Universidade Federal de Minas Gerais
Escola de Engenharia

ELT091 - Redes TCP IP

Turma TEE

UFMG

Trabalho prático: Simulação e controle de um robô manipulador para paletização de caixas

Danilo Siqueira Santos - 2018013917
Isabela Braga da Silva - 2018020590
Lucas Henrique Gomes Ferreira - 2018020018

1 de julho de 2022

Sumário

1 Introdução

Os manipuladores robóticos permitem que diversas atividades sejam executadas com confiabilidade e flexibilidade dentro de ambientes industriais. À vista disso, este trabalho detalha o desenvolvimento de uma simulação de um robô utilizado para paletizar caixas.

Um manipulador com esse fim especificado é amplamente utilizado em ambientes em que se faz necessário o manuseio e empilhamento de caixas contendo algum produto com peso considerável. O exemplo que inspirou o trabalho detalha o uso do robô para paletização de caixas em uma indústria de processamento de peixes [2].

O trabalho aborda a definição do projeto, estabelecendo os requisitos do mesmo, e o desenvolvimento realizado para cumpri-los. Ao fim do trabalho são apresentados os resultados e discussões para avaliação da solução proposta.

1.1 Objetivos

O manipulador robótico deve ser capaz de cumprir a trajetória especificada de forma a atender os seguintes requisitos:

- Simular um manipulador robótico e o cenário com que ele interage;
- Respeitar os limites de configuração e velocidade do robô conforme manual;
- Não colidir com o ambiente e nem com ele próprio;
- Capturar e soltar caixas nas posições desejadas;
- Não movimentar a caixa de forma brusca ou desordenada, causando perda total ou parcial de material.

2 Desenvolvimento

Essa parte do trabalho compreende o desenvolvimento do código para simular o ambiente e controlar o robô manipulador, de forma que ele seja capaz de concluir sua tarefa sem colisões ao longo do trajeto.

2.1 Definição do problema

O robô manipulador se encontra instalado em uma fábrica dedicada ao processamento de peixes. As caixas contendo os produtos chegam através de uma esteira transportadora e são recolhidas pelo efetuador do robô, que realiza um movimento

para armazená-las em um palete. As caixas tem um peso máximo de 5 quilogramas, valor escolhido para respeitar as características suportadas pelo manipulador selecionado, que será detalhado na próxima seção.

O palete utilizado para transporte dos peixes possui largura para acomodar no máximo duas caixas horizontalmente, sendo permitido criar pilhas de até 3 caixas, totalizando 6 caixas por palete. Após a conclusão, o palete deve ser removido por um operador e um vazio deve ser colocado no lugar. O robô só deve realizar o movimento para capturar uma caixa quando houver caixa na esteira transportadora e houver um palete com número de caixas menor que seis. Um esquemático do funcionamento da planta pode ser visto na Figura 1.

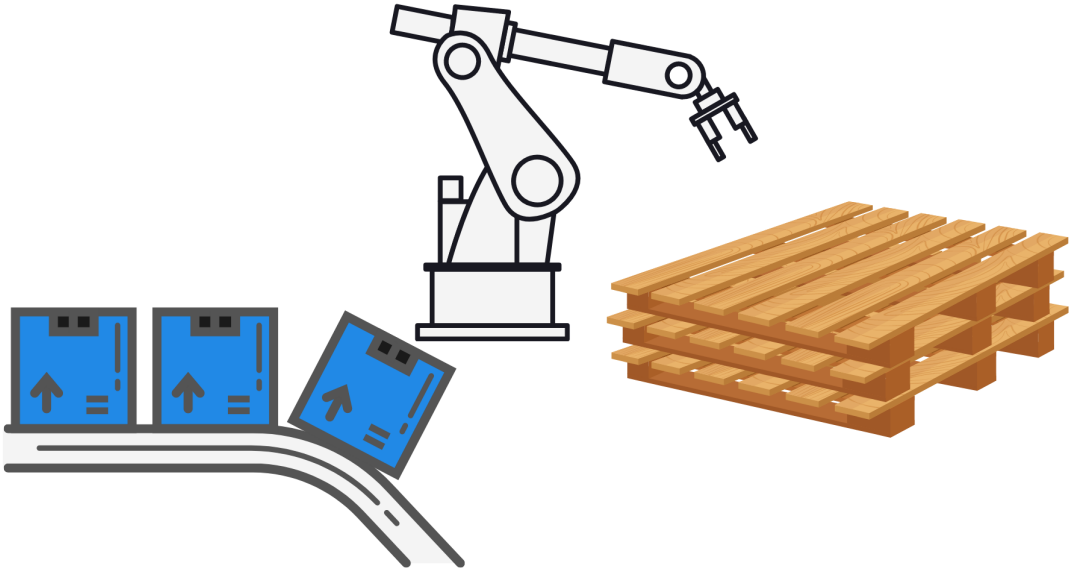


Figura 1: Esquemático de funcionamento da planta.

2.2 Robô e cenário

O robô selecionado para cumprir a tarefa proposta é o KUKA KR5 R850, cujo manual técnico pode ser visualizado em [3]. Ele possui 6 juntas rotativas e a carga máxima de trabalho é de 5 quilogramas. Na Tabela 1 é possível observar as informações sobre os limites de posição e a velocidade de cada uma das seis juntas. Para identificar as juntas e os sentidos de rotação de robô no robô físico, observa-se a Figura 2.

Além do manipulador robótico, o cenário também é composto por uma correia transportadora, um palete, uma caixa para transporte do produto e duas bases para o

Tabela 1: Limites das juntas e velocidades para uma carga nominal (por [3]).

Junta	Limites	Velocidade com carga de 5kg
1	$\pm 170^o$	$250^o/s$
2	$+45^o$ a -190^o	$250^o/s$
3	$+165^o$ a -119^o	$250^o/s$
4	$\pm 190^o$	$410^o/s$
5	$\pm 120^o$	$410^o/s$
6	$\pm 358^o$	$650^o/s$

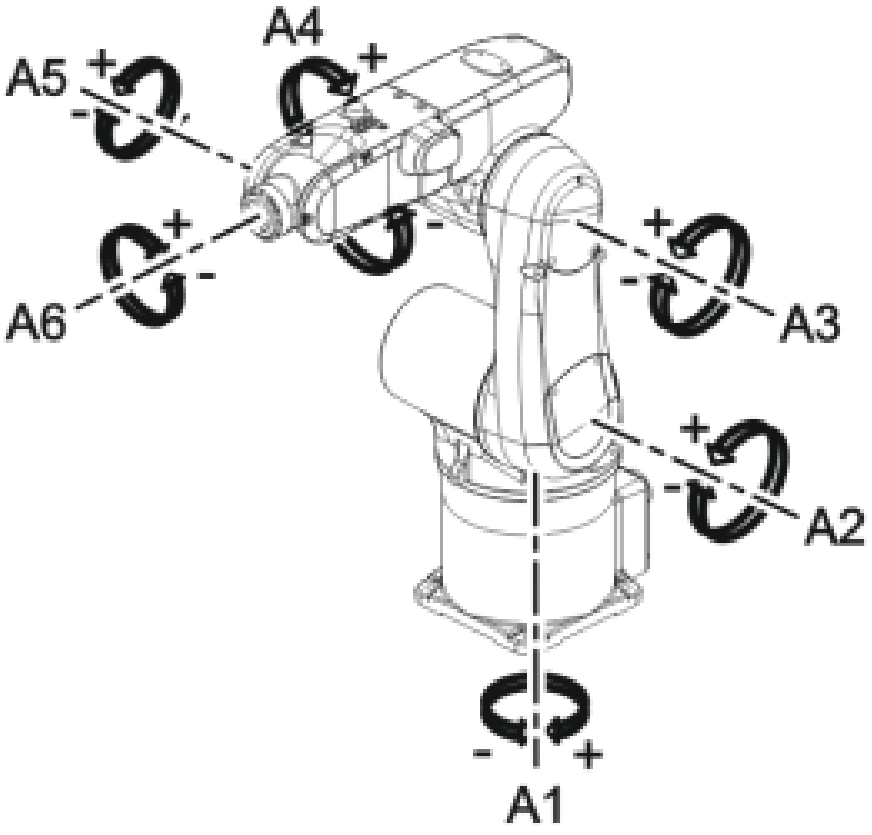


Figura 2: Robô manipulador KUKA KR5 850 com representação das juntas e sentidos de rotação (Fonte: [3]).

suporte do robô. O palete e os suportes do manipulador foram construídos através de elementos da própria biblioteca, já para a caixa e a correia transportadora foram importados de modelos 3D, como visto abaixo.

```
robot = ub.Robot.create_kuka_kr5(htm = ub.Utils.trn([0,0,
    base_quadrado.height+base_cilindro.height]), color = 'gray')
texture_conveyor = ub.Texture(
    url='https://raw.githubusercontent.com/IsabelaBraga96/
    TP_Manipuladores_Roboticos/master/Textura_aluminio.jpg',
    wrap_s='RepeatWrapping', wrap_t='RepeatWrapping', repeat=[1,
    1])

texture_box = ub.Texture(
    url='https://raw.githubusercontent.com/IsabelaBraga96/
    TP_Manipuladores_Roboticos/master/Textura_plastico.jpg',
    wrap_s='RepeatWrapping', wrap_t='RepeatWrapping', repeat=[1,
    1])


texture_wood = ub.Texture(
    url='https://raw.githubusercontent.com/IsabelaBraga96/
    TP_Manipuladores_Roboticos/master/Textura_madeira.jpg',
    wrap_s='RepeatWrapping', wrap_t='RepeatWrapping', repeat=[1,
    1])

material_caixa = ub.MeshMaterial(texture_map=texture_box, roughness
    =1, metalness=0, opacity=1,color='#004d99',reflectivity=0,
    clearcoat=0, emissive='#3c3939')
material_correia = ub.MeshMaterial(texture_map=texture_conveyor,
    roughness=0.364, metalness=0.415, opacity=1,color='#4e4f50',
    reflectivity=1, clearcoat=0.16, emissive='#3c3939')
material_wood = ub.MeshMaterial(texture_map=texture_wood, roughness
    =0.364, metalness=0.7, opacity=1)

conveyor_1 = ub.Model3D(url = 'https://raw.githubusercontent.com/
    IsabelaBraga96/TP_Manipuladores_Roboticos/master/
    Correia_Expansiva.obj', scale= 0.005, mesh_material=
    material_correia)
box_1 = ub.Model3D(url = 'https://raw.githubusercontent.com/
    IsabelaBraga96/TP_Manipuladores_Roboticos/master/Caixa_Plastico.
    obj', scale= 0.0035, mesh_material= material_caixa)

conveyor = ub.RigidObject(name = 'conveyor', list_model_3d=[
    conveyor_1],htm = ub.Utils.trn([0.7,0.2,0]) @ ub.Utils.rotz(np.
    pi/2))
box= ub.RigidObject(name = 'box', list_model_3d=[box_1],htm = ub.
    Utils.trn([0.69,0.2,0.41]) @ ub.Utils.rotz(np.pi/2))
base_pallet = ub.Box(name="base_pallet", htm = ub.Utils.trn
    ([-0.73,0.1,0.05]), width=0.8, depth=0.8, height=0.1,
```

```
    mesh_material=material_wood)
25
26 material_table = ub.MeshMaterial(roughness=0.35, metalness=1,
    opacity=1,color='#888B8D',reflectivity=1)
27 base_quadrado = ub.Box(name="base_quadrado", htm = ub.Utils.trn
    ([0,0,0.05]), width=0.5, depth=0.5, height=0.10, mesh_material=
    material_table)
28 base_cilindro = ub.Cylinder(name='base_cilindro',htm = ub.Utils.trn
    ([0,0,0.2]), radius=0.24, height=0.2, mesh_material=
    material_table)
```

Listing 1: cenario.py

2.3 Controle

2.3.1 Controle Cinemático

A técnica de controle implementada foi o controle cinemático. Nesse controlador, envia-se a velocidade de configuração desejada para cada junta do robô e assume-se que essa velocidade de junta desejada é igual a velocidade real do robô. Como a variável de processo é a própria configuração, basta que o controlador seja um integrador, como mostrado na Figura 3.

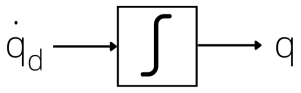


Figura 3: Controlador cinemático.

O manipulador deve remover o objeto de uma posição e colocá-lo em outra, de forma que ele deve traçar uma trajetória para interpolar os pontos alvos. Isso foi realizado de maneira discreta, assim, a referência de posição é chaveada para atingir as poses desejadas.

A pose da caixa na correia transportadora é sempre igual, ou seja, sempre que o robô buscar uma nova caixa para depositá-la no palete, ela estará na mesma posição da caixa anterior. Esse cenário não se repete quando a caixa é depositada no palete. Devido ao empilhamento, as posições serão sempre diferentes das anteriores. Apesar da orientação ser igual, ocorre um deslocamento no eixo z do mundo.

Para atender aos objetivos de controle, definiu-se, em primeiro lugar, a função de tarefa para concluir a ação esperada para o robô. Para o presente controlador, empregou-se a função de tarefa destacada na equação (1). Como o controlador implementa posições discretas, as componentes $[x_d\ y_d\ z_d]$ da função de tarefa são chaveadas durante o tempo. Entretanto, cada elemento do conjunto de posições

desejadas é invariante no tempo. Por essa razão, considera-se que a função de tarefa para a pose não depende do tempo. A implementação da função de tarefa no UAIBot é apresentada a seguir. Para computar a posição atual($S_e(q)$), calcula-se a cinemática direta do efetuador.

$$r(q) = \begin{pmatrix} S_e(q) - S_d \\ 1 - x_d^T x_e(q) \\ 1 - y_d^T y_e(q) \\ 1 - z_d^T z_e(q) \end{pmatrix} \tag{1}$$

```
# Calcula a cinematica direta e Jacobiana para o efetuador
 nessavconfiguracao
Jg, fk = robot.jac_geo(q)
#Faz a extracao de x_e, y_e, z_e e s_e
x_e = fk[0:3,0]
y_e = fk[0:3,1]
z_e = fk[0:3,2]
s_e = fk[0:3,3]

# Faz a extracao dos elementos x_d, y_d, z_d e s_d
x_d = htm_d[h][0:3,0]
y_d = htm_d[h][0:3,1]
z_d = htm_d[h][0:3,2]
s_d = htm_d[h][0:3,3]

# Monta o vetor de tarefa
r = np.matrix(np.zeros((6,1)))
r[0:3] = s_e - s_d
r[3] = 1- x_d.T * x_e
r[4] = 1- y_d.T * y_e
r[5] = 1- z_d.T * z_e
```

Listing 2: FuncaoTarefa.py

Em seguida, a função de controle escalar (FCE) $f(r)$, que faz parte do controlador, é selecionada. Nesse momento, escolhe-se uma velocidade de queda para a função de tarefa que é constante, tendo a forma apresentada na equação 2. Com o objetivo de definir uma FCE contínua, determina-se uma tolerância (ω_{tol}) que é usada para avaliar se a i-ésima componente da função de tarefa está praticamente completa. A seguir, apresenta-se a implementação da FCE no uaibot com uma velocidade de junta de 0.25 e uma tolerância de 0.01.

$$f(\omega) = \begin{cases} -A & \omega \geq \omega_{tol} \\ -A\frac{-\omega}{\omega_{tol}} & |\omega| < w_{tol} \\ A & \omega \leq \omega_{tol} \end{cases} \tag{2}$$

```
# Cria a funcao F:
def fun_F(r):
    A = [0.25, 0.25, 0.25, 0.25, 0.25, 0.25]
    w_tol = [0.01, 0.01, 0.01, 0.01, 0.01, 0.01]
    F = np.matrix(np.zeros((6, 1)))
    for i in range(6):
        if abs(r[i, 0]) < w_tol[i]:
            F[i, 0] = -A[i] * (r[i, 0] / w_tol[i])
        elif r[i, 0] >= w_tol[i]:
            F[i, 0] = -A[i]
        else:
            F[i, 0] = A[i]
    return F
```

Listing 3: FCE.py

Por fim, define-se a Jacobiana da Tarefa, necessária para calcular a ação de controle. A forma da jacobiana de tarefa é apresentada na equação (3). A seguir, é possível observar a implementação da Jacobiana de tarefa no UAIBot. A Jacobiana da Velocidade (J_v) foi calculada anteriormente durante a etapa de cálculo da função de tarefa.

$$J_r(q) = \begin{pmatrix} J_v(q) \\ x_d^T S(x_e(q)) J_\omega(q) \\ y_d^T S(x_y(q)) J_\omega(q) \\ z_d^T S(x_z(q)) J_\omega(q) \end{pmatrix} \tag{3}$$

```
  # Monta a Jacobiana de tarefa
  Jr = np.matrix(np.zeros((6,n)))

  Jr[0:3,:] = Jg[0:3,:]
  Jr[3,:] = x_d.T * ub.Utils.S(x_e) * Jg[3:6,:]
  Jr[4,:] = y_d.T * ub.Utils.S(y_e) * Jg[3:6,:]
  Jr[5,:] = z_d.T * ub.Utils.S(z_e) * Jg[3:6,:]
```

Listing 4: JacobianaTarefa.py

De posse da FCE, função de tarefa e Jacobiana de Tarefa, é possível calcular a ação de controle que será enviada para o robô. Em (4), apresenta-se a expressão para a

ação de controle. A implementação do cálculo da ação de controle no UAIBot pode ser encontrado a seguir.

$$u = \dot{q} = -J_r(q,t)^{\dagger(\varepsilon)} f(r) \tag{4}$$

```
  # Calcula a acao de controle
  u = ub.Utils.dp_inv(Jr,0.001)*fun_F(r)
```

Listing 5: AcaoControle.py

2.3.2 Geração da referência de posição

Para realizar a ação proposta no trabalho, diferentes posições devem ser atingidas durante o movimento do robô. As posições desejadas foram previamente determinadas e são selecionadas durante a execução da rotina de controle. O critério de seleção da referência de posição é baseado na própria função de tarefa definida para o controle cinemático. Conforme definição, a função de tarefa é zero se e somente se a tarefa for concluída, logo, quando seu valor se anula na execução do código do robô, uma nova posição é selecionada e o controle continua. A seguir, apresenta-se as posições previamente definidas para a ação proposta.

```
##Crias as posicoes para o robo
h=0
htm_d = []
# Pega a primeira caixa
htm_d.append(POS @ ub.Utils.trn([0,0,0.3]) @ ub.Utils.rotx(m.pi))
    #0
htm_d.append(base_pallet.htm @ ub.Utils.trn([0,-0.15,0.75]) @ ub.
    Utils.rotx(m.pi)) #1
htm_d.append(htm_d[len(htm_d)-1] @ ub.Utils.trn([0,0,0.53])) #3

# Pega a segunda caixa
htm_d.append(POS @ ub.Utils.trn([0,0,0.3]) @ ub.Utils.rotx(m.pi))
    #4
htm_d.append(base_pallet.htm @ ub.Utils.trn([0,0.15,0.75]) @ ub.
    Utils.rotx(m.pi)) #5
htm_d.append(htm_d[len(htm_d)-1] @ ub.Utils.trn([0,0,0.53])) #6

# Pega a terceira caixa
htm_d.append(POS @ ub.Utils.trn([0,0,0.3]) @ ub.Utils.rotx(m.pi))
    #7
htm_d.append(base_pallet.htm @ ub.Utils.trn([0,-0.15,0.75]) @ ub.
    Utils.rotx(m.pi)) #8
htm_d.append(htm_d[len(htm_d)-1] @ ub.Utils.trn([0,0,0.38])) #9

# Pega a quarta caixa
```

```
htm_d.append(POS @ ub.Utils.trn([0,0,0.3]) @ ub.Utils.rotx(m.pi))
    #10
htm_d.append(base_pallet.htm @ ub.Utils.trn([0,0.15,0.75]) @ ub.
    Utils.rotx(m.pi)) #11
htm_d.append(htm_d[len(htm_d)-1] @ ub.Utils.trn([0,0,0.38])) #12

# Pega a quinta caixa
htm_d.append(POS @ ub.Utils.trn([0,0,0.3]) @ ub.Utils.rotx(m.pi))
    #13
htm_d.append(base_pallet.htm @ ub.Utils.trn([0,-0.15,0.75]) @ ub.
    Utils.rotx(m.pi)) #14
htm_d.append(htm_d[len(htm_d)-1] @ ub.Utils.trn([0,0,0.23])) #15

# Pega a sexta caixa
htm_d.append(POS @ ub.Utils.trn([0,0,0.3]) @ ub.Utils.rotx(m.pi))
    #16
htm_d.append(base_pallet.htm @ ub.Utils.trn([0,0.15,0.75]) @ ub.
    Utils.rotx(m.pi)) #17
htm_d.append(htm_d[len(htm_d)-1] @ ub.Utils.trn([0,0,0.23])) #18

htm_d.append(HOME) #19 - FIM
```

Listing 6: SetpointPosicao.py

Além da seleção da próxima posição para o robô, a rotina de seleção de posição também deve comandar o efetuador do robô. Assim, dependendo da posição, comanda-se o efetuador para pegar a caixa ou soltar a caixa. A rotina de seleção de referência de posição e comando do efetuador é apresentada a seguir.

```
  if(h<(len(htm_d)-1) and all(p <= 0.0001 for p in r)):
    h = h+1; #Nova posicao
    if (h==1):
      robot.attach_object(box)
    if(h==3):
      robot.detach_object(box)
    if(h==4):
      robot.attach_object(box1)
    if(h==6):
      robot.detach_object(box1)
    if(h==7):
      robot.attach_object(box2)
    if(h==9):
      robot.detach_object(box2)
    if(h==10):
      robot.attach_object(box3)
    if(h==12):
      robot.detach_object(box3)
    if(h==13):
      robot.attach_object(box4)
```

```
    if(h==15):
      robot.detach_object(box4)
    if(h==16):
      robot.attach_object(box5)
    if(h==18):
      robot.detach_object(box5)
```

Listing 7: SelecaoPosicao.py

2.3.3 Limites das Juntas

Foi adicionado ainda a lógica para limitar o movimento e a velocidade de cada junta, de forma a seguir a especificação do manual [3]. A lógica consiste em um simples saturador que mantém os sinais dentro do *range* de operação.

```
# Limita a velocidade maxima de junta
 for j in range(n):
   if u[j] > limites_de_vel[j]:
     u[j] = limites_de_vel[j]
   elif u[j] < -limites_de_vel[j]:
     u[j] = -limites_de_vel[j]

 # Limita o movimento da junta
 for k in range(n):
   if (q[k] + u[k]*dt) > limites_de_junta[k][0]:
     u[k] = (limites_de_junta[k][0] - q[k])/dt
   elif (q[k] + u[k]*dt) < limites_de_junta[k][1]:
     u[k] = (limites_de_junta[k][0] - q[k])/dt
```

Listing 8: LimitesJuntas.py

3 Resultados e discussão

Analisando a simulação e os gráficos produzidos, percebe-se que o robô respeitou as restrições estabelecidas (limites de juntas e de velocidade de juntas) e cumpriu a tarefa determinada, ou seja, empilhou as fileiras de caixas a fim de montar o palete.

Na Figura 4 é possível ver o sinal de controle não ultrapassa os limites definidos na Tabela 1. Já na Figura 5 pode-se observar o comportamento da função de tarefa ao longo da simulação. No momento em que as funções são iguais a zero significa que o manipulador atingiu sua posição desejada e o controlador é chaveado para outra posição.

Ademais, observa-se que os limites de junta foram respeitados, pois as rotações das juntas estiveram sempre dentro do intervalo especificado como visto na Figura 6.

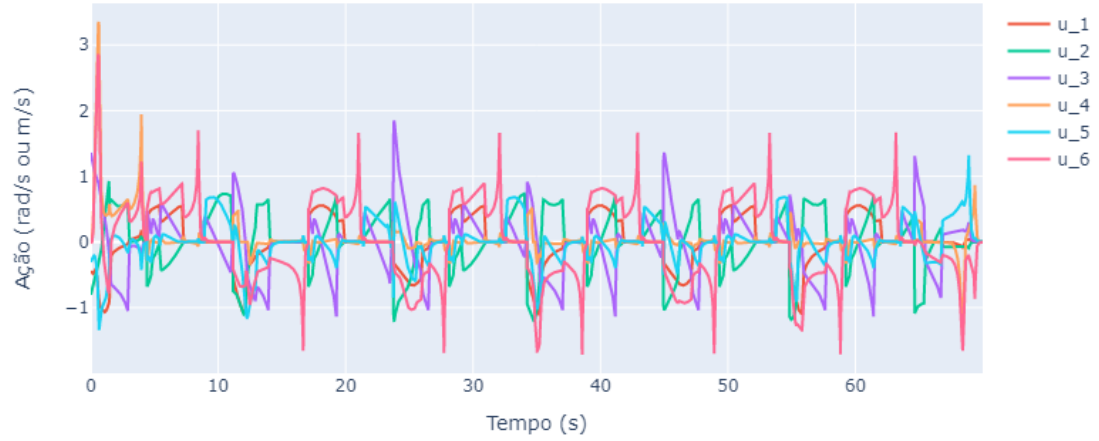


Figura 4: Ação de controle.

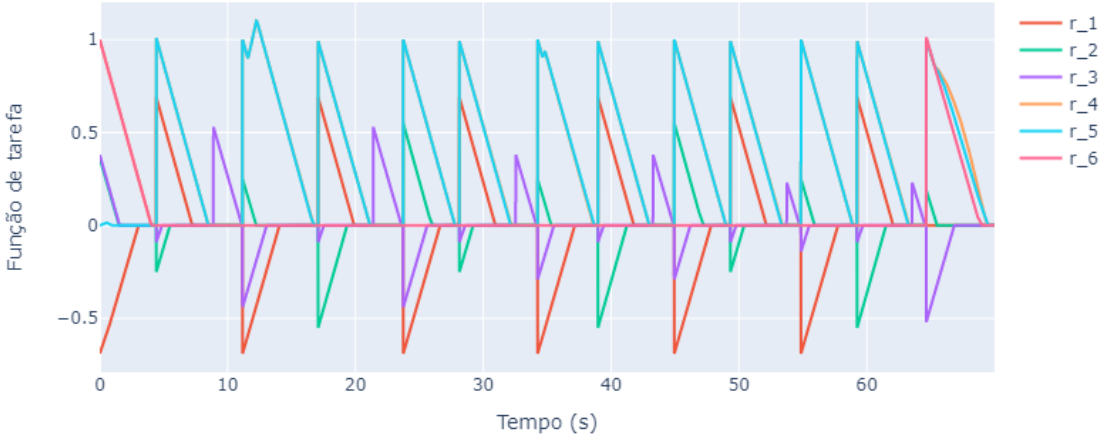
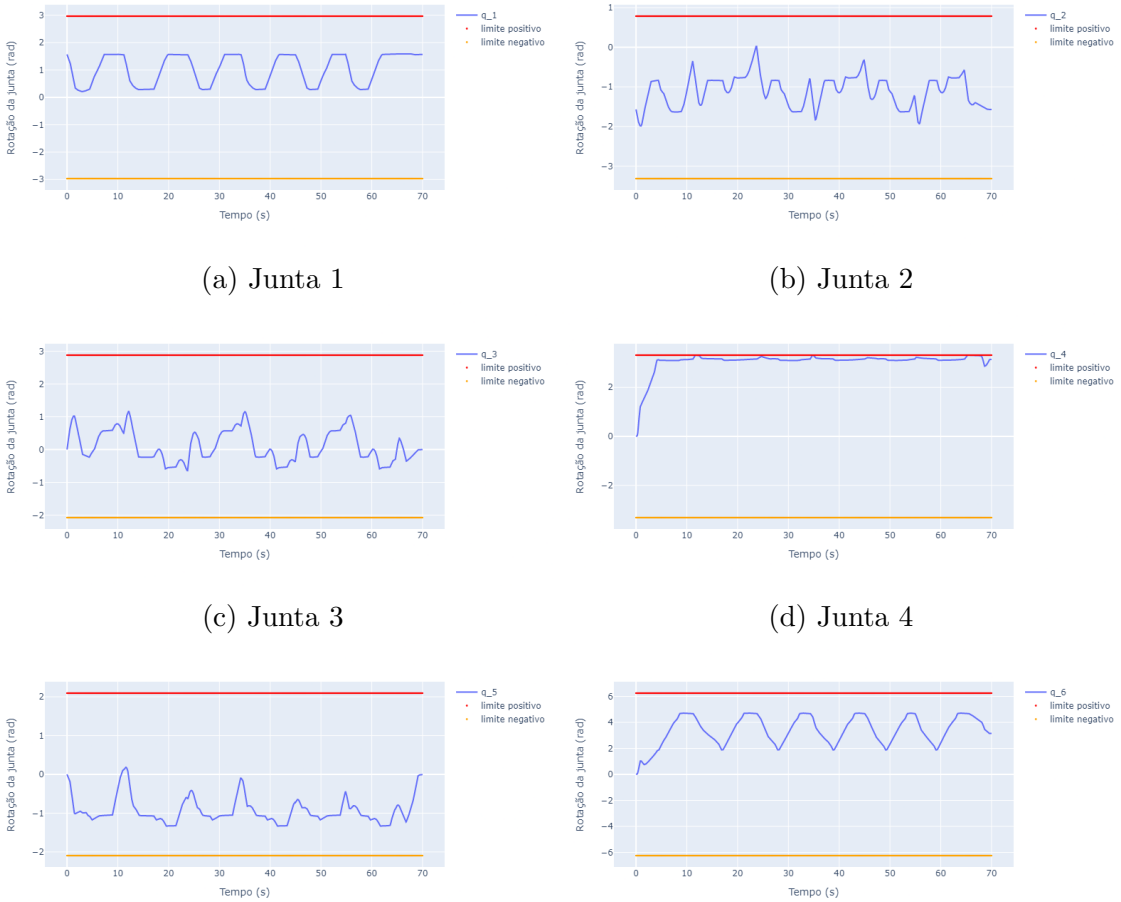


Figura 5: Função de tarefa.



(a) Junta 1

(b) Junta 2

(c) Junta 3

(d) Junta 4

(e) Junta 5

(f) Junta 6

Figura 6: Limites referentes a cada junta.

Por fim, é possível ver o desempenho do manipulador cumprindo a tarefa desejada através do link disponibilizado em [1]. É fácil identificar o fato do robô interpolar pontos alvos diferentes, realizando uma trajetória. Apesar dessa interpolação não ser contínua, ela não apresenta um movimento brusco e o robô consegue se deslocar entre um ponto e outro de forma suave. A estratégia de seleção de referência de posição permite que novas posições sejam inseridas na trajetória com facilidade, o que flexibiliza a quantidade máxima de caixas que o robô pode empilhar e o local em que elas serão posicionadas.

A caixa é sempre mantida para cima, evitando que o produto caia no chão. A cada fileira a posição alvo da caixa é incrementada para não colidir com a fileira abaixo. A tarefa é concluída quando as seis caixas são devidamente alocadas no palete, não restando nenhuma sobre a esteira. Durante todo o trajeto o robô não sofre nenhuma colisão com o ambiente ou com ele próprio, aproximando-se de uma aplicação real.

4 Conclusão

O robô manipulador proposto é capaz de empilhar três fileiras com duas caixas em cada palete. Essa simulação foi inspirada em uma fábrica de processamento de peixes, em que o manipulador robótico capta os produtos de uma correia transportadora e os deposita em uma base de palete para posterior transporte e armazenamento.

O desafio do trabalho consistiu-se em empilhar seis caixas em um palete completo para demonstrar o funcionamento da solução. O robô foi capaz de cumprir a tarefa respeitando os requisitos de não colisão e limites de movimento e velocidade especificados.

Foram utilizados modelos 3D para simular alguns itens do cenário e aproximar a solução de uma situação real. O controle realizado no manipulador é o cinemático, responsável pela manipulação das velocidades de junta. Mesmo com as limitações de movimento e velocidade, o robô conseguiu atingir seu objetivo e realizar a paletização das caixas dispostas na simulação.

Para melhorar a simulação, ela deveria ser capaz de simular o surgimento de caixas em tempos diferentes, assim seria possível tornar o controlador mais genérico, só podendo ser atuado quando houvesse produto na esteira. Outro ponto importante seria a remoção de objetos do palete, liberando o robô para continuar o trabalho de empilhamento de mais caixas. Isso tornaria a simulação mais próxima do que ocorre na realidade.

Referências

[1] Manipulador robótico usado na paletização de caixas. https://youtu.be/RuAgqOUBFWE. Vídeo no YouTube contendo a simulação realizada durante o desenvolvimento do trabalho.

[2] ©KUKA AG 2022. Robô kuka de paletização de caixas de peixe na pakfish. https://www.kuka.com/pt-br/ramos-de-atividade/banco-de-dados-de-solu%c3%a7%c3%b5es/2022/02/pakfish-e-stawiany. Visão geral de projetos utilizando robôs manipuladores.

[3] ©KUKA Roboter GmbH. Kr 5 sixx r650, r850 - specification. https://github.com/viniciusmgn/uaibot_content/blob/master/contents/Manual%20PDFs/KUKA%20KR%205%20R850.pdf. Manual Técnico do Robô KR5 R850.