The Journal of Open Source Software

ManipulaPy: A GPU-Accelerated Python Framework for Robotic Manipulation, Perception, and Control

M. I. M. AboElNasr [1]

1 Universität Duisburg-Essen

DOI: 10.21105/joss.08490

Software

- Review
- Repository
- Archive

Editor: Owen Lockwood

Reviewers:

- @sea-bass
- @cadojo

Submitted: 03 May 2025
Published: 18 October 2025

Summary

ManipulaPy is an open-source Python toolbox that unifies the entire manipulation pipeline—from URDF parsing to GPU-accelerated dynamics, vision-based perception, planning and control—within a single API. Built on the Product-of-Exponentials (PoE) model (Lynch & Park, 2017), PyBullet (Coumans & Bai, 2019), CuPy (Okuta et al., 2017) and custom CUDA kernels (Liang et al., 2018), the library enables researchers to move from robot description to real-time control with up to a 40× speedup over CPU implementations. DOF-agnostic GPU trajectory kernels accelerate 6-DOF and higher manipulators, while specialized inverse-dynamics prototypes achieve up to 3600× speedups for batch processing. Performance claims are reproducible via benchmarks in the repository.

Statement of Need

Modern manipulation research requires tight integration of geometry, dynamics, perception, planning, and control within a unified computational framework. However, existing open-source tools address only portions of this pipeline, forcing researchers to write substantial integration code:

Library	Core Strengths	Integration Challenges
MoveIt (Chitta et al., 2012)	Mature sampling-based planners	Custom ROS nodes for sensor integration, external plugins for real-time dynamics, no native GPU acceleration
Pinocchio (Carpentier et al., 2025)	High-performance PoE dynamics (C++)	CPU-only; perception & planning must be synchronized manually
CuRobo (Sundaralingam et al., 2023)	GPU collision checking & trajectory optimization	Planning-focused; lacks perception pipeline and closed-loop control
Python Robotics Toolbox (Corke & Haviland, 2021)	Educational algorithms, clear APIs	CPU-only; users build simulation/control/vision components separately

These integration challenges manifest as sensor-planner gaps, dynamics-control mismatches, GPU memory fragmentation, and synchronization complexity between components.

ManipulaPy eliminates these integration burdens through a unified Python API that maintains data consistency across the entire manipulation pipeline with GPU acceleration throughout.

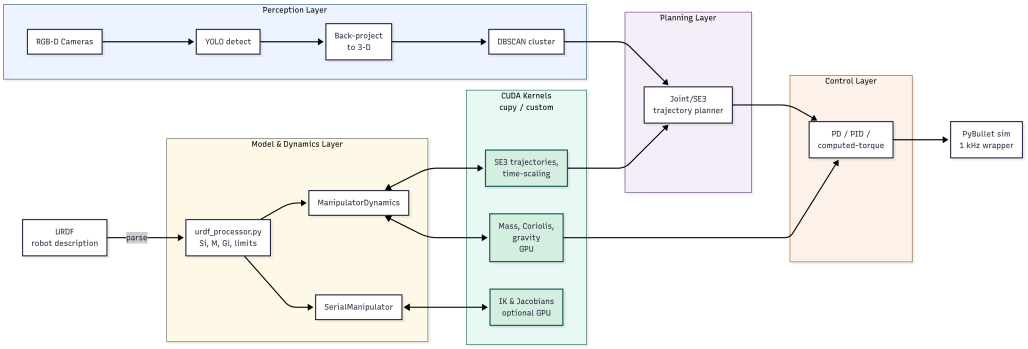

Figure 1: System architecture of ManipulaPy showing the unified manipulation pipeline. The framework integrates URDF processing, GPU-accelerated kinematics and dynamics, motion planning with collision avoidance, multiple control strategies, and PyBullet simulation within a single API. Data flows consistently between components without manual synchronization, while GPU acceleration provides 40× speedup for trajectory generation and real-time dynamics computation.

Library Architecture

ManipulaPy implements a unified manipulation pipeline with coherent data flow where each component builds upon shared representations:

Robot Model Processing converts URDF descriptions into PoE representations, extracting screw axes, mass properties, and joint constraints through PyBullet integration. This creates fundamental `SerialManipulator` and `ManipulatorDynamics` objects used throughout the system.

Kinematics and Dynamics provide vectorized FK/IK, Jacobians, and GPU-accelerated trajectory time-scaling that is DOF-agnostic. GPU dynamics kernels are shape-agnostic but simplified (per-joint/diagonalized), while fully coupled n-DOF spatial dynamics remain on the CPU path for exactness.

Motion Planning generates collision-free trajectories using GPU-accelerated time-scaling functions, supporting both joint-space and Cartesian-space planning with real-time obstacle avoidance.

Control Systems implement classical (PID, computed torque) and modern (adaptive, robust) control algorithms with automatic gain tuning, operating on the same dynamic model used in planning.

Simulation Framework provides PyBullet integration with synchronized camera rendering, physics simulation, and control execution.

Vision and Perception Pipeline

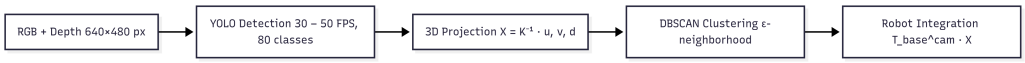

Figure 2: ManipulaPy vision and perception pipeline architecture. The five-stage pipeline processes raw sensor data from stereo cameras and RGB-D sensors through object detection using YOLO v8, transforms 2D detections to 3D world coordinates, applies DBSCAN clustering for object segmentation, and maintains multiple obstacle representations (point clouds, geometric primitives, SDFs) for robot integration at 5-15 Hz refresh rates during trajectory execution.

ManipulaPy's perception system converts raw sensor data into actionable robot knowledge through a five-stage pipeline:

Sensor Fusion handles stereo cameras (RGB+depth via OpenCV rectification), RGB-D sensors, and point cloud input with temporal alignment across sensor types.

Object Detection integrates YOLO v8 (Jocher et al., 2023) for real-time 2D bounding box detection and supports custom detectors for domain-specific models.

3D Integration transforms pixel coordinates to 3D world positions using camera intrinsics, performs multi-frame fusion to reduce noise, and applies calibrated transforms to register sensor data to robot coordinates.

Spatial Clustering applies DBSCAN clustering (Chu et al., 2021) to group 3D points using ε-neighborhoods for object segmentation and generates hierarchical representations.

Robot Integration maintains multiple obstacle representations simultaneously (geometric primitives, point clouds, SDFs) with 5–15 Hz refresh rates during trajectory execution.

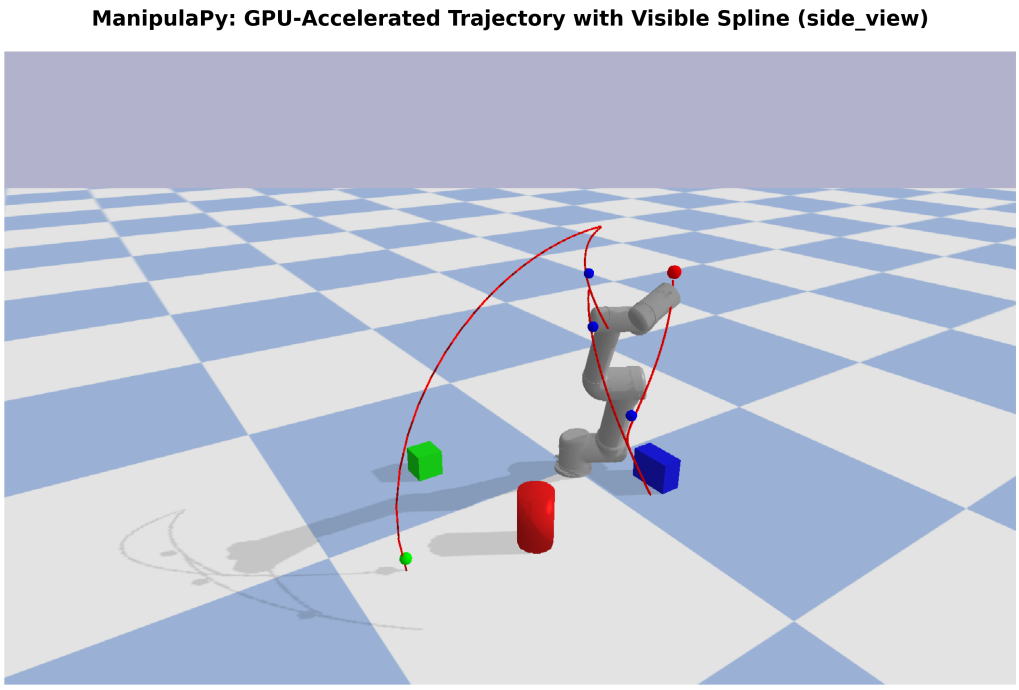

Figure 3: GPU-accelerated trajectory execution demonstration in PyBullet simulation. A 6-DOF robotic manipulator executes a complex trajectory while avoiding dynamic obstacles in real-time. The trajectory planning utilizes GPU acceleration for 40× speedup over CPU implementation, enabling 1 kHz control rates with real-time collision avoidance through potential field methods integrated with CUDA kernels.

Design Considerations

ManipulaPy provides tiered functionality that scales gracefully from CPU-only to GPU-accelerated operation while maintaining a balance between performance, flexibility, and precision.
This section outlines the computational tiers, vision dependencies, and design trade-offs that guided the framework's architecture.

CPU-Only Features

Core robotics modules include URDF processing, forward and inverse kinematics, Jacobian analysis, small-scale trajectory planning ($N < 1000$ points), basic control, and simulation setup. Performance characteristics include single trajectory generation in **~10–50 ms** for 6-DOF robots and real-time control frequencies up to **~100 Hz**, primarily limited by Python's Global Interpreter Lock (GIL).

GPU-Accelerated Features

High-performance modules support large-scale trajectory planning ($N > 1000$ points) achieving up to **a 40× speedup**, batch inverse-dynamics computation, real-time control above **1 kHz**, workspace analysis via Monte Carlo sampling, and GPU-accelerated potential fields.
Performance typically includes large trajectory generation in **~1–5 ms** for 6-DOF manipulators, enabling real-time planning and control at **1 kHz** rates.

Vision and Perception Dependencies

Camera and perception functionality rely on **OpenCV** for camera operations, **YOLO models** for object detection, and **graphics libraries** for visualization.
These dependencies support multi-camera setups, object detection, and spatial clustering but may require additional system-level libraries that are not always available in containerized environments.

Performance and Design Trade-offs

- **Performance Constraints**: Consumer GPUs (8 GB) limit trajectory planning to approximately **50,000 points**. GPU acceleration provides tangible benefits primarily when $N > 1000$ due to kernel-launch overhead. CPU-only control performance is constrained by Python's GIL, limiting real-time execution to ~100 Hz.

- **Integration Scope**: ManipulaPy operates independently of ROS middleware. While this ensures modularity, integration with ROS-based systems currently requires manual bridging.

- **Algorithmic Focus**: The current implementation focuses on potential-field and polynomial-interpolation methods. The framework is optimized for **serial kinematic chains**, and supporting parallel mechanisms would require architectural adaptation.

- **Intended Use**: Designed for **research and education**, ManipulaPy is not intended for industrial deployment. It does not include safety certifications or formal real-time verification mechanisms found in production systems.

Future Development

Planned enhancements include native ROS2 integration, advanced sampling-based planners, multi-robot support with GPU acceleration, direct hardware interfaces, safety monitoring with

Control Barrier Functions (Morton & Pavone, 2025), and enhanced GPU utilization techniques.

Acknowledgements

Work supported by Universität Duisburg-Essen and inspired by Modern Robotics (Lynch & Park, 2017), PyBullet (Coumans & Bai, 2019), Pinocchio (Carpentier et al., 2025), and Ultralytics YOLO (Jocher et al., 2023) projects.

References

Carpentier, J., Mansard, N., Valenza, F., Mirabel, J., Saurel, G., & Budhiraja, R. (2025). *Pinocchio - Efficient and versatile Rigid Body Dynamics algorithms* (Version 3.7.0). https://github.com/stack-of-tasks/pinocchio

Chitta, S., Sucan, I. A., & Cousins, S. (2012). MoveIt!: An introduction. *AI Matters*, *3*(4), 4–9. https://moveit.ros.org/

Chu, Y., Wang, L., & Zhang, M. (2021). An approach to boundary detection for 3D point clouds based on DBSCAN clustering. *Pattern Recognition*, *124*, 108431. https://doi.org/10.1016/j.patcog.2021.108431

Corke, P., & Haviland, J. (2021). Robotics, vision and control: Fundamental algorithms in MATLAB, Python and Julia. *Springer Tracts in Advanced Robotics*, *1*. https://petercorke.com/books/robotics-vision-control-python-the-practice-of-robotics-vision/

Coumans, E., & Bai, Y. (2019). *PyBullet: Physics simulation for games, robotics, and machine learning*. http://pybullet.org

Jocher, G., Qiu, J., & Chaurasia, A. (2023). *Ultralytics YOLO* (Version 8.0.0). https://github.com/ultralytics/ultralytics

Liang, H., Du, X., & Xiao, J. (2018). GPU-based high-performance robot dynamics computation. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2396–2402.

Lynch, K. M., & Park, F. C. (2017). *Modern robotics: Mechanics, planning, and control*. Cambridge University Press. http://modernrobotics.org

Morton, D., & Pavone, M. (2025). Safe, task-consistent manipulation with operational space control barrier functions. *arXiv Preprint arXiv:2503.06736*.

Okuta, R., Unno, Y., Nishino, D., Hido, S., & Loomis, C. (2017). CuPy: A NumPy-compatible library for NVIDIA GPU calculations. *Proceedings of Workshop on Machine Learning Systems (LearningSys) at NeurIPS*. https://cupy.dev

Sundaralingam, B., Hari, S. K. S., Fishman, A., Garrett, C., Van Wyk, K., Blukis, V., Millane, A., Oleynikova, H., Handa, A., Ramos, F., Ratliff, N., & Fox, D. (2023). CuRobo: Parallelized collision-free robot motion generation. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 8112–8119. https://doi.org/10.1109/ICRA48891.2023.10160765