**Media Engineering and Technology Faculty**
**German University in Cairo**


# MCTR911
# Robotics Programming Project

**Milestone 2**
**Report**

Team 6: Sara Wasfy, 52-0356
Mayar Bahnacy, 52-7251
Mazen Mostafa, 52-12990
Kareem Elkaderi, 52-1441
Teodore Farag, 52-3837
Ahmed Hemeida, 43-18800

Submission Date: 2 October, 2024

# Contents

# Chapter 1

# Project Flow

## 1.1 Milestone 1: Literature Review and Initial Setup

- **Literature Review:** Research and write a two-page review on Industrial Robotics Applications focusing on pick-and-place robots (5-10 papers starting from 2015).
- **Select Application:** Choose the pick-and-place robot for industrial applications (e.g., packaging, assembly line). Propose a draft project flow to guide development.
- **CAD Model Search:** Search and select desktop robotic manipulators with 3-4 DOF from GrabCAD.
- **Simulator Setup:** Download and install CoppeliaSim for robot simulation. GitHub Setup: Create a GitHub repository with a folder named "Milestone 01".
- **Include:** Two-page literature review. CAD models of robotic manipulators. Video showing the installation of CoppeliaSim.
- **Deliverables:** GitHub Repository with "Milestone 01" folder: Literature review. CAD models. Video of simulator setup.

## 1.2 Milestone 2: Kinematic Analysis and Simulation

- **Frame Assignment:** Assign coordinate frames to the robotic arm joints and base in CoppeliaSim.
- **Denavit-Hartenberg (DH) Parameters:** Develop the DH parameters to define the kinematics of the robot. Forward and Inverse Kinematics: Use MATLAB/Simulink or Python to derive the forward and inverse kinematics of the robotic arm. Test and verify the results using simulations in CoppeliaSim.

- **Simulation Setup:** Import the CAD model of the robot into CoppeliaSim. Simulate the robot's movement using joint inputs and verify motion paths.
- **Deliverables:** GitHub Repository with "Milestone 02" folder: Report on DH convention and kinematic equations. Simulation code (MATLAB/Simulink or Python). Video showing the simulation of robot motion in CoppeliaSim.

## 1.3 Milestone 3: Velocity, Acceleration Kinematics, and Trajectories

- **Velocity and Acceleration Kinematics:** Derive forward and inverse velocity/acceleration kinematics using MATLAB/Simulink or Python. Simulate and validate these in CoppeliaSim by testing the robotic arm's movement with velocity and acceleration inputs.
- **Task-Space and Joint-Space Trajectories:** Generate Task-Space Trajectories for the end-effector to follow a path. Develop Joint-Space Trajectories for the robotic joints based on the end-effector path.
- **Simulate and Validate Trajectories:** Implement the task-space and joint-space trajectories in CoppeliaSim and validate the results.
- **Deliverables:** GitHub Repository with "Milestone 03" folder: Report on velocity/acceleration kinematics and trajectory planning. Code for velocity, acceleration kinematics, and trajectory tracking. Video showing simulation and trajectory validation.

## 1.4 Milestone 4: Trajectory Validation in GUI

- **GUI and Environment Building:** Continue building the GUI in CoppeliaSim to represent the environment (conveyor belts, workspace, etc.).
- **Trajectory Validation:** Validate the Task-Space and Joint-Space trajectories in the CoppeliaSim environment. Simulate the robot's performance following these paths and fine-tune based on observations.
- **Hardware Integration (Optional):** If applicable, integrate the simulation with a hardware system and validate the robot's movement on real hardware.
- **Deliverables:** GitHub Repository with "Milestone 04" folder: Videos of trajectory validation in CoppeliaSim. Code for trajectory validation. Video showing hardware validation (if applicable).

## 1.5 Milestone 5: Control and Final Validation

- **Position Control (PID):** Implement a PID control algorithm to regulate the robotic arm's joint positions.
- **Closed-Loop System Testing:** Test the closed-loop system in CoppeliaSim to ensure the robot can follow the desired trajectories accurately using position feedback.
- **Force Control (Optional):** Implement force control for more advanced manipulation tasks, such as handling delicate or variable-weight objects.
- **Final Validation:** Perform final validation of the complete pick-and-place robot system in CoppeliaSim and (if applicable) integrate with hardware for real-world testing.
- **Deliverables:** GitHub Repository with "Milestone 05" folder: Report and code for control systems. Video showing the complete robot system in simulation and hardware (if applicable).

# Chapter 2

# Literature Review

Sherwani et al. (2020) emphasize the increasing role of collaborative robots (cobots) in Industry 4.0, particularly in tasks such as pick-and-place. Cobots, with their adaptability, safety, and ease of programming, are ideal for shared workspaces in industries like manufacturing, where repetitive tasks such as material handling and packaging are essential. This study highlighted the efficiency gains of cobots in smart factories powered by IoT and AI, which directly influenced our decision to focus on pick-and-place operations. The use of cobots in such environments can be accurately simulated in CoppeliaSim, allowing us to refine their performance before deployment[7].

Sanchez et al. (2018) review robotic manipulation of deformable objects, which are often encountered in industrial pick-and-place tasks. Their insights into the challenges of handling objects like cables, fabrics, and soft materials informed our choice of application, as CoppeliaSim allows us to simulate these complex dynamics. By testing how our robot interacts with these materials, we can ensure that it performs tasks like sorting and packaging with precision, an essential capability in industries dealing with delicate or flexible objects[6].

Javaid et al. (2021) emphasize the critical role of pick-and-place robots in high-speed manufacturing environments, where speed and precision are paramount. Their discussion of the integration of robots with IoT and AI for real-time monitoring and predictive maintenance supported our decision to focus on pick-and-place. Using CoppeliaSim, we can simulate these operations to optimize the robot's movements and ensure its efficiency in high-demand environments like electronics assembly, where continuous operation without fatigue is a key advantage[4].

Villani et al. (2018) focus on human-robot collaboration (HRC) and how pick-and-place robots contribute to improving safety and productivity in industrial settings. Their findings on the importance of intuitive interfaces and safety protocols reinforced our choice to prioritize pick-and-place applications, where seamless human-robot interaction is critical. CoppeliaSim allows us to simulate and test these safety mechanisms, ensuring that our robot can perform repetitive tasks like packaging and inspection in environments shared with human workers without compromising safety[8].

Collins et al. (2019) discuss the importance of simulators in robotics research, particularly for testing robotic systems in safe and cost-effective environments. Their emphasis on simulators like CoppeliaSim as a tool for optimizing robotic performance informed our decision to use it for pick-and-place operations. CoppeliaSim provides a robust platform to simulate and refine the robot's movements and interactions with various objects, ensuring accuracy and efficiency before implementing it in real-world manufacturing[1].

Feng et al. (2024) introduce a knowledge graph-based framework that enables robots to autonomously infer operational parameters, which influenced our decision to incorporate intelligent decision-making in our pick-and-place application. Testing such a framework in CoppeliaSim ensures that our robot can autonomously adapt to different tasks, such as picking various items in a production line, without requiring constant reprogramming. This aligns with our goal of improving robot autonomy and flexibility in manufacturing[3].

Wang and Hauser (2020) examine the optimization of robot-assisted packing in environments with unpredictable item arrival. Their algorithms for handling nondeterministic item sequences were directly applicable to our pick-and-place focus, where the robot must efficiently manage varying sequences of objects. Simulating these scenarios in CoppeliaSim ensures that the robot can handle the uncertainty of item arrival in warehouse and logistics environments, optimizing space and time efficiency[9].

Tehrani et al. (2022) review robotics in industrialized construction, focusing on tasks like pick-and-place for offsite and onsite assembly. Their findings on the need for greater automation in construction informed our choice of application, as CoppeliaSim allows us to simulate these complex tasks in a controlled environment. Testing the robot's ability to handle construction materials ensures that it can perform efficiently and safely in real-world construction environments[5].

Dzedzickis et al. (2022) highlight the advancements in collaborative robotics across sectors like healthcare, agriculture, and manufacturing, with a specific focus on pick-and-place tasks. The integration of AI and sensor technologies for precise operations guided our choice to prioritize pick-and-place, as CoppeliaSim allows for the simulation of sensor-driven, precision-based interactions. This is particularly useful in industries where accuracy in handling small, delicate objects is essential[2].

Yuan and Lu (2023) discuss the impact of industrial robots on maintaining competitive production rates in global value chains, particularly through pick-and-place tasks that improve efficiency. Their insights into how automation enhances productivity and reduces errors supported our decision to focus on pick-and-place operations, which can be optimized in CoppeliaSim. This allows us to test and refine the robot's performance to meet the demands of a highly competitive global market[10].

In conclusion, insights from the reviewed literature highlight the relevance and importance of pick-and-place operations in various industrial applications. By using CoppeliaSim, we can simulate and optimize these operations to ensure that the robot performs with precision, safety, and efficiency, making it well-suited for real-world industrial environments.

# Chapter 3

# GrabCAD Models

## 3.1 Model 1

This model is named in zip folders as 4-dof-robot-arm-4.snapshot.4. The robotic arm was designed with four degrees of freedom and programmed to accomplish accurate material lifting tasks to assist in the production line in any industry. It consists of 4 servo motors, one at the base, 2 at the body, and one in the end effector.
Link: https://grabcad.com/library/4-dof-robot-arm-4

Figure 3.1: Model 1

Figure 3.2: Model 1

Figure 3.3: Model 1

## 3.2 Model 2

This model named in zip folders as 4-dof-robotic-arm-1.snapshot.3. The robotic arm or a mechanical arm is a programmable robot with 4 degrees of freedom. It is designed to perform functions that are similar to a human arm. The arm can act individually and as a part of some more complex Robot. The links of these arms are connected by joints connected at the end with the end effector.
Link: https://grabcad.com/library/4dof-robotic-arm-1

Figure 3.4: Model 2

## 3.3 Model 3

This model named in zip folders as robot-arm-13.snapshot.1. The robotic arm was designed with four or five degrees of freedom and used to lift or move objects.
Link: https://grabcad.com/library/robot-arm-13

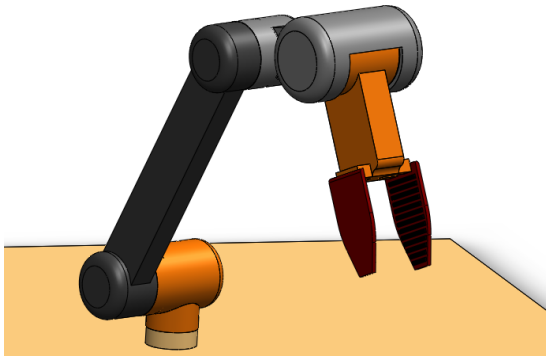


Figure 3.5: Model 3

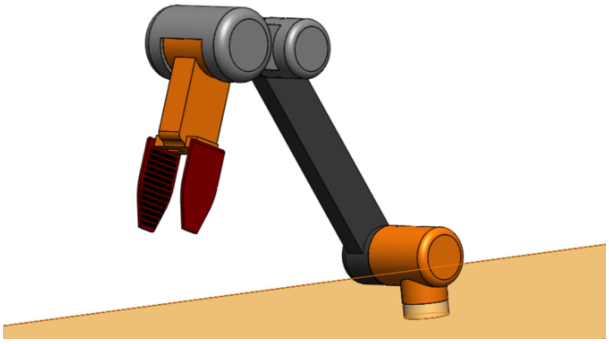


Figure 3.6: Model 3

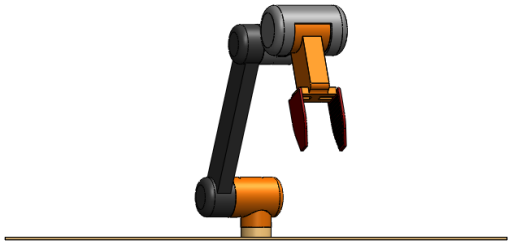


Figure 3.7: Model 3

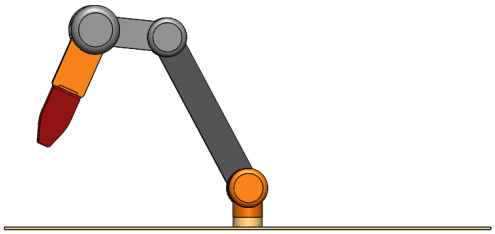


Figure 3.8: Model 3

# Chapter 4

# Coordinate Frame

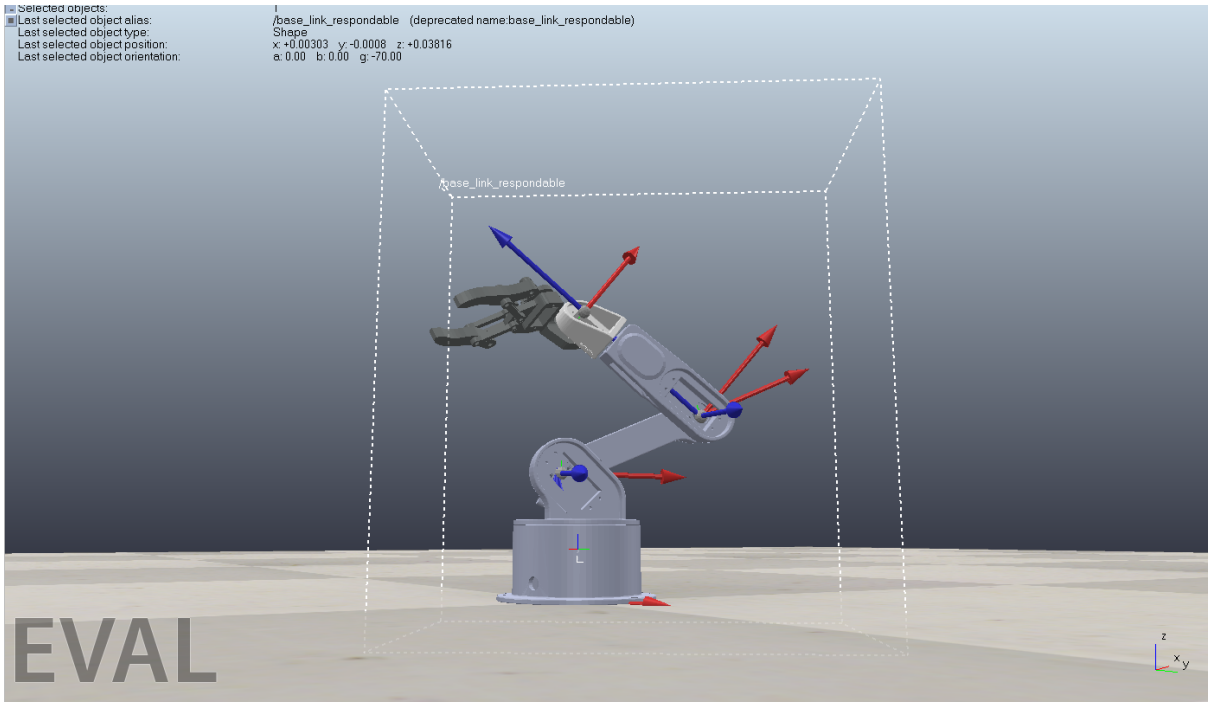



Figure 4.1: Robot Coordinate Frames

In Figure 4.1, we can see a robotic arm with coordinate frames assigned according to the Denavit-Hartenberg (DH) convention. The coordinate frames are represented by colored axes: red for the X-axis,blue for the Z-axis, and green for the Y-axis, positioned at various joints and links of the robot.

**Base (Joint 1):**

- The bottom part of the robotic arm (attached to the ground) is the base.
- This part rotates about the Z-axis (denoted by the blue arrow).

- It represents the first revolute joint, allowing rotation about the vertical axis. The DH frame is assigned here, with the Z-axis pointing upwards and the X-axis (red) pointing along the arm's rotational axis.

**Link 1:**

- The first link extends from the base to the next joint.
- According to the DH-convention, the X-axis is aligned along the common normal between the joint axes, and the Z-axis is aligned with the axis of the first joint.

**Joint 2:**

- The next joint controls the vertical motion of the arm.
- This is another revolute joint, where the rotation occurs in a different plane. The Z-axis at this joint points along the axis of rotation.

**Link 2:**

- Extending from Joint 2, this link holds the next part of the robotic arm.
- The DH frame follows the convention, with the Z-axis pointing in the direction of the rotational axis of Joint 2.

**Joint 3:**

- The third joint is located at the elbow-like structure, controlling the arm's bending motion.
- It is another revolute joint, and the Z-axis aligns with the elbow's rotational axis.

**End-Effector:**

- The robotic arm's end-effector is visible at the top, represented by a gripper.
- The end-effector is responsible for manipulating objects.
- The DH frame for the end-effector is assigned based on the last link's positioning and the orientation of the end-effector.

# Chapter 5

# DH-Convention

## 5.1 Table

| Dh Convention | | | | |
|---|---|---|---|---|
| | | | | |
| Joint | theta | delta | a | alpha |
| 1 | q1 | l1 = 0.09522 | 0 | 90 |
| 2 | q2 | 0 | l2 | 0 |
| 3 | q3 | 0 | 0 | -90 |
| ee | q4 | l3+l4 | 0 | 0 |
| | | | | |

Figure 5.1: DH-Convention Table

## 5.2 Final Matrix

```
[[ 8.08134280e-01 -4.60830980e-01 -3.66815747e-01 -3.57486755e-02]
 [ 4.95984940e-01  8.68329205e-01  1.82498791e-03  1.77850030e-04]
 [ 3.17675815e-01 -1.83409922e-01  9.30291824e-01  2.14726319e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

Figure 5.2: DH-Convention Final Matrix

Figure 5.2 illustrates the total transformation matrix from the base reference frame to the end effector reference frame. This transformation is crucial for understanding the position and orientation of the end effector relative to its base frame.

# Chapter 6

# Simulation

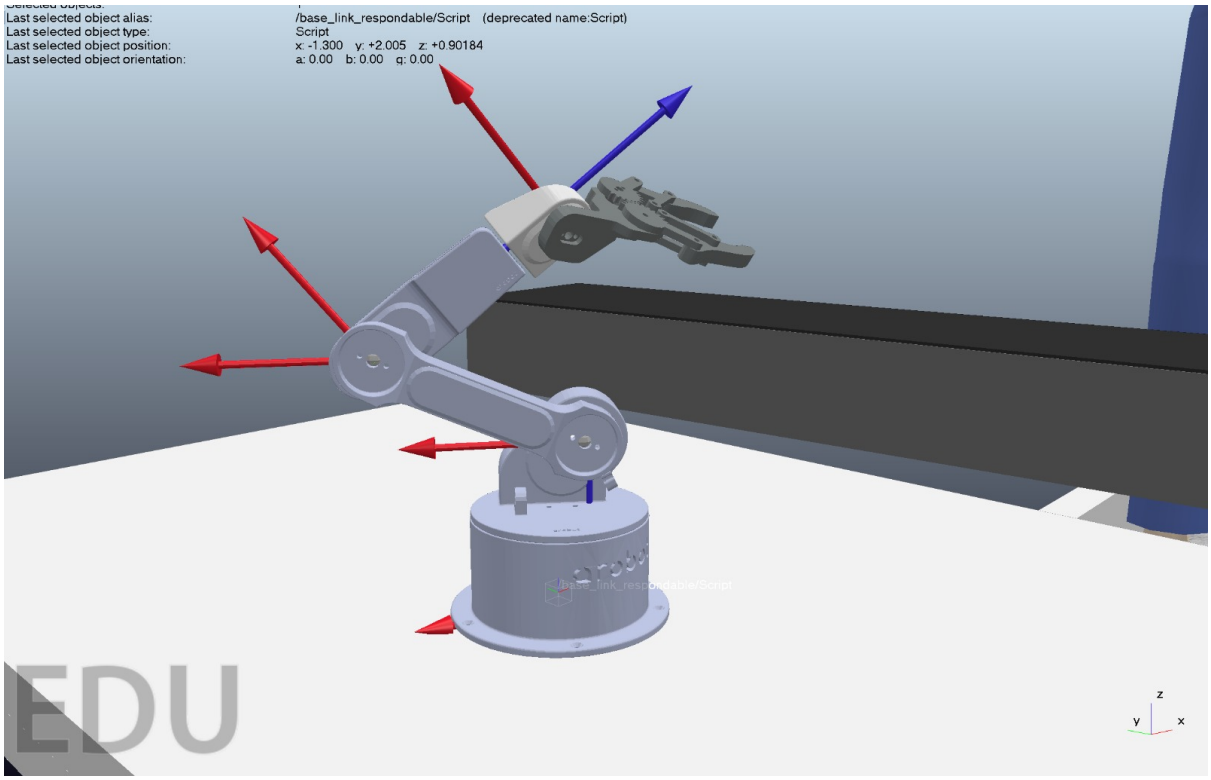



Figure 6.1: Robot Manipulator Simulation - Joint Movements

In these images, we observe a robotic manipulator simulated in a 3D environment. The manipulator's joints are actively moving, and the corresponding coordinate frames—based on the Denavit-Hartenberg (DH) convention are visualized at various key points.

The images showcase the manipulator in various positions as its joints move, simulating the arm's interaction with its environment. The coordinate frames illustrate the current orientation and position of the joints during the simulation.

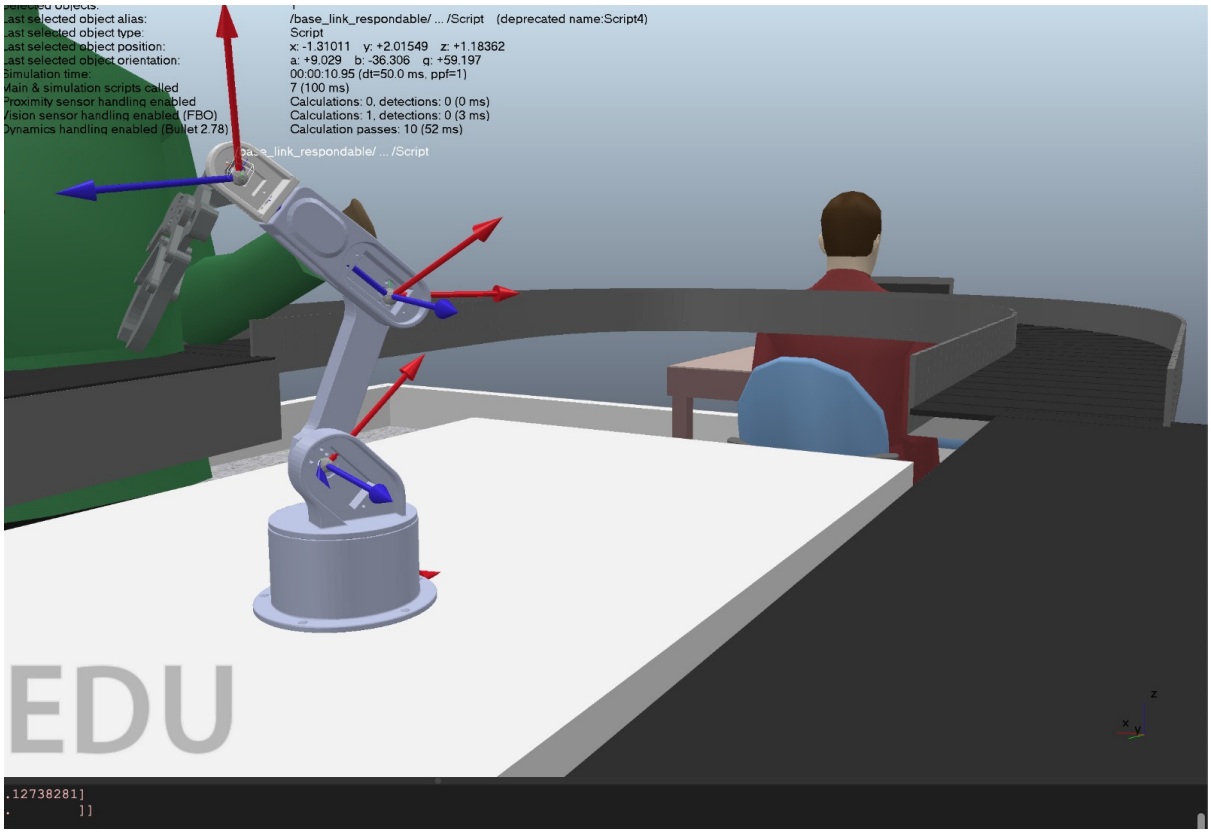



Figure 6.2: Robot Manipulator Simulation - Advanced Joint Position

In the background, we notice additional objects and characters, such as a person seated at a desk, indicating that the robot might be part of a larger simulation for interaction with surrounding components, such as the visible conveyor belt. This setup suggests the robot is likely involved in tasks like pick-and-place operations or object manipulation within the environment.

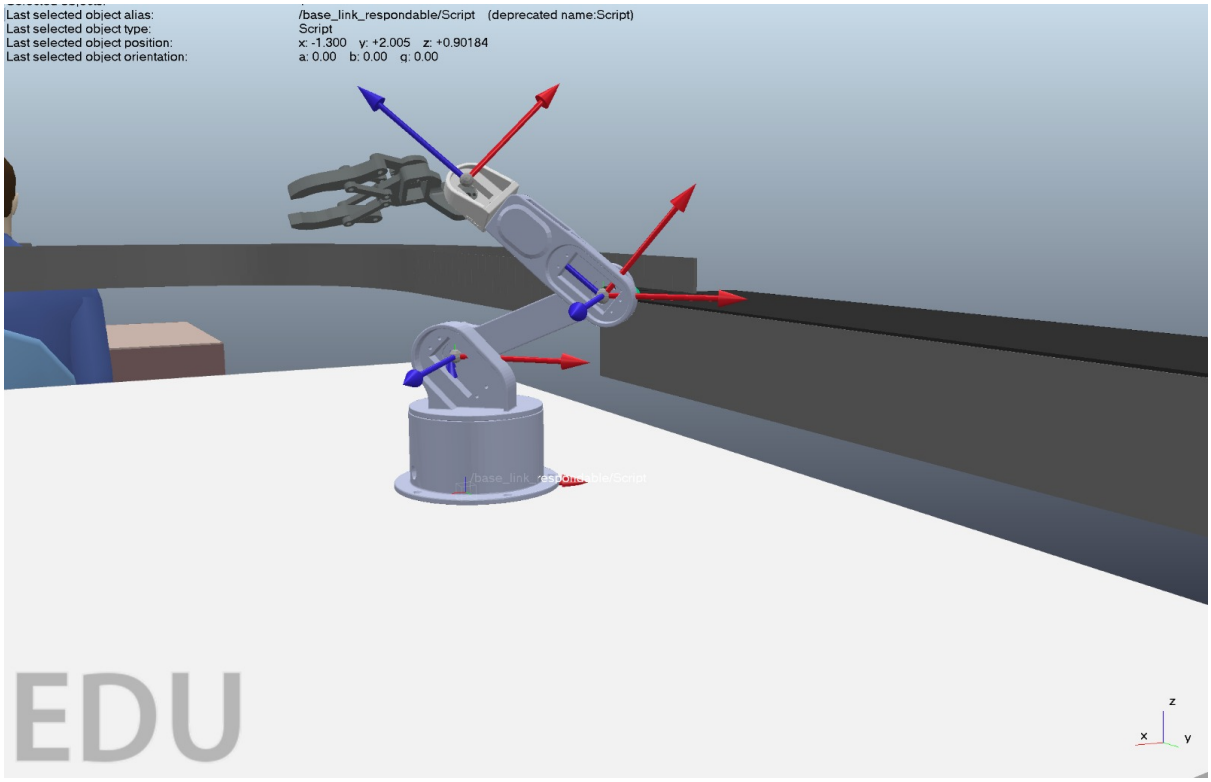
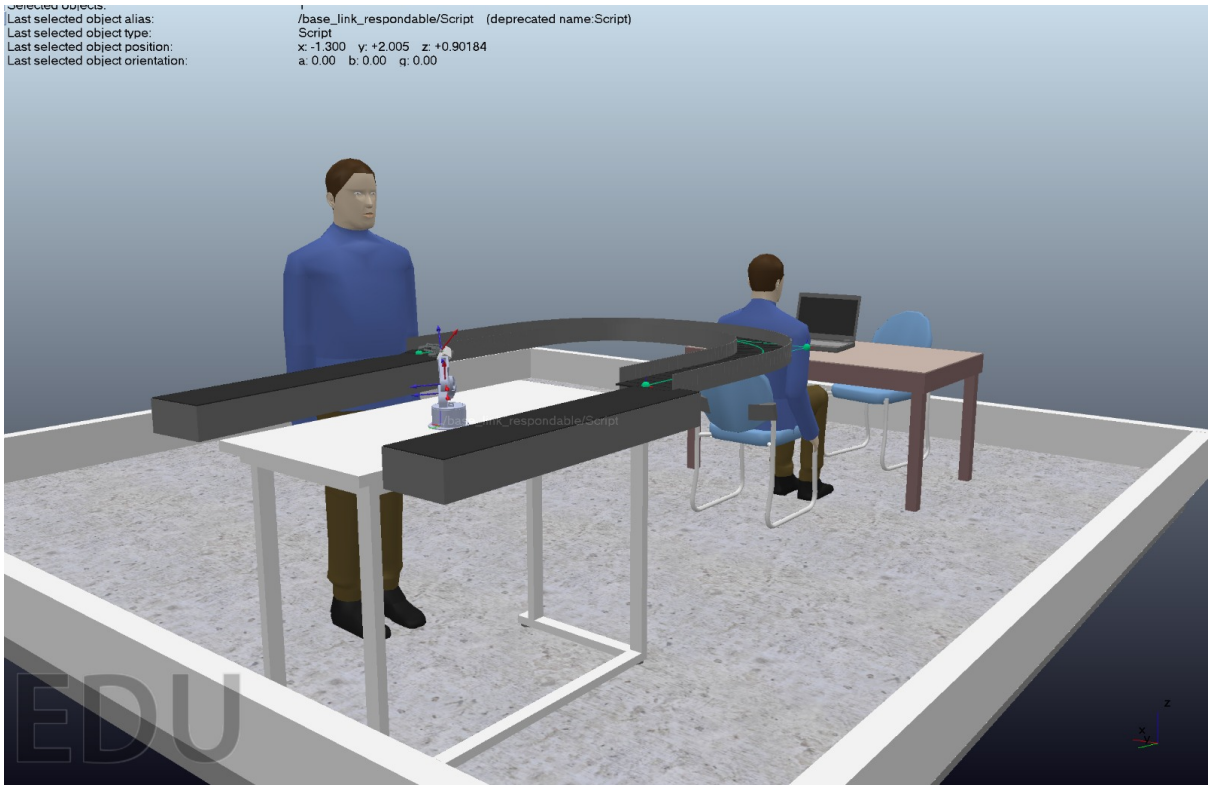



Figure 6.3: Robot Manipulator Simulation - Full Setup

# Chapter 7

# GUI

Figure 7.1 illustrates a user interface for an office environment. One person is seen working at their desk, while another individual oversees the industrial pipeline, ensuring that the robotic arm is operating correctly. The robotic arm is responsible for picking and placing products along the conveyor belt. This setup highlights the integration of human supervision and automated processes, reflecting a modern industrial workflow where technology and human expertise collaborate seamlessly for efficient production.



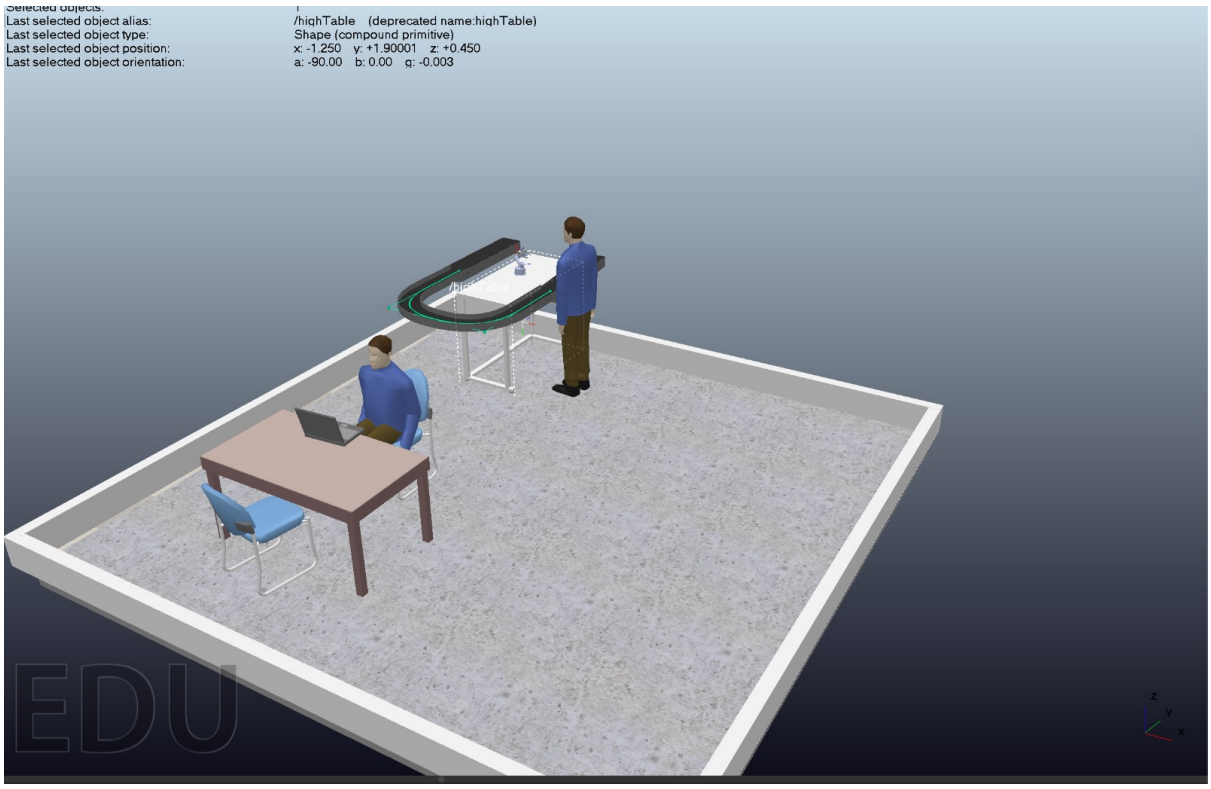




Figure 7.1: GUI

# Chapter 8

# Inverse Position Kinematics Using Newton-Raphson Method

## 8.1 Introduction

Inverse Position Kinematics (IPK) involves finding the joint angles $(q_1, q_2, q_3, q_4)$ required for the end-effector of a robotic arm to reach a specified position in Cartesian space $(X, Y, Z)$. This chapter describes the Newton-Raphson numerical method, which is employed to iteratively solve the nonlinear equations governing the robotic manipulator's motion.

## 8.2 Methodology

The Newton-Raphson method is a numerical approach to solving nonlinear equations iteratively. The joint angles are updated using the general formula:

$$\mathbf{\Theta}_{k+1} = \mathbf{\Theta}_k - J^{-1}(\mathbf{\Theta}_k) \cdot \mathbf{F}(\mathbf{\Theta}_k)$$

Where:

- $\mathbf{\Theta}_k = [q_1, q_2, q_3, q_4]^T$ is the vector of joint angles at iteration $k$.
- $J^{-1}$ is the inverse Jacobian matrix.
- $\mathbf{F}(\mathbf{\Theta})$ is the error vector representing the difference between the current end-effector position and the target position.

## 8.3 Symbolic Calculations

### 8.3.1 Forward Kinematics Equations

The forward kinematics equations for the end-effector position are given as:

$$
\begin{aligned}
X &= L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) + L_3 \cos(q_1 + q_2 + q_3) + L_4 \cos(q_1 + q_2 + q_3 + q_4) \\
Y &= L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) + L_3 \sin(q_1 + q_2 + q_3) + L_4 \sin(q_1 + q_2 + q_3 + q_4) \\
Z &= arctan(Y/X)
\end{aligned}
$$

### 8.3.2 Error Vector

The error vector represents the difference between the current and target end-effector positions:

$$
\mathbf{F}(\mathbf{\Theta}) = \begin{bmatrix} X - \big(L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) + L_3 \cos(q_1 + q_2 + q_3) + L_4 \cos(q_1 + q_2 + q_3 + q_4)\big) \\ Y - \big(L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) + L_3 \sin(q_1 + q_2 + q_3) + L_4 \sin(q_1 + q_2 + q_3 + q_4)\big) \\ Z - \text{Current Z} \end{bmatrix}
$$

### 8.3.3 Jacobian Matrix

The Jacobian matrix relates the joint velocities to the end-effector's velocities. For a robotic arm with $N$ joints, it can be represented as follows:

$$
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_{6\times 1} = \begin{bmatrix} J_{v1} & J_{v2} & \cdots & J_{vN} \\ J_{\omega 1} & J_{\omega 2} & \cdots & J_{\omega N} \end{bmatrix}_{6\times N} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_N \end{bmatrix}_{N\times 1}
$$

where:

- The top $3 \times N$ block, $J_{vi}$, represents the contribution of each joint $i$ to the translational velocity components $(\dot{x}, \dot{y}, \dot{z})$.
- The bottom $3 \times N$ block, $J_{\omega i}$, represents the contribution of each joint $i$ to the rotational velocity components $(\omega_x, \omega_y, \omega_z)$.

#### Symbolic Jacobian Matrix for a 4-DOF Robotic Arm

For our specific 4-DOF robotic arm, the symbolic Jacobian matrix $J$ is given by:

$$J = \begin{bmatrix} -0.14\sin(q_1) - 0.13\sin(q_1+q_2) - 0.12\sin(q_1+q_2+q_3) - 0.1\sin(q_1+q_2+q_3+q_4) & -0.13\sin(q_1+q_2) - 0.12\sin(q_1+q_2+q_3) - 0.1\sin(q_1+q_2+q_3+q_4) & -0.12\sin(q_1+q_2+q_3) - 0.1\sin(q_1+q_2+q_3+q_4) & -0.1\sin(q_1+q_2+q_3+q_4) \\ 0.14\cos(q_1) + 0.13\cos(q_1+q_2) + 0.12\cos(q_1+q_2+q_3) + 0.1\cos(q_1+q_2+q_3+q_4) & 0.13\cos(q_1+q_2) + 0.12\cos(q_1+q_2+q_3) + 0.1\cos(q_1+q_2+q_3+q_4) & 0.12\cos(q_1+q_2+q_3) + 0.1\cos(q_1+q_2+q_3+q_4) & 0.1\cos(q_1+q_2+q_3+q_4) \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

**Error Vector**

The error vector $\mathbf{F}(\mathbf{\Theta})$ represents the difference between the desired end-effector position $(X, Y, Z)$ and the position calculated by the forward kinematics:

$$\mathbf{F}(\mathbf{\Theta}) = \begin{bmatrix} X - (L_1\cos(q_1) + L_2\cos(q_1+q_2) + L_3\cos(q_1+q_2+q_3) + L_4\cos(q_1+q_2+q_3+q_4)) \\ Y - (L_1\sin(q_1) + L_2\sin(q_1+q_2) + L_3\sin(q_1+q_2+q_3) + L_4\sin(q_1+q_2+q_3+q_4)) \\ Z - \text{Current Z} \end{bmatrix}$$

**Partial Derivatives**

The partial derivatives of $X$ with respect to each joint angle $q_i$ (used in the Jacobian matrix) are computed as follows:

$$\frac{\partial X}{\partial q_1} = -L_1\sin(q_1) - L_2\sin(q_1+q_2) - L_3\sin(q_1+q_2+q_3) - L_4\sin(q_1+q_2+q_3+q_4)$$

Similarly, the partial derivatives for $Y$ and $Z$ can be computed symbolically for each $q_i$.

## 8.4 Newton-Raphson Iteration Steps

The Newton-Raphson method is implemented as follows:

1. Start with an initial guess for $\mathbf{\Theta}_0 = [q_1, q_2, q_3, q_4]^T$.
2. Compute the error vector $\mathbf{F}(\mathbf{\Theta}_k)$ and the Jacobian matrix $J(\mathbf{\Theta}_k)$.
3. Update the joint angles using:
$$\mathbf{\Theta}_{k+1} = \mathbf{\Theta}_k - J^{-1} \cdot \mathbf{F}(\mathbf{\Theta}_k)$$
4. Repeat until the error $|\mathbf{F}(\mathbf{\Theta}_k)|$ is within a predefined tolerance.

## 8.5 Simulation Results

In this section, we present the results of the simulations performed in CoppeliaSim. The following screenshots demonstrate different stages and configurations of the robotic arm during testing. We discuss the observed results for each test case and comment on the system's performance and limitations.

### 8.5.1 Test Case 1: Initial Position and Expected Output

- **Q Values:** $[-1.1699, 0.6911, -1.4259, 0.7330]$
- **Expected Position:** $[0.0464, -0.1096, 0.3283]$
- **Calculated Position:** $[0.0801, -0.1890, 0.0447]$
- **Inverse Position Values:** $[-1.1703, 0.6909, -1.4255, 0.7347]$

```
Q Values:
[-1.1699, np.float64(0.6911503837897546), np.float64(-1.4259339988793673), np.float64(0.7330382858376184)]
Expected Position:
[0.04641933090981021, -0.10964196674541116, 0.32832397974498295]
Calculated Position:
(np.float64(0.08014337684855113), np.float64(-0.18908421610631687), np.float64(0.044695300833633655))
Inverse Position Values:
[np.float64(-1.1703054783190918), np.float64(0.6909513179146511), np.float64(-1.425492185408967), np.float64(0.7346791408896668)]
```

Figure 8.1: Q values and expected vs. calculated positions during initial test case

The following screenshots capture the robotic arm in various positions and orientations, illustrating the execution of inverse kinematics based on different input values.

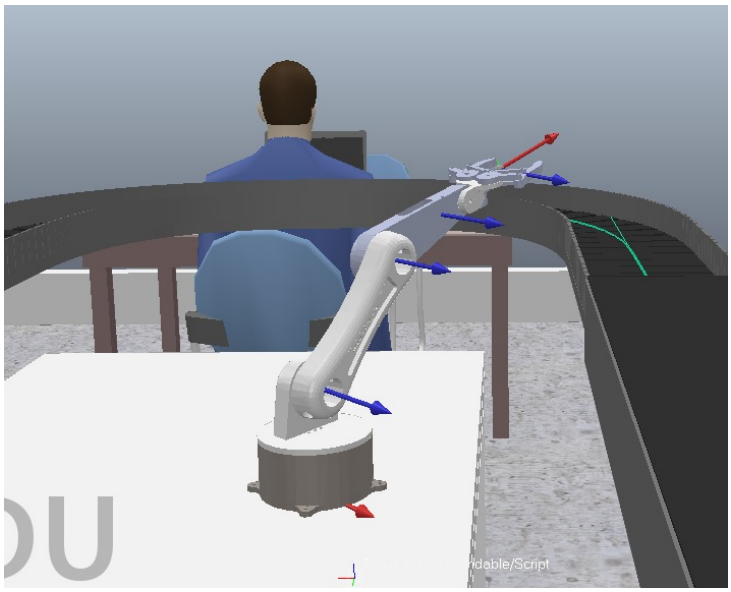


Figure 8.2: Close-up of the robotic arm performing tasks near the conveyor

## 8.6 Comments on System Performance

The robotic arm demonstrated acceptable accuracy in positioning based on the input joint values and target positions. However, there were some deviations between the expected and calculated positions, as observed in Test Case 1. These discrepancies could be attributed to numerical precision limits or slight misconfigurations in the inverse kinematics equations.

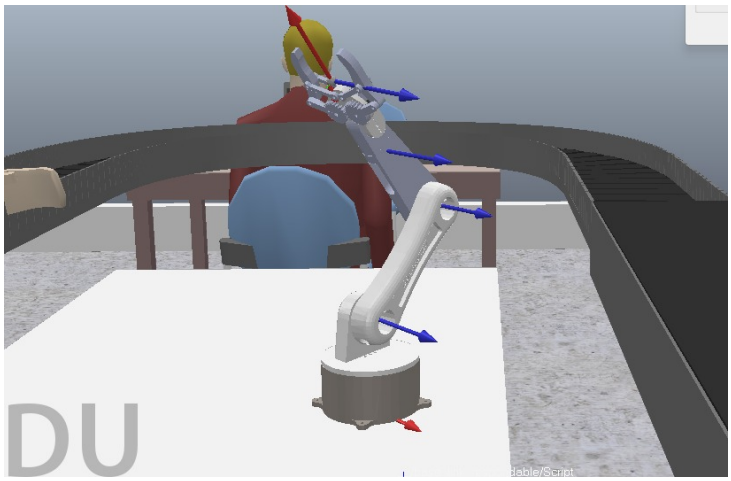


Figure 8.3: Side view of the robotic arm reaching towards an object on the conveyor

## 8.7 Limitations and Constraints of the System

- **Precision Limits:** The system's precision is affected by numerical approximations, particularly in complex inverse kinematics calculations. This may lead to slight inaccuracies in the end-effector position.
- **Workspace Constraints:** The robotic arm has a limited workspace, constrained by joint angles and physical dimensions. Reaching targets outside this range is not feasible.
- **Performance Under Dynamic Conditions:** When testing under dynamically changing inputs, the system exhibited slight delays in updating positions, indicating a potential need for optimization in real-time control.
- **Simulation Frame Rate:** The simulation's frame rate can impact performance, especially when multiple tasks are performed simultaneously. In this case, an average frame rate of approximately 7.94 fps was observed.

## 8.8 Conclusion

The simulation demonstrates the capability of the robotic arm to perform basic tasks in an industrial environment. Despite minor inaccuracies and system constraints, the setup provides a valuable platform for further testing and optimization.

# Chapter 9

# The Forward and Inverse Velocity Kinematics

## Forward Velocity Kinematics

The forward velocity kinematics equation relates the joint velocities, $\dot{q}$, to the linear and angular velocities of the end-effector, represented as:

$$V = J \cdot \dot{q}$$

Here, $J$ is the Jacobian matrix, which is derived from the partial derivatives of the forward kinematics equations.

The forward kinematics equations are:

$$X = L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) + L_3 \cos(q_1 + q_2 + q_3) + L_4 \cos(q_1 + q_2 + q_3 + q_4)$$

$$Y = L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) + L_3 \sin(q_1 + q_2 + q_3) + L_4 \sin(q_1 + q_2 + q_3 + q_4)$$

$$Z = 0 \quad \text{(planar motion assumed)}.$$

The Jacobian matrix is obtained by taking the partial derivatives of these equations with respect to the joint variables $(q_1, q_2, q_3, q_4)$:

$$J = \begin{bmatrix} \frac{\partial X}{\partial q_1} & \frac{\partial X}{\partial q_2} & \frac{\partial X}{\partial q_3} & \frac{\partial X}{\partial q_4} \\ \frac{\partial Y}{\partial q_1} & \frac{\partial Y}{\partial q_2} & \frac{\partial Y}{\partial q_3} & \frac{\partial Y}{\partial q_4} \\ \frac{\partial Z}{\partial q_1} & \frac{\partial Z}{\partial q_2} & \frac{\partial Z}{\partial q_3} & \frac{\partial Z}{\partial q_4} \end{bmatrix}.$$

After substituting the symbolic forward kinematics equations into the derivatives, the Jacobian matrix becomes:

$$J = \begin{bmatrix} -L_1 \sin(q_1) - L_2 \sin(q_1 + q_2) - L_3 \sin(q_1 + q_2 + q_3) - L_4 \sin(q_1 + q_2 + q_3 + q_4) & \dots & \dots & \dots \\ L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) + L_3 \cos(q_1 + q_2 + q_3) + L_4 \cos(q_1 + q_2 + q_3 + q_4) & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Using numerical joint angles $(q_1, q_2, q_3, q_4)$, the Jacobian matrix $J$ is computed, and the end-effector velocity is found as:

$$V = J \cdot \dot{q},$$

where $\dot{q}$ is the vector of joint velocities.

# Inverse Velocity Kinematics

The inverse velocity kinematics involves calculating the joint velocities, $\dot{q}$, from a given end-effector velocity, $V$. This is achieved by solving:

$$\dot{q} = J^{-1} \cdot V,$$

where $J^{-1}$ represents the pseudo-inverse of the Jacobian matrix if $J$ is not square or is singular.

The symbolic joint velocities required to achieve a desired end-effector velocity are given by:

$$\dot{q} = \begin{bmatrix} 0.0218924632498228 \cdot v_x + 3.85002337452884 \cdot v_y \\ -0.0284811298757441 \cdot v_x + 2.70997247477341 \cdot v_y \\ 4.83990574739114 \cdot v_x - 11.2390735431116 \cdot v_y \\ 0.0463109269324299 \cdot v_x + 0.786658966599982 \cdot v_y \end{bmatrix}.$$

This symbolic solution demonstrates the relationship between the end-effector velocity components $v_x$ and $v_y$ and the required joint velocities $\dot{q}$.

# Chapter 10

# Finalized Environment

The finalized simulated environment in CoppeliaSim demonstrates a setup designed for testing and validating robotic kinematics. Below is a detailed description of the components and features present.

## 10.1 Components of the Environment

### 10.1.1 Robotic Arm and Base Frame

- The robotic arm, modeled with multiple links and joints, is positioned on a base frame labeled as `base_link_visual`. This base serves as the primary anchor for the arm's movements and is essential for maintaining stability and alignment during operations.
- Each segment of the arm, such as `Arm1_visual`, `Arm2_visual`, and others, corresponds to specific joints or sections that allow for controlled movement and flexibility, facilitating various tasks like reaching and manipulating objects.

### 10.1.2 End-Effector

- The end-effector is represented by `Grip_respondable` and `Grip_visual`, which together form the gripping mechanism that interacts with the environment. This gripper is designed to pick up or manipulate objects within its reachable workspace, playing a crucial role in testing kinematic and path-planning algorithms.

### 10.1.3 Conveyor System

- A conveyor belt system (`efficientConveyor0` and `efficientConveyor1`) is present in the environment, enabling dynamic interactions with moving objects. This setup allows the robot to pick up items from the conveyor, simulating a realistic assembly line scenario and providing an opportunity to test response times and adaptability.

### 10.1.4 Human Models

- Human models are placed around the robotic system, providing context to the environment as an industrial or collaborative workspace. This feature is helpful in validating the robot's spatial awareness and safety mechanisms when operating near people.

### 10.1.5 Workspace and Boundaries

- The workspace is defined within a confined area, delineated by walls and furniture items such as tables (`customizableTable0`). This boundary serves to test the robot's range of motion and ensures that the arm operates within safe, pre-defined limits, minimizing the risk of collision with surrounding structures.

### 10.1.6 Sensors and Control Interface

- A `Vision_sensor` is included, possibly for monitoring object positions or detecting obstacles. Such sensors enable closed-loop control, where feedback from the environment can dynamically adjust the robot's actions based on real-time data.

### 10.1.7 Software and Simulation Control

- The CoppeliaSim interface displays the hierarchical structure of components, facilitating detailed control over each part of the system. The Lua scripting section provides a platform for implementing custom control algorithms, enhancing the robot's adaptability to different tasks and inputs.

## 10.2 Environment Features

This setup effectively replicates an industrial automation scenario, providing a controlled environment to:

- Test inverse and velocity kinematics algorithms.
- Validate the robot's ability to handle various target positions on the conveyor.
- Assess system constraints, such as joint limits and workspace boundaries.

The finalized environment showcases a comprehensive simulation platform, ideal for developing and refining robotic functions in an industrial or collaborative setting.

The finalized simulated environment in CoppeliaSim, as demonstrated in the images below, showcases a setup for testing and validating robotic kinematics.

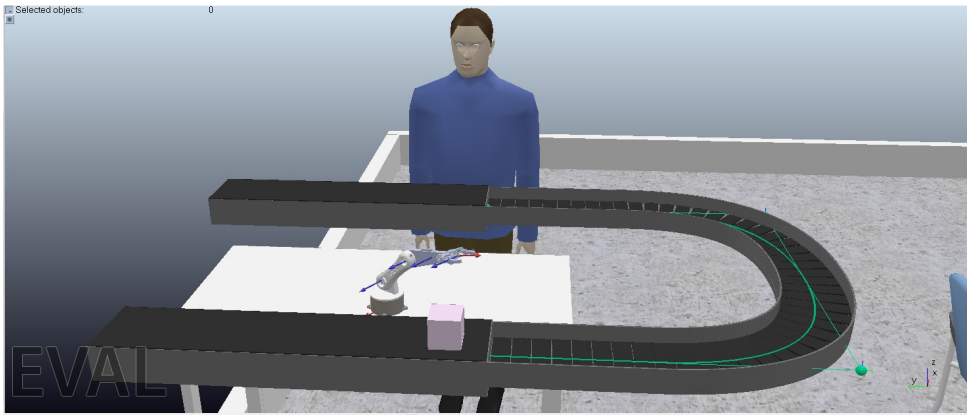



Figure 10.1: Close-up of the robotic arm on the conveyor with human models in the environment

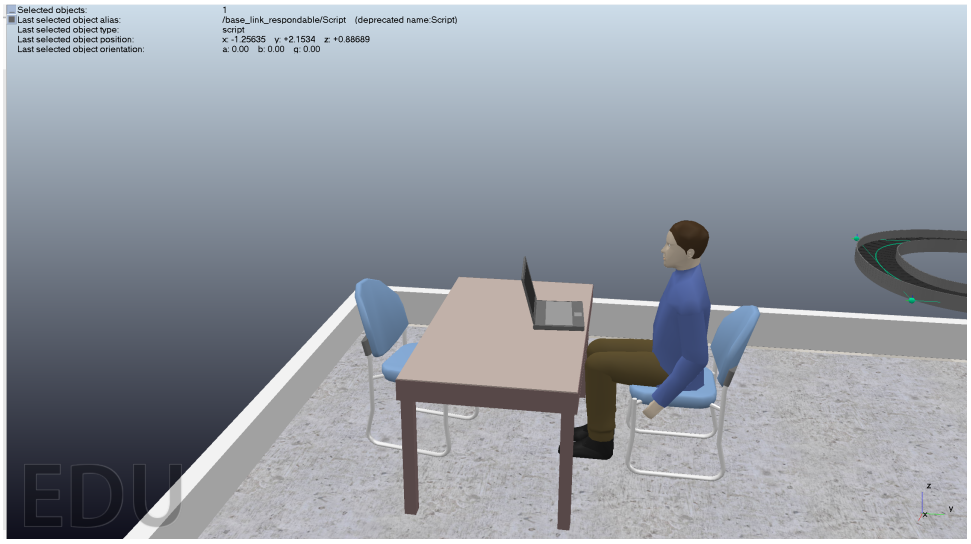



Figure 10.2: Side view of the simulated environment showing a human model working at a desk

## 10.3 Description of the Environment

### 10.3.1 Robotic Arm and Conveyor System

The environment features a robotic arm positioned on a table with a conveyor system. The conveyor allows for dynamic interaction with objects, simulating an assembly line scenario where the robot can pick or manipulate items.

### 10.3.2 Human Models and Workspace Layout

Human models are placed in the environment, providing context to simulate a collaborative industrial workspace. One model is positioned near the robotic arm, while another

is seated at a desk, replicating a typical work environment.

# Chapter 11

# Trajectory

## 11.1 Trajectory Types

### 11.1.1 Linear Trajectories

Linear trajectories are generated to provide a direct path between two points in space. The function `task_traj` computes the slope for each coordinate ($X$, $Y$, $Z$) based on the difference between the initial and final positions, and interpolates positions at discrete time steps. This approach is efficient for straightforward tasks like pick-and-place operations, where the end-effector moves in a straight line.

The linear slope for each coordinate is defined as:

$$\text{Slope} = \frac{X_f - X_0}{T_f},$$

where:

- $X_0, X_f$: Initial and final positions of the coordinate.
- $T_f$: Total time for the trajectory.

The main characteristics of linear trajectories are:

- **Simplicity:** The motion is direct and computationally lightweight.
- **Predictability:** The trajectory follows a straight line, making it easy to control and plan.
- **Limitations:** Linear paths may not be suitable for applications requiring smooth or continuous curves.

### 11.1.2 Elliptical Trajectories

Elliptical trajectories are used for generating smooth and continuous motion paths in the $X$-$Y$ plane, with optional linear progression along the $Z$-axis. The function `task_traj_ellipse` computes the parameters of the ellipse, such as the semi-major ($a$) and semi-minor ($b$) axes, and interpolates positions based on the angular velocity.

The elliptical trajectory is defined by the following equations:

$$x(t) = a \cdot \cos(\theta(t)) + X_0,$$

$$y(t) = b \cdot \sin(\theta(t)) + Y_0,$$

$$z(t) = Z_0 + \text{Slope}_z \cdot t.$$

Here:

- $a, b$: Semi-major and semi-minor axes of the ellipse.
- $X_0, Y_0, Z_0$: Center offset of the trajectory.
- $\theta(t)$: Angular parameter varying with time.
- $\text{Slope}_z$: Slope controlling the $z$-axis progression.

The main characteristics of elliptical trajectories are:

- **Smooth Motion:** The motion is continuous and ideal for tasks requiring non-linear paths, such as inspection or arc welding.
- **Flexibility:** The trajectory can be adjusted for different radii and angular velocities.
- **Complexity:** Elliptical paths require more computation compared to linear trajectories.

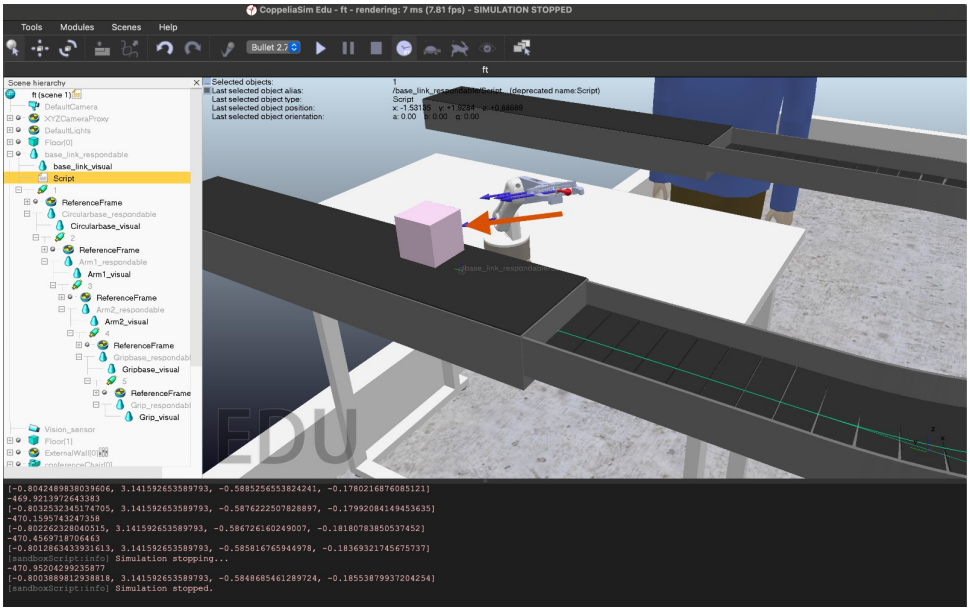



Figure 11.1: The desired robot's motion.

# Bibliography


[1] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431, 2021.

[2] Andrius Dzedzickis, Jurga Subačiūtė-Žemaitienė, Ernestas Šutinys, Urtė Samukaitė-Bubnienė, and Vytautas Bučinskas. Advanced applications of industrial robotics: New trends and possibilities. *Applied Sciences*, 12(1):135, 2021.

[3] Bohan Feng, Xinzhe Juan, Xinyi Gao, Qi Zhou, and Youyi Bi. A robotic manipulation framework for industrial human–robot collaboration based on continual knowledge graph embedding. *The International Journal of Advanced Manufacturing Technology*, pages 1–17, 2024.

[4] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, and Rajiv Suman. Substantial capabilities of robotics in enhancing industry 4.0 implementation. *Cognitive Robotics*, 1:58–75, 2021.

[5] Behnam M. Tehrani, Samer BuHamdan, and Aladdin Alwisy. Robotics in assembly-based industrialized construction: A narrative review and a look forward. *International Journal of Intelligent Robotics and Applications*, 7(3):556–574, 2023.

[6] Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey. *The International Journal of Robotics Research*, 37(7):688–716, 2018.

[7] Fahad Sherwani, Muhammad Mujtaba Asad, and Babul Salam Kader K Ibrahim. Collaborative robots and industrial revolution 4.0 (ir 4.0). In *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, pages 1–5. IEEE, 2020.

[8] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018.

[9] Fan Wang and Kris Hauser. Robot packing with known items and nondeterministic arrival order. *IEEE Transactions on Automation Science and Engineering*, 18(4):1901–1915, 2020.

[10] Wenhua Yuan and Weixiao Lu. Research on the impact of industrial robot application on the status of countries in manufacturing global value chains. *Plos one*, 18(6):e0286842, 2023.