PyBullet Industrial Path Planner: Handling Transitional Movements in Constrained Robot Cell Environments

Philipp Schulz*, Malte Hansjosten*, Alexander Puchta, Jürgen Fleischer

Karlsruhe Institute of Technology (KIT), Institute for Production Science (wbk)

*Shared first authorship. Contact: malte.hansjosten@kit.edu

1 Summary

This work extends the PyBullet Industrial (PBI) simulation library. PBI was developed to address the limitations of simulators like CoppeliaSim [1] and Gazebo [2] by integrating process modeling into robotic simulations with a focus on industrial applications [3]. A key feature is the simulation of additive and subtractive processes. However, the library lacks functionalities for planning transitional movements, such as tool changes and part handling, which are essential for creating complete, executable process sequences.

To address this, the **PyBullet Industrial Path Planner** extends PBI by integrating the Open Motion Planning Library (OMPL) [4]. This enables the planning of collision-free joint paths, guided by process-specific constraints and customizable cost functions. In several example scenarios, planned paths were evaluated in simulation and parameterized via integrated numerical control (NC) code modules for deployment on physical robots.

2 Motivation

At the wbk Institute of Production Science, research focuses on using industrial robots to automate complex tasks. The PyBullet Industrial (PBI) library supports this research through the simulation of robots performing process operations such as welding, milling, and gripping.

A representative research area is the automated generation of adaptive disassembly sequences for end-of-life electric motors using 6-axis industrial robots [5]. While PBI enables the simulation of individual process operations generated by process planning modules[6], it lacks motion planning capabilities for transitional movements. Typical use cases include connecting process steps, performing tool changes, or executing manipulation tasks. Without proper planning, these transitions can lead to collisions, constraint violations, or robot singularities [7]. This work aims to close this gap by introducing the PBI Path Planner, a generalized solution for planning constraint-aware transitional movements.

3 Overview

The PBI Path Planner addresses the planning of transitional movements by framing them as a path planning problem with known start and goal configurations in a static environment. The planning process considers several types of constraints:

- **Kinematic Constraints:** Structure of the kinematic chain, including multiple links and joints, as well as variable end-effector tools and grasped objects.
- **Collision Constraints:** Avoidance of undesired geometric interference within the moving kinematic chain and with external static objects, while accounting for allowed interactions.
- **Additional Constraints:** Incorporation of task-specific or system-imposed restrictions that further refine the planning problem (e.g., workspace restrictions).

While constraints ensure the validity of a solution path, the following optimality criteria are relevant for the usability in industrial applications:

- **Smoothness:** Gradual, continuous end-effector motion that avoids unnecessary detours.
- **Path Length Objective:** Preference for the shortest feasible path as a default behavior.
- **Use-Case Objectives:** Ability to optimize paths for specific use cases (e.g., clearance margins).

The framework is designed for offline planning, with solution paths stored as sequences of equally spaced joint states. To convert these into executable trajectories that respect the differential constraints of the robotic system [7], downstream modules are provided for post-processing and translation into machine commands.

4 Software Framework

Sampling-based planners are widely used in industrial robot motion planning due to their efficiency in high-dimensional configuration spaces [8]. These algorithms explore the search space by rapidly evaluating static states through randomized sampling schemes and connecting valid states to construct feasible solution paths [9]. They are generally proven to be probabilistically complete, and some variants have been shown to achieve asymptotic optimality [10].

The Open Motion Planning Library (OMPL) is built around state-of-the-art sampling-based algorithms and supports both constrained and optimal planning. Its modular architecture and widespread adoption in frameworks like MoveIt [11] have established it as a standard for geometric motion planning in research.

Figure 1 illustrates the three-layer design of the PBI Path Planner framework. The bottom layer consists of extended OMPL modules tailored for integration with PyBullet Industrial. The middle layer provides an interface where user-configurable inputs define a planning instance using the *Simple Setup* class, designed for user-friendly configuration and planning. The top layer contains modules specific to the PyBullet Industrial environment.

User inputs are categorized as either required or optional. During initialization, the planning instance is linked to a fixed robot model, while all other inputs can be configured dynamically through subsequent planning queries. Inputs can be provided as class instances or callables, allowing function-based configuration at runtime for greater flexibility.

The specific components of the framework are presented in relation to the previously introduced constraints, optimality criteria, and the generation of solution paths (see Section 3).

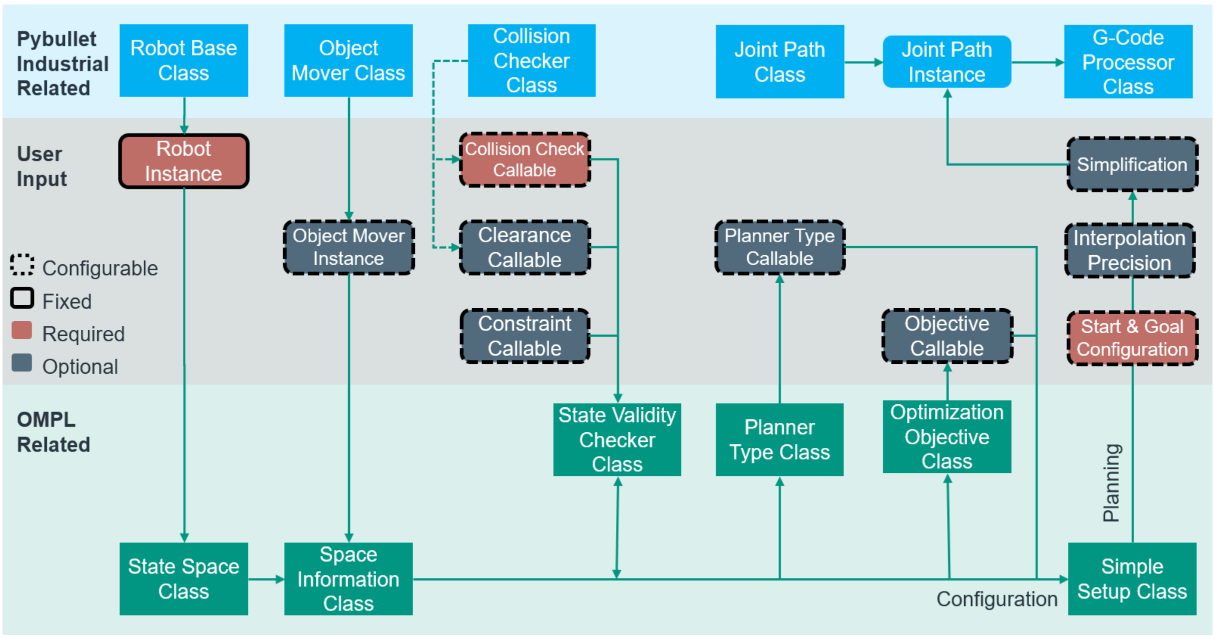

Figure 1: Overview of the PBI Path Planner Framework.

4.1 Incorporation of Kinematic Constraints

To account for the kinematic constraints and the high-frequency displacements inherent in sampling-based planning algorithms, the *State Space*, *Object Mover*, and *Space Information* classes play a central role.

The *State Space* class defines the configuration space of the robot, encompassing all valid joint configurations. It is constructed based on the number and type of movable joints provided by the *Robot Base* instance. Currently, the framework supports any serial kinematic chain consisting of prismatic and revolute joints with defined upper and lower limits.

The *Object Mover* class extends the robot's kinematic structure beyond the end-effector. In industrial applications, tool changes are common, and tools are often used to transport components such as unscrewed bolts or partial assemblies. The class therefore allows users to update the current state of the complete kinematic structure.

The *Space Information* class serves as a central access point for the planning process. It receives the configured *Robot Base* and *Object Mover* instances via user input and enables efficient static state sampling by displacing the entire kinematic structure.

4.2 Incorporation of Collision & Additional Constraints

While the *Space Information* class maps the kinematic chain into sampled states, the *State Validity Checker* class evaluates their feasibility based on user-defined collision and additional constraints.

During each evaluation cycle, the mandatory collision check function is called first. To support its formulation, the *Collision Checker* class offers tools for managing collision pairs, allowing them to be categorized as allowed, internal, or external. It also includes support for clearance callables, which compute the minimum distances between objects. Although clearance values do not directly determine feasibility, they can be used by use-case-specific optimization objectives.

Additional constraints restrict the search to relevant subspaces [9]. A typical example is transporting a glass of water, which requires the end-effector to maintain an upright orientation throughout the motion. While OMPL provides a dedicated constrained planning module [12] for such cases, it was not available in the pre-built Python wheels at the time of this work. As a result, constraint handling was integrated directly into the *State Validity Checker* class. To increase the likelihood of finding feasible samples, constraint callables should incorporate tolerance ranges.

This design choice comes with the limitation that direct sampling within the constrained subspace is not possible. Instead, the framework relies on sampling strategies that steer towards promising regions.

4.3 Incorporation of Optimality Criteria

Planner types and optimization objectives are handled as user-defined callables derived from OMPL-related classes. This approach supports deferred initialization, where the complete *Space Information* instance is passed dynamically at runtime.

Users can configure planning algorithms flexibly by directly accessing OMPL's extensive library. Optimal planners default to the path length objective, gradually optimizing toward short and smooth paths. To support use-case-specific goals, the framework provides additional custom objectives. The *Clearance Objective* class encourages maximum distance from nearby obstacles using the clearance callable, while the *Multi-Objective* class allows weighted combinations of multiple cost terms.

4.4 Determination of Solution Paths and NC Code Generation

Determination of solution paths with the configured *Space Information* instance requires defining start and goal configurations. If enabled, OMPL's optional path simplification improves path smoothness by eliminating unnecessary detours.

The resulting output is an equally spaced joint path, generated based on a predefined interpolation precision. This path is stored and managed through the provided functionalities of the *Joint Path* class. Depending on the specific robot setup, conversion to executable control code may be required for deployment. PBI supports this by providing a *G-Code Processor* module for converting joint paths into NC code.

5 Features & Capabilities

Like PyBullet Industrial, the PBI Path Planner was designed to support research in the field of industrial robotics. The focus was on both immediate usability and extensibility for specialized applications. To improve accessibility, a graphical user interface (GUI) was developed, providing an intuitive platform for setting up and managing path planning scenarios.

To demonstrate the planner's functionality, the following subsections present a set of preconfigured planning scenarios created using the GUI. These examples are included in the demonstration files provided with the package.

5.1 Planning in Cluttered Environments

This scenario demonstrates the planner's capability to generate smooth paths in cluttered environments. Figure 2 shows the resulting solution pat, where the robot moves an electric motor to a designated storage position. The trajectory avoids a barrier while guiding the robot to place the workpiece inside a storage box positioned at a tilted angle, increasing the complexity of the planning task.

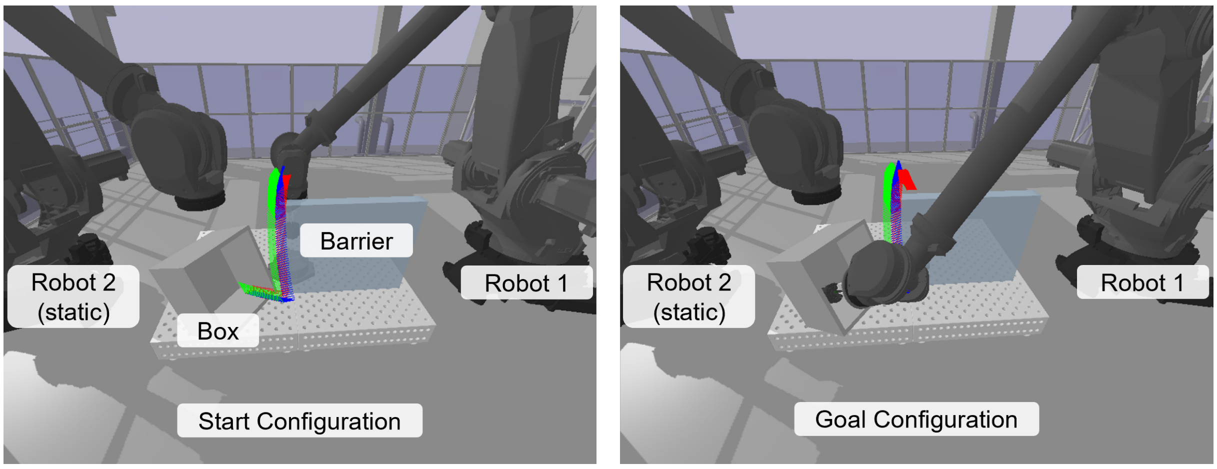

Figure 2: Solution path navigating through cluttered environment.

5.2 Planning with Additional Constraints

This scenario demonstrates the planner's ability to handle tasks with additional constraints, specifically maintaining an upright end-effector orientation throughout the motion. As depicted in Figure 3, a direct comparison is made between a constrained and an unconstrained planning task. The resulting paths are visualized along with the corresponding end-effector frames at discrete interpolation steps. In the unconstrained case, the end-effector tilts during motion, as indicated by red x-axis deviating from a parallel alignment with the floor. In contrast, the constrained path maintains the x-axis parallel to the floor, preserving the required upright orientation.

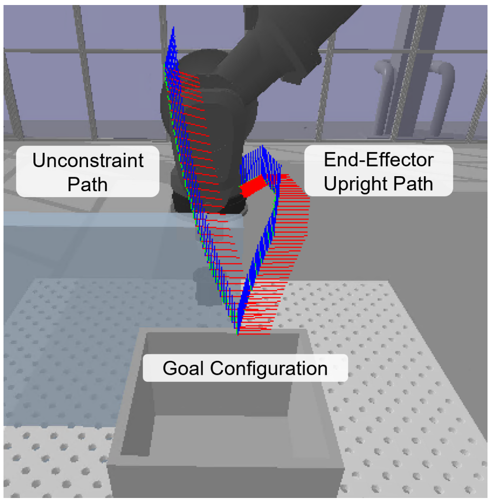

Figure 3: Comparison of unconstrained and end-effector-upright solution paths.

5.3 Multi-Objective Optimization Planning

This scenario demonstrates path planning with customized objectives using the *Multi-Optimization Objective* class, which optimizes a weighted trade-off between path length and clearance. The task involves one robot moving over another, which is treated as a static obstacle. Adjusting the weighting revealed the expected trade-off between directness and maintaining safety margins. However, the outcome is highly sensitive to the selected weights. While the motion cost for path length behaves linearly, the clearance cost increases non-linearly due to the inverse formulation of the underlying state cost. This imbalance adds complexity to weight tuning and makes it difficult to achieve a predictable trade-off. As a result, defining weight configurations that generalize well across different planning scenarios remains a challenge.

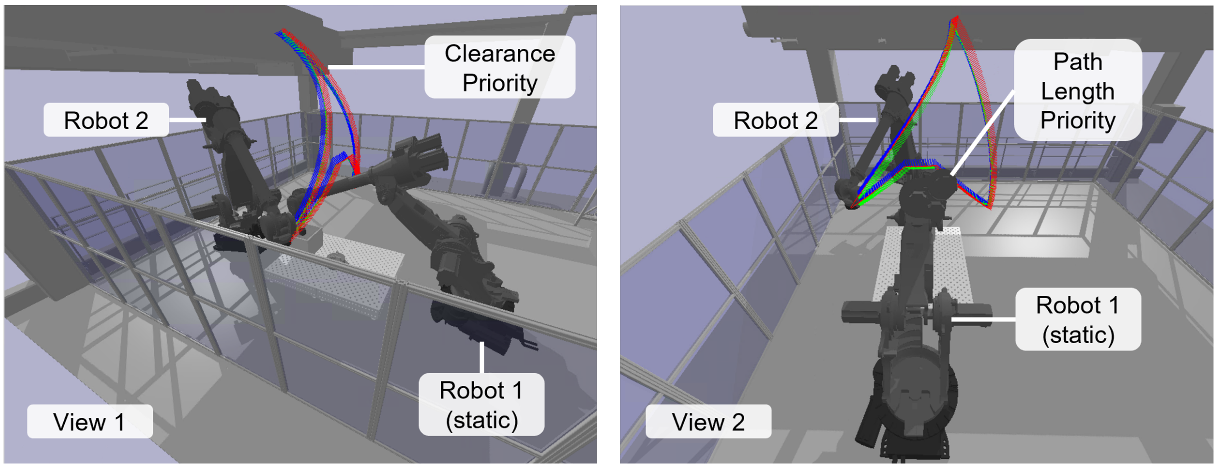

Figure 4: Computed solution paths using multi-objective optimization with different weightings for clearance and path length objectives.

5.4 System Deployment

As shown in Figure 5, the planning scenario presented in the previous subsection was successfully executed in a real robotic cell. The resulting trajectory closely matched the behavior observed in simulation.

The joint path was converted into NC-code and further processed using the PBI module *G-Code Simplifier*. This included assigning a constant feed rate to define execution speed along the sequence of states, as well as additional adjustments to comply with the machine code conventions of the robotic system.

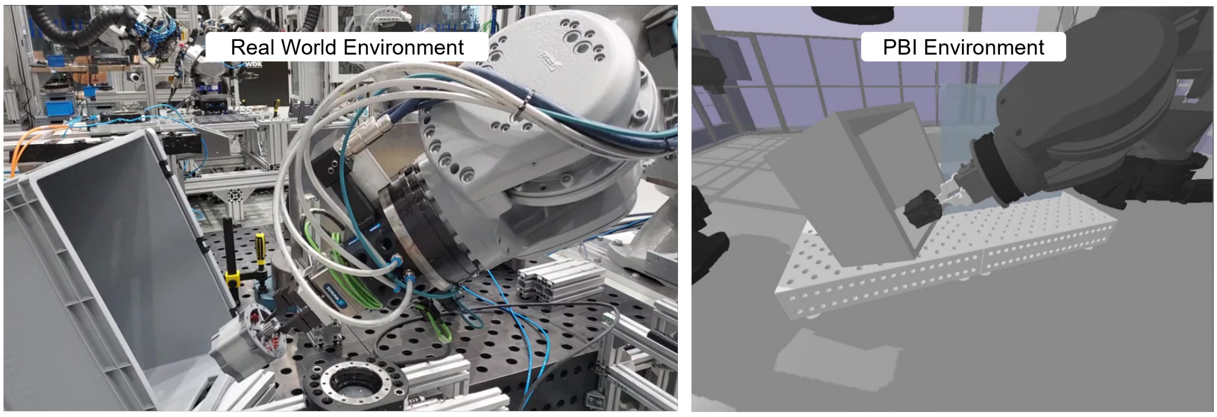

Figure 5: System deployment showing alignment between physical execution and simulation.

6 Conclusion

This work addresses the planning of transitional movements in robotic systems, contributing to the broader goal of advancing industrial automation through the generation of complete and executable NC code for entire process workflows.

The PBI Industrial Path Planner fills a functional gap in PyBullet Industrial by adding advanced path planning capabilities suitable for real-world deployment. It currently enables transition planning for various serial kinematic robots. By integrating OMPL through tailored extensions, it provides access to advanced sampling-based algorithms adapted to industrial requirements. The system also supports additional use-case constraints and customizable optimization objectives, allowing it to handle a wide range of application scenarios.

While the system provides a solid functional base, further development will focus on enhancing its practical applicability and enabling higher levels of automation and integration. In particular, fully automated solutions will require high-level modules capable of extracting relevant process information from both preceding and subsequent operations. Additionally, complex planning scenarios can result in runtimes of up to two minutes; future work will focus on reducing the overhead from collision checking [13] and improving computational performance through parallelization.

Recent trends in motion planning suggest strong potential in combining sampling-based methods with machine learning [8] to further improve adaptability and efficiency, particularly in environments with recurring geometric structures. Although OMPL does not currently support learning-based planning, the presented framework offers a solid foundation for implementing such approaches. This could significantly contribute to future research aimed at achieving full industrial automation of complex robotic operations in constrained cell environments.

References

[1] E. Rohmer, S. P. N. Singh, and M. Freese. “CoppeliaSim (Formerly v-REP): A Versatile and Scalable Robot Simulation Framework”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013.

[2] Nathan Koenig and Andrew Howard. “Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sendai, Japan, Sept. 2004, pp. 2149–2154.

[3] Jan Baumgärtner et al. “PyBullet Industrial: A Process-Aware Robot Simulation”. In: *Journal of Open Source Software* 8.85 (May 18, 2023), p. 5174. ISSN: 2475-9066. DOI: 10.21105/joss.05174. URL: https://joss.theoj.org/papers/10.21105/joss.05174 (visited on 02/25/2025).

[4] Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. “The Open Motion Planning Library”. In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012), pp. 72–82. ISSN: 1070-9932, 1558-223X. DOI: 10.1109/MRA.2012.2205651. URL: https://ieeexplore.ieee.org/document/6377468/ (visited on 04/24/2025).

[5] Malte Hansjosten and Jürgen Fleischer. "Towards Autonomous Adaptive Disassembly of Permanent-Magnet Synchronous Motors with Industrial Robots". In: *Manufacturing Letters* 35 (Aug. 2023), pp. 1336–1346. ISSN: 22138463. DOI: `10.1016/j.mfglet.2023.08.084`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S2213846323001414` (visited on 05/10/2025).

[6] Malte Hansjosten, Jan Baumgärtner, and Jürgen Fleischer. "Generalized Partially Destructive Disassembly Planning for Robotic Disassembly". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024 IEEE International Conference on Robotics and Automation (ICRA). May 2024, pp. 1978–1984. DOI: `10.1109/ICRA57147.2024.10610546`. URL: `https://ieeexplore.ieee.org/document/10610546/` (visited on 05/10/2025).

[7] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. 1st ed. Cambridge University Press, May 25, 2017. ISBN: 978-1-316-66123-9 978-1-107-15630-2 978-1-316-60984-2. DOI: `10.1017/9781316661239`. URL: `https://www.cambridge.org/core/product/identifier/9781316661239/type/book` (visited on 10/15/2024).

[8] Mehran Ghafarian Tamizi, Marjan Yaghoubi, and Homayoun Najjaran. "A Review of Recent Trend in Motion Planning of Industrial Robots". In: *International Journal of Intelligent Robotics and Applications* 7.2 (June 1, 2023), pp. 253–274. ISSN: 2366-598X. DOI: `10.1007/s41315-023-00274-2`. URL: `https://doi.org/10.1007/s41315-023-00274-2` (visited on 08/27/2024).

[9] Steven M. LaValle. *Planning Algorithms*. 1st ed. Cambridge University Press, May 29, 2006. ISBN: 978-0-521-86205-9 978-0-511-54687-7. DOI: `10.1017/CBO9780511546877`. URL: `https://www.cambridge.org/core/product/identifier/9780511546877/type/book` (visited on 04/03/2025).

[10] Sertac Karaman and Emilio Frazzoli. *Sampling-Based Algorithms for Optimal Motion Planning*. May 5, 2011. DOI: `10.48550/arXiv.1105.1186`. arXiv: `1105.1186 [cs]`. URL: `http://arxiv.org/abs/1105.1186` (visited on 04/27/2025). Pre-published.

[11] Sachin Chitta, Ioan Sucan, and Steve Cousins. "MoveIt! [ROS Topics]". In: *IEEE Robotics & Automation Magazine* 19.1 (Mar. 2012), pp. 18–19. ISSN: 1558-223X. DOI: `10.1109/MRA.2011.2181749`. URL: `https://ieeexplore.ieee.org/document/6174325/` (visited on 04/24/2025).

[12] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. "Sampling-Based Methods for Motion Planning with Constraints". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (May 28, 2018), pp. 159–185. ISSN: 2573-5144, 2573-5144. DOI: `10.1146/annurev-control-060117-105226`. URL: `https://www.annualreviews.org/doi/10.1146/annurev-control-060117-105226` (visited on 03/24/2025).

[13] Christer Ericson. *Real-Time Collision Detection*. 0th ed. CRC Press, Dec. 22, 2004. ISBN: 978-0-08-047414-4. DOI: `10.1201/b14581`. URL: `https://www.taylorfrancis.com/books/9780080474144` (visited on 04/24/2025).