Jacobiana reduzida (6x6)

% Erros de posição e orientação
p = transl(T); % Extração da posição
R = SO3(T).R; % Extração da rotação
p_err = pd' - p; % Erro de posição
nphi = rotm2axang2(Rd * R'); % Erro de rotação (em eixo-ângulo)
nphi_err = nphi(1:3) * nphi(4); % Vetor n * phi (parte do erro angular)

% Vetor de erro combinado
e_ant = e; % Atualizar erro anterior
e = [p_err'; nphi_err'];

% Resolve o controle com a jacobiana reduzida
u_reduced = pinv(J_reduced) * (K * e); % Movimento das juntas 2 a 7

% Atualiza apenas as juntas 2 a 7
theta(2:end) = theta(2:end) + u_reduced;

% Junta 1 permanece fixa
theta(1) = 0;

% Visualização e armazenamento de dados
robot.plot(theta');
plot3(p(1), p(2), p(3), 'b.', 'MarkerSize', 3);
control_sig(:, j) = [0; u_reduced]; % Adiciona 0 como movimento da junta 1
err(j) = norm(e); % Armazena a norma do erro
joint_angles(:, j) = theta; % Armazena os ângulos das juntas para cada iteração
q_seq(:, j) = theta(2:7);
err_rot(:, j) = [nphi_err(1) nphi_err(2) nphi_err(3)];
disp('Ângulos finais das juntas:');
disp(theta);
end
%
% Open a new figure for plotting control signals
figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, j_ant:j);

% Plot control signals for each joint
hold on;
for m = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);

```

```

end
hold off;

% Add labels, title, and legend
xlabel('Iterações');
ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta de P3 a P4');
legend('show'); % Display joint labels in the legend
grid on;

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for m = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas de P3 a P4');
legend('show'); % Exibe a legenda
grid on;

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm from P3 to P4', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
plot(err(j_ant:j), 'LineWidth', 1.5);
xlabel('Tempo (s)');
ylabel('Erro de Posição (mm)');
title('Erro de Posição de P3 a P4');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw from P1 to P2', 'NumberTitle', 'off'); % Abre uma

```

```
% Plotar os ângulos para cada junta
hold on;
for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');
ylabel('Erro de Orientação (graus)');
title('Erro de Orientação de P3 a P4');
legend('Roll', 'Pitch', 'Yaw');
grid on;
```

Código no MATLAB - move_trajectory_P1.m

```
K = 1; % Define ganho
epsilon = 10e-3; % Define critério de parada
e_ant = 1;
e = 0;

T1 = robot.fkine(q_solucao);

% % Control loop visualization
figure(2);
robot.plot(theta');
hold on;
%T1.plot('rgb');
title('Robot Path During Control Loop');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
grid on;
view(3);

% Redundancy resolution factor (null space control)
j_ant = j + 1;
n_iteracoes = 300; % Número total de iterações desejadas
tempo_total = 30;
deltat = tempo_total / n_iteracoes; %

% Control loop
tic;
for ts = 1:n_iteracoes
    iter_start = tic;
    t = (ts - 1) * deltat; % Tempo atual
    %fprintf('Tempo atual: %.2f segundos\n', t);
```

```

x = P4(1) + t * (P1(1) - P4(1))/tempo_total;
y = P4(2) + t * (P1(2) - P4(2))/tempo_total;
z = P4(3) + t * (P1(3) - P4(3))/tempo_total;

pd = [x y z].';
i = i + 1; % Counter
j = j + 1; % Counter

% Calcula a jacobiana completa e cinemática direta
J_full = robot.jacob0(theta); % Jacobiana completa (6x7)
T = robot.fkine(theta); % Pose atual do efetuador

% Reduz a jacobiana (remove a contribuição da junta 1)
J_reduced = J_full(:, 2:end); % Jacobiana reduzida (6x6)

% Erros de posição e orientação
p = transl(T); % Extração da posição
R = S03(T).R; % Extração da rotação
p_err = pd' - p; % Erro de posição
nphi = rotm2axang2(Rd * R'); % Erro de rotação (em eixo-ângulo)
nphi_err = nphi(1:3) * nphi(4); % Vetor n * phi (parte do erro angular)

% Vetor de erro combinado
e_ant = e; % Atualizar erro anterior
e = [p_err'; nphi_err'];

% Resolve o controle com a jacobiana reduzida
u_reduced = pinv(J_reduced) * (K * e + [0 -1.2/tempo_total 0 0 0 0].'); % Movimento

% Atualiza apenas as juntas 2 a 7
theta(2:end) = theta(2:end) + u_reduced;

% Junta 1 permanece fixa
theta(1) = 0;

% Visualização e armazenamento de dados
robot.plot(theta');
plot3(p(1), p(2), p(3), 'r.', 'MarkerSize', 3);
control_sig(:, j) = [0; u_reduced]; % Adiciona 0 como movimento da junta 1
err(j) = norm(e); % Armazena a norma do erro
joint_angles(:, j) = theta; % Armazena os ângulos das juntas para cada iteração
q_seq(:, j) = theta(2:7);
err_rot(:, j) = [nphi_err(1) nphi_err(2) nphi_err(3)];
elapsed = toc(iter_start); % Tempo gasto na iteração
pause_time = deltat - elapsed; % Tempo restante para o próximo passo
if pause_time > 0
    pause(pause_time); % Pausa ajustada
end
% Extraia a posição atual do efetuador usando cinemática direta
end

```

```

toc;

%
% Open a new figure for plotting control signals
figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, j_ant:j);
% Plot control signals for each joint

hold on;
for n = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(n, :), 'DisplayName', ['Junta ', num2str(n)]);
end
hold off;

% Add labels, title, and legend
xlabel('Iterações');
ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta de P4 a P1');
legend('show'); % Display joint labels in the legend
grid on;

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for m = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(m, :), 'DisplayName', ['Joint ', num2str(m)]);
end
hold off;

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(n, :), 'DisplayName', ['Junta ', num2str(n)]);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas de P4 a P1');
legend('show'); % Exibe a legenda
grid on;

```

```

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm from P4 to P1', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
plot(err(j_ant:j), 'LineWidth', 1.5);
xlabel('Tempo (s)');
ylabel('Erro de Posição (mm)');
title('Erro de Posição de P4 a P1');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw from P1 to P2', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');
ylabel('Erro de Orientação (graus)');
title('Erro de Orientação de P4 a P1');
legend('Roll', 'Pitch', 'Yaw');
grid on;

```

Código no MATLAB - move_trajectory_P2.m

```

K = 1; % Define ganho
epsilon = 10e-3; % Define critério de parada
e_ant = 1;
e = 0;
T1 = robot.fkine(q_solucao);

% % Control loop visualization
figure(2);
robot.plot(theta');
hold on;
%T1.plot('rgb');
title('Robot Path During Control Loop');
xlabel('X-axis (m)');

```

```

ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
grid on;
view(3);

% Redundancy resolution factor (null space control)
j_ant = j + 1;
n_iteracoes = 300; % Número total de iterações desejadas
tempo_total = 30;
deltat = tempo_total / n_iteracoes; %

% Control loop
tic;
for ts = 1:n_iteracoes
    iter_start = tic;
    t = (ts - 1) * deltat; % Tempo atual
    %fprintf('Tempo atual: %.2f segundos\n', t);
    x = P1(1) + t * (P2(1) - P1(1))/tempo_total;
    y = P1(2) + t * (P2(2) - P1(2))/tempo_total;
    z = P1(3) + t * (P2(3) - P1(3))/tempo_total;

    pd = [x y z].';
    i = i + 1; % Counter
    j = j + 1; % Counter

    % Calcula a jacobiana completa e cinemática direta
    J_full = robot.jacob0(theta); % Jacobiana completa (6x7)
    T = robot.fkine(theta); % Pose atual do efetuador

    % Reduz a jacobiana (remove a contribuição da junta 1)
    J_reduced = J_full(:, 2:end); % Jacobiana reduzida (6x6)

    % Erros de posição e orientação
    p = transl(T); % Extração da posição
    R = SO3(T).R; % Extração da rotação
    p_err = pd' - p; % Erro de posição
    nphi = rotm2axang2(Rd * R'); % Erro de rotação (em eixo-ângulo)
    nphi_err = nphi(1:3) * nphi(4); % Vetor n * phi (parte do erro angular)

    % Vetor de erro combinado
    e_ant = e; % Atualizar erro anterior
    e = [p_err'; nphi_err'];

    % Resolve o controle com a jacobiana reduzida
    u_reduced = pinv(J_reduced) * (K * e + [0 0 -0.7/tempo_total 0 0 0].');

    % Atualiza apenas as juntas 2 a 7
    theta(2:end) = theta(2:end) + u_reduced;

    % Junta 1 permanece fixa

```

```

theta(1) = 0;

% Visualização e armazenamento de dados
robot.plot(theta');
plot3(p(1), p(2), p(3), 'r.', 'MarkerSize', 3);
control_sig(:, j) = [0; u_reduced]; % Adiciona 0 como movimento da junta 1
err(j) = norm(e); % Armazena a norma do erro
joint_angles(:, j) = theta; % Armazena os ângulos das juntas para cada iteração
q_seq(:, j) = theta(2:7);
err_rot(:, j) = [nphi_err(1) nphi_err(2) nphi_err(3)];
% Extraia a posição atual do efetuador usando cinemática direta

elapsed = toc(iter_start); % Tempo gasto na iteração
pause_time = deltat - elapsed; % Tempo restante para o próximo passo
if pause_time > 0
    pause(pause_time); % Pausa ajustada
end
toc;
% Open a new figure for plotting control signals
figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, j_ant:j);

% Plot control signals for each joint
hold on;
for m = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Add labels, title, and legend
xlabel('Iterações');
ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta de P1 a P2 Movimento Linear');
legend('show'); % Display joint labels in the legend
grid on;

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for m = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end

```

```

end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas de P1 a P2 Movimento Linear');
legend('show'); % Exibe a legenda
grid on;

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm from P1 to P2 linear speed', 'NumberTitle', 'off'); % Abre uma figura com nome personalizado

% Plotar os ângulos para cada junta
plot(err(j_ant:j), 'LineWidth', 1.5);
xlabel('Tempo (s)');
ylabel('Erro de Posição (mm)');
title('Erro de Posição de P1 a P2 Movimento Linear');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw from P1 to P2', 'NumberTitle', 'off'); % Abre uma figura com nome personalizado

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');
ylabel('Erro de Orientação (graus)');
title('Erro de Orientação de P1 a P2 Movimento Linear');
legend('Roll', 'Pitch', 'Yaw');
grid on;
%

```

Código no MATLAB - move_trajectory_PC.m

```

K = 1; % Define ganho
epsilon = 10e-3; % Define critério de parada
e_ant = 1;
e = 0;

```

```

T1 = robot.fkine(q_solucao);

% % Control loop visualization
figure(2);
robot.plot(theta');
hold on;
%T1.plot('rgb');
title('Robot Path During Control Loop');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
grid on;
view(3);

% Redundancy resolution factor (null space control)
j_ant = j + 1;
n_iteracoes = 300; % Número total de iterações desejadas
tempo_total = 30;
deltat = tempo_total / n_iteracoes; %
PC = [1; 0; 0.3];
% Control loop
tic;
for ts = 1:n_iteracoes
    iter_start = tic;
    t = (ts - 1) * deltat; % Tempo atual
    %fprintf('Tempo atual: %.2f segundos\n', t);
    x = P2(1) + t * (PC(1) - P2(1))/tempo_total;
    y = P2(2) + t * (PC(2) - P2(2))/tempo_total;
    z = P2(3) + t * (PC(3) - P2(3))/tempo_total;

    pd = [x y z].';
    i = i + 1; % Counter
    j = j + 1; % Counter

    % Calcula a jacobiana completa e cinemática direta
    J_full = robot.jacob0(theta); % Jacobiana completa (6x7)
    T = robot.fkine(theta); % Pose atual do efetuador

    % Reduz a jacobiana (remove a contribuição da junta 1)
    J_reduced = J_full(:, 2:end); % Jacobiana reduzida (6x6)

    % Erros de posição e orientação
    p = transl(T); % Extração da posição
    R = S03(T).R; % Extração da rotação
    p_err = pd' - p; % Erro de posição
    nphi = rotm2axang2(Rd * R'); % Erro de rotação (em eixo-ângulo)
    nphi_err = nphi(1:3) * nphi(4); % Vetor n * phi (parte do erro angular)

    % Vetor de erro combinado

```

```

e_ant = e; % Atualizar erro anterior
e = [p_err'; nphi_err'];

% Resolve o controle com a jacobiana reduzida
u_reduced = pinv(J_reduced) * (K * e + [0 0.6/tempo_total 0 0 0] .'); % Movimento

% Atualiza apenas as juntas 2 a 7
theta(2:end) = theta(2:end) + u_reduced;

% Junta 1 permanece fixa
theta(1) = 0;

% Visualização e armazenamento de dados
robot.plot(theta');
plot3(p(1), p(2), p(3), 'r.', 'MarkerSize', 3);
control_sig(:, j) = [0; u_reduced]; % Adiciona 0 como movimento da junta 1
err(j) = norm(e); % Armazena a norma do erro
joint_angles(:, j) = theta; % Armazena os ângulos das juntas para cada iteração
q_seq(:, j) = theta(2:7);
err_rot(:, j) = [nphi_err(1) nphi_err(2) nphi_err(3)];
% Extraia a posição atual do efetuador usando cinemática direta

elapsed = toc(iter_start); % Tempo gasto na iteração
pause_time = deltat - elapsed; % Tempo restante para o próximo passo
if pause_time > 0
    pause(pause_time); % Pausa ajustada
end
end
toc;
% Open a new figure for plotting control signals
figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, j_ant:j);

% Plot control signals for each joint
hold on;
for m = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Add labels, title, and legend
xlabel('Iterações');
ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta de P2 a PC');
legend('show'); % Display joint labels in the legend
grid on;

```

```

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for m = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas de P2 a PC');
legend('show'); % Exibe a legenda
grid on;

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm from P2 to PC', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
plot(err(j_ant:j), 'LineWidth', 1.5);
xlabel('Tempo (s)');
ylabel('Erro de Posição (mm)');
title('Erro de Posição de P2 a PC');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw from P1 to P2', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');

```

```

ylabel('Erro de Orientação (graus)');
title('Erro de Orientação de P2 a PC');
legend('Roll', 'Pitch', 'Yaw');
grid on;

```

Código no MATLAB - move_trajectory_circlee.m

```

K = 1; % Define ganho
epsilon = 10e-3; % Define critério de parada
e_ant = 1;
e = 0;

T1 = robot.fkine(q_solucao);

% % Control loop visualization
figure(2);
robot.plot(theta');
hold on;
%T1.plot('rgb');
title('Robot Path During Control Loop');
xlabel('X-axis (m)');
ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
grid on;
view(3);

% Parâmetros
Pcentro = [1; 0; 0.65]; % Centro do círculo
raio = 0.35; % Raio do círculo
j_ant = j + 1;
n_iteracoes = 300; % Número total de iterações desejadas
tempo_volta = 30; % Tempo para uma volta completa
n_voltas = 2; % Número de voltas
tempo_total = n_voltas * tempo_volta;
deltat = tempo_total / n_iteracoes; %
omega = 2 * pi / tempo_volta; % Velocidade angular (rad/s)

phi = atan2((PC(3) - Pcentro(3)) / raio, (PC(2) - Pcentro(2)) / raio);

% Control loop
tic;
for ts = 1:n_iteracoes
    iter_start = tic;
    t = (ts - 1) * deltat; % Tempo atual
    %fprintf('Tempo atual: %.2f segundos\n', t);
    x = Pcentro(1);
    y = Pcentro(2) + raio * cos(omega * t + phi);
    z = Pcentro(3) + raio * sin(omega * t + phi);

```

```

pd = [x y z].';
i = i + 1; % Counter
j = j + 1; % Counter

% Calcula a jacobiana completa e cinemática direta
J_full = robot.jacob0(theta); % Jacobiana completa (6x7)
T = robot.fkine(theta); % Pose atual do efetuador

% Reduz a jacobiana (remove a contribuição da junta 1)
J_reduced = J_full(:, 2:end); % Jacobiana reduzida (6x6)

% Erros de posição e orientação
p = transl(T); % Extração da posição
R = S03(T).R; % Extração da rotação
p_err = pd' - p; % Erro de posição
nphi = rotm2axang2(Rd * R'); % Erro de rotação (em eixo-ângulo)
nphi_err = nphi(1:3) * nphi(4); % Vetor n * phi (parte do erro angular)

% Vetor de erro combinado
e_ant = e; % Atualizar erro anterior
e = [p_err'; nphi_err'];

% Resolve o controle com a jacobiana reduzida
u_reduced = pinv(J_reduced) * (K * e + [0 -raio*omega*sin(omega*t + phi) raio*omega*cos(omega*t + phi)]);

% Atualiza apenas as juntas 2 a 7
theta(2:end) = theta(2:end) + u_reduced;

% Junta 1 permanece fixa
theta(1) = 0;

% Visualização e armazenamento de dados
robot.plot(theta');
plot3(p(1), p(2), p(3), 'r.', 'MarkerSize', 3);
control_sig(:, j) = [0; u_reduced]; % Adiciona 0 como movimento da junta 1
err(j) = norm(e); % Armazena a norma do erro
joint_angles(:, j) = theta; % Armazena os ângulos das juntas para cada iteração
q_seq(:, j) = theta(2:7);
err_rot(:, j) = [nphi_err(1) nphi_err(2) nphi_err(3)];
% Extraia a posição atual do efetuador usando cinemática direta

elapsed = toc(iter_start); % Tempo gasto na iteração
pause_time = deltat - elapsed; % Tempo restante para o próximo passo
if pause_time > 0
    pause(pause_time); % Pausa ajustada
end
end
toc;
% Open a new figure for plotting control signals

```

```

figure('Name', 'Control Signals', 'NumberTitle', 'off'); % Opens a new, named window

% Trim unused columns from control_sig (up to the current iteration)
control_sig_trimmed = control_sig(:, j_ant:j);

% Plot control signals for each joint
hold on;
for m = 1:size(control_sig_trimmed, 1) % Loop over all joints
    plot(control_sig_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Add labels, title, and legend
xlabel('Iterações');
ylabel('Sinal de Controle: u (rad/s)');
title('Sinal de Controle para cada Junta Movimento Circular');
legend('show'); % Display joint labels in the legend
grid on;

% Remover colunas não usadas
joint_angles_trimmed = joint_angles(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Joint Angles', 'NumberTitle', 'off'); % Abre uma nova janela

% Plotar os ângulos para cada junta
hold on;
for m = 1:size(joint_angles_trimmed, 1) % Loop sobre todas as juntas
    plot(joint_angles_trimmed(m, :), 'DisplayName', ['Junta ', num2str(m)]);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Iterações');
ylabel('Angulo das Juntas(rad)');
title('Angulos das Juntas Movimento Circular');
legend('show'); % Exibe a legenda
grid on;

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Norm Circular Moviment', 'NumberTitle', 'off'); % Abre uma nova

% Plotar os ângulos para cada junta
plot(err(j_ant:j), 'LineWidth', 1.5);
xlabel('Tempo (s)');

```

```

ylabel('Erro de Posição (mm)');
title('Erro de Posição Movimento Circular');
grid on;

% Remover colunas não usadas
err_rot_trimmed = err_rot(:, j_ant:j);

% Abrir uma nova figura para os ângulos das juntas
figure('Name', 'Error Row Pitch Yaw from P1 to P2', 'NumberTitle', 'off'); % Abre uma

% Plotar os ângulos para cada junta
hold on;
for n = 1:size(err_rot_trimmed, 1) % Loop sobre todas as juntas
    plot(err_rot_trimmed(n, :), 'DisplayName', ['Row', 'Pitch', 'Yaw']);
end
hold off;

% Adicionar rótulos, título e legenda
xlabel('Tempo (s)');
ylabel('Erro de Orientação (graus)');
title('Erro de Orientação Movimento Circular');
legend('Roll', 'Pitch', 'Yaw');
grid on;

```

Código no MATLAB - reproducaoJuntasCoppelia.m

```

% preambulo
clc;

% carrega as bibliotecas de apoio
addpath('./lib/libCoppelia/');

%% Parametros

%%%% Posicao inicial em graus no espaco de juntas
q_i = [0      0 -90      0  0 0] .';

% delta t do controlador
deltaT = 0.01;

%% Configuracoes iniciais

% cria o objeto de comando do comau no coppeliasim
comau = ComauSim();

% inicia a simulacao
comau.startSimulation();

```

```

% Define a posicao atual do sm6 como o vetor inicial
q = deg2rad(q_i);

% envia o robo para a posicao inicial
comau.setJointTargetPosition(q);
pause(2);
%q_seq = [45 0 -90 0 -90 0].';
%q_seq = theta(2:7);
% q_seq = [-0.5061 0.3033 -1.4529 -0.7540 -0.6942 0.8025].';

%% Loop de envio das posicoes de juntas ao robo
total_points = size(q_seq,2);
for i=1:total_points

    fprintf('Enviando ponto: %d/%d \n', i, total_points);

    % envia target position para as juntas do sm6
    % comau.setJointTargetPosition(deg2rad(q_seq(:,i)));
    comau.setJointTargetPosition(q_seq(:,i));

    % pequeno pause de acordo com o deltaT estipulado
    pause(deltaT);
end
pause(5);

% finaliza a simulacao
comau.stopSimulation();

```

Referências

- [1] SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. *Robot modeling and control.* [S.l.]: John Wiley & Sons, 2020.
- [2] CORKE, P. *Robotics, Vision and Control: fundamental algorithms in Python.* [S.l.]: Springer Nature, 2023. v. 146.