Dual Robot Manipulation Sandbox: A Goal-Conditioned Platform for Imitation and Reinforcement Learning

Cheng Yin
The School of Mechanical Science and Engineering
Huazhong University of Science and Technology, China

October 30, 2025

Abstract

This technical report documents the design, implementation, and intended research workflow of the *Dual Robot Manipulation Sandbox*[1]. The project offers a reproducible bridge between physics-based simulation, heuristic demonstration harvesting, imitation learning, and reinforcement learning for dual- and single-arm UR5 manipulators. We provide an overview of the software architecture, the environment abstractions that enable goal-conditioned control, and the data and learning pipelines that turn those abstractions into working policies. Particular attention is paid to the integration of Hindsight Experience Replay (HER), weighted goal-conditioned supervision, and adversarial imitation—three algorithmic pillars that make the sandbox a compelling venue for rapid experimentation. Beyond code descriptions, this report narrates the evolution of the system into an integrated story that can guide future extensions and deployment onto physical hardware.

1 Introduction

Manipulation research increasingly depends on modular software stacks that lower the cost of iterating over environment variants, collecting demonstrations, and training agents. The `Dual Robot Manipulation Sandbox` was conceived to meet this need for goal-conditioned UR5 tasks in PyBullet, bringing three complementary environments (dual-arm pick-and-place, single-arm pick-and-place, and single-arm reach) under a unified interface while bundling imitation and reinforcement learning baselines for each task [1]. The repository foregrounds reproducibility: environment definitions, data collection policies, training harnesses, and evaluation scripts all share the same observation and action conventions, allowing researchers to switch between simulation-first iteration and real-robot deployment with minimal friction.

This report captures the latest state of the sandbox, telling the story of how its components interact. We begin with the environment abstractions that keep dual- and single-arm variants aligned, move on to the robot embodiment layer that mediates between high-level commands and joint-level actuation, and then turn to the demonstration and learning pipelines that populate the sandbox with skill.

[1] Code available at https://github.com/wadeKeith/Dual-Robot-Manipulation-Sandbox-A-Goal-Conditioned-Platform-for-Im

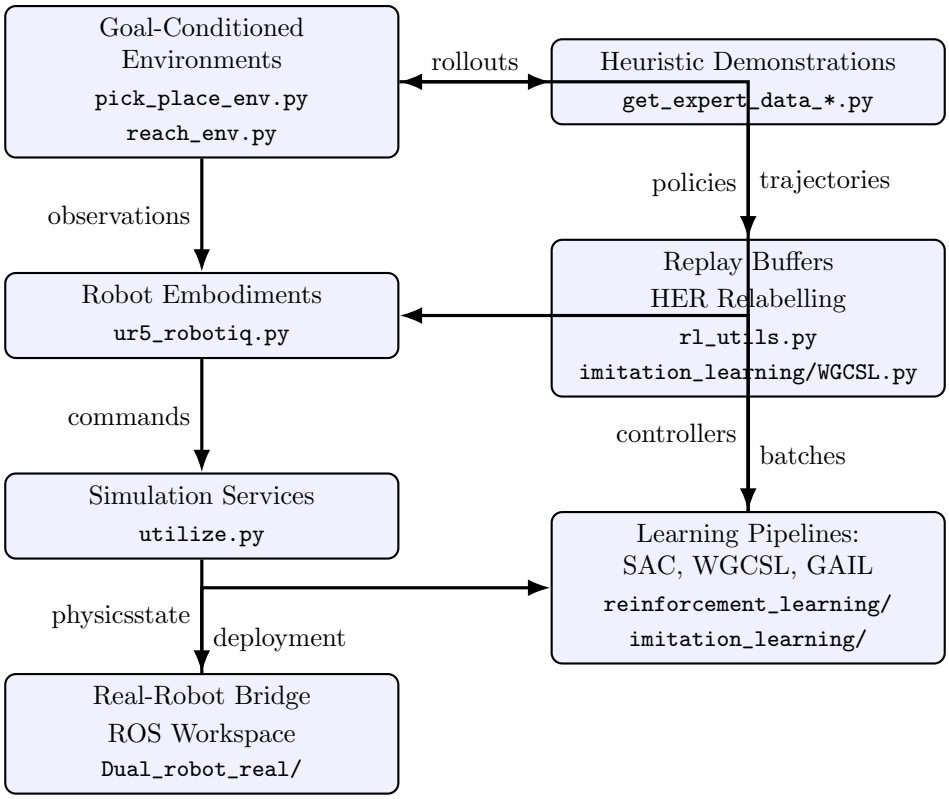

Figure 1: Information flow in the Dual Robot Manipulation Sandbox. Simulation-facing modules on the left provide goal-conditioned rollouts that feed the demonstration and learning pipelines on the right. Learned policies close the loop by issuing commands back to both simulated and real robot embodiments.

2 Software Architecture

At the heart of the sandbox lies a set of Gymnasium-style environments built around a goal-conditioned observation dictionary. A mixin class flattens these dictionaries into arrays for algorithm compatibility while preserving the semantic grouping of observation, achieved goal, and desired goal components [2]. Simulation services, including PyBullet connection management, camera control, and general utilities such as distance computations, are centralised in a reusable helper module [3]. Figure 1 illustrates the resulting flow of information between layers.

3 Goal-Conditioned Task Suite

3.1 Dual-Arm Pick-and-Place

The legacy dual-arm task instantiates two UR5 arms equipped with Robotiq 85 grippers and exposes either joint-space or end-effector control. A training episode begins with random sampling of both the initial object pose and the desired goal, subject to a configurable distance threshold that prevents trivial resets [4]. The environment normalises each observation channel relative to its action bounds, composes a goal-conditioned dictionary, and exposes both dense and sparse success metrics through reward and termination logic [5, 6].

A key feature is the ability to switch between joint and Cartesian control without modifying

downstream code. The environment computes inverse kinematics for each arm when operating in end-effector mode and enforces identical termination and truncation rules regardless of control scheme [7, 8].

3.2 Single-Arm UR5e Pick-and-Place

The single-arm variant mirrors the dual-arm API while specialising to a UR5e equipped with a Robotiq 140 gripper. Optional toggles disable the gripper during training for policies that rely on externally scripted grasps. The environment samples tabletop goals within a narrow workspace, constructs observation bounds consistent with the embodiment's controllable degrees of freedom, and provides shared reward logic with the dual-arm task to ease cross-transfer [9]. End-effector control clamps motion within a safe vertical range to prevent physically implausible commands during optimisation [10].

3.3 UR5e Reach Task

The reach task reuses the single-arm embodiment but recasts the goal as the average position of the gripper finger pads. Goals are sampled by perturbing a canonical handle position, with optional visualisation via debug points for qualitative inspection [11]. Observations concatenate historical joint states to capture motion context, and HER-friendly rewards penalise distances above the threshold [12].

4 Robot Embodiments

The environment wrappers delegate low-level actuation to dedicated robot classes. The dual-arm wrapper loads a composite URDF, sets up mimic joint constraints for the grippers, and provides helper functions for computing inverse kinematics and constructing observation vectors rich in end-effector pose and velocity information [13, 15]. Gripper actuation is translated into target finger separations through trigonometric mapping from desired opening lengths to joint angles, ensuring symmetric closure [14].

The single-arm wrapper follows the same philosophy but emphasises configurability: the tool-centre-point link can be overridden, gripper mimics are generalised through gear constraints, and joint bounds are enforced whenever joint-space control is requested [16].

5 Simulation Services

PyBullet clients are instantiated with consistent physics settings, regardless of GUI usage, ensuring reproducible dynamics across headless and interactive runs [3]. Camera utilities provide projective transformations for rendering RGB-D observations or mapping depth pixels back to world coordinates, supporting vision-based extensions without intruding on the core control loops [17].

6 Demonstration Harvesting

Heuristic scripts record expert trajectories by steering the gripper towards the object or goal depending on the state of the manipulation sequence. The dual-arm collector decides which hand should engage based on instantaneous distance and modulates gripper closing schedules to secure the object before lift-off [18]. Each trajectory is inserted into a replay buffer that supports optional

HER relabelling, ensuring compatibility with both imitation and reinforcement learning consumers [19].

7 Learning Pipelines

7.1 Reinforcement Learning

Soft Actor-Critic (SAC) with HER forms the primary reinforcement learning baseline. The training script seeds all stochastic components, constructs goal-augmented state vectors by concatenating observation, desired goal, and achieved goal channels, and alternates between trajectory collection and buffered updates [20]. Learning rate decay, adaptive entropy targeting, and periodic evaluation checkpoints are built in to support long training schedules. Replay buffers capture full trajectories and perform HER sampling with adjustable ratios before each update [21].

Utility functions shared across reinforcement learning pipelines provide generic replay buffers, on-policy training loops, and advantage estimators for PPO variants, highlighting the sandbox's emphasis on code reuse [22].

7.2 Imitation Learning

Weighted Goal-Conditioned Supervised Learning (WGCSL) serves as the supervised baseline, combining policy and value estimators with HER-augmented sampling and soft target updates [23]. Adversarial imitation is implemented via a GAIL harness that couples a PPO policy with a discriminator trained on expert rollouts serialized from the heuristic collector [24]. The training loop refreshes the discriminator with on-policy data each episode and evaluates checkpoints against held-out goals to monitor policy recovery.

8 Real-Robot Bridge

While simulation remains the primary iteration environment, the repository includes a ROS Noetic catkin workspace for interfacing with a physical dual-UR5 platform, Intel RealSense cameras, and Robotiq grippers. The codebase is therefore positioned to support a sim-to-real workflow, with simulation and learning scripts remaining decoupled from ROS dependencies [25].

9 Research Narrative

Developing the sandbox meant orchestrating multiple moving parts so that researchers could focus on algorithmic ideas rather than glue code. The journey began with harmonising observation spaces across tasks—a pragmatic decision that later enabled HER to function identically in both imitation and reinforcement learning pipelines. The dual-arm environment, initially a bespoke script, was refactored into a class that shares a mixin with the single-arm tasks, reducing bugs when expanding the observation vector.

Next came the recognition that reproducible demonstrations were essential. The heuristic collector, while simple, embodies years of intuition about how to stage gripper motions and when to commit to a grasp. Its trajectories not only bootstrap imitation learners but also offer dense coverage for HER, effectively transforming sparse rewards into learning signals without manual reward shaping.

Finally, the reinforcement and imitation learning stacks were designed to tell complementary stories: SAC+HER emphasises exploration under sparse rewards, WGCSL showcases the power of reweighting demonstrations, and GAIL closes the loop by adversarially aligning agent and expert behaviour. Together, these pipelines create a virtuous cycle of data collection, policy refinement, and evaluation that can be extended to new tasks with minimal friction.

10 Conclusion

The `Dual Robot Manipulation Sandbox` delivers a cohesive platform for studying goal-conditioned manipulation across simulation and physical embodiments. By integrating environment design, robot abstraction, heuristic expertise, and learning algorithms, the project offers an extensible foundation for future research. Ongoing work aims to tighten the sim-to-real gap, incorporate vision-based policies through the existing camera utilities, and broaden the task suite to include collaborative manipulation scenarios.

Acknowledgements

The author thanks the PyBullet, Gymnasium, and PyTorch communities for the tooling that made this sandbox feasible, as well as colleagues who provided feedback on the dual-arm platform.

References

[1] README, "Dual Robot Manipulation Sandbox," commit version accessed October 2024.

[2] `pick_place_env.py`, lines 10–79.

[3] `utilize.py`, lines 11–142.

[4] `pick_place_env.py`, lines 109–158.

[5] `pick_place_env.py`, lines 45–82.

[6] `pick_place_env.py`, lines 159–205.

[7] `pick_place_env.py`, lines 214–245.

[8] `pick_place_env.py`, lines 186–198.

[9] `pick_place_env.py`, lines 262–404.

[10] `pick_place_env.py`, lines 324–401.

[11] `reach_env.py`, lines 73–121.

[12] `reach_env.py`, lines 124–180.

[13] `ur5_robotiq.py`, lines 7–190.

[14] `ur5_robotiq.py`, lines 88–148.

[15] `ur5_robotiq.py`, lines 300–340.

[16] `ur5_robotiq.py`, lines 343–624.

[17] `utilize.py`, lines 88–142.

[18] `get_expert_data_pick_place.py`, lines 44–129.

[19] `imitation_learning/WGCSL.py`, lines 25–96.

[20] `reinforcement_learning/train_sac.py`, lines 31–120.

[21] `reinforcement_learning/train_sac.py`, lines 121–199.

[22] `rl_utils.py`, lines 1–90.

[23] `imitation_learning/WGCSL.py`, lines 99–200.

[24] `imitation_learning/train_GAIL.py`, lines 1–188.

[25] README, lines 30–133.