Design and Simulation of a 4-DoF Robotic Manipulator in MATLAB/Simulink for Pick-and-Place Operations

Ahmed Kamel
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI, USA
Email: kamelah1@msu.edu

***Abstract*—Pick-and-place is a canonical benchmark task in industrial automation because it requires accurate end-effector positioning, smooth motion generation, and reliable grasp sequencing. This paper presents a MATLAB/Simulink simulation of a 4-degrees-of-freedom (DoF) manipulator executing waypoint-based pick-and-place. The approach combines (i) task-space trajectory generation using cubic interpolation, (ii) waypoint tracking and gripper sequencing implemented as state machines, (iii) kinematic mapping using forward kinematics (FK) and numerical inverse kinematics (IK), and (iv) joint-level feedback control using PID compensation. Simulation results demonstrate stable convergence to waypoints and repeatable grasp/release execution under a configurable tolerance.**

***Index Terms*—Robotics, Manipulator, Inverse Kinematics, Trajectory Planning, Cubic Spline, PID Control, MATLAB Simulink, Pick-and-Place.**

I. Introduction

Robotic manipulators are widely used in manufacturing and logistics because they can repeatedly execute spatially precise motions. Even for a basic pick-and-place task, successful execution requires coordinated solutions for kinematics, smooth trajectory generation, feedback control, and sequencing logic for grasping and releasing. This work implements a complete simulation pipeline in MATLAB/Simulink for a 4-DoF manipulator using modular subsystems for (1) trajectory/waypoint generation, (2) waypoint tracking and gripper logic, (3) kinematics, and (4) motion actuation and visualization [1].

II. System Architecture

Fig. 1 shows the top-level model. An *Input Trajectory* block generates a pre-computed Cartesian trajectory (75 points in the configuration used here) and a set of discrete waypoints. A *Waypoint Tracking Logic* subsystem compares the current end-effector pose against the current target, advances when the error is within tolerance, and issues (i) a target position reference and (ii) a gripper command. A *Waypoint Tracking Environment* subsystem consumes these commands, computes IK, actuates the robot in simulation, and outputs the measured end-effector position and gripper force for feedback.

Course project for ECE 818 (Robotics), Spring 2024.

A. Trajectory and Waypoints

The *Input Trajectory* block initializes trajectory points and waypoints in the MATLAB workspace. The project configuration uses a *Complex* scenario with a *Cubic* trajectory type and $N = 75$ trajectory samples (Fig. 2). Waypoints are also visualized in 3D as spheres, while the reference trajectory is rendered as a spline curve (Fig. 3).

B. Waypoint Tracking and Gripper Sequencing

Waypoint progression and gripper actions are implemented using Stateflow logic (Fig. 4). Let $\boldsymbol{p}(t) \in \mathbb{R}^3$ be the measured end-effector position and $\boldsymbol{p}_k$ be the current target (waypoint or trajectory sample). The position difference used by the state machine is

$$d_k(t) = \|\boldsymbol{p}(t) - \boldsymbol{p}_k\|_2 . \tag{1}$$

When $d_k(t) \le \epsilon$ (the waypoint threshold), the logic advances to the next index. Gripper logic sets `gripCmd` to open/close and uses a force/torque-related feedback signal (`gripForce`) to detect a successful grasp and to flag faults (e.g., timeout with no grasp detected) [1].

III. Kinematic Modeling

A. Forward Kinematics

Forward kinematics maps joint coordinates to the end-effector pose. Using the Denavit–Hartenberg (DH) convention, each joint/link pair is represented by a homogeneous transform

$$^{i-1}A_i(\theta_i, d_i, a_i, \alpha_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

The base-to-end-effector transform is

$$^{0}T_e = \prod_{i=1}^{n} {}^{i-1}A_i, \qquad n = 4. \tag{3}$$

In Simulink/Simscape, FK is also obtained via a transform query from the end-effector frame to the world frame, followed

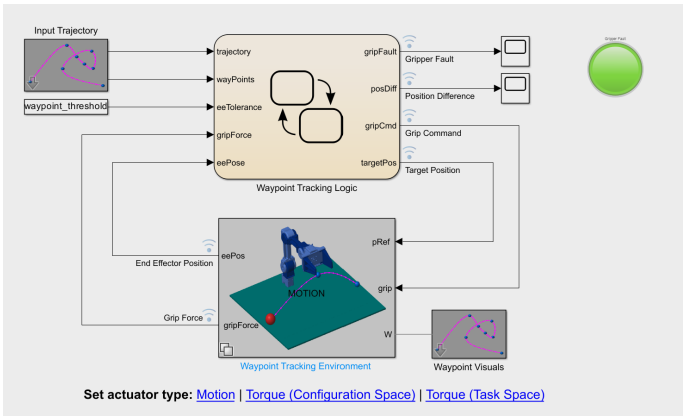

Fig. 1. Top-level Simulink architecture integrating trajectory/waypoints, waypoint tracking logic, gripper fault/diagnostics, and the simulated environment.

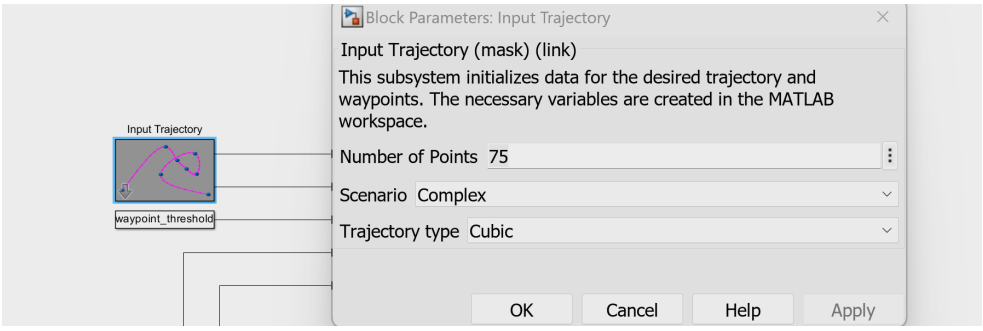

Fig. 2. Input Trajectory mask showing key configuration parameters (number of points, scenario, and trajectory type).

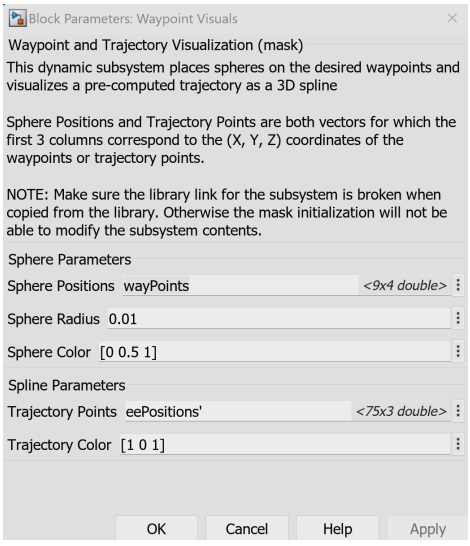

Fig. 3. Waypoint/trajectory visualization mask: waypoints are rendered as spheres (radius set in the mask), and the pre-computed trajectory is shown as a 3D spline.

by conversion from homogeneous transform to translation vector (Fig. 5).

B. Inverse Kinematics (Numerical)

Inverse kinematics (IK) solves for joint angles that realize a desired end-effector position $\boldsymbol{p}_d \in \mathbb{R}^3$. This project uses an iterative numerical solver based on the Jacobian pseudo-inverse. Let $\boldsymbol{\theta} \in \mathbb{R}^4$ be the joint vector, $\boldsymbol{p}(\boldsymbol{\theta})$ be FK position, and define the error

$$\boldsymbol{e}_k = \boldsymbol{p}(\boldsymbol{\theta}_k) - \boldsymbol{p}_d. \tag{4}$$

With Jacobian $J(\boldsymbol{\theta}) = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{\theta}}$, the update is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - J^{\dagger}(\boldsymbol{\theta}_k)\, \boldsymbol{e}_k, \tag{5}$$

where $J^{\dagger}$ denotes the Moore–Penrose pseudo-inverse. Iterations continue until $\|e_k\|_2 \leq \epsilon_{\text{IK}}$ or a maximum iteration count is reached [1].

IV. Trajectory Generation and Control

A. Cubic Trajectory Interpolation

Between waypoints, the desired Cartesian path is smoothed using cubic interpolation. For each segment i over $t \in [t_i, t_{i+1}]$, a cubic polynomial is used per coordinate:

$$\boldsymbol{p}_i(t) = \boldsymbol{a}_i + \boldsymbol{b}_i(t - t_i) + \boldsymbol{c}_i(t - t_i)^2 + \boldsymbol{d}_i(t - t_i)^3, \tag{6}$$

with coefficients selected to ensure continuity of position and (typically) velocity at waypoint boundaries [1].

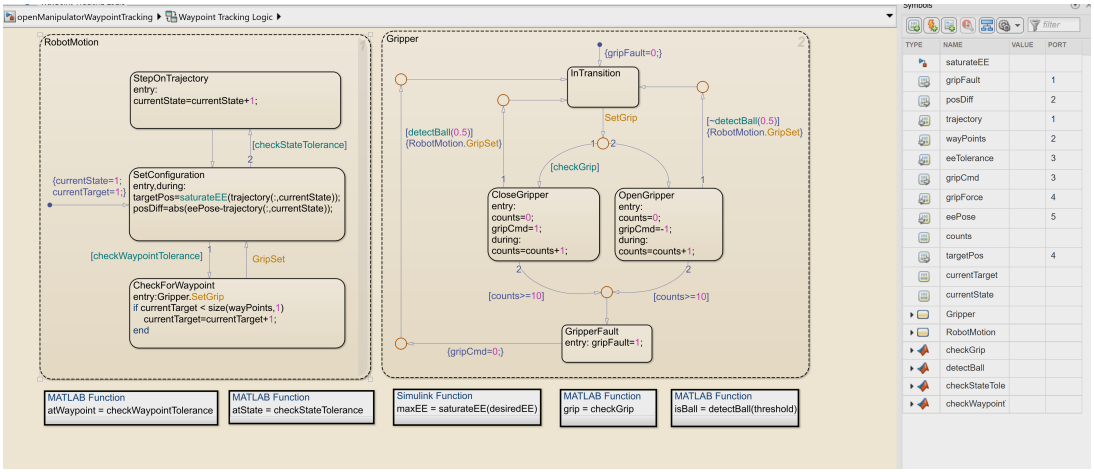

Fig. 4. Waypoint tracking and gripper Stateflow logic. Left: robot motion states progress through trajectory samples and waypoints based on tolerance checks. Right: gripper states command open/close and detect grasp success/faults using feedback.

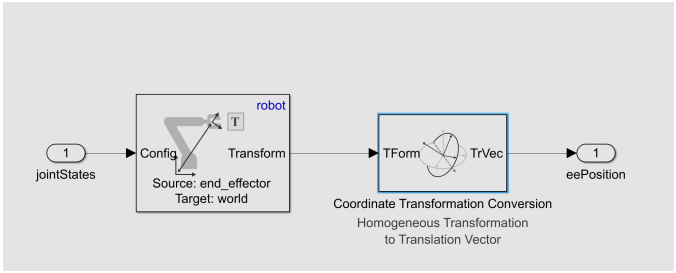

Fig. 5. Forward kinematics extraction in Simulink: transform from end-effector to world converted to a translation vector used as the end-effector position output.

B. Joint-Level PID Control

Joint actuation is performed using PID feedback to reduce tracking error between desired and measured joint signals. For a scalar tracking error $e(t)$, the PID law is

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)\, d\tau + K_D \frac{de(t)}{dt}. \qquad (7)$$

In simulation, the commanded joint positions (from IK) and gripper command are routed to the motion-actuated robot and environment (Fig. **??**).

V. Simulation Environment and Signal Flow

Fig. **??** shows the environment-level interconnection between the IK subsystem, the motion-actuated robot, and the forward kinematics block that returns end-effector position. Fig. **??** shows a more detailed Simscape Multibody view of joint/link components and the routing of sensor outputs (joint positions and gripper signals) used for monitoring and feedback.

VI. Results and Discussion

The simulated manipulator repeatedly converges to successive waypoints under the configured tolerance ϵ (Eq. 1) while following a smooth cubic path (Eq. 6). The state-machine sequencing provides a clear mechanism to (i) hold at a target until convergence, (ii) command gripper actions, and (iii) detect grasp completion or fault conditions (Fig. 4). Practical improvements for future work include damping in the Jacobian pseudo-inverse near singularities, explicit joint-limit handling, and dynamics-aware control (e.g., computed-torque) for improved transient response.

VII. Conclusion

This paper presented an end-to-end MATLAB/Simulink simulation of a 4-DoF manipulator performing waypoint-based pick-and-place. The design integrates cubic trajectory generation, Stateflow-based waypoint/gripper logic, numerical Jacobian pseudo-inverse IK, FK pose extraction, and PID joint control [1]. The resulting model provides a modular baseline for extending toward perception-driven grasping and dynamics-aware control.

References

[1] A. Kamel, "Design and Simulation of a Robotic Manipulator Using MATLAB Simulink for Pick and Place Operations," course report, ECE 818, Michigan State University, 2024.
[2] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Pearson, 2005.
[3] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2006.
[4] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, 2nd ed. Springer, 2017.