UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DA COMPUTAÇÃO
CURSO SUPERIOR DE BACHALERADO EM ENGENHARIA DE COMPUTAÇÃO

Cicero Rogério Lima Tenório Filho

Matheus Macário dos Santos
Maria Eduarda Cardoso Aciole
Rafael Ramos Pimentel Santana

RELATÓRIO PARTE 2

Maceió, AL
2025

RESUMO

RELATÓRIO PARTE 2

Cicero Rogério Lima Tenório Filho

Matheus Macário dos Santos
Maria Eduarda Cardoso Aciole
Rafael Ramos Pimentel Santana
ORIENTADOR: Glauber R. Leite

Este relatório tem como objetivo apresentar o processo de desenvolvimento e as aplicações de uma simulação do braço robótico Denso VP-6242, acoplado a uma garra Robotiq modelo 2F-85. O projeto consiste em uma tarefa simples de pick and place, utilizando técnicas de Jacobiana e cinemática inversa implementadas em Python, conforme abordado em sala de aula. Além disso, serão discutidos aspectos como o espaço de trabalho, os graus de liberdade do sistema, bem como suas principais vantagens e limitações.

Palavras-chave: Denso VP-6242, Robotiq 2F-85, Jacobiana, Cinemática Inversa, ROS 2, Simulação Robótica, Pick and Place, CoppeliaSim, Python

SUMÁRIO

1 INTRODUÇÃO

Pick and Place é um dos problemas mais conhecidos e amplamente utilizados em aplicações de automação e controle, em que se exige que um atuador manipule um objeto ou componente, transportando-o até um ponto específico ou de uma maneira determinada. Esse problema consiste, essencialmente, em controlar um manipulador robótico dentro de seu espaço de configuração padrão, com o objetivo de agarrar o objeto e movê-lo até outra posição no espaço.

Este trabalho refere-se ao projeto final da disciplina de Robótica, no qual a aplicação do problema Pick and Place será implementada utilizando o braço robótico Denso VP-6242 em conjunto com a garra 2F-85 da Robotiq, por meio de uma simulação no ambiente CoppeliaSim. Além da aplicação prática, serão abordadas, de forma sucinta, algumas características construtivas e funcionais dos manipuladores, bem como suas configurações. Por conseguinte, serão apresentadas as técnicas e equações envolvidas, como cinemática direta, cinemática inversa, Jacobiana e pseudo-inversa, utilizadas tanto para descrever espacialmente a posição do robô no espaço euclidiano quanto para analisar e calcular seu comportamento e movimento durante a execução da tarefa.

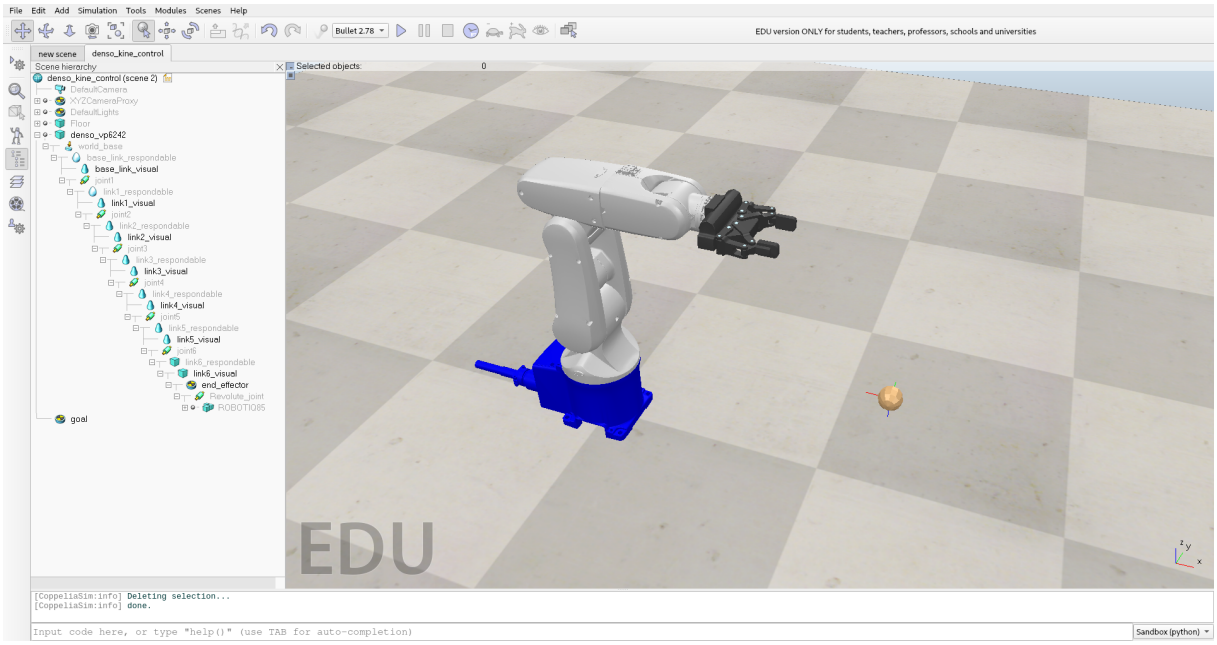

Figura 1.1 – Modelo 3D do Denso visto no CoppeliaSim

2 CARACTERÍSTICAS DO ROBÔ

Nesta seção, são apresentados os manipuladores utilizados, suas características construtivas, espaço de configuração e espaço de trabalho, além das equações que descrevem seu comportamento cinemático, como as de cinemática direta, inversa e a Jacobiana.

2.1 DESCRIÇÃO DOS MANIPULADORES

O Denso é um manipulador serial de seis graus de liberdade, com articulações rotacionais acionadas por servomotores AC de baixa potência. Cada motor é equipado com encoder absoluto, possibilitando leitura precisa da posição articular. As juntas também possuem freios, conferindo segurança e estabilidade ao sistema durante operações críticas. O braço é projetado para aplicações em ambientes compactos, mantendo elevada repetibilidade e controle.

A garra 2F-85, montada no efetuador final do braço, é um atuador elétrico adaptativo de dois dedos, desenvolvido para manipulação versátil em tarefas industriais. Seu funcionamento se dá por um mecanismo subatuado, onde um único motor controla ambos os dedos, permitindo adaptação automática ao formato dos objetos. A garra apresenta robustez construtiva e facilidade de integração, sendo compatível com controladores industriais e plataformas robóticas diversas.

2.2 ESPAÇOS DE CONFIGURAÇÃO E TRABALHO

O espaço de configuração do Denso é definido por suas seis juntas rotacionais, caracterizando um espaço $\mathbb{R}^6$. Esse arranjo permite ampla liberdade de movimento, possibilitando a realização de trajetórias complexas no espaço tridimensional. O alcance máximo do braço é de 624 mm a partir da base, com uma repetibilidade de ±0,02 mm, formando um espaço de trabalho aproximadamente esférico, adequado para tarefas de precisão e manipulação em áreas restritas.

A garra da Robotiq opera em um espaço de trabalho linear, com amplitude de abertura dos dedos de 0 a 85 mm. A força de preensão é ajustável entre 20 e 235 N, e a velocidade de fechamento varia de 20 a 150 mm/s. A repetibilidade de posicionamento é de ±0,05 mm, permitindo manipulação segura de objetos com diferentes tamanhos e características físicas, sem comprometer a precisão exigida por aplicações industriais.

3 PROBLEMAS E FUNDAMENTACAO TEORICA

A tarefa central deste trabalho e a implementacao de uma rotina de *pick and place* utilizando o robo Denso VP-6242 e a garra Robotiq 2F-85. Isso envolve o planejamento e a execucao de movimentos para que o robo colete um objeto em uma posicao inicial e o deposite em uma posicao final. A solucao para esse problema requer a aplicacao de conceitos fundamentais da cinematica de manipuladores roboticos, detalhados a seguir.

3.1 TECNICAS E EQUACOES

Com base nos Capitulos 2.9 a 3.7 do livro de Siciliano *Robotics: Modelling, Planning and Control*, foram aplicadas as seguintes tecnicas:

- Cinematica direta para obter a pose inicial do robo;
- Cinematica inversa para calcular as configuracoes articulares a partir de posicoes e orientacoes desejadas;
- Jacobiana para a modelagem dos movimentos continuos entre pontos discretos e para a cinematica inversa numerica;
- Composicao de rotacoes para definicao de orientacoes do efetuador.

Cada ponto de movimento e definido por uma configuracao θ, obtida por cinematica inversa. As matrizes de rotacao sao extraidas da parte rotacional de T_0^6, resultante da sequencia de matrizes A_i^{i+1}.

3.1.1 Cinematica Direta

$$T_0^6(\theta) = A_0^1 A_1^2 A_2^3 A_3^4 A_4^5 A_5^6 \tag{3.1}$$

Cada matriz A_i^{i+1} e definida segundo a convencao de Denavit-Hartenberg:

$$A_i(q_i) = \text{Rot}_z(\theta_i) \cdot \text{Trans}_z(d_i) \cdot \text{Trans}_x(a_i) \cdot \text{Rot}_x(\alpha_i)$$

Para uma junta rotacional, $\theta_i = q_i$, e os demais parametros sao constantes. Para uma junta prismatica, $d_i = q_i$.

A transformacao total entre a base (elo 0) e o efetuador (elo 6) e:

$$T_0^6(q) = \prod_{i=1}^{6} A_i(q_i) = \begin{bmatrix} R_6^0(q) & p_6^0(q) \\ 0 & 1 \end{bmatrix}$$

3.1.2 Cinematica Inversa

Com base no Capitulo 3.7 do livro de Siciliano, para manipuladores com punho esferico, a cinematica inversa pode ser dividida em duas etapas: posicao e orientacao.

$$p_3 = p_d - d_6 \cdot \hat{z}_6 \tag{3.2}$$

$$R_{36} = R_0^3(\theta_1, \theta_2, \theta_3)^T \cdot R_d \tag{3.3}$$

A decomposicao de R_{36} fornece os $\theta_4, \theta_5, \theta_6$.

3.1.3 Matriz Jacobiana

Com base no Capitulo 3.1, a Jacobiana relaciona velocidades articulares e operacionais:

$$\dot{x} = J(q)\dot{q}, \quad \Rightarrow \quad \dot{q} = J(q)^{-1}\dot{x} \tag{3.4}$$

Cada coluna J_i e:

- Se a junta i e rotacional:

$$J_i = \begin{bmatrix} z_{i-1} \times (p_e - p_{i-1}) \\ z_{i-1} \end{bmatrix}$$

- Se prismatica:

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

3.1.4 Composicao de Rotacoes do Pulso

A orientacao do efetuador e composta por rotacoes sequenciais:

$$R = R_z(\theta_4)R_y(\theta_5)R_z(\theta_6)$$

3.1.5 Pseudo-Inversa da Jacobiana e Cinematica Inversa Numerica

$$\Delta x \approx J(q)\Delta q \tag{3.5}$$

Se $J(q)$ e nao quadrada ou singular, utiliza-se a pseudo-inversa:

$$J^{\dagger} = J^T(JJ^T)^{-1} \tag{3.6}$$

Em regioes de singularidade, aplica-se a pseudo-inversa amortecida:

$$J^{\dagger} = J^T(JJ^T + \lambda^2 I)^{-1} \tag{3.7}$$

O algoritmo iterativo segue:

1. Calcular o erro: $e = x_d - x(q_k)$
2. Enquanto $||e|| > \varepsilon$:
   (a) Calcular $J(q_k)$
   (b) Calcular $\Delta q_k = J^{\dagger}(q_k)e$
   (c) Atualizar $q_{k+1} = q_k + \alpha\Delta q_k$
   (d) Recalcular e

4 IMPLEMENTAÇÃO E RESULTADOS

A implementação da tarefa de *pick and place* será realizada em Python, aproveitando bibliotecas para operações matriciais e a interface do simulador CoppeliaSim.

4.1 IMPLEMENTAÇÃO EM PYTHON

O seguinte código em Python implementa a lógica completa de controle para o braço robótico Denso VP-6242 e a garra Robotiq 2F-85 no ambiente de simulação **CoppeliaSim**, com integração da *Remote API*:

```python
import sim # CoppeliaSim remote API client
import time
import math
import numpy as np # For potential future kinematic calculations

# --- Simulation Configuration ---
COPPELIASIM_IP = '127.0.0.1'
COPPELIASIM_PORT = 19997 # Default port, use 19999 for non-blocking

# --- Robot and Gripper Configuration ---
ROBOT_JOINT_NAMES = [
    'joint1', 'joint2', 'joint3',
    'joint4', 'joint5', 'joint6'
]
GRIPPER_JOINT_NAME = 'Revolute_joint'
GRIPPER_OPEN_POS_DEG = 0
GRIPPER_CLOSED_POS_DEG = 35

TARGET_OBJECT_NAME = 'goal'
GRIPPER_TCP_NAME = 'robotiq_gripper'

# --- Predefined Robot Configurations ---
Q_REST = [0, 30, 60, 0, -90, 0]
Q_APPROACH_PICK = [0, 30, 60, 0, -90, 0]
Q_GRASP_OBJECT = [0, 45, 75, 0, -75, 0]
Q_RETREAT_BACK = [0, 30, 60, 0, -90, 0]
ALPHA_ROTATION_DEG = -90
Q_ROTATE_BASE = [ALPHA_ROTATION_DEG, 30, 60, 0, -90, 0]
Q_APPROACH_RELEASE = [ALPHA_ROTATION_DEG, 30, 60, 0, -90, 0]
Q_RELEASE_OBJECT = [ALPHA_ROTATION_DEG, 45, 75, 0, -75, 0]

# --- Helper Functions ---

def connect_to_coppeliasim():
    sim.simxFinish(-1)
    client_id = sim.simxStart(COPPELIASIM_IP, COPPELIASIM_PORT, True, True, 5000, 5)
    if client_id != -1:
        print("Connected to CoppeliaSim")
    else:
        print("Failed to connect to CoppeliaSim")
```

```
41                  exit()
42              return client_id
43
44          def get_object_handles(client_id):
45              handles = {'joints': [], 'gripper_joint': None, 'target_object': None, 'tcp': None}
46              for name in ROBOT_JOINT_NAMES:
47                  err_code, handle = sim.simxGetObjectHandle(client_id, name, sim.simx_opmode_blocking)
48                  if err_code == sim.simx_return_ok:
49                      handles['joints'].append(handle)
50              err_code, handles['gripper_joint'] = sim.simxGetObjectHandle(client_id, GRIPPER_JOINT_NAME,
                ↪  sim.simx_opmode_blocking)
51              err_code, handles['target_object'] = sim.simxGetObjectHandle(client_id, TARGET_OBJECT_NAME,
                ↪  sim.simx_opmode_blocking)
52              err_code, handles['tcp'] = sim.simxGetObjectHandle(client_id, GRIPPER_TCP_NAME,
                ↪  sim.simx_opmode_blocking)
53              return handles
54
55          def set_joint_target_positions(client_id, joint_handles, target_q_deg):
56              print(f"Moving to: {target_q_deg}")
57              for i, handle in enumerate(joint_handles):
58                  target_rad = math.radians(target_q_deg[i])
59                  sim.simxSetJointTargetPosition(client_id, handle, target_rad, sim.simx_opmode_oneshot)
60              time.sleep(0.5)
61              while True:
62                  all_settled = True
63                  for i, handle in enumerate(joint_handles):
64                      err_code, q_rad = sim.simxGetJointPosition(client_id, handle,
                        ↪  sim.simx_opmode_buffer)
65                      if err_code == sim.simx_return_ok:
66                          if abs(q_rad - math.radians(target_q_deg[i])) > math.radians(2):
67                              all_settled = False
68                      else:
69                          all_settled = False
70                          break
71                  if all_settled:
72                      break
73                  time.sleep(0.1)
74              print("Movement complete.")
75
76          def control_gripper(client_id, gripper_joint_handle, action, open_pos_deg, closed_pos_deg):
77              if action == "open":
78                  target_pos_rad = math.radians(open_pos_deg)
79                  print("Opening gripper...")
80              elif action == "close":
81                  target_pos_rad = math.radians(closed_pos_deg)
82                  print("Closing gripper...")
83              else:
84                  print(f"Invalid gripper action: {action}")
85                  return
86              sim.simxSetJointTargetPosition(client_id, gripper_joint_handle, target_pos_rad,
                ↪  sim.simx_opmode_oneshot)
87              time.sleep(1.0)
88              print(f"Gripper action '{action}' complete.")
89
90          def attach_object_to_gripper(client_id, tcp_handle, object_handle):
91              sim.simxSetObjectParent(client_id, object_handle, tcp_handle, True,
                ↪  sim.simx_opmode_oneshot_wait)
92              print(f"Object attached to gripper TCP.")
93
```

```
94              def detach_object_from_gripper(client_id, object_handle, scene_object_handle=-1):
95                  sim.simxSetObjectParent(client_id, object_handle, scene_object_handle, True,
                    ↪  sim.simx_opmode_oneshot_wait)
96                  print(f"Object detached from gripper.")
97
98              # --- Main Pick and Place Logic ---
99              def perform_pick_and_place(client_id, handles):
100                 print("Initializing: Moving to REST and opening gripper.")
101                 control_gripper(client_id, handles['gripper_joint'], "open", GRIPPER_OPEN_POS_DEG,
                    ↪  GRIPPER_CLOSED_POS_DEG)
102                 set_joint_target_positions(client_id, handles['joints'], Q_REST)
103
104                 print("\nStep 1: Moving to pick object...")
105                 set_joint_target_positions(client_id, handles['joints'], Q_APPROACH_PICK)
106                 set_joint_target_positions(client_id, handles['joints'], Q_GRASP_OBJECT)
107                 control_gripper(client_id, handles['gripper_joint'], "close", GRIPPER_OPEN_POS_DEG,
                    ↪  GRIPPER_CLOSED_POS_DEG)
108                 attach_object_to_gripper(client_id, handles['tcp'], handles['target_object'])
109                 time.sleep(0.5)
110
111                 print("\nStep 2: Retreating with object...")
112                 set_joint_target_positions(client_id, handles['joints'], Q_RETREAT_BACK)
113
114                 print("\nStep 3: Rotating base...")
115                 set_joint_target_positions(client_id, handles['joints'], Q_ROTATE_BASE)
116
117                 print("\nStep 4: Moving to release object...")
118                 set_joint_target_positions(client_id, handles['joints'], Q_APPROACH_RELEASE)
119                 set_joint_target_positions(client_id, handles['joints'], Q_RELEASE_OBJECT)
120
121                 print("\nStep 5: Releasing object...")
122                 control_gripper(client_id, handles['gripper_joint'], "open", GRIPPER_OPEN_POS_DEG,
                    ↪  GRIPPER_CLOSED_POS_DEG)
123                 detach_object_from_gripper(client_id, handles['target_object'])
124                 time.sleep(0.5)
125
126                 print("\nStep 6: Returning to rest...")
127                 set_joint_target_positions(client_id, handles['joints'], Q_APPROACH_RELEASE)
128                 set_joint_target_positions(client_id, handles['joints'], Q_REST)
129
130                 print("\nPick and Place sequence completed!")
131
132             # --- Script Execution ---
133             if __name__ == "__main__":
134                 client_id = connect_to_coppeliasim()
135
136                 if client_id != -1:
137                     object_handles = get_object_handles(client_id)
138
139                     if object_handles and all(object_handles['joints']):
140                         for handle in object_handles['joints']:
141                             sim.simxGetJointPosition(client_id, handle, sim.simx_opmode_streaming)
142                         time.sleep(0.1)
143
144                         try:
145                             input("Press Enter to start the Pick and Place sequence...")
146                             sim.simxStartSimulation(client_id, sim.simx_opmode_oneshot_wait)
147                             perform_pick_and_place(client_id, object_handles)
148                         except Exception as e:
```

```
149                         print(f"An error occurred: {e}")
150                     finally:
151                         sim.simxStopSimulation(client_id, sim.simx_opmode_oneshot_wait)
152                         sim.simxFinish(client_id)
153                         print("Disconnected from CoppeliaSim.")
154                 else:
155                     print("Could not obtain all necessary object handles. Exiting.")
156                     sim.simxFinish(client_id)
```

4.2 DESAFIOS DE IMPLEMENTAÇÃO

Durante o desenvolvimento do projeto de *Pick and Place* utilizando o ambiente de simulação **CoppeliaSim**, foram encontrados diversos desafios práticos. Um dos principais foi estabelecer corretamente a comunicação entre os scripts em `Python` e o simulador, por meio da *Remote API* e da biblioteca `sim`. A configuração adequada da API, incluindo a definição das portas de comunicação, o gerenciamento das conexões e o controle do fluxo de dados entre o cliente Python e o servidor CoppeliaSim, exigiu ajustes e testes detalhados.

Outro desafio foi o controle do braço robótico **Denso VP-6242**. Durante a fase inicial, ocorreram erros relacionados à modelagem do sistema no CoppeliaSim, como inconsistências nos arquivos de configuração, parâmetros incorretos e ajustes necessários nos modelos cinemáticos. Foi necessário revisar e corrigir a modelagem das juntas e das hierarquias de objetos para garantir um comportamento coerente com o esperado.

Além disso, a integração e o controle da garra **Robotiq Q85** apresentaram dificuldades. Foi necessário identificar corretamente o atuador da garra, incluí-lo na modelagem do sistema e ajustar os comandos de controle para sincronizar a preensão do objeto alvo (`goal`) com os movimentos do braço.

5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento e a simulação de uma aplicação de *pick and place* utilizando o manipulador robótico **Denso VP-6242** acoplado à garra **Robotiq Q85**, operando no ambiente de simulação **CoppeliaSim**. O projeto teve como objetivo integrar conceitos fundamentais da robótica — como a modelagem cinemática e a análise da matriz Jacobiana — com a prática de controle de robôs em ambiente virtual, utilizando programação em Python e a *Remote API* do CoppeliaSim.

Durante o desenvolvimento, foi realizada uma análise teórica consistente sobre a cinemática direta e inversa, com base na convenção de Denavit-Hartenberg, bem como o estudo detalhado da matriz Jacobiana e da sua pseudo-inversa para controle diferencial. Esses conceitos foram documentados e compreendidos, conforme a fundamentação teórica apresentada. Entretanto, no escopo desta implementação prática, não foi possível integrar completamente esses métodos para controlar o robô de forma automática por meio da Jacobiana ou da cinemática inversa numérica em tempo real. A execução da tarefa de *pick and place* foi realizada por meio de comandos diretos de posição articular (target de juntas), com sequências de ângulos previamente definidos e calibrados empiricamente.

Essa limitação prática decorreu de desafios técnicos, tais como a complexidade da integração das bibliotecas de cálculo cinemático com o CoppeliaSim, a necessidade de um mapeamento exato entre o modelo geométrico e o modelo utilizado no simulador e escopo do projeto. Ainda assim, o sistema foi capaz de executar com sucesso a sequência de movimentação planejada, incluindo a manipulação do objeto alvo (`goal`) com a garra.

O trabalho também evidenciou desafios típicos do desenvolvimento de sistemas robóticos em simulação, tais como a configuração e depuração da comunicação Python–CoppeliaSim, o ajuste fino dos modelos cinemáticos e o controle preciso da garra adaptativa.

Em suma, o projeto consolidou conhecimentos teóricos essenciais da disciplina de Robótica e promoveu habilidades práticas no uso de ferramentas modernas para simulação e controle de manipuladores, constituindo uma base sólida para o aprofundamento futuro na área.

REFERÊNCIAS BIBLIOGRÁFICAS

Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani e Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. London: Springer-Verlag, 2009. ISBN: 978-1-84628-641-4. Disponível em: <https://link.springer.com/book/10.1007/978-1-84628-642-1>. Acesso em: 1 jun. 2025.

Glauber R. Leite. *Denso Pick and Place using ROS 2 and MoveIt 2*. 2023. Disponível em: <https://github.com/glauberrleite/denso-pick-and-place>. Acesso em: 1 jun. 2025.

Robotics Ninja. *Pick and Place in CoppeliaSim (vídeo)*. 2023. Disponível em: <https://youtu.be/9X8QVcuJvQ4?si=DVyIJmF6Ux1BMqCu>. Acesso em: 1 jun. 2025.

IFRA-Cranfield. *ros2_RobotSimulation*. 2023. Disponível em: <https://github.com/IFRA-Cranfield/ros2_RobotSimulation>. Acesso em: 1 jun. 2025.

A. Price. *robotiq_arg85_description*. 2020. Disponível em: <https://github.com/a-price/robotiq_arg85_description>. Acesso em: 1 jun. 2025.

Aboul Ella Hassanien e Hesham A. Hefny. Fuzzy logic and genetic algorithms for intelligent control and robotics. *Journal of Advanced Transportation*, v. 50, n. 6, p. 1210–1224, 2016. Disponível em: <https://onlinelibrary.wiley.com/doi/full/10.1155/2016/5720163>.

Antonyviser. *Inverse Kinematics and Jacobian (Playlist)*. 2022. Disponível em: <https://www.youtube.com/playlist?list=PLNX7fogrcaPLlaHoQMGVMbBE05OyyiQQG>. Acesso em: 1 jun. 2025.

Bruno G. Lima. *vp6242_robotiq85_ros*. 2023. Disponível em: <https://github.com/bglima/vp6242_robotiq85_ros>. Acesso em: 1 jun. 2025.

DENSO ROBOTICS. *Ficha técnica: VP-5243 e VP-6242 – Especificações técnicas*. Disponível em: <https://www.densorobotics-europe.com/fileadmin/Products/VP-5243_and_VP-6242_technical_Data_Sheet/VP-5243_and_VP-6242_technical_Data_Sheet.pdf>. Acesso em: 2 jun. 2025.

Saadi-Tech. *Pick and Place and Color-Based Sorting using KUKA KR16 Robot in CoppeliaSim and MATLAB*. 2023. Disponível em: <https://github.com/saadi-tech/Pick-n-place-and-color-based-sorting-using-KUKA-Kr16-robot-in-Coppeliasim-MATLAB/tree/main/KUKA-KR16%20Sequence%20pick%26place>. Acesso em: 1 jun. 2025.

Sagar Chotalia. *Pick and Place Robot Eklavya*. 2021. Disponível em: <https://github.com/sagarchotalia/Pick-and-Place-Robot-Eklavya>. Acesso em: 1 jun. 2025.