Desktop Based Industrial Robot for Pick and Place Application

Mohamed Sabry*, Mohammed Saeed*, Shaheer Sherif* , Mariam Fathi*, Lina Ghonim*, Youssef Mahran*
*German University in Cairo (GUC), Egypt
Emails: mohamed.amer@student.guc.edu.eg, mohammed.saeed@student.guc.edu.eg, shaheer.aziz@student.guc.edu.eg, mariam.fathi@student.guc.edu.eg, lina.ghonim@student.guc.edu.eg, youssef.mahran@student.guc.edu.eg

***Abstract*—This paper presents the development of a desktop-based industrial robot tailored for pick-and-place applications. The Denavit-Hartenberg (DH) convention and Newton-Raphson method are utilized for forward and inverse kinematics, respectively, ensuring precise end-effector positioning. The integration of simulation tools like MATLAB Simscape and CoppeliaSim facilitated digital twin validation, enabling trajectory optimization in both task and joint space. Hardware implementation involved the fabrication of modular components using 3D printing and the deployment of pre-defined trajectory arrays for reliable operation. Validation experiments demonstrated the alignment of simulated results with real-world performance, highlighting the robot's potential for flexible industrial automation.**

I. Introduction

The use of serial robotic manipulators in pick-and-place industrial applications has been extensively studied, given their precision, adaptability, and efficiency in repetitive tasks. Serial manipulators, characterized by a chain of rigid links connected by rotary or prismatic joints, excel in applications requiring high dexterity and workspace flexibility, such as in industrial setups.

Custudio et al. (2020) [1] demonstrated the implementation of a six-axis manipulator for sorting and packaging, highlighting the system's ability to dynamically adjust to varied object sizes and weights using computer vision integration. Similarly, Ginnante et al. (2019) [2]investigated trajectory optimization techniques for reducing cycle times, emphasizing the role of inverse kinematics and dynamic control algorithms in improving throughput. Advances in sensor integration, such as the use of vision-guided systems and force-torque sensors, have further enhanced the capabilities of these manipulators in handling delicate objects with minimal damage (Ahmed and Zhou, 2021 [3]).

Moreover, the adoption of machine learning has opened new avenues for improving pick-and-place efficiency. Studies by Andrea and Tosello (2021) [4] highlighted the use of reinforcement learning to train robotic arms for adaptive placement strategies, leading to significant improvements in unstructured environments. However, challenges remain in achieving real-time responsiveness and energy efficiency, particularly for large-scale operations (Chea et al., 2020 [5]). Addressing these limitations requires advancements in lightweight materials, actuators, and decentralized control systems. These studies collectively underline the pivotal role of serial robotic manipulators in modern industrial automation, driven by continuous innovations in hardware and intelligent control systems.

II. Methodology

A. Forward Position Kinematics

To determine the forward position kinematics of the robotic arm, the Denavit-Hartenberg (DH) convention was employed as a systematic and standardized framework. This methodology enabled the precise mathematical modeling of the robot's kinematic chain, establishing the relationship between the joint parameters and the position and orientation of the end-effector in the Cartesian space. The first step in using the DH convention involved assigning coordinate frames to each link of the robotic arm. A frame was attached to every joint such that:

1) The **Z-axis** of each frame represented the axis of rotation for revolute joints or the direction of translation for prismatic joints.
2) The **X-axis** aligned with the common normal between two adjacent Z-axes or was perpendicular to the Z-axis if they were parallel.

This consistent frame assignment ensured that all relative transformations between links could be expressed in a unified format. For each joint i, the following four DH parameters were identified:

1) **Link length** (a_i): The distance between the Z_i and Z_{i+1} axes along the X_i axis.
2) **Link twist** (α_i): The angle of rotation about the X_i axis that aligns Z_i with Z_{i+1}.
3) **Joint offset** (d_i): The distance between the origins of the frames along the Z_i axis.
4) **Joint angle** (θ_i): The angle of rotation about the Z_i axis that aligns the X_{i-1} and X_i axes.

These parameters were organized into a tabular format, serving as the foundation for constructing the transformation matrices. Using the identified DH parameters, the transformation matrix A_i for each joint was constructed as follows:

$$T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Each matrix encapsulated the relative transformation (rotation and translation) from one link to the next. To compute the forward kinematics, the individual transformation matrices were sequentially multiplied:

$$T = T_1 \cdot T_2 \cdot T_3 \cdot \ldots \cdot T_n$$

Here, T represents the final homogeneous transformation matrix that relates the base frame to the end-effector frame. This matrix contains:

- The rotational component in the top-left 3×3 submatrix.
- The translational component (position) in the top-right 3×1 vector.

The resulting transformation matrix T was used to extract the end-effector's position (x, y, z) and orientation (roll, pitch, yaw or equivalent representations). This information is crucial for tasks such as path planning, motion control, and interaction with the environment. The DH convention streamlined the process of obtaining forward kinematics by reducing the complexity of modeling a multi-link robotic system. Its systematic nature ensured consistency in frame assignment and parameter identification, minimizing the potential for errors in computation. Additionally, the homogeneous transformation matrices facilitated easy integration with downstream algorithms, such as inverse kinematics and dynamic modeling.

This methodology provided the foundation for accurate and efficient kinematic analysis, enabling the robotic arm to achieve its desired tasks in a reliable manner.

B. Inverse Position Kinematics

Inverse position kinematics (IK) of a serial robotic manipulator is necessary to calculate the joint angles required to place the end-effector at a desired position and orientation in the workspace. This is a critical step in robot control, as it determines the configuration of the manipulator's joints for a given task. One numerical approach to solving the inverse kinematics problem is the Newton-Raphson method, which is an iterative technique used to find successively better approximations of the roots of a system of nonlinear equations. In the context of serial manipulators, the inverse kinematics equations are typically nonlinear and involve the manipulator's geometric and kinematic parameters. The goal of inverse kinematics is to find the joint angles $\theta_1, \theta_2, \ldots, \theta_n$ that achieve a desired end-effector position. This can be solved iteratively using the Newton-Raphson method.

Let the forward kinematics function $\mathbf{f}(\boldsymbol{\theta})$ obtained earlier represent the position of the end-effector as a function of the joint angles:

$$\mathbf{f}(\boldsymbol{\theta}) = \begin{bmatrix} x(\boldsymbol{\theta}) \\ y(\boldsymbol{\theta}) \\ z(\boldsymbol{\theta}) \end{bmatrix}$$

The goal is to find the joint angles $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_n]^T$ such that:

$$\mathbf{f}(\boldsymbol{\theta}) = \mathbf{p}_d$$

where $\mathbf{p}_d$ is the desired position of the end-effector.

Using the Newton-Raphson method, we update the joint angles iteratively:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - J^{-1}(\boldsymbol{\theta}_k) \cdot \mathbf{e}(\boldsymbol{\theta}_k)$$

where:

$$\mathbf{e}(\boldsymbol{\theta}_k) = \mathbf{f}(\boldsymbol{\theta}_k) - \mathbf{p}_d$$

is the error vector, and $J(\boldsymbol{\theta}_k)$ is the Jacobian matrix of the system, which relates the change in joint angles to the change in the end-effector position:

$$J(\boldsymbol{\theta}_k) = \frac{\partial \mathbf{f}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

The Jacobian matrix can be computed as:

$$J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \cdots & \frac{\partial x}{\partial \theta_n} \\ \frac{\partial y}{\partial \theta_1} & \cdots & \frac{\partial y}{\partial \theta_n} \\ \frac{\partial z}{\partial \theta_1} & \cdots & \frac{\partial z}{\partial \theta_n} \end{bmatrix}$$

The process continues iteratively until the error $\mathbf{e}(\boldsymbol{\theta})$ is sufficiently small, i.e., when:

$$\|\mathbf{e}(\boldsymbol{\theta})\| < \epsilon$$

where ϵ is a small threshold indicating convergence.

C. Digital Twin

A robot CAD model was exported as an URDF to be imported into MATLAB Simscape simulation and Coppeliasim python based simulator. A digital twin is crucial in deveoping a desktop based industrial robot in order to validate the codes in a safe environment with zero risks.

D. Trajectory Generation

Trajectory generation is a critical process in the motion planning of serial robotic manipulators, focusing on determining feasible paths or joint configurations to execute tasks such as object manipulation or trajectory tracking. It involves adhering to the manipulator's physical constraints—such as joint limits, velocities, and accelerations—as well as environmental constraints, including obstacles and task-specific requirements. Two simulation softwares were used which are Simscape and CoppeliaSim . In Simscape, two Simulink subsystems were designed. The first is a task space trajectory algorithm which takes the initial and final position of the robot, the time duration, and initial guess for the joint angles as inputs. The output of this block is the required joint angles which are calculated using an inverse position kinematics algorithm. The second Simulink subsystem implements the joint space trajectory equations which are calculated externally and simply inputted into the Simulink subsystem. The initial and final positions of the robot for both the task-space and joint-space trajectories. In coppeliasim, two python codes were designed. The first is a task space trajectory code which takes an array of joint angles generated from matlab script and sends it to the robot.

E. Hardware Implementation

From the simulations implemented, we obtained an array of joint angles that represent the desired positions for the robotic system to execute the planned motion. This array serves as a predefined set of joint configurations to achieve the intended task. These angles were statically defined in the Arduino code to ensure precise and repeatable execution of the movements. By embedding the array directly into the Arduino, we simplified the implementation and reduced the dependency on real-time calculations. The algorithm used to process and execute the joint angles is presented below in pseudocode, highlighting the sequence of operations performed by the Arduino to control the robotic system effectively.

Algorithm 1 Pseudocode for trajectory tracking algorithm

Initialize the struct object for the defined trajectory
while Trajectory array length not reached **do**
Move Base Servo with the desired angle at the corresponding index in the trajectory array.
Move Link 1 Servo with the desired angle at the corresponding index in the trajectory array.
Move Link 2 Servo with the desired angle at the corresponding index in the trajectory array.
Move Gripper pitvh Servo with the desired angle at the corresponding index in the trajectory array.
end while
Set Gripper to Open or Close based on whether it is a pick or place trajectory.
=0

III. Results

A. Hardware Assembly

The fabrication of the robot involved a detailed and precise process, starting with the 3D printing of the robot's components. 3D printing was chosen due to its flexibility in creating complex geometries with high precision, allowing for customized parts that fit the specific design of the robot. Various materials, including PLA and ABS, were used based on their strength, durability, and weight considerations. For example, the structural components that needed to bear more load were printed in ABS, while lighter, non-load-bearing parts were made from PLA. The 3D printed components were designed to be modular, facilitating easier assembly and maintenance.

Once the parts were printed, the focus shifted to the installation of the motors, which are essential for driving the robot's movement. The motors were selected based on the torque requirements and their compatibility with the robot's design. These motors were mounted into pre-designed motor housings, ensuring that they were securely fixed and aligned correctly. Special attention was given to wiring and ensuring the motors were connected to the power supply and control system, allowing for smooth operation.

After the motors were installed, the robot's links—rigid connections between different parts of the robot—were assembled. The links were carefully attached to the motor shafts, ensuring that each joint allowed for smooth movement while maintaining the robot's stability. The assembly of the links required precise alignment, as even minor misalignments could affect the robot's functionality and performance. Each link was secured with screws or bolts, and the joints were designed to minimize friction and maximize mobility.

The final steps involved testing and calibrating the assembled robot to ensure everything was functioning as expected. This included adjusting the motors' response to input signals, fine-tuning the movement of the links, and ensuring the overall balance and functionality of the robot. The entire fabrication and assembly process was iterative, with adjustments made to improve performance and ensure the robot met the desired specifications.

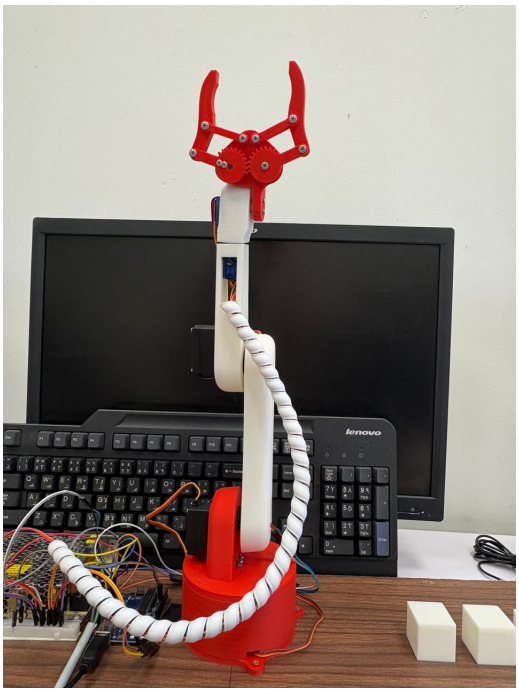


Fig. 1: Front view of the fabricated Robotic Arm

The circuit for the robot was designed using a simulator to provide a clear and easily visualized representation of the electrical components and their connections. This approach allowed us to organize and plan the wiring between the motors, sensors, microcontroller, and power supply efficiently. By designing the circuit in a virtual environment, we ensured that each component was correctly placed and interconnected, offering a straightforward overview of how the system would function once assembled. The figure of the circuit below illustrates the complete design, providing a clear schematic of the components and their connections for better understanding and further development.

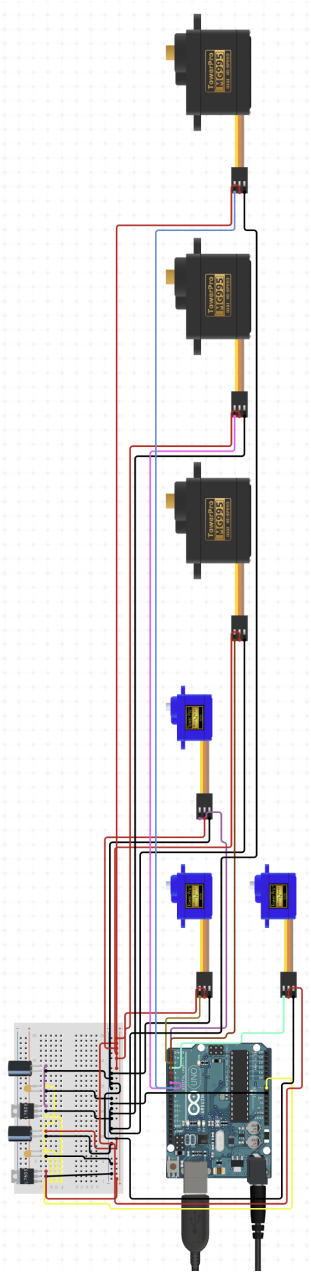


Fig. 2: Motor and Arduino Circuitry

B. Forward Position Kinematics

Joint (i)	$\theta(i)$	$d(i)$	$a(i)$	$\alpha(i)$
1	q_1	ℓ_1	0	$\frac{\pi}{2}$
2	q_2	0	ℓ_2	0
3	q_3	0	ℓ_3	0
4	q_4	0	ℓ_4	0

1) Simscape Validation: Two cases are used to simulate the robot and ensure correctness between the DH convention equations and the simscape end effector position. The first case is the equilibrium position where the robot is standing vertically upward and the joint angles are all equal to 0. The values obtained from the MATLAB code and Simscape simulations are $[0, 0.2182, 317.5499]^T$ and $[5.203, 0, 317.4]^T$ respectively. It is apparent that they are approximately equal within a small tolerance due to dimensions not taken into account in the DH convention such as the small depth values between the links. Finally, the robot's position in 3D is readily seen in Figure 3.

Fig. 3: Case 1 Robot Position

$$q_1 = 0.5\ rad \tag{1}$$
$$q_2 = 0.4\ rad \tag{2}$$
$$q_3 = 0.5\ rad \tag{3}$$
$$q_4 = 0\ rad \tag{4}$$

The values obtained from the MATLAB code and Simscape simulations are $[-140.0417, -76.3126, 255.7942]^T$ and $[-135.1, -73.8, 256.7]^T$ which are also quite similar considering the tolerance. Since the joint q_1 is actuated, the robot has a position on the Y-axis as if q_1 was not actuated, then the joints q_2 and q_3 would simply move the robot in the Z-X plane. Note that q_1 is a rotation around a vertical axis and q_2 and q_3 are rotations around axes out of the page. Finally, q_4 is the rotation of the gripper also around an axis out of the page. It is apparent that they are approximately equal within a small tolerance due to dimensions not taken into account in the DH convention such as the small depth values between the links. Finally, the robot's position in 3D is readily seen in Figure 4.

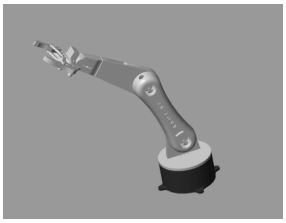


Fig. 4: Case 2 Robot Position

2) Coppeliasim Validation: Forward position kinematics were coded in python language and simulated on Coppeliasim. The results of two cases on input angles where the robot position from Coppeliasim is compared to that of the DH convention are shown in this section.

The first case is the equilibrium position as shown in Fig. 5 where the robot is standing vertically upward and the joint

angles are all equal to 0. The values obtained from the python code where found to be [0, 0, 0.25] while the values from Coppeliasim where [0, 0, 0.3]. It is apparent that they are approximately equal within a small tolerance due to dimensions not taken into account in the DH convention such as the small depth values between the links.

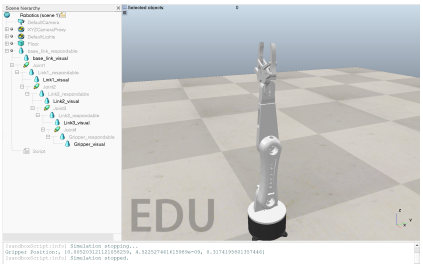


Fig. 5: Case 1 Coppeliasim Results

The second case is a more general case as shown in Fig. 6 where:

$$q_1 = 0.5\ rad \tag{5}$$
$$q_2 = 0.4\ rad \tag{6}$$
$$q_3 = 0.5\ rad \tag{7}$$
$$q_4 = 0\ rad \tag{8}$$

The python code result was [-0.14, -0.07, 0.25] and the coppeliasim was [-0.15, 0, 0.25]. Again small tolerances appear in the results due to inconsistencies in the DH convention.

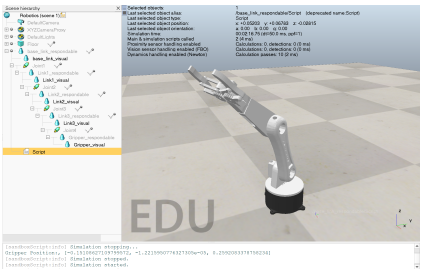


Fig. 6: Case 2 Coppeliasim Results

C. Inverse Position Kinematics

1) Simscape Validation: Two cases are utilized to validate the employed inverse position kinematics algorithm based on the Newton-Raphson method. First, a certain desired position which consists of three components: (X_d, Y_d, Z_d) is chosen. Second, the inverse position kinematics function is run to obtain the required joint angles which achieve this desired position. With the joint angles in hand, they are given as input to both the forward position kinematics function and the Simscape joints to validate that they yield the same position as the aforementioned desired position (X_d, Y_d, Z_d).

The first case consists of a position near the equilibrium position of the robot where the desired position is defined as $(-0.1266, -0.0535, 0.2711)$. The required joint angles to achieve this position were determined using inverse position kinematics MATLAB function which is called: "inverse _position_nr". These joint angles are $[0.4, 0.3, 0.5, 1]^T$. After obtaining these values, they are inputted into the forward position kinematics function to ensure its output is equal to the desired position which is called "forward_kinematics". Finally, the required joint angles obtained were inputted into the Simscape model of the robot and the final robot position can be observed in Figure 7 respectively. The final robot position was $[-0.1215, -0.05136, 0.2711]^T$ which is sufficiently close to that of the desired position and hence, this further validates the inverse-position kinematics algorithm.

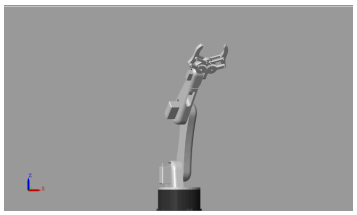


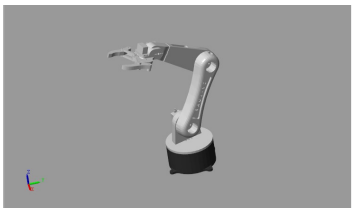
Fig. 7: Case 1 Robot Position

The second position is more consistent with that of a pick and place application. The desired coordinates are $[-0.125, -0.125, 0.08]^T$. Using the aforementioned MATLAB functions, the required joint angles were calculated to be $[3.927, 5.7406, 4.57805, 1.000]^T$. When these joint angles were inputted to the simscape model, the resultant position was $[-0.1274, -0.1245, 0.085]^T$ which is sufficiently close to the desired position further validating our inverse position kinematics algorithm. The resultant robot position can be seen in Figure 8.

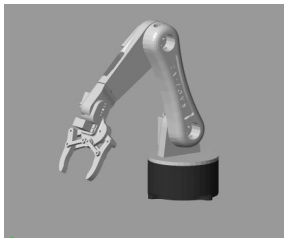
Fig. 8: Case 2 Robot Position

D. Trajectory Tracking Results

1) Simscape Validation: Figure 9a and Figure 9b show the initial and final positions of the robot when simulating one linear task-space and one joint space trajectory with the same initial and final points. The methodology regarding these simulations is discussed in detail in the Methodology section.



(a) Initial Robot Position in Simscape



(b) Final Robot Position in Simscape

Fig. 9: Robot Positions in Task and Joint Space Trajectories in Coppeliasim

Regarding the full application trajectories, a MATLAB function was created which outputs the polynomial coefficients of each joint given initial and final joint positions and velocities. The X-O application consists of three pick and place operations each of which consists of three separate trajectories and hence a total of 9 trajectories are simulated one after the other. The three trajectories for a certain pick and place operation consist of the following:

1) Moving from the home position to the pick position
2) Moving from the pick position to the place position
3) Moving from the place position back to the home position

Of course, this process is repeated 3 times, each with different pick and place positions however, the home position remains the same. The equations of the trajectories and the corresponding simulation video can be seen in Team 7's github. An example of the joint equations for the initial trajectory from the home position to the first pick position is shown below.

Consider the 3rd order polynomial equation to represent the joint space trajectories for each of the joints.

$$q(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3 \tag{9}$$

The initial and final joint angles as well as the initial and final joint velocities were inputted into a MATLAB function to get the set of coefficients of each joint each of which are 4x1 vectors since there are four joint angles. From this function it was found that:

$$C_0 = [0.2110, 0.4222, 0.7983, -0.2849]^T \tag{10}$$
$$C_1 = [0, 0, 0, 0]^T \tag{11}$$
$$C_2 = [0.1632, -0.0747, 0.1952, 0.1135]^T \tag{12}$$
$$C_3 = [-0.0218, 0.0096, -0.0260, -0.0151]^T \tag{13}$$

This process was repeated for the 9 trajectories corresponding to the 3 pick and place operations and the end-effector trajectory of the robot is as shown in Figure 10. Each triangular plane represents 1 pick and place operation where the robot moves from the home position to the pick position to the place position. The specific positions were extracted from the hardware setup.

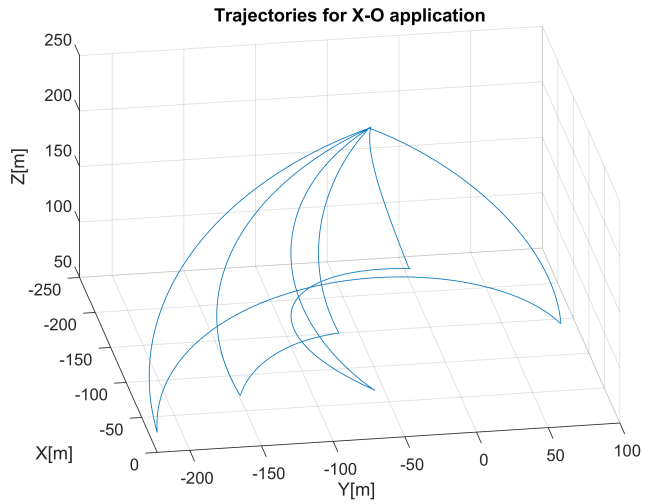


Fig. 10: Full X-O application trajectories

2) Coppeliasim Validation: The output of this block is the required joint angles which are calculated using an inverse position kinematics algorithm. The second is a joint space trajectory code where it computes the joint values online using the joint space equations and sends it to the robot. The trajectories were simulated in Coppelia, and the video results can be seen in Team 7's github. The initial position of the robot can be seen in Figure 11a and the final position of the robot can be observed in Figure 11b.

(a) Initial Robot Position

(b) Final Robot Position

Fig. 11: Robot Positions in Task and Joint Space Trajectories in Coppeliasim

As shown Coppelia sim achieved consistent results with simscape.

IV. Conclusion

This paper successfully developed and validated a desktop-based industrial robot for pick-and-place tasks, combining advanced simulation techniques and efficient hardware implementation. The integration of forward and inverse kinematics ensured precise positioning, while trajectory generation optimized the robot's movements. Validation using MATLAB Simscape and CoppeliaSim demonstrated strong closeness between simulated and experimental results, proving the effectiveness

of the system. This work highlights the potential of accessible robotic systems in industrial applications and sets the stage for further enhancements, including real-time adaptability and extended task automation capabilities.

REFERENCES

[1] L. Custodio and R. Machado, "Flexible automated warehouse: a literature review and an innovative framework," *The International Journal of Advanced Manufacturing Technology*, vol. 106, pp. 533–558, 2020.

[2] A. Ginnante, "Design, analysis and kinematic control of highly redundant serial robotic arms," 2024.

[3] A. A. Hayat, S. Chaudhary, R. A. Boby, A. D. Udai, S. D. Roy, S. K. Saha, and S. Chaudhury, *Vision based identification and force control of industrial robots*, vol. 404. Springer, 2022.

[4] A. Franceschetti, E. Tosello, N. Castaman, and S. Ghidoni, "Robotic arm control and task training through deep reinforcement learning," in *International Conference on Intelligent Autonomous Systems*, pp. 532–550, Springer, 2021.

[5] C. P. Chea, Y. Bai, X. Pan, M. Arashpour, and Y. Xie, "An integrated review of automation and robotic technologies for structural prefabrication and construction," *Transportation Safety and Environment*, vol. 2, no. 2, pp. 81–96, 2020.