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.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

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 model (Lynch & Park, 2017) (similar to Pinocchio (Carpentier et al., 2025) but with GPU acceleration), 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 **13× overall performance improvement** and **3600× faster inverse dynamics** on a 6-DOF UR5 compared to NumPy baseline. Performance claims are reproducible via benchmarks in `benchmarks/README.md`.

Statement of Need

Modern manipulation research requires tight integration of geometry, dynamics, perception, planning, and control—ideally within a single, real-time computational loop on GPU hardware. 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	Requires custom ROS nodes to bridge sensor data with planning; external plugins needed for real-time dynamics; no native GPU acceleration
Pinocchio (Carpentier et al., 2025)	High-performance PoE dynamics (C++)	CPU-only; separate perception and planning libraries must be manually synchronized; requires Python bindings for integration
CuRobo (Sundaralingam et al., 2023)	GPU-accelerated collision checking and trajectory optimization	Planning-focused; lacks perception pipeline and closed-loop control; requires external sensor processing
Python Robotics Toolbox (Corke & Haviland, 2021)	Educational algorithms with clear APIs	CPU-only implementation; users must implement their own simulators, controllers, and sensor processing
PyRoKi (Kim* et al., 2025)	JAX-accelerated kinematics	Early development stage; limited dynamics and no perception support
CBFPy (Morton & Pavone, 2025)	Control barrier functions with JAX	Specialized for safety-critical control; requires manual integration with perception and planning

These integration challenges manifest as: - **Sensor-planner gaps**: Converting camera data to collision geometries requires custom OpenCV → ROS → MoveIt pipelines - **Dynamics-control mismatches**: Real-time controllers need consistent mass matrices, but most libraries compute dynamics separately from control loops
- **GPU memory fragmentation**: Transferring data between CPU planners and GPU dynamics creates performance bottlenecks - **Synchronization complexity**: Keeping sensors, planners, and controllers temporally aligned requires careful threading and message passing

ManipulaPy eliminates these integration burdens through a unified Python API that maintains data consistency across the entire manipulation pipeline:

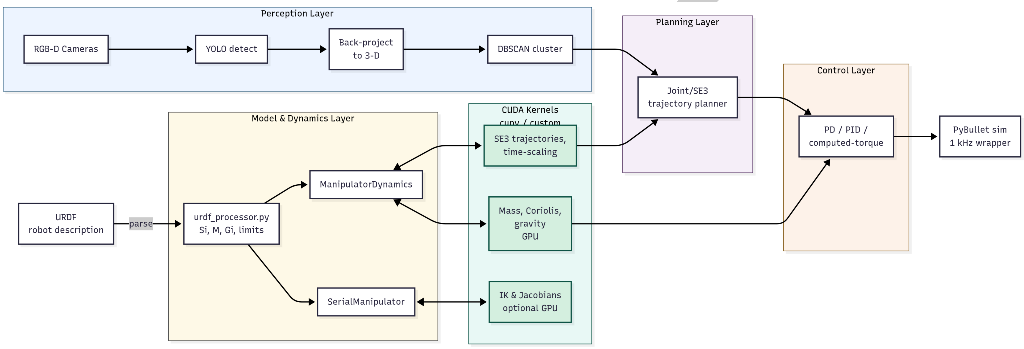

Figure 1: ManipulaPy manipulation pipeline architecture showing unified data flow from sensors through planning to control, with GPU acceleration throughout.

Core design principles:

1. **Unified data structures**
   All components share consistent representations (PoE screws, SE(3) transforms, GPU tensors)
2. **GPU-first architecture**
   Trajectories, dynamics, and perception processing execute on GPU without CPU round-trips
3. **Temporal synchronization**
   Built-in 1 kHz control loop keeps sensors, planners, and actuators phase-locked
4. **Extensible perception**
   Multiple obstacle representations (primitives, point clouds, SDFs) supported simultaneously

Performance benchmarks demonstrating the claimed **13× overall speedup** are reproducible via `Benchmarks/performance_benchmark.py` (requires CUDA-capable GPU).

The Journal of Open Source Software

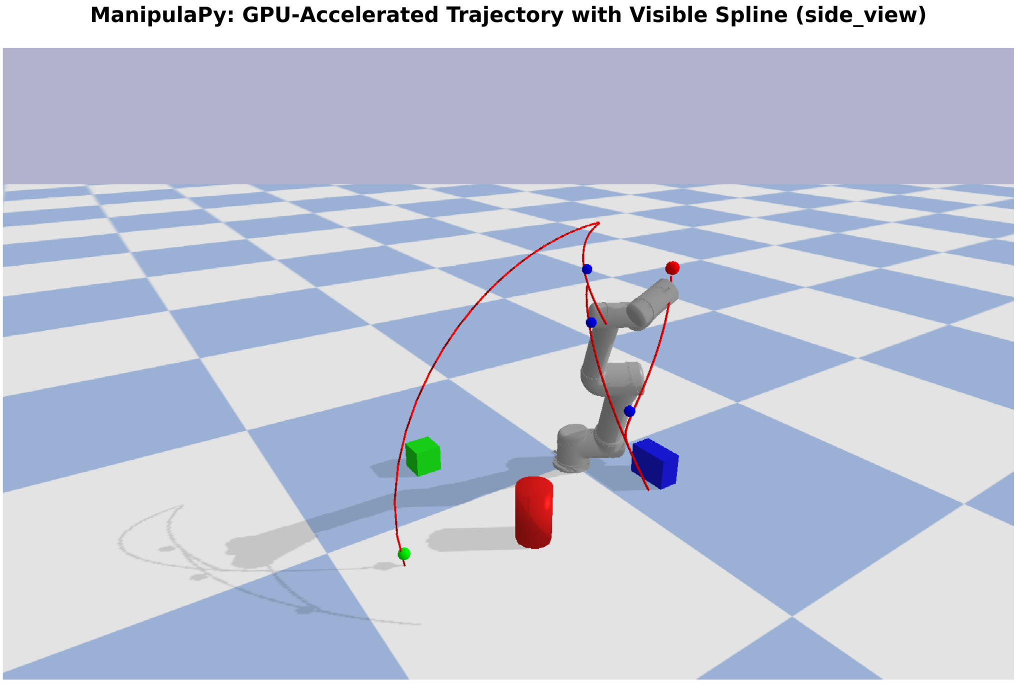

Figure 2: ManipulaPy PyBullet simulation showing GPU-accelerated trajectory execution with real-time collision avoidance. The robot smoothly navigates around dynamically detected obstacles while maintaining 1 kHz control rates.

Library Architecture

ManipulaPy's architecture centers on a **unified manipulation pipeline** that maintains data consistency from sensor input to motor commands. Rather than loosely coupled modules, the system implements a coherent data flow where each component builds upon shared representations:

Core Pipeline Components:

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

Kinematics and Dynamics implement GPU-accelerated forward/inverse kinematics, Jacobian computation, and Newton-Euler dynamics. Custom CUDA kernels optimize critical operations for 6-DOF manipulators, enabling real-time performance at 1 kHz control rates.

Perception Integration processes sensor data through a multi-stage pipeline supporting diverse input modalities. The `vision.py` module handles low-level camera operations (stereo rectification, calibration, image capture), while `perception.py` provides high-level semantic processing (object detection, clustering, obstacle representation). This separation allows users to plug in custom sensors while maintaining consistent 3D obstacle representations.

Motion Planning generates collision-free trajectories using GPU-accelerated time-scaling functions. The system supports both joint-space and Cartesian-space planning with real-time obstacle avoidance based on vision feedback.

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

Simulation Framework provides PyBullet integration with synchronized camera rendering, physics simulation, and control execution. This enables seamless transition from simulation to real hardware.

Key Architectural Decisions:

- **Shared GPU Memory**: All components operate on GPU tensors, eliminating CPU-GPU transfer bottlenecks
- **Consistent Time Base**: 1 kHz control loop synchronizes all components

- **Modular Perception**: Multiple obstacle representations coexist (geometric primitives, point clouds, signed distance fields)
- **Extensible Design**: New sensors, planners, and controllers integrate through well-defined interfaces

Vision and Perception Pipeline

ManipulaPy's perception system addresses the challenge of converting raw sensor data into actionable robot knowledge through a five-stage pipeline that supports multiple obstacle representations:

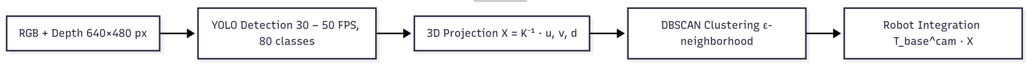

Figure 3: ManipulaPy perception pipeline showing sensor fusion, object detection, 3D integration, spatial clustering, and robot integration stages.

Stage 1: Sensor Fusion - **Stereo cameras**: RGB+depth via OpenCV rectification and SGBM disparity computation - **RGB-D sensors**: Direct depth integration from RealSense, Kinect, or similar devices
- **Point cloud input**: Direct processing of PCL/Open3D data structures - **Multi-modal fusion**: Temporal alignment and calibration across sensor types

Stage 2: Object Detection
- **YOLO v8 integration** (Jocher et al., 2023): Real-time 2D bounding box detection at 30-50 FPS - **Custom detector support**: Pluggable interface for domain-specific models - **Geometric primitive detection**: Built-in recognition of spheres, boxes, cylinders from URDF specifications

Stage 3: 3D Integration - **Depth projection**: Camera intrinsics K transform pixel coordinates (u, v) to 3D world positions
- **Multi-frame fusion**: Temporal averaging reduces sensor noise and handles partial occlusions - **Coordinate transformation**: Calibrated transforms T_{base}^{cam} register sensor data to robot coordinates

Stage 4: Spatial Clustering - **DBSCAN clustering** (Chu et al., 2021): Groups 3D points using ϵ-neighborhoods for object segmentation - **Hierarchical representations**: Octree/Octomap structures for large-scale environment mapping - **Implicit surfaces**: Signed distance field generation for smooth collision checking

Stage 5: Robot Integration - **Multi-representation support**: Simultaneously maintains geometric primitives, point clouds, and SDFs - **Dynamic obstacle updates**: 5-15 Hz refresh rate during trajectory execution - **Collision geometry generation**: Automatic conversion to convex hulls, bounding spheres, or custom shapes

Supported Obstacle Representations:

Unlike manipulation frameworks that handle only geometric primitives or require external mapping servers, ManipulaPy natively supports: - **Geometric primitives**: Fast collision checking with spheres, boxes, cylinders - **Unstructured point clouds**: Direct processing without conversion to meshes - **Signed distance fields**: Smooth gradients for optimization-based planning - **Octrees/Octomaps**: Hierarchical voxel representation for large environments - **Hybrid representations**: Multiple formats coexist for different planning algorithms

This flexibility allows researchers to choose optimal representations for their specific applications while maintaining real-time performance through GPU acceleration.

Theory and Implementation

Product-of-Exponentials Kinematics

Like Pinocchio (Carpentier et al., 2025), ManipulaPy adopts the Product-of-Exponentials formulation for robot kinematics. However, while Pinocchio achieves performance through highly optimized C++ implementations, ManipulaPy provides GPU acceleration across the entire manipulation pipeline:

$$T(\theta) = e^{S_1\theta_1} \cdots e^{S_n\theta_n} M$$

where each screw axis $S_i \in \mathbb{R}^6$ encodes joint motion and $M \in SE(3)$ represents the home configuration. The space-frame Jacobian becomes:

$$J(\theta) = \left[\mathrm{Ad}_{T_1} S_1, \ldots, S_n\right]$$

GPU-Accelerated Dynamics

Custom CUDA kernels parallelize the recursive Newton-Euler algorithm for the fundamental dynamics equation:

$$\tau = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta)$$

The mass matrix $M(\theta) = \sum_{i=1}^{n} \mathrm{Ad}_{T_i}^T G_i \mathrm{Ad}_{T_i}$ computation is optimized for 256-thread blocks, achieving up to **3600× speedup for inverse dynamics** and **8× speedup for trajectory generation** on 6-DOF manipulators compared to NumPy implementations.

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

Kim*, C. M., Yi*, B., Choi, H., Ma, Y., Goldberg, K., & Kanazawa, A. (2025). *PyRoki: A modular toolkit for robot kinematic optimization*. https://arxiv.org/abs/2505.03728

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