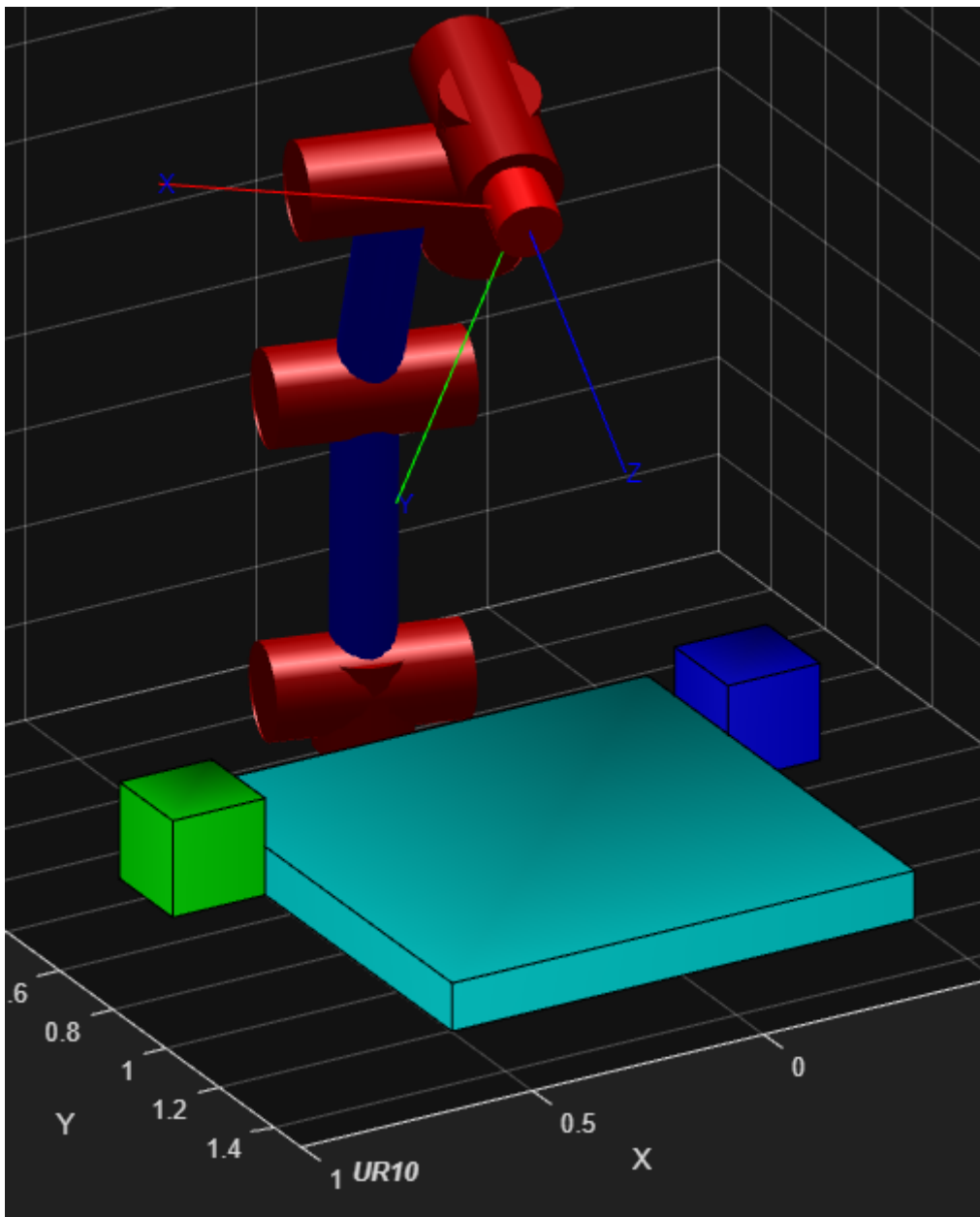


Trabalho de Introdução à Robótica

Licenciatura em Automação Industrial



Tomás Espincho (110486)

1 de novembro de 2025

Conteúdo

1	Introdução	2
2	Explicação do Código	2
2.1	Função Principal	2
2.2	Partes da Função Principal	3
2.2.1	Movimento dos blocos por Cinemática inversa	3
2.2.2	Desenho da linha através do jacobiano	3
2.3	Funções Auxiliares	3
3	Referências e Links Úteis	4

1 Introdução

Neste projeto, o objetivo consiste em utilizar o *MATLAB* para simular o robô **UR10** a realizar uma tarefa de manipulação. A simulação envolve o robô a pegar em dois blocos e a colocá-los sobre uma mesa, mantendo uma distância de 50 cm entre eles.

O robô inicia o movimento a partir da posição vertical (*home*), pega num dos blocos e posiciona-o sobre a bancada com redundância do cotovelo para cima e movimento por juntas, calculado através da cinemática inversa ([SerialLink.ikine](#)) e a posição da ponta através da cinemática direta ([SerialLink.fkine](#)). Em seguida, retorna à posição inicial (*home*) e repete o processo com o segundo bloco, desta vez com a redundância do cotovelo para baixo. Após posicionar ambos os blocos, o robô utiliza a sua ferramenta para desenhar uma linha reta sobre a bancada, movimento por juntas, calculado através do Jacobiano ([SerialLink.jacob0](#)), a ligar dois vértices dos blocos.

2 Explicação do Código

O código **MATLAB** utilizado neste trabalho é apresentado e explicado nas subsecções seguintes.

2.1 Função Principal

A função principal é responsável por inicializar as variáveis e executar o programa.

Listing 1: Função principal, e iniciação de parametros

```
1 function tp1_110486(offset)
2
3 %% Parametros Iniciais
4
5 if nargin < 0.01
6     offset = 0; % valor por omissão
7 end
8
9 %% Parametros Base do Robo
10
11 % Carregar Robo
12
13 mdl_ur10
14
15 % Criar a ponta (linha de 15 cm no eixo z do último elo)
16
17 TamanhoPonta = 0.15;
18 ur10.tool = transl(0, 0, TamanhoPonta);
19
20 Home = [-pi/2 -pi/2 0 -pi/2 -pi/2 0]; % Posição Home
21 ur10.base = transl(0, 0, 0); % Situar a Base no (0,0,0)
```

2.2 Partes da Função Principal

2.2.1 Movimento dos blocos por Cinemática inversa

1. **Definição das Posições:** São definidas as matrizes de transformação para os pontos: uma posição de aproximação segura acima do bloco, a posição exata de "pegar" (*pick*), uma posição de segurança para mover o bloco, e a posição final de "largar" (*place*). Todas as posições incluem uma rotação de 180° em torno de X para a ferramenta apontar para baixo.
2. **Cálculo da Cinemática Inversa:** Para cada posição (ex: MPickVrd), as juntas correspondente ao calculo com a função `ur10.ikine(MPickVrd)`.
3. **Geração da Trajetória:** Com as posições de juntas (ex: Home e JPickVrd), a função `jtraj(Home, JPickVrd, NSteps)` é usada para criar a trajetória.
4. **Animação e Atualização do Bloco:** O robô é animado por `ur10.animate()` ou `ur10.plot()` dentro de um ciclo `for`. Durante os movimentos em que o bloco está "agarrado", a função auxiliar `AtualizaBloco` é chamada para re-desenhar os vértices do bloco, fazendo-o seguir o robô.

2.2.2 Desenho da linha através do jacobiano

1. **Definição da Trajetória Cartesiana:** Calculo da trajetória em 3D através do `ctrj`, com o ponto inicial (MStart) e final (MFinish).
2. **Cálculo da velocidade das juntas:** Calculo da velocidades cartesianas através da função `tr2delta`.
3. **Jacobiano:** Calculo do jacobiano, através do `jacob0`, as velocidades lineares e angulares das juntas pelo jacobiano inverso através do `pinv`.

É implementado pelo seguinte ciclo `for`:

Listing 2: Ciclo for para implementar o jacobiano

```
1 for k = 1:(NSteps - 1)
2     Pos = TrajLine(k, :); % Posição atual das juntas
3     J = ur10.jacob0(Pos); % Obter o Jacobiano (6x6) para
        a configuração atual
4     Ji = pinv(J);
5     dq = Ji * DeltaPath(:, k);
6     TrajLine(k+1, :) = Pos + dq';
7 end
```

2.3 Funções Auxiliares

Complementam o funcionamento da função principal.

Listing 3: Função auxiliar usada para Criar Blocos e a "mesa"

```
1 function B = CriaBloco(Tipo, Pos, Cor, Dimensoes)
2 (... )
3 end
```

Função usa a função **Patch** do MATLAB para criar os blocos, através dos vértices e das faces do cubo/retângulo.

Listing 4: Função auxiliar para atualizar a posição do bloco

```
1 function AtualizaBloco(Bloco, traj, Robo, i)
2 (... )
3 end
```

Esta função usa o *i* de um ciclo *for*, para receber a posição anterior do bloco e atualiza-o para a próxima posição usando cinemática direta.

3 Referências e Links Úteis

- Documentação MATLAB: <https://www.mathworks.com/help/matlab/>
- Documentação do SerialLink Peter Corke:
<https://www.petercorke.com/RTB/r9/html/SerialLink.html>
- Jacobian matrix and determinant Wikipedia:
https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant
- Resposta no Matlab Help Central :
https://www.mathworks.com/matlabcentral/answers/1578550-how-to-plot-a-3d-cube-answer_823944
- Prompts usados em LLM's (Não estão em ordem cronológica)
 - **Interação 1:** Prompt do Utilizador: "Estou com dificuldades a meter acentos e tiles e ç na parte de `lstlisting` como resolvo"
 - **Interação 2:** Prompt do Utilizador: "Como faço um `linespace` em 3d matlab ?"
 - **Interação 3:** Prompt do Utilizador: "Quando atualizo o bloco, como faço para ele ficar na ponta dos 15 cm ?"
 - **Interação 4:** Prompt do Utilizador: "Quando uso o `patch` é normal deixar as faces translucidas ? Se sim não há forma de as deixar completamente opacas ?"
 - **Interação 5:** Prompt do Utilizador: "O plot esta muito grande como faço para dar plot inclinado mais pequeno"
 - **Interação 6 (Revisão):** O LLM foi usado para rever a lógica do código MATLAB (incluindo a função `AtualizaBloco` e a implementação do Jacobiano) e para ajustar o código LaTeX (resolvendo erros de compilação e acentuação).