Trajectory Generation for Pick-and-Place Tasks: RRT vs P2P Comparison

Mehmet Baha Dursun – medur25@student.sdu.dk
University of Southern Denmark (SDU) – Robotics and Computer Vision 2025

1 Introduction

This report presents a comparative analysis of two trajectory generation methods for robotic pick-and-place tasks: **RRT (Rapidly-exploring Random Tree)** and **P2P (Point-to-Point)**. The experiments were conducted using a UR5 robot with a Robotik Hand-E gripper in a MuJoCo physics simulation environment. Three objects (**box, cylinder, and T-block**) were manipulated to evaluate the performance of both approaches.

The primary goal is to understand the trade-offs between automatic path planning (RRT) and pre-recorded motion sequences (P2P). The analysis focuses on joint-space trajectories, velocity profiles, acceleration characteristics, and end-effector paths computed via forward kinematics.

2 Methods

2.1 Differences and Similarities

Two fundamentally different trajectory generation approaches were implemented and compared:

P2P (Point-to-Point): A deterministic, open-loop method commonly used in structured industrial environments. The environment is assumed static and fully known. Joint configurations (waypoints) are manually recorded <similar to a teach pendant> meaning collision-free paths are guaranteed by the human operator during the "teaching" phase. This removes the need for real-time collision checking or complex inverse kinematics (IK) during execution, but lacks flexibility: any environmental change requires manual re-teaching.

RRT (Rapidly-exploring Random Tree): A sampling-based, closed-loop planning method [1]. Only the target goal is known, and the path is autonomously generated. We use the OMPL library [2] with MuJoCo [3] for collision checking. The system samples the 6-DOF configuration space (C-space) and performs real-time collision checking. A key feature is **dynamic collision modeling**: once an object is grasped, the system computes the grasp offset and treats the held object as an extension of the robot's kinematic chain.

Industrial Context: P2P is preferred in mass production due to lower computational costs and perfect repeatability. RRT becomes necessary when environment uncertainty exists or when the system must autonomously navigate around obstacles.

Common Element: Both methods use the **Trapezoidal Velocity Profile (LSPB)** for smooth interpolation between waypoints, ensuring bounded acceleration and preventing motor damage.

2.2 Trapezoidal Velocity Profile (LSPB)

The Problem with Linear Interpolation: Consider linear interpolation between two joint positions:

$$q(t) = q_i + \frac{q_f - q_i}{t_f - t_i} \cdot t \tag{1}$$

Taking the derivative to find velocity:

$$\dot{q}(t) = \frac{d}{dt}q(t) = \frac{q_f - q_i}{t_f - t_i} = \text{constant} \tag{2}$$

The problem occurs at $t = 0$: velocity must jump from $\dot{q} = 0$ (rest) to this constant value *instantly*. Computing the required acceleration:

$$\ddot{q} = \frac{\Delta\dot{q}}{\Delta t} = \frac{V - 0}{0} = \frac{V}{0} = \infty \tag{3}$$

Infinite acceleration is physically impossible and would damage motor gears. To solve this, we implement the **Linear Segment with Parabolic Blends (LSPB)** [4], which bounds acceleration to a finite value $\ddot{q}_c$. The motion is divided into three phases:

Phase 1 - Acceleration $(0 \le t \le t_c)$:

$$q(t) = q_i + \frac{1}{2}\ddot{q}_c \cdot t^2 \tag{4}$$

$$\dot{q}(t) = \ddot{q}_c \cdot t \tag{5}$$

Phase 2 - Cruise $(t_c < t \le t_f - t_c)$:

$$q(t) = q_i + \ddot{q}_c \cdot t_c \cdot (t - \frac{t_c}{2}) \tag{6}$$

$$\dot{q}(t) = \dot{q}_c \quad \text{(constant velocity)} \tag{7}$$

Phase 3 - Deceleration $(t_f - t_c < t \le t_f)$:

$$q(t) = q_f - \frac{1}{2}\ddot{q}_c \cdot (t_f - t)^2 \tag{8}$$

$$\dot{q}(t) = \ddot{q}_c \cdot (t_f - t) \tag{9}$$

where t_c is the blend time, $\ddot{q}_c$ is the cruise acceleration, and $\dot{q}_c$ is the cruise velocity.

Trapezoidal vs Parabolic Blend: These terms are related but differ in perspective. Trapezoidal profile is *velocity-based* (acceleration → cruise → deceleration), while parabolic blend is *position-based* (linear segments with parabolic blends at transitions).

Limitation: Jerk ($\dddot{q}$) is infinite at phase transitions. Alternative methods exist:

- **Cubic polynomial** (3rd order): Smoother, but no constant velocity phase. Requires 4 boundary conditions (position + velocity at both ends).
- **Quintic polynomial** (5th order): Finite jerk, smoothest motion. Requires 6 boundary conditions (position + velocity + acceleration at both ends).

Trapezoidal remains preferred for industrial applications due to its simplicity and predictable acceleration limits.

2.3 Pseudo Algorithms

Algorithm 1 RRT Path Planning

Input: Start config q_{start}, Goal frame T_{goal}, Held object O
Output: Collision-free trajectory τ

$IK_{sols} \leftarrow$ SweepJ1(T_{goal}, $n = 64$) ▷ J1 sweep
$GoalRegion \leftarrow$ OMPL.GoalSampleable(IK_{sols})
$Space \leftarrow$ RealVector(6), bounds $[-\pi, \pi]$
$Validator \leftarrow$ StateValidator(O) ▷ MuJoCo
if $O \neq$ None **then**
    $Offset \leftarrow$ ComputeGraspOffset(O)
    Validator.SetHeldObject(O, $Offset$)
end if
$Planner \leftarrow$ RRT($Space$, $Validator$)
$Path \leftarrow$ Planner.Solve(q_{start}, $GoalRegion$, $t = 15s$)
$\tau \leftarrow$ ApplyLSPB($Path$, $dt = 0.002$)
return τ

Algorithm 2 P2P Waypoint Execution

Input: Object name obj, Robot state
Output: Executed trajectory τ

$Waypoints \leftarrow$ LoadConfig(obj) ▷ config.yaml
$close_{idx} \leftarrow$ Config.gripper_close_after
$open_{idx} \leftarrow$ Config.gripper_open_after
$q_{curr} \leftarrow$ Robot.GetCurrentQ()
for $i = 0$ to $|Waypoints| - 1$ **do**
    $q_{target} \leftarrow Waypoints[i]$
    $\tau_i \leftarrow$ LSPB(q_{curr}, q_{target}, $t = 500ms$)
    Execute(τ_i)
    $q_{curr} \leftarrow q_{target}$
    if $i = close_{idx}$ **then**
        Gripper.Close()
    else if $i = open_{idx}$ **then**
        Gripper.Open()
    end if
end for
return τ

Key Differences:

- **Planning:** RRT samples C-space randomly with 15s timeout; P2P uses pre-recorded waypoints instantly.
- **Collision:** RRT checks at resolution 0.005 rad; P2P assumes safe paths.
- **IK:** RRT uses J1 sweep (64 samples) for goal region; P2P stores joint configs directly.
- **Repeatability:** P2P is identical every run; RRT varies due to random sampling.

3 Results and Discussion

3.1 Joint Position Trajectories

Figure 1 shows the joint position comparison for the box manipulation task. Each subplot represents one of the six UR5 joints. RRT trajectories (blue solid) exhibit more exploratory paths due to random sampling in C-space, while P2P trajectories (red dashed) follow direct, predetermined routes.

Similar patterns were observed for the cylinder (Figure 2) and T-block objects, with RRT consistently producing longer trajectories due to its sampling-based nature.

3.2 Velocity Profiles

Figure 3 presents the joint velocity profiles for the box task. The expected trapezoidal shape (ramp-up, constant, ramp-down) is visible in both methods.

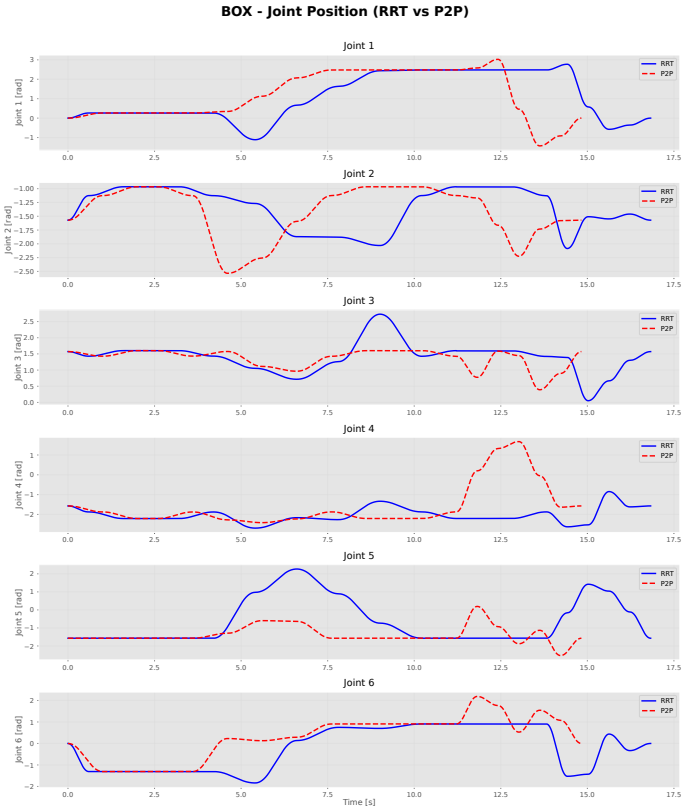

Figure 1: Box - Joint position trajectories (6 DOF). RRT (blue) shows exploratory behavior; P2P (red dashed) follows direct paths.

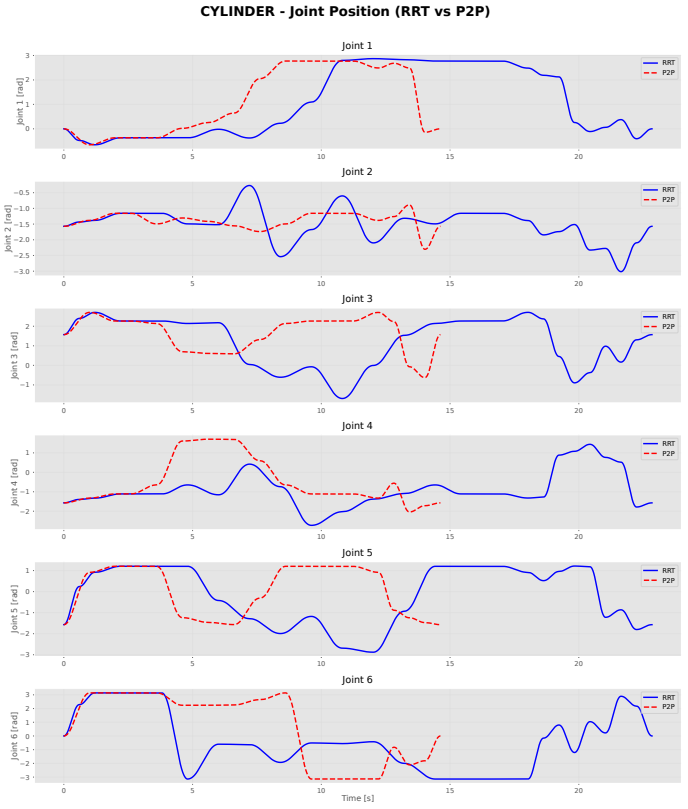

Figure 2: Cylinder - Joint position trajectories. RRT requires more waypoints due to complex obstacle avoidance.

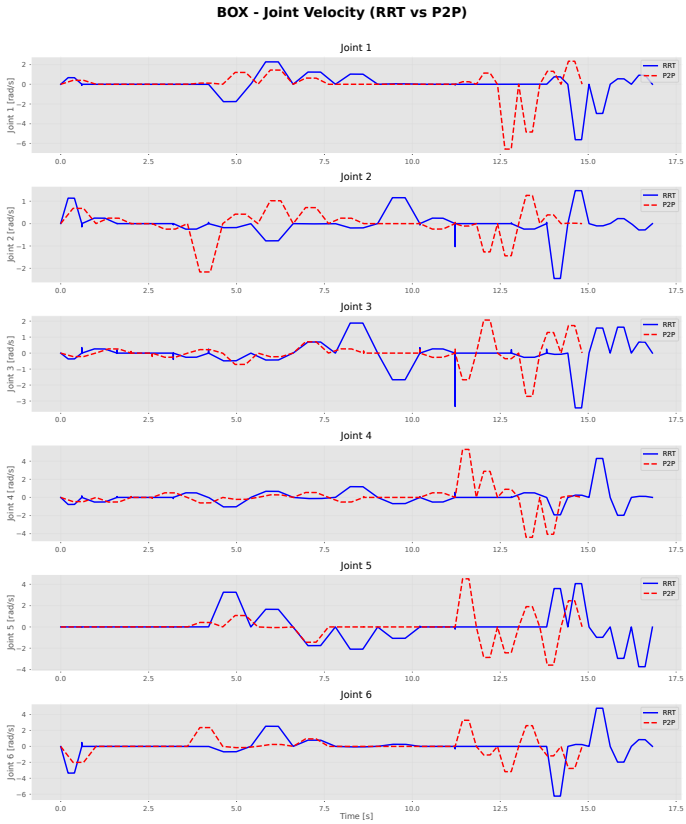

Figure 3: Box - Joint velocity profiles by DOF. Both RRT and P2P show trapezoidal shape with visible spikes at segment boundaries.

Velocity Spikes Analysis: Sharp velocity spikes are observed at segment transition points. These arise from three sources:

1. **Zero-velocity boundary conditions:** Each LSPB segment forces $\dot{q} = 0$ at start and end, creating discontinuous jumps when segments connect.
2. **Numerical differentiation:** Velocity is computed as $\dot{q} \approx \Delta q / \Delta t$ using `np.diff()`. Small position steps amplify to large velocity values. For solving you can add calculation filtering this to the code.
3. **Multi-segment concatenation:** The `generate_via_point_trajectory()` function concatenates independent LSPB profiles, causing velocity resets at each waypoint.

3.3 Acceleration Profiles

Figure 4 displays the acceleration profiles. The characteristic step pattern of LSPB is evident: constant positive acceleration (Phase 1), zero acceleration during cruise (Phase 2), and constant negative acceleration (Phase 3).

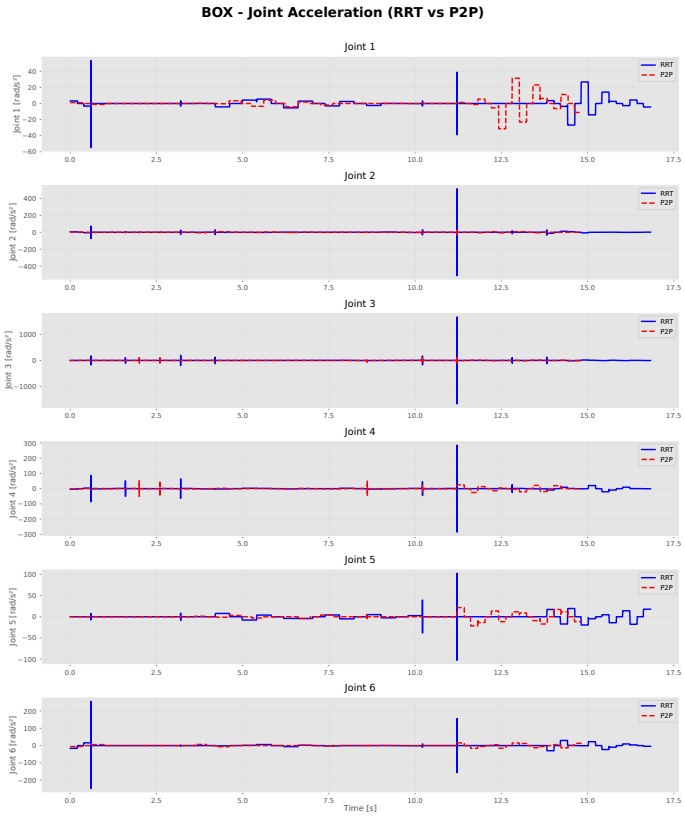

Figure 4: Box - Joint acceleration profiles by DOF. Step pattern confirms LSPB implementation. Large spikes indicate infinite jerk at phase transitions.

Acceleration Spikes Analysis: The prominent spikes represent **infinite jerk** a fundamental limitation of trapezoidal profiles. Consider the phase transition at $t = t_c$ (end of acceleration phase):

Before transition ($t < t_c$): $\ddot{q} = \ddot{q}_c$ (constant acceleration)

After transition ($t > t_c$): $\ddot{q} = 0$ (cruise phase, no acceleration)

Computing jerk (rate of change of acceleration):

$$\dddot{q} = \frac{d\ddot{q}}{dt} = \frac{\Delta \ddot{q}}{\Delta t} = \frac{0 - \ddot{q}_c}{0} = -\frac{\ddot{q}_c}{0} = -\infty \tag{10}$$

This infinite jerk occurs at all four phase transitions: $t = 0$, $t = t_c$, $t = t_f - t_c$, and $t = t_f$. In practice, motor controllers have finite bandwidth that filters these spikes, but they can cause mechanical vibration and wear.

Solution: Quintic (5th order) polynomial trajectories guarantee finite jerk by enforcing $\ddot{q}(t_i) = \ddot{q}(t_f) = 0$ as boundary conditions, ensuring smooth acceleration transitions.

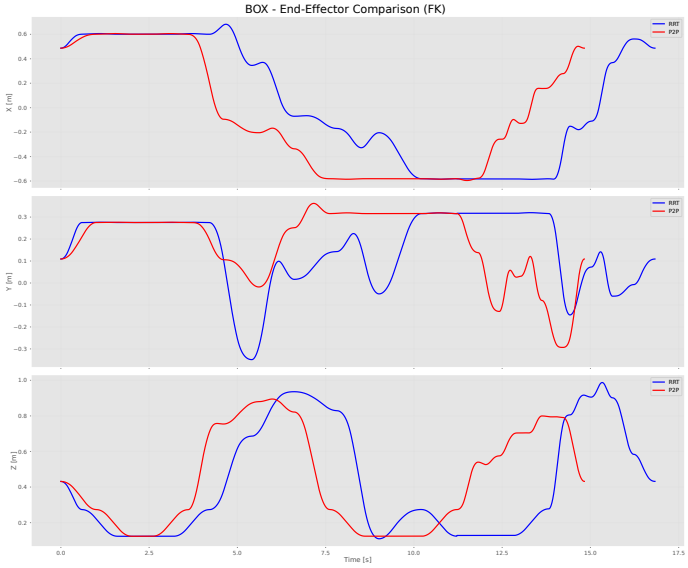

Figure 5: Box - End-effector position by Cartesian DOF (X, Y, Z). RRT path shows more oscillation; P2P follows smoother Cartesian trajectory.

3.4 Forward Kinematics (End-Effector Paths)

Figure 5 displays the end-effector (TCP) position computed via forward kinematics using the roboticstoolbox [5]. The three subplots show X, Y, and Z Cartesian coordinates over time, computed as:

$$T_{ee}(t) = FK(q(t)) = \prod_{i=1}^{6} A_i(q_i(t)) \tag{11}$$

where A_i are the DH transformation matrices for the UR5.

FK Analysis: Despite different joint-space trajectories, both methods reach the same pick and place positions. RRT's exploratory joint motion translates to more complex end-effector paths, while P2P's direct joint interpolation produces smoother Cartesian motion. This demonstrates that joint-space optimality does not guarantee Cartesian-space optimality.

3.5 T-block Analysis

Figure 6 shows the T-block manipulation trajectories. The asymmetric geometry of the T-block requires more careful gripper alignment, resulting in slightly different joint configurations compared to symmetric objects.

3.6 Performance Metrics

Table 1 summarizes the trajectory statistics across all three objects.

Analysis:

- RRT requires 13-56% more trajectory points due to exploratory sampling.
- Execution time is 13-56% longer for RRT, but planning provides collision safety.
- P2P is faster but cannot adapt to environmental changes.
- The cylinder task shows the largest difference due to its complex placement location requiring more RRT exploration.

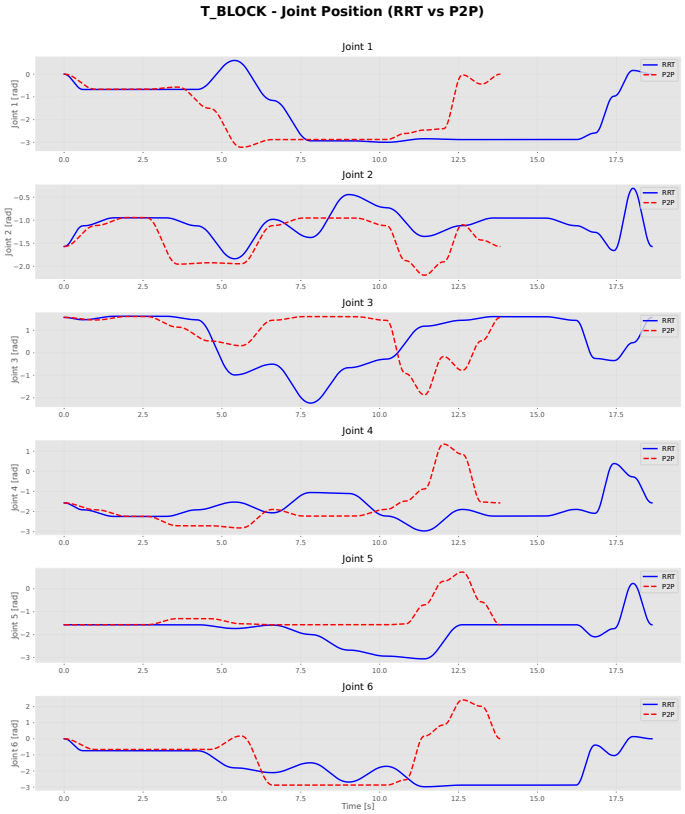

Figure 6: T-block - Joint position trajectories. Asymmetric object geometry leads to unique grasp configurations.

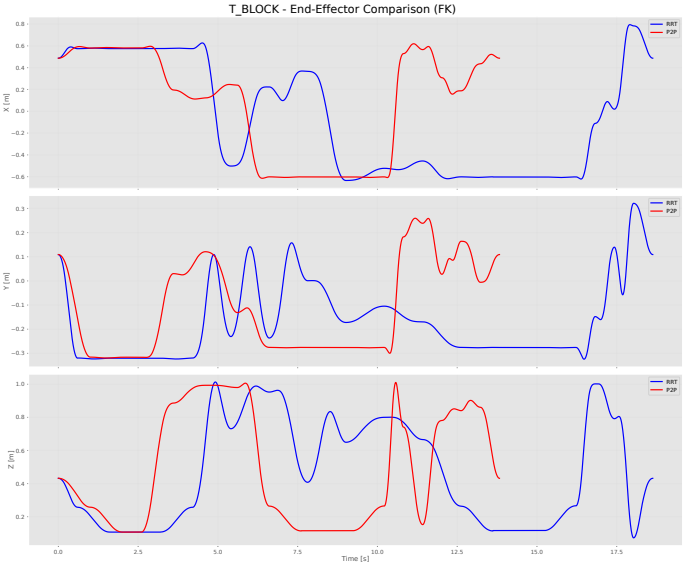

Figure 7: T-block - End-effector FK comparison. Both methods successfully reach pick/place positions despite different joint-space paths.

Table 1: Trajectory Statistics Comparison

Metric	Box	Cylinder	T-block
RRT Points	8415	11422	9316
P2P Points	7415	7314	6914
RRT Time (s)	16.83	22.84	18.63
P2P Time (s)	14.83	14.63	13.83

4 Conclusion

This study compared RRT and P2P trajectory generation for pick-and-place tasks. Key findings:

P2P Advantages: Faster execution, perfect repeatability, no computational overhead, ideal for static industrial environments with known layouts.

RRT Advantages: Automatic collision avoidance, adapts to unknown obstacles, dynamic collision modeling for held objects, essential for flexible automation.

Common Strength: Both methods successfully utilize the trapezoidal velocity profile (LSPB) for smooth, bounded-acceleration motion.

Future Improvements:

- Implement RRT* for asymptotically optimal paths
- Use bidirectional RRT for faster convergence
- Develop hybrid approach: P2P with RRT fallback for obstacle avoidance
- Replace LSPB with S-curve profiles to eliminate infinite jerk

Supplementary Materials: Video demonstrations are available online:

- P2P Data Collection: https://youtu.be/ljMGeYzblWg
- P2P Execution: https://youtu.be/nj_jDb7eGxQ
- RRT Execution: https://youtu.be/tpHh2QvFxpo

References

1. LaValle, S. M. *Rapidly-Exploring Random Trees: A New Tool for Path Planning* tech. rep. TR 98-11 (Iowa State University, 1998).
2. Sucan, I. A., Moll, M. & Kavraki, L. E. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* **19,** 72–82 (2012).
3. Todorov, E., Erez, T. & Tassa, Y. *MuJoCo: A Physics Engine for Model-Based Control* in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2012), 5026–5033.
4. Sciavicco, L., Siciliano, B., Villani, L. & Oriolo, G. *Robotics: Modelling, Planning and Control* 2nd. ISBN: 978-1-84628-641-4 (Springer Science & Business Media, 2012).
5. Corke, P. *Robotics, Vision and Control: Fundamental Algorithms in Python* https://petercorke.github.io/robotics-toolbox-python/ (Springer Nature, 2023).