FurnitureBench: Reproducible Real-World Benchmark for Long-Horizon Complex Manipulation

Minho Heo*
KAIST

Youngwoon Lee*
UC Berkeley

Doohyun Lee
KAIST

Joseph J. Lim
KAIST

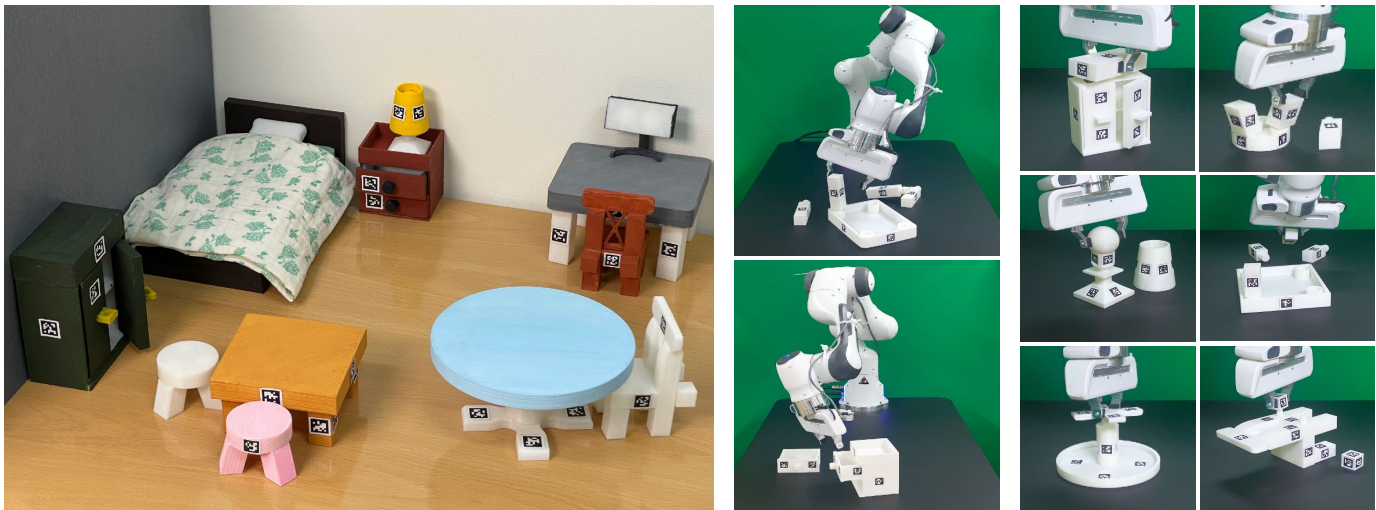https://clvrai.com/furniture-bench

Fig. 1: **FurnitureBench: reproducible real-world furniture assembly benchmark.** Benchmarking furniture assembly poses to address many robotic manipulation challenges: long-horizon planning, dexterous control, and visual perception. FurnitureBench is designed to be *easy-to-reproduce* and *easy-to-use* with the 3D printable furniture models, robot control software stack, environment setup guide, and large demonstration data. (Left) A decorated room in the real world with furniture models our robot assembled. (Right) A suite of 8 furniture models in our benchmark.

Abstract—**Reinforcement learning (RL), imitation learning (IL), and task and motion planning (TAMP) have demonstrated impressive performance across various robotic manipulation tasks. However, these approaches have been limited to learning simple behaviors in current real-world manipulation benchmarks, such as pushing or pick-and-place. To enable more complex, long-horizon behaviors of an autonomous robot, we propose to focus on real-world furniture assembly, a complex, long-horizon robot manipulation task that requires addressing many current robotic manipulation challenges to solve. We present FurnitureBench, a reproducible real-world furniture assembly benchmark aimed at providing a low barrier for entry and being easily reproducible, so that researchers across the world can reliably test their algorithms and compare them against prior work. For ease of use, we provide 200+ hours of pre-collected data (5000+ demonstrations), 3D printable furniture models, a robotic environment setup guide, and systematic task initialization. Furthermore, we provide FurnitureSim, a fast and realistic simulator of FurnitureBench. We benchmark the performance of offline RL and IL algorithms on our assembly tasks and demonstrate the need to improve such algorithms to be able to solve our tasks in the real world, providing ample opportunities for future research.**

I. INTRODUCTION

Everyday physical tasks we want to automate with robots, such as doing laundry, tidying messy rooms, cooking meals, and assembling furniture, require a robot to understand environments, plan, and execute complex long-horizon behaviors. These activities have long time spans, consist of diverse semantically composable behaviors, and require dexterous, precise manipulation skills. How can we enable autonomous robots to perform such complex, long-horizon tasks?

Recently, deep reinforcement learning (RL), imitation learning (IL), and task and motion planning (TAMP) have presented impressive results in robotic manipulation [33, 52, 2, 22, 19, 11]. Yet, many of these works are hardly reproducible due to the absence of standardized robots, environments, and software used for experiments. There have been efforts on improving reproducibility in robotic manipulation benchmarks [29, 58, 1, 7, 15, 17, 13, 5]; however, they are still limited to simple short-horizon tasks, such as pick-and-place.

To further robotics research toward solving people's everyday tasks, it is crucial to tackle challenges in more complex and longer-horizon tasks. This calls for reproducible benchmarks

(a) Robot system

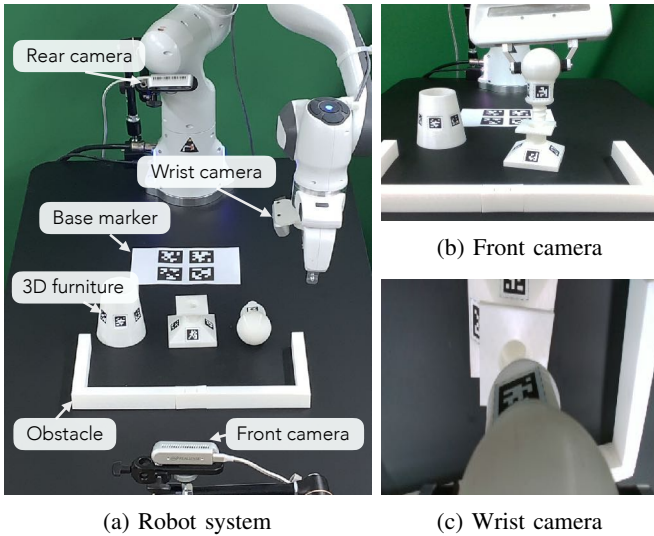(b) Front camera

(c) Wrist camera

Fig. 2: **Real-world furniture assembly environment.** Our reproducible robot system (a) and visual observations from the front-view camera (b) and wrist camera (c).

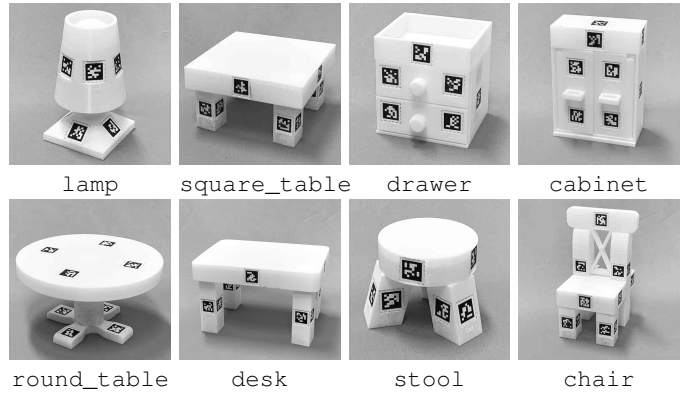lamp square_table drawer cabinet

round_table desk stool chair

Fig. 3: **3D printed furniture models.** Each furniture is designed inspired by IKEA furniture. Due to the limitations imposed by using a single robotic arm, we modify some furniture pieces feasible to be assembled with one hand.

of long-horizon complex robotic manipulation tasks. To this end, we propose to focus on furniture assembly as the next milestone for complex, long-horizon robotic manipulation, and present FurnitureBench, a reproducible real-world furniture assembly benchmark, as shown in Figures 1 to 3.

Furniture assembly is a proper task suite to benchmark a difficult, long-horizon manipulation task through which many challenges in robotic manipulation must be addressed to solve. It has a hierarchical task structure, requiring reasoning over long horizons (e.g., deciding in which order and how to assemble various furniture pieces). In addition, connecting different furniture pieces together requires complex path planning and dexterous manipulation skills (e.g., robust grasping, accurate alignment of attachment points, and deliberate force control for inserting and screwing).

FurnitureBench is a *reproducible* and *easy-to-use* benchmark that can evaluate such challenges in furniture assembly. It aims to enable any lab to easily set up a standardized robotic environment and evaluate the capability of algorithms on realistic complex long-horizon tasks. We believe that FurnitureBench will help robot manipulation researchers identify challenges in solving complex long-horizon tasks, easily compare the performance of their approaches over prior work, and eventually solve diverse complex long-horizon real-world tasks.

The main contributions of this paper are as follows:

- We introduce FurnitureBench, a **real-world** furniture assembly benchmark, which allows robotics researchers to investigate RL, IL, and TAMP algorithms on a *realistic and complex* task beyond pick-and-place (Sections III and IV, Figure 2).
- We ensure **reproducibility** by incorporating 3D printable furniture pieces (Figure 3) and providing comprehensive environment setup and evaluation guides (Section IV-A).
- We collected **200+ hours of large teleoperation demon-**

strations that reduce barriers to real-world experiments on offline RL and IL algorithms (Section IV-B).
- We evaluate diverse subtasks required for furniture assembly, e.g., grasping, inserting, and screwing skills, in our **single-skill benchmark** (Section VI-A), which identifies challenges in learning "inserting" and "screwing" skills.
- In the **full-assembly benchmark**, we evaluate the full furniture assembly tasks, where IL and offline RL methods achieve up to 2 out of 12 subtasks on average (Section VI-B).
- We develop FurnitureSim, a **simulator** of our real-world benchmark, which can boost fast iteration of experiments for new methods (Sections IV-C and VI-C).

II. RELATED WORK

Simulated manipulation benchmarks. Deep reinforcement learning (RL) has made rapid progress with the advent of standardized, simulated benchmarks, such as Atari [6] and continuous control [10, 55] benchmarks. In robotic manipulation, most existing simulated environments have been focused on tasks like picking and placing [10, 30, 21, 60, 59], in-hand manipulation [45, 49], door opening [56], peg insertion [14, 57], and screwing [42]. However, these tasks, which consist of lifting, stacking, and picking and placing, are limited to short-horizon primitive skills.

Composite manipulation tasks, such as block stacking [16], ball serving [31], kitchen tasks [18, 47], and table-top manipulation [40], have been introduced but are still limited to small variations in shapes and physical properties of objects. BEHAVIOR [51, 34] and Habitat [39, 54] present simulated benchmarks including diverse long-horizon mobile manipulation tasks while focusing on high-level planning and abstracting complex low-level control problems. In contrast, the IKEA furniture assembly environment [32] simulates a complex manipulation task, *furniture assembly*, to evaluate long-term planning and generalization of learned skills across various shapes of objects. Although simulated benchmarks are easily accessible and useful for quickly verifying ideas, they do

		FurnitureBench (Ours)	NIST Assembly Task Boards [25]	RGB-Stacking [29]	REPLAB [58]	DGBench [12]	GRASPA 1.0 [7]	ROBEL D'Claw [1]	In-Hand Manipulation [15]	Bimanual Cloth Manipulation [17]	Bimanual Manipulation [13]	Real Robot Challenge [3, 4]
Reproducibility	Robot and workspace	✓	✗	✓	✓	✓	✗	✓	✗	✗	✗	✓
	Availability of objects	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Long horizon	Task horizon[1]	2300 steps, 230 sec	-	400 steps	-	-	-	80 steps, 8 sec	2 sec	20-140 sec	60 sec	120 sec
Complexity	DoF	6D	-	4D	6D	3D	-	6D	-	-	6D	6D
	Skills[2]	PnP, P, F, I, S	PnP, I, S	PnP, F	G, R	DG	G, R	PO, S	IR	PnP, F	H, I, B	P, L, RO
Diversity	Number of tasks	8	4	2	2	2	3	3	10	3	2	2
	Different levels of evaluation per task	3	1	5	1	1	3	2-3	3	4-5	1	4
	Object categories[3]	R, A	R, D	R	R, A, D	R	R	R	R	D	R, D	R
Demonstrations	Expert teleoperation	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Dataset size	200hrs / 5k ep.	-	-	92k ep.	-	-	-	-	-	1200 ep.	10k ep.
Simulation	Simulator	✓	✗	✓	✗	✗	✗	✓	✓	✗	✓	✓

[1] Average or maximum episode lengths or expert execution time [2] PnP: Pick-and-place / P: Push / F: Flip / I: Insert / S: Screw / G: Grasp / R: Reach / DG: Dynamic grasp / PO: Pose / IR: In-hand re-orientation / H: Hold / B: Bend / L: Lift / RO: Rotate [3] R: Rigid objects, A: Articulated objects, D: Deformable objects

TABLE I: **Comparison of real-world robotic manipulation benchmarks.** Our furniture assembly benchmark is reproducible, testing a complex, long-horizon task, and providing a large dataset and a simulator for easy benchmarking.

not ensure that algorithms can directly work on real robots due to the complexity and stochastic nature of the real environment.

Real-world manipulation benchmarks. Real-world robot learning involves many challenges that are not considered in simulated environments, such as collision, noisy and delayed sensory inputs, automatic environment resetting, and providing reward signals. As summarized in Table I, recent robotic manipulation benchmarks have introduced reproducible benchmarking environments with an affordable robot and easy-to-make cage [58, 1], with 3D printed objects [29], and by evaluating in cloud [3, 4]. However, these works focus on much simpler and short-horizon tasks (e.g., stacking blocks, re-orienting objects) than furniture assembly.

Robotic assembly benchmarks. Robotic furniture assembly has been studied in instrumented and constrained settings [43, 27, 53]. Yet, these works are not suitable for benchmarking as they use their own furniture models, robots, and software. NIST Assembly Task Board [25] provides a suite of complex manufacturing tasks and evaluation metrics, which is most similar to ours. However, it mainly focuses on providing task specifications and lacks a standard experimental setup, namely a robot, hardware, observation, and action. This motivates us to implement a *standardized* real-world manipulation benchmark consisting of long-horizon complex tasks with careful designs for reproducibility and usability.

III. FURNITURE ASSEMBLY: LONG-HORIZON COMPLEX MANIPULATION TASK

To advance robotics research from simple and artificial tasks to complex and realistic tasks, we need a benchmark that helps researchers identify challenges in solving such complex, long-horizon real-world tasks, and allows them to easily evaluate and compare various approaches. In this paper, we propose to benchmark the task of furniture assembly, which is a meaningful, long-horizon, complex, real-world task.

Even for humans, furniture assembly is not a simple task. We need to understand how pieces are assembled into the final configuration and make a plan to assemble them in the right order. Furniture assembly also involves many complex manipulation skills, such as the accurate alignment of two parts and sophisticated control to screw them together. Therefore, furniture assembly is a comprehensive robotic manipulation task [43, 27, 53] requiring high-level long-term planning, sophisticated control, and reliable 3D perception, making it a suitable benchmark for robotic manipulation research.

Long-horizon tasks. The furniture assembly task is long-horizon and hierarchical, as illustrated in Figure 4. Furniture assembly can be accomplished by repeating 1-step assembly as follows: (1) selecting two parts to be assembled, (2) grasping one part[1], (3) moving the part towards the other, (4) aligning the attachable points (e.g., a screw and hole), and (5) firmly attaching them via screwing or insertion, until all parts are assembled. Thus, furniture assembly has a much longer horizon on average (60-230 sec, 600-2300 low-level steps) than prior manipulation benchmarks (e.g., 15-120 sec for Roboturk [38]).

Diverse dexterous skills. This seemingly simple procedure requires diverse, challenging manipulation skills, as illustrated in Figure 4. First, **grasping** a furniture piece for assembly requires a specific grasping pose for the following sub-tasks. Attaching two parts sometimes requires re-orienting the furniture pieces both in hand and on the table through picking-and-placing or pushing. As the task proceeds, maneuvering an arm and manipulating furniture parts need more thorough path planning to **avoid collisions** with irrelevant parts. For example, a robot needs to avoid colliding with already attached table legs; otherwise, it can move irrelevant furniture pieces around. Lastly, **inserting** and **screwing**, by themselves, are challenging manipulation skills. Inserting and screwing skills require precise alignment between a screw and a hole, then repeated rotation of the screw while gently pressing it. Moreover, due to the limited wrist joint rotation, screwing (i.e., rotating a part for $540°$) requires at least five $90°$-screwing and re-grasping motions.

---

[1] This benchmark focuses on single-arm manipulation, which is the most common robot arm setup in research labs. Using one robotic arm results in the limited dexterity to handle multiple objects simultaneously or a large object. We overcome this issue by adding an obstacle fixed on the table (see Figure 2), which can be used to hold an object, and slightly modifying 3D furniture models suited for one-hand manipulation. We leave extending this work to multi-arm and mobile manipulation for future work.
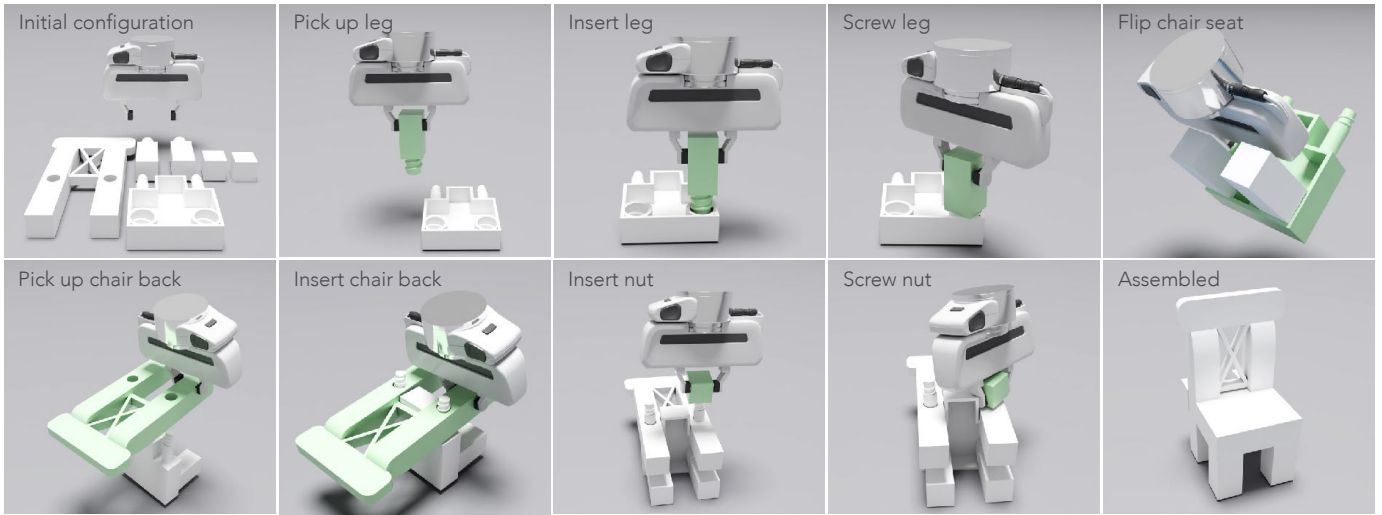
Fig. 4: **Overview of a furniture assembly task, `chair`.** Furniture assembly requires many dexterous skills, including grasping, re-orienting, inserting, and screwing. The part in contact with the robot gripper is highlighted in green and some repetitive steps, e.g., screwing the second leg and nut, are omitted.

IV. FURNITUREBENCH: REPRODUCIBLE REAL-WORLD FURNITURE ASSEMBLY BENCHMARK

While bringing real-world complexity into the robot manipulation community, our benchmark is also designed to be reproducible and easy-to-use. In this section, we first describe our **reproducible** real-robot experimental setup, including the robotic system design and furniture models. For **ease of use**, we provide a plug-and-play robot control stack and a large teleoperated demonstration dataset. We then introduce a simulated environment of our real-world benchmark for easy and fast verification of various algorithms.

A. Reproducible Real-World Benchmark Environment

Reproducible system design. To make the real-robot environment easy to reproduce, we opt for widely used products across the world and 3D-printing objects. As shown in Figure 2a, our real-world system mainly consists of one 7-DoF Franka Emika Panda robot arm with a default parallel gripper and three Intel RealSense D435 RGB-D cameras, which are widely used in robotics community. As a workspace, we propose to use the black IKEA TOMMARYD table, which is available in a wide range of countries, and attach 3D-printed obstacles to the table. The green photography backdrop is used to provide a consistent background and a single light is used to vary lighting conditions in color temperature (4600 K-6000 K) and brightness ($\leq$4000 lm). We provide detailed system setup instructions (Section H) and software tools, which allow a user to build a new environment with nearly the same configuration based on furniture and table poses estimated with AprilTag [44].

Reproducible furniture models. For reproducibility, we use 3D-printed furniture models (Figure 3); thus, any lab can make the same furniture parts by simply downloading and 3D-printing our model files.[2] Our benchmark features 8 different

[2]Our furniture models require a 3D printer with the capacity larger than 25 cm×25 cm×15 cm and printing one furniture model takes up to 24 hours.
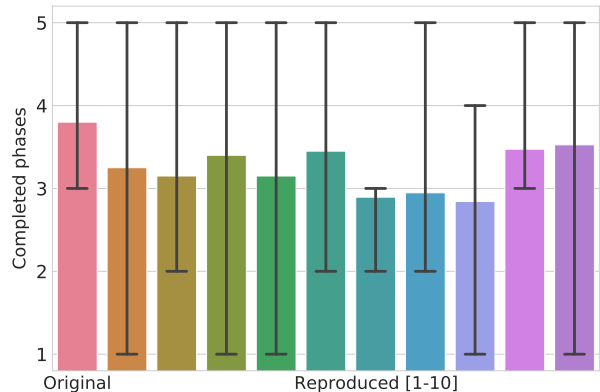

Fig. 5: **Reproducibility analysis.** We evaluate the number of completed subtasks (i.e., phases) of an IQL policy on the `one_leg` assembly task in 10 newly built environments under the low randomness level. The new environments achieve on average 84% of the performance in the original environment, which ensures the reproducibility of our benchmark.

furniture models, each modeled after an existing piece of IKEA furniture, as shown in Figure 17. These 8 furniture models are selected to introduce their own challenges and interactions (e.g., `drawer` requires precise alignment between the rail and drawer; already assembled legs in `desk` can interrupt assembly of other legs). More details about the furniture models and their assembly procedures are described in Section F.

Reproducibility analysis. To verify the reproducibility of our benchmark, we conduct the reproducibility test by asking 10 participants to set up an environment from scratch following the instruction in Section H. Each participant independently builds the system, and this takes about 3 hours. We then evaluate a fully-trained IQL-R3M policy on the `one_leg` task in these 10 new environments. Figure 5 shows the average number of completed phases (i.e., subtasks) over the 10 new environments

TABLE II: **Statistics of the tasks and demonstrations.** Each task includes three different levels with respect to the randomness in initialization: low, medium, and high in order.

	Parts	Phases	Demos	Avg. length	Total hrs
lamp	3	7	150	594	4.9
			150	598	5.0
			50	768	2.1
square_table	5	16	150	1689	14.1
			150	1660	13.8
			50	1682	4.7
desk	5	16	100	1531	8.5
			100	1914	10.6
			50	1687	4.7
drawer	3	8	250	571	7.9
			250	520	7.2
			50	781	2.2
cabinet	4	11	150	883	7.4
			150	814	6.8
			50	1166	3.2
round_table	3	8	100	847	4.7
			100	867	4.8
			50	1060	2.9
stool	4	11	100	1231	6.8
			100	1419	7.9
			50	1273	3.5
chair	6	17	100	1817	10.1
			100	2282	12.7
			50	2066	5.7
one_leg	2	5	1000	374	20.8
			1000	429	23.8
			500	461	12.8
Overall	2 - 6	5 - 17	5100	374 - 2282	219.6

are consistently within 75-93% of the performance in the original (best-performing) environment. This ensures that our benchmarking result can be reproduced in a new environment built by a new subject with slightly different camera viewpoints, backgrounds, and lighting conditions. Although the results are reproducible, we found that the pre-trained policy is sensitive to the changes in the background and viewpoints of the front and wrist cameras. Therefore, we highlight how to minimize the errors in camera viewpoints in our step-by-step instruction (Section H). The experimental details can be found in Section C-A.

Observation and robot control. The observation space for the environment consists of front-view and wrist camera inputs (Figure 2) and a proprioceptive robot state. Two Intel RealSense D435 cameras are used to obtain the RGB inputs. The front-view image (1280×720) is first downsampled to 455×256 and then center-cropped to 224×224. The wrist camera image is directly downsampled to 224×224 for the wider view. Note that estimated poses of furniture parts with AprilTags [44] are **not** used as an observation but used for making the benchmark reproducible. The proprioceptive robot state includes the end-effector position, orientation (in quaternion), velocity, and gripper width. We use end-effector delta-pose control at $10\,\mathrm{Hz}$ implemented using Operational Space Control [24].
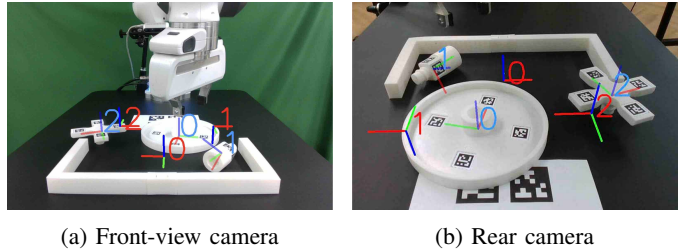

(a) Front-view camera   (b) Rear camera

Fig. 6: **Task initialization GUI tool.** To evaluate our benchmarks with the proper distribution of initial poses of furniture parts, we provide a task initialization tool. Through two camera views in (a-b), a user requires to match the poses of furniture parts (transparent axis, blue index) with the target configuration sampled from a pre-defined distribution (solid axis, red index).

B. Easy-to-Use Benchmark

Finally, to make our benchmark easy to use with as little human effort as possible, we provide a plug-and-play robot control software, a task initialization tool for evaluation, and a dataset of successful demonstrations for every assembly task.

Robot control stack. To lower the barrier to entry into a real-robot manipulation setup, our robot control stack is provided as a Docker image. The robot control stack is developed on top of a Python-based controller, Polymetis [35], which is easy to install, use, and adapt.

Task initialization tool. Initial configurations set by users without any guidance can be easily biased. This biased task initialization makes experimental results from different users not comparable since evaluation results can vary depending on initial state distributions. To evaluate algorithms under the same initial state distribution, we provide a task initialization GUI tool as shown in Figure 6. This tool samples an initial configuration from a pre-defined distribution and guides a user to match the sampled initial pose of each furniture part.

Teleoperated demonstration dataset. Our furniture assembly benchmark proposes the suite of challenging long-horizon manipulation tasks described above. Due to complex manipulation skills and the long task time horizon, learning these tasks from scratch may require millions of real-world interactions for the state-of-the-art RL methods. To make this benchmark more feasible, we collect a large demonstration dataset using teleoperation and evaluate IL and offline RL methods trained on this dataset.

We collected 219.6 hours of successful demonstrations using an Oculus Quest 2 controller and a keyboard. The VR controller makes general data collection easier and faster while a keyboard is used to rotate the wrist without moving the position of the gripper for fine-grained control, such as screwing. We summarize the statistics of the collected demonstrations in Table II. For each model and task initialization level, 2 to 24 hours of demonstration data is collected. Each demonstration is around 300-3000 steps long due to the long-horizon nature of the task.

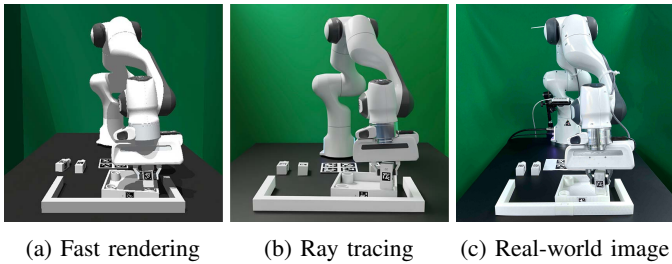(a) Fast rendering   (b) Ray tracing   (c) Real-world image

Fig. 7: **FurnitureSim: simulated version of FurnitureBench.** FurnitureSim provides realistic physics simulation and rendering: (a) fast online rendering and (b) photorealistic offline rendering with ray tracing similar to the real-world image (c).

C. FurnitureSim: Simulated Environment

Slow and costly evaluation is one of the major drawbacks of real-robot benchmarks. Introducing a simulator can help researchers quickly understand the tasks and test their ideas. To this end, we provide FurnitureSim, a simulated version of FurnitureBench, as shown in Figure 7. For realistic simulation, FurnitureSim is built upon Isaac Gym [37] and Factory [42], which provides a fast and accurate simulation of screws along with realistic rendering. To closely resemble the real-world environment, we use the same 3D furniture models and robot controller used for the real-world environment.

Correlation between FurnitureBench and FurnitureSim. Note that there still exist domain gaps between the real-world and simulated environments; yet, the simulator can be still useful for quickly verifying the correctness of algorithms and providing a proxy for real-world performance. In Figure 8, we evaluate IL (BC [48] with the ResNet-18 encoder [20]) and offline RL (IQL [28] with R3M features [41]) in both the real world and simulation. The result clearly shows the positive correlation between the simulation and real-world performances in success rates. The performances in the simulation are generally better than the ones from the real world, which implies additional challenges in the real-world benchmark. We leave closing simulation-to-real domain gaps as future work.
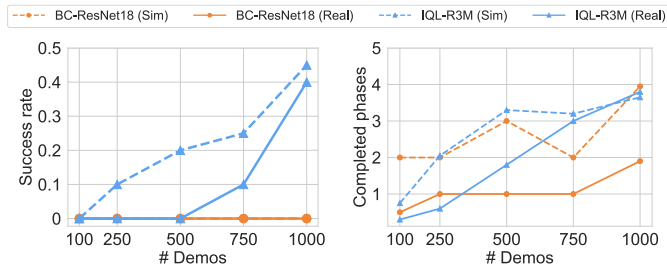

Fig. 8: **Correlation between FurnitureBench and FurnitureSim.** We compare the performance of IL and offline RL methods with respect to the dataset size between simulation and the real world on the `one_leg` assembly task under the low randomness level. We compare both the success rate (left) and the number of completed phases (right). The results in simulation and the real world show similar trends with respect to the data size as well as the choice of algorithm.

V. EXPERIMENTAL SETUP

Baselines. We evaluate our benchmark with imitation learning (BC) and the state-of-the-art offline RL (IQL) methods. Please refer to Section D for implementation details.

- **BC** (Behavioral Cloning [48]) fits a policy to the demonstration state-action pairs $(\mathbf{s}, \mathbf{a})$ with supervised learning.
- **IQL** (Implicit Q Learning [28]) is the state-of-the-art offline RL, performing advantage-weighted BC with a value function trained with expectile regression loss.

Evaluation metric. Our benchmark first measures the number of completed part assembly. However, due to its difficulty, most results are 0 with the tested algorithms. To provide more fine-grained measures of progress, we report the number of completed **phases** (i.e., subtasks defined in Section B-D) in each episode. A trained model is evaluated for 10 episodes, where their initial states are set following the provided task initialization guide tool.

Environment initialization randomness levels. Although the ultimate goal of autonomous robotic manipulation is to handle all possible configurations, a wide variety of initial configurations makes the problem challenging. Thus, we benchmark both tractable and challenging scenarios in three different levels with respect to the randomness in the initial furniture part configuration, as illustrated in Figure 9:

- **Low**: The pose of each furniture piece is fixed; but, there can be a small noise during resetting.
- **Medium**: Based on the pre-defined poses in **Low**, we perturb each part with translation noise between $[-5\,\mathrm{cm}, 5\,\mathrm{cm}]$ and rotational noise between $[-45°, 45°]$.
- **High**: Furniture parts are randomly initialized on the workspace.
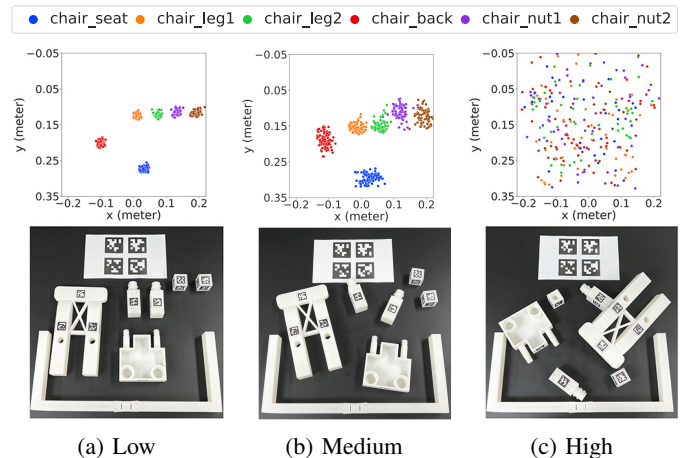

(a) Low   (b) Medium   (c) High

Fig. 9: **Three initialization randomness levels.** The higher the randomness in the initial state is, the more generalization capability is required for an agent. We provide three different levels of randomness in the initial states: (a) **low** fixes part poses and allows only a little human error, (b) **medium** randomizes the part poses around the given arrangement, and (c) **high** randomly places all pieces. We visualize the initial part position distributions in our data (top) and example instances (bottom).

Evaluation procedure. Each individual trial is conducted according to the following procedure:

1) The robot arm moves to the initial pose.
2) The furniture parts are rearranged using our provided tool according to the initialization level (low, med, high).
3) A policy controls the robot until it completes the task, stops motions for 5 sec, shows unsafe movements, exceeds 350 steps per skill, or exceeds 3000 steps in total.
4) The success rate and phase completion score are recorded.

VI. BENCHMARKING RESULTS

We evaluate an IL algorithm (BC [48] with the ResNet-18 encoder [20]) and the state-of-the-art offline RL algorithm (IQL [28] with R3M features [41]) on our reproducible, real-world furniture assembly benchmark.[3] We first benchmark the performance of selected individual skills in our single-skill benchmark (Section VI-A). We then benchmark the full furniture assembly tasks in Section VI-B, where none of the algorithms make meaningful progress. To validate our benchmark for the full-assembly evaluation, we additionally benchmark a simpler task, `one_leg` assembly, with 1000 demonstrations. Finally, we benchmark our simulated environment in Section VI-C.

A. Benchmark I: Single-Skill Benchmark

Furniture assembly is a challenging manipulation task due to its long horizon and complex control. Thus, we investigate the difficulty of each individual subtask of furniture assembly, such as grasping, pushing, inserting, and screwing, as defined in Figure 18. For the single-skill benchmark, we train a skill policy using manually segmented trajectories corresponding to the skill. Then, we measure the success rate of each skill by initializing the robot and objects to one of the skill's initial states following Appendix, Section C-B. We benchmark the first five skills of each furniture model.

The single-skill benchmark results in Table III demonstrate that individual skill policies can successfully learn "grasping" and "placing" skills. The "pushing" skill in `drawer` achieves 30% success rate, which is slightly worse than that of the "grasping" skill (60%), with BC. This is mainly due to the stochastic dynamics of non-prehensile interactions. On the other hand, both algorithms struggle at "inserting" skill, which shows from 0% to 20% success rates. "Inserting" requires precise control to correctly align a screw and a hole, and when the agent fails to align them, it often gets stuck or easily goes into an out-of-distribution state from its training data.

Interestingly, the "screwing" skill turns out to be not too difficult for IQL, e.g., achieving 90% and 70% success rates for `square_table` and `desk` on the low randomness level, respectively. However, in `lamp` and `round_table`, where the round-shaped parts need to be screwed, IQL struggles and achieves only 10% and 0% success rates, respectively.

TABLE III: **Single-skill benchmark results.** First 5 skills of each furniture task are independently evaluated and the average success rates over 10 trials are reported in percentage. Each skill is evaluated on two different initialization levels: low and medium randomnesses.

	Skill 1 low	Skill 1 med	Skill 2 low	Skill 2 med	Skill 3 low	Skill 3 med	Skill 4 low	Skill 4 med	Skill 5 low	Skill 5 med
`lamp`	Grasping 1		Placing		Grasping 2		Inserting		Screwing	
BC	0%	0%	40%	30%	30%	20%	0%	0%	10%	0%
IQL	70%	40%	90%	20%	0%	20%	20%	0%	10%	0%
`square_table`										
BC	40%	0%	40%	80%	0%	20%	0%	0%	60%	0%
IQL	80%	70%	100%	70%	60%	40%	10%	0%	90%	30%
`desk`										
BC	60%	50%	80%	60%	0%	0%	0%	0%	0%	0%
IQL	100%	60%	100%	60%	60%	20%	20%	20%	70%	20%
`round_table`										
BC	0%	50%	50%	30%	30%	0%	0%	0%	0%	0%
IQL	80%	90%	30%	40%	50%	10%	10%	10%	0%	10%
`stool`										
BC	70%	0%	50%	30%	60%	0%	0%	0%	0%	0%
IQL	90%	60%	80%	40%	90%	10%	0%	0%	60%	0%
`chair`										
BC	30%	10%	60%	30%	20%	0%	10%	0%	20%	0%
IQL	60%	50%	100%	40%	40%	10%	0%	0%	50%	10%
`drawer`									Pushing	
BC	50%	60%	60%	20%	90%	60%	0%	0%	10%	30%
IQL	80%	50%	70%	0%	70%	20%	0%	0%	80%	50%
`cabinet`									Grasping 3	
BC	70%	30%	70%	30%	0%	20%	0%	0%	0%	0%
IQL	60%	70%	70%	60%	20%	50%	0%	0%	30%	50%

B. Benchmark II: Full-Assembly Benchmark

We present full furniture assembly evaluation results in Figure 10. Overall, neither BC nor IQL achieve a single part assembly on the full-assembly tasks. Across all furniture and initialization levels, IQL performs better than BC, and lower randomness in the initial configuration leads to higher performance.

Qualitatively, the learned policies grasp the first target furniture part in most cases, and often place it in the designated position and grasp the second target part. However, the policies mostly fail to achieve more difficult phases, such as inserting and screwing, as can be observed in the single-skill benchmark.

The failure of these algorithms to even attach a pair of furniture parts despite the high-quality demonstration dataset highlights the need for further algorithmic improvements and suggests a good utility of our FurnitureBench for such algorithmic progression. More qualitative results are shown in Appendix, Figure 20 and our website.

Is this benchmark tractable? To verify the feasibility of the proposed benchmark and provide a more tractable benchmark, we devise an easier benchmarking task, `one_leg` assembly, which is a part of `square_table` consisting of five phases: (1) grasping a tabletop, (2) placing it in the corner, (3) picking up a table leg, (4) inserting the leg into the hole on the tabletop, and (5) screwing it, as illustrated in Appendix, Figure 18b. The task horizon is approximately 500 timesteps. We collected 1000 demonstrations for each of the low and medium initialization randomness levels.
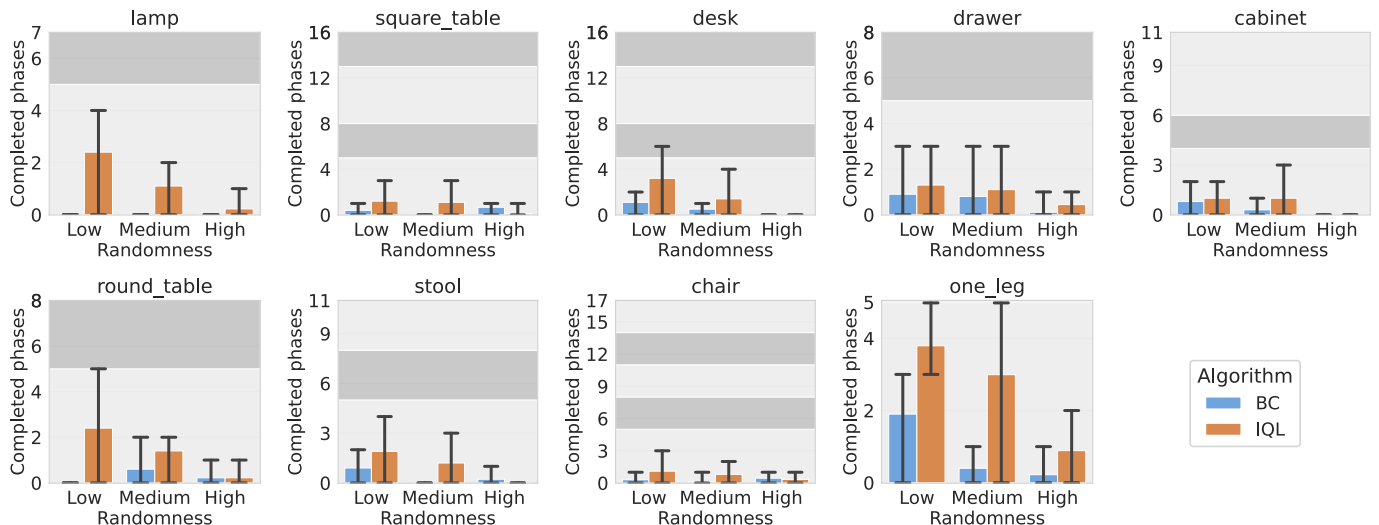
Fig. 10: **Full-assembly benchmark results.** We report the number of completed phases averaged over 10 episodes and the error bars indicating the minimum and maximum completed phases. The background color indicates each part assembly and the maximum phase in each plot represents the task success. Each furniture model is evaluated on three different initialization levels (low, medium, and high).

Figures 8 and 10 show that IQL-R3M achieves 4 phases on average and 40% success rate on the low randomness level. It always achieves the phase 3 (grasping the leg) but fails at inserting 60% of the time. Once it succeeds in inserting, it mostly succeeds in screwing. This result reassures that "inserting" is the most challenging skill as it involves stochastic and frequent collisions.

Does more data improve the performance? In Figure 8, we show that both BC-ResNet18 and IQL-R3M improve their performances with more demonstration data. We further scale the data by combining trajectories collected from the low and medium randomness levels. In Table IV, the IQL-R3M policies trained with 2000 demonstrations in the mixed dataset (**Mixed data**) show significant improvements, 4.6 and 3.7 completed phases on average on the low and medium randomness levels, compared to the policies trained only with 1000 demonstrations (**Original**), 3.8 and 3.0, respectively. These results suggest the need for learning generalizable policies with limited data.

How critical the diversity of training data is? To see the benefit of diverse data, we compare IQL-R3M policies trained using the low randomness level data (**Original low**) and the medium randomness level data (**Original med**). In Table IV, **Original low** shows a higher performance, 3.8, than **Original med**, 3.0, which implies that the benchmarking algorithms are not good at utilizing diverse data but require a large number of data on the same distribution for evaluation.

Is a wrist camera essential for furniture assembly? In our benchmark, we use visual observations from the front-view camera and wrist camera. We test whether the wrist camera is helpful for furniture assembly by training a policy only with the front-view camera input (**Front camera only**). Without the wrist camera input, the performance drops significantly from 3.8 and 3.0 to 2.0 and 1.3 on the low and medium randomness levels, respectively. This result means that the

policies mostly fail to grasp the table leg without the wrist camera. This suggests that (1) the wrist camera provides information about the occluded area from the front camera and (2) some manipulation skills are easier to train and generalize better with the wrist-camera observation.

Does a vision-based policy cheat on AprilTag? One can concern about a policy learns to identify and exploit AprilTags for learning manipulation skills. Thus, we evaluate the IQL-R3M policy with the furniture models after removing AprilTags (**No AprilTag**) and after attaching random AprilTag markers different from the original ones (**Random AprilTag**) in Table IV. Due to the visual domain gaps in AprilTags, **Random AprilTag** shows slightly worse performance, 3.4 and 2.7 than **Original**, 3.8 and 3.0, under the low and medium randomness levels, respectively. **No AprilTag** has a little more performance drop and achieves 3.1 and 2.4, respectively. Despite these inferior results, the policy performs reasonably well in both cases, which ensures that the vision-based policies do not exploit the AprilTag marker information.

In summary, the one_leg assembly experiments verify that our benchmark is *tractable* but the algorithms require a huge amount of data to solve a part of a full furniture assembly task.

TABLE IV: **Analysis on one-leg assembly.** We evaluate IQL-R3M under diverse settings and report the average completed phases over 10 episodes.

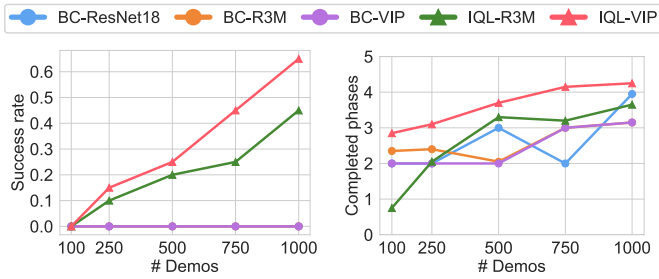|	low	med
Original low (low randomness data)	3.8	-
Original med (medium randomness data)	3.0	3.0
Mixed data	4.6	3.7
Front camera only	2.0	1.3
No AprilTag	3.1	2.4
Random AprilTag	3.4	2.7

Fig. 11: **Simulation benchmark results.** Using our FurnitureSim simulator, we evaluate BC and IQL with diverse visual encoders in the `one_leg` assembly task on the low randomness level. We measure both success rates and average completed phases.

C. Benchmark III: Simulation Benchmark

For easy testing and debugging of algorithms, we provide the simulated environment, FurnitureSim, as described in Section IV-C. Although direct transfer of knowledge from the simulator to the real world is difficult due to the simulation-to-real domain gap, the simulated results can provide a good proxy for how well an algorithm may work in the real world.

Figure 11 shows the performance of simulated `one_leg` assembly of BC and IQL with various visual encoders: ResNet-18 [20], R3M [41], and VIP [36]. For the ResNet-18 encoder, we randomly initialize its parameters and train the encoder together with the policy. For R3M and VIP, we use the pre-trained parameters without fine-tuning. Similar to the real-world benchmark, IQL shows higher success rates than BC. In terms of average completed phases, BC achieves comparable performance with IQL but BC fails at accurate insertion and screwing as these contact-rich interactions quickly bring an environment out of distribution. In the simulation, IQL-VIP shows higher performance than IQL-R3M; however, we found that IQL-R3M is slightly more robust in the real-world experiments and thus, we use R3M for the real-world benchmarks.

VII. Conclusion

We propose FurnitureBench, a *reproducible, real-world* furniture assembly benchmark, as a novel benchmark for testing complex long-horizon manipulation tasks on real robots. To serve a reproducible and easy-to-use benchmarking environment, we provide 3D printable furniture part models, a step-by-step environment setup guide, software tools, large demonstration data, and a simulator, FurnitureSim. Our benchmarking results show that common imitation learning and the state-of-the-art offline reinforcement learning methods struggle at solving furniture assembly due to its long-horizon nature and complex manipulation skills. Therefore, it is a well-suited benchmark to encourage robot learning researchers to tackle complex long-horizon real-world manipulation tasks with better sample and demonstration efficiency.

Limitations and Future work. While FurnitureBench includes many perspectives of real-world furniture assembly,

the 3D furniture models are still tailored to common robotic arms for research, e.g., all pieces have widths larger than $2\,cm$ for easy grasping, which is larger than the tiny screws used in real-world IKEA furniture. Furthermore, our furniture models are much smaller in scale compared to the real-world furniture, which requires robots with a wider workspace and higher payload. Despite such simplification, FurnitureBench still serves most of common challenges in solving complex long-horizon manipulation tasks in the real world, such as temporal credit assignment, exploration, perception, and dexterous manipulation; thus, we believe solving this benchmark will make us a step closer to solving real-world furniture assembly.

Our benchmark is mainly evaluated using single-task criteria with a single robot. Leveraging shared information across different furniture assembly tasks in a multi-task RL perspective (e.g., diverse furniture categories, shapes, materials) or a multi-embodiment setup is worth to be examined.

FurnitureBench uses a single Franka Emika Panda arm and using only a single hand limits the dexterity of possible manipulation skills, resulting in the limited interactions tested in the benchmark. Extending our benchmark to multi-arm or multi-robot collaboration can endow robots with more dexterity.

Lastly, FurnitureSim has simulation-to-real gaps in both visual and physical domains. Especially, we found that system identification is very challenging. For example, the same torque command does not lead to the same robot trajectories in simulation and the real world due to inaccurate robot modeling (e.g., mass of each robot part, friction of joints). We leave closing the simulation-to-real gaps as future work. Simulation-to-real policy transfer under an imperfect simulator could be another interesting direction.

References

[1] Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on Robot Learning*, pages 1300–1313. PMLR, 2020.

[2] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al.

Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[3] Stefan Bauer, Manuel Wüthrich, Felix Widmaier, Annika Buchholz, Sebastian Stark, Anirudh Goyal, Thomas Steinbrenner, Joel Akpo, Shruti Joshi, Vincent Berenz, et al. Real robot challenge: A robotics competition in the cloud. In *Neural Information Processing Systems 2021 Competitions and Demonstrations Track*, pages 190–204. PMLR, 2021.

[4] Stefan Bauer, Manuel Wüthrich, Felix Widmaier, Annika Buchholz, Sebastian Stark, Anirudh Goyal, Thomas Steinbrenner, Joel Akpo, Shruti Joshi, Vincent Berenz, Vaibhav Agrawal, Niklas Funk, Julen Urain De Jesus, Jan Peters, Joe Watson, Claire Chen, Krishnan Srinivasan, Junwu Zhang, Jeffrey Zhang, Matthew Walter, Rishabh Madan, Takuma Yoneda, Denis Yarats, Arthur Allshire, Ethan Gordon, Tapomayukh Bhattacharjee, Siddhartha Srinivasa, Animesh Garg, Takahiro Maeda, Harshit Sikchi, Jilong Wang, Qingfeng Yao, Shuyu Yang, Robert McCarthy, Francisco Sanchez, Qiang Wang, David Bulens, Kevin McGuinness, Noel O'Connor, Redmond Stephen, and Bernhard Schölkopf. Real robot challenge: A robotics competition in the cloud. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, volume 176 of *Proceedings of Machine Learning Research*, pages 190–204. PMLR, 06–14 Dec 2022.

[5] Stefan Bauer, Manuel Wüthrich, Felix Widmaier, Annika Buchholz, Sebastian Stark, Anirudh Goyal, Thomas Steinbrenner, Joel Akpo, Shruti Joshi, Vincent Berenz, et al. Real robot challenge: A robotics competition in the cloud. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 190–204. PMLR, 2022.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[7] Fabrizio Bottarel, Giulia Vezzani, Ugo Pattacini, and Lorenzo Natale. Graspa 1.0: Graspa is a robot arm grasping performance benchmark. *IEEE Robotics and Automation Letters*, 5(2):836–843, 2020.

[8] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[9] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[11] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*, 2022.

[12] Ben Burgess-Limerick, Chris Lehnert, Jürgen Leitner, and Peter Corke. Dgbench: An open-source, reproducible benchmark for dynamic grasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3218–3224. IEEE, 2022.

[13] Konstantinos Chatzilygeroudis, Bernardo Fichera, Ilaria Lauzana, Fanjun Bu, Kunpeng Yao, Farshad Khadivar, and Aude Billard. Benchmark for bimanual robotic manipulation of semi-deformable objects. *IEEE Robotics and Automation Letters*, 5(2):2443–2450, 2020. doi: 10.1109/LRA.2020.2972837.

[14] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *IEEE International Conference on Robotics and Automation*, 2019.

[15] Silvia Cruciani, Balakumar Sundaralingam, Kaiyu Hang, Vikash Kumar, Tucker Hermans, and Danica Kragic. Benchmarking in-hand manipulation. *IEEE Robotics and Automation Letters*, 5(2):588–595, 2020. doi: 10.1109/LRA.2020.2964160.

[16] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in Neural Information Processing Systems*, pages 1087–1098, 2017.

[17] Irene Garcia-Camacho, Martina Lippi, Michael C. Welle, Hang Yin, Rika Antonova, Anastasiia Varava, Julia Borras, Carme Torras, Alessandro Marino, Guillem Alenyà, and Danica Kragic. Benchmarking bimanual cloth manipulation. *IEEE Robotics and Automation Letters*, 5(2): 1111–1118, 2020. doi: 10.1109/LRA.2020.2965891.

[18] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *Conference on Robot Learning*, 2019.

[19] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning*, pages 24–33, 2021.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[21] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

[22] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002, 2021.

[23] H-J Kang and Robert A Freeman. Joint torque optimization of redundant manipulators via the null space damping method. In *IEEE International Conference on Robotics and Automation*, pages 520–521, 1992.

[24] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[25] Kenneth Kimble, Karl Van Wyk, Joe Falco, Elena Messina, Yu Sun, Mizuho Shibata, Wataru Uemura, and Yasuyoshi Yokokohji. Benchmarking protocols for evaluating small parts robotic assembly systems. *IEEE robotics and automation letters*, 5(2):883–889, 2020.

[26] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[27] Ross A Knepper, Todd Layton, John Romanishin, and Daniela Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *IEEE International Conference on Robotics and Automation*, pages 855–862, 2013.

[28] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

[29] Alex X Lee, Coline Manon Devin, Yuxiang Zhou, Thomas Lampe, Konstantinos Bousmalis, Jost Tobias Springenberg, Arunkumar Byravan, Abbas Abdolmaleki, Nimrod Gileadi, David Khosid, et al. Beyond pick-and-place: Tackling robotic stacking of diverse shapes. In *Conference on Robot Learning*, 2021.

[30] Youngwoon Lee, Edward S. Hu, Zhengyu Yang, and Joseph J. Lim. To follow or not to follow: Selective imitation learning from observations. In *Conference on Robot Learning*, 2019.

[31] Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S. Hu, and Joseph J. Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rygrBhC5tQ.

[32] Youngwoon Lee, Edward S Hu, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. In *IEEE International Conference on Robotics and Automation*, 2021. URL https://clvrai.com/furniture.

[33] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[34] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, 2022.

[35] Yixin Lin, Austin S. Wang, Giovanni Sutanto, Akshara Rai, and Franziska Meier. Polymetis. https://facebookresearch.github.io/fairo/polymetis/, 2021.

[36] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*, 2022.

[37] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu based physics simulation for robot learning. In *Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[38] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893, 2018.

[39] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *IEEE International Conference on Computer Vision*, 2019.

[40] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 2022.

[41] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. In *Conference on Robot Learning*, 2022.

[42] Yashraj Narang, Kier Storey, Iretiayo Akinola, Miles Macklin, Philipp Reist, Lukasz Wawrzyniak, Yunrong Guo, Adam Moravanszky, Gavriel State, Michelle Lu, et al. Factory: Fast contact for robotic assembly. In *Robotics: Science and Systems*, 2022.

[43] Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, 2013.

[44] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *IEEE International Conference on Robotics and Automation*, pages 3400–3407, 2011.

[45] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[47] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning*, 2020.

[48] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, pages 305–313, 1989.

[49] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*, 2018.

[50] Şenol Şirin, Enes Aslan, and Gülşah Akincioğlu. Effects of 3d-printed pla material with different filling densities on coefficient of friction performance. *Rapid Prototyping Journal*, 29(1):157–165, 2023.

[51] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, et al. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Conference on Robot Learning*, 2021.

[52] Francisco Suárez-Ruiz and Quang-Cuong Pham. A framework for fine robotic assembly. In *IEEE International Conference on Robotics and Automation*, pages 421–426, 2016.

[53] Francisco Suárez-Ruiz, Xian Zhou, and Quang-Cuong Pham. Can robots assemble an ikea chair? *Science Robotics*, 3(17), 2018.

[54] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Neural Information Processing Systems*, 2021.

[55] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[56] Yusuke Urakami, Alec Hodgkinson, Casey Carlin, Randall Leu, Luca Rigazio, and Pieter Abbeel. Doorgym: A scalable door opening environment and baseline agent. *arXiv preprint arXiv:1908.01887*, 2019.

[57] Jun Yamada, Youngwoon Lee, Gautam Salhotra, Karl Pertsch, Max Pflueger, Gaurav S. Sukhatme, Joseph J. Lim, and Peter Englert. Motion planner augmented reinforcement learning for obstructed environments. In *Conference on Robot Learning*, 2020.

[58] Brian Yang, Jesse Zhang, Vitchyr Pong, Sergey Levine, and Dinesh Jayaraman. Replab: A reproducible low-cost arm benchmark platform for robotic learning. *arXiv preprint arXiv:1905.07447*, 2019.

[59] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, 2019.

[60] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

To enhance the main paper, we additionally provide the following supplementary materials:

- **Environment details**: We elaborate on the hardware configurations, software details, simulator, reward function, action and observation space, and AprilTag pose estimation in Section B.
- **Evaluation details**: We elaborate on the detailed evaluation protocols of our benchmarks in Section C.
- **Training details**: We elaborate on how we train BC and IQL for our benchmarks in Section D.
- **Dataset details** reveal details of the data collection process and characters of datasets Section E.
- **Furniture model details**: We describe a suite of furniture models and their assembly processes in Section F.
- **Qualitative results**: Videos of the qualitative results of BC and IQL can be found in https://clvrai.com/furniture-bench.
- **Code** and **data**: https://github.com/clvrai/furniture-bench
- **Step-by-step environment setup guide**: We describe our reproducible environment setup instruction in Section H.

A. Hardware Specifications

FurnitureBench aims to provide a reproducible real-robot system by adopting products widely available across the world and 3D printing objects. Here is the list of products used to build our benchmark system:

- Franka Emika Panda (link) and bench clamp (link)
- 3x Intel RealSense D435 (link)
- 2x Manfrotto variable friction arm with camera bracket (link)
- 2x Manfrotto super clamp (link)
  - For the front camera arm, 1x Ulanzi ULS01 camera mount (link) can be used instead of one Manfrotto variable friction arm and super clamp.[4]
- Oculus Quest 2 (optional for data collection) (link)
- Any keyboard and mouse
- Black IKEA TOMMARYD table (link)
- Green photography backdrop (larger than $w \times h$: $280\,\mathrm{cm} \times 400\,\mathrm{cm}$ for wide-angle coverage)
- Background frame (larger than $w \times h$: $190\,\mathrm{cm} \times 150\,\mathrm{cm}$)
- 2x Standing pole to hang the green backdrop on the sides of the table
- LED light panel supporting color temperature ($4600\,\mathrm{K}$-$6000\,\mathrm{K}$) and brightness ($\sim 4000\,\mathrm{lm}$)

---

[4]We offer two options for the front camera mount. In most of our evaluations, we used the Manfrotto variable friction arm and super clamp for the front camera. However, we observed that some inexperienced participants struggled at manipulating the Manfrotto variable friction arm in high precision. We found that Ulanzi ULS01 camera mount is more intuitive to set up due to its ability to independently move the camera arm along different axes, and we could reproduce our original results with the new camera arm. Both these options are detailed in our step-by-step setup guide.
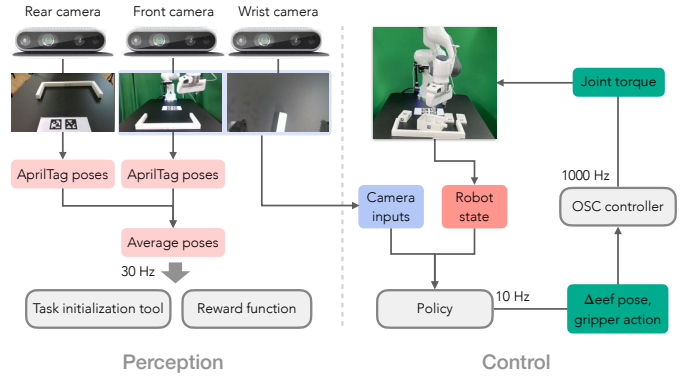

Fig. 12: **Overview of our robot system design.** The agent receives proprioceptive robot states and images from the front and wrist cameras and takes action (delta end-effector pose and a binary gripper action) at a frequency of $10\,\mathrm{Hz}$. For the reward function and initialization tool, we use images collected via the front and rear cameras to estimate the poses of the furniture parts using AprilTag. Note that the estimated furniture part poses are used only for environment functionalities, not for the learning agent. We choose the front and rear camera poses that can provide holistic views of the workspace as well as the high AprilTag detection accuracy.

- 4x 10 feet USB Type-C to Type-A 3.1 cables (link)
- Server computer (an Intel 8th generation i7 or AMD Ryzen 5 5600X processor, or better)
- Client computer (an Intel 8th generation i7 or AMD Ryzen 5 5600X processor, four USB 3 Type-A ports with high bandwidth, and an NVIDIA RTX 3070 GPU, or better)
- White PLA 3D printer filament
- Double-sided rubber tape
- 2x M3 x $10\,\mathrm{mm}$ hex socket screw for the wrist camera
- 1x M6 x $25\,\mathrm{mm}$ hex socket screw for the wrist camera

Our system requires two computers, Server and Client machines. The server is directly connected to the robot and dedicated to real-time robot control. The client manages all other aspects, including taking sensory inputs from cameras as well as the robot, computing a deep neural network policy, and sending action commands to the robot. In this paper, we use the server machine with an AMD Ryzen 5 5600X CPU and the client machine with an AMD Ryzen 9 5950X CPU and an NVIDIA RTX 3090 GPU.

B. Software Details

Our robot control stack is built on top of Polymetis [35], which provides a Python interface for the robot control stack, including basic robot controllers and network communication interfaces with the robot arm and gripper.

We implement Operational Space Control (OSC [24]) in PyTorch [46] and optimize its performance with just-in-time compilation. Although the controller is optimized, OSC's computation may be slower than $1000\,\mathrm{Hz}$, which is the control frequency of Franka Emika Panda. Thus, we compute a joint torque every three low-level control steps, which is approximately $333\,\mathrm{Hz}$ by repeating the same torque three times.

Finally, as illustrated in Figure 12, a policy action in delta end-effector pose (10 Hz) is converted into approximately 33 joint torques, and each of these torques repeats three times.

C. FurnitureSim Implementation

We develop FurnitureSim, a simulated version of FurnitureBench, for easy and fast evaluation of algorithms. To minimize gaps between simulation and the real world, we adopt the code from FurnitureBench as much as we can. Specifically, we simply replace the real-world interactions with fast contact simulation supported by Factory [42] and IsaacGym [37] with the PhysX physics engine. For example, the 3D furniture models of FurnitureBench are used for FurnitureSim after converting their formats to the Signed Distance Function format (SDF). The simulator provides camera observations and robot proprioceptive states. An agent then receives this information and predicts its action just as the input comes from the real-world sensors. The robot controller code used in FurnitureBench takes input from the action command by the agent and computes low-level action signals.

Although the same 3D models are used, estimating their exact physical properties is not trivial. We estimate the density of each 3D model by first measuring its mass using a 3D printing slicing tool and then measuring its volume using Blender 3D Print Toolbox. With the mass and volume of a 3D part, we can estimate its density.

3D-printed objects with PLA may have diverse friction coefficients [50]. We found that a friction coefficient between 0.15 and 0.42 works robustly for diverse furniture configurations in FurnitureSim. In this paper, we use 0.15 as a friction coefficient.

D. Reward Function

The reward function is essential for reinforcement learning and, more generally, evaluating the task's success. Our reward function is simple: $+1$ when a pair of parts are assembled and 0, otherwise. Thus, a successful assembly will achieve a total reward of $N - 1$, where N is the number of parts. Note each pair of parts is rewarded only once, meaning disassembling and reassembling will not provide a reward again.

The reward function is computed based on the relative poses between parts, which are estimated using AprilTags. To discriminate the success of assembly, we pre-define ground truth relative poses between matching parts when they are assembled. Two parts are considered assembled when the distance between the current relative pose and ground truth pose remain lower than a threshold (cosine similarity of each column vector of a rotation matrix is larger than 0.96, and the absolute distance in each of x, y, z axes is smaller than $7\,\text{mm}$).

Figure 18 illustrates example assembly procedures for all furniture models. The images with red boundaries indicate that two parts are assembled, i.e., a $+1$ reward is provided. These assembly orders are not the only solutions. An agent can achieve rewards with different assembly orders, e.g., assembling the table leg to either the top left corner or the bottom right corner of the table top will lead to a reward of $+1$.

Phase. Furniture assembly tasks are complex and long-horizon – even getting the first reward requires a large number of environmental steps. Therefore, instead of benchmarking with reward, we propose to measure more fine-grained progress, which we refer to as a "phase". The phase score indicates the number of completed subtasks. The human demonstrators always accomplished each phase in the same order during data collection. During the evaluation, in contrast to the automated reward function, the success of each phase is judged by a human operator. Note that this phase information is not used for training an agent but for evaluating and analyzing a learned agent. Phase numbers for each furniture model can be found in Figure 18 on the top left corners.

E. Action Space

We use an 8D action space, which consists of delta end-effector (x, y, z)-position in meter (3D), delta orientation (quaternion, 4D), and gripper action (1D). The action space is bounded between -1 and $+1$.

Although the gripper action is binary (-1 for opening and $+1$ for closing the gripper), a policy can output a continuous value between -1 to 1. To prevent the gripper from repeating opening and closing actions, the gripper action is only activated when its absolute value is greater than a threshold of 0.019.

The robot arm is controlled using OSC [24], which computes joint torques to reach a given end-effector goal pose (3D Cartesian coordinate and 4D quaternion). As shown in Figure 12, the robot arm takes an end-effector control command at $10\,\text{Hz}$, and the OSC controller produces joint torques in real-time ($1000\,\text{Hz}$). The resulting joint torques are damped to prevent unstable movements due to high DoFs [23], and these conservative motions help maintain easy and safe robot control.

Setting the desired end-effector goal too far away from the current end-effector position can generate excessive joint torques since the torques are computed based on the difference between the current and the goal positions. We use two strategies to avoid such instability: interpolation to the desired end-effector goal and clipping the delta action to a fixed amount. The interpolation is done by equally dividing the distance between the current end-effector position and the desired end-effector position into several steps. The total number of steps is defined by the controller frequency divided by the action command frequency. And the clipping range of the delta end-effector position is set to be $\pm 10\,\text{cm}$.

F. Observation Space

The observations available from our environment are listed in Table V. Although other sensory inputs are available via environmental steps and stored while data collection, only a portion of those are used for the agent's input. The observation space for the policy consists of two RGB images from the front and wrist camera and robot proprioceptive states. The front camera image is first downsampled to 455×256 (smaller edge match 256 while keeping the image ratio) and then center-cropped to 224×224. The wrist camera is directly downsampled to 224×224 to keep the wider view. We use

INTER_AREA interpolation in OpenCV [9] for downsampling for all the images. The proprioceptive robot state consists of the end-effector position, orientation (in quaternion), velocity, and gripper width.

TABLE V: Observations available in FurnitureBench. The observations used for the input to the policy are check marked.

Name	Shape	Unit	Policy
Front camera RGB	(1280, 720, 3)	pixel value (0 - 255)	✓
Wrist camera RGB	(1280, 720, 3)	pixel value (0 - 255)	✓
Rear camera RGB	(1280, 720, 3)	pixel value (0 - 255)	
Front camera depth	(1280, 720)	(0 - 65536) mm	
Wrist camera depth	(1280, 720)	(0 - 65536) mm	
Rear camera depth	(1280, 720)	(0 - 65536) mm	
Object poses	(# parts $\times$ 7)	m and quaternion	
EE position	(3)	m	✓
EE orientation	(4)	quaternion	✓
EE linear velocity	(3)	m/s	✓
EE rotational velocity	(3)	rad/s	✓
Joint positions	(7)	rad	
Joint velocities	(7)	rad/s	
Joint torques	(7)	N $\cdot$ m	
Gripper width	(1)	m	✓

G. Multi-camera pose estimation using AprilTag

AprilTag pose estimation using a single camera is often inaccurate and can even fail to detect object poses when markers are occluded or angled. To compensate for this detection failure and inaccurate pose estimation, we use two parallel AprilTag detection modules using two cameras (front and rear cameras as illustrated in Figure 2a) and aggregate the estimated poses. Thus, the resulting pose estimations can be more accurate. We choose the original poses of the front and rear cameras in that the viewpoints provide holistic views of the workspace as well as clearly observe the AprilTags in multiple orientations. To handle the noise (e.g., flipping axis) in AprilTag pose estimation, we filter outliers that are different from the last five detection results for each camera.

For accurate and easy marker placement, all 3D models have placeholders for markers with their AprilTag IDs, as shown in Figure 13. We further improve the reliability and accuracy of pose estimation using multiple markers for each furniture part. For each furniture part, the poses of the detected markers are converted to the canonical furniture part pose, where the position is its center of mass and the orientation follows the marker with the smallest ID. Then, the furniture pose is estimated by averaging the canonical poses computed from all detected markers.

To sum it up, we improve the stability and accuracy of pose estimation by (1) averaging over estimated poses from two cameras, (2) filtering outliers, and finally, (3) averaging over estimated poses of multiple markers on each furniture part.

APPENDIX C
EXPERIMENT DETAILS

A. Reproducibility Experiment Details

In the reproducibility experiments, we investigate the difficulty of our environment setup and confirm the reproducibility

Fig. 13: **AprilTag placeholder.** For easy and accurate marker placement, all 3D models have AprilTag placeholders on their surfaces with corresponding AprilTag IDs.

of our evaluation results. We ask participants to build the environment from scratch, providing all the necessary components for our benchmarking environment.

At the starting point, the robot's pose differs from our specified arrangement, so participants need to adjust its position according to the provided instruction. Since lifting and relocating the robot alone is physically demanding, an additional person assisted during the robot's mounting process. Additionally, the required software has been pre-installed, as our primary goal is to assess the reproducibility of the real-world environment setup rather than software installation. The initial configuration is shown in Figure 14.

After finishing the hardware installation, the pre-trained IQL-R3M policy is evaluated 20 rollouts for the `one_leg` assembly task, and the average number of completed phases is measured. For evaluation, we use the low initialization level, and the policy is trained with 1000 demonstrations.

Fig. 14: **Initial configuration for reproducibility experiments.** Each participant is asked to recreate the environment with the provided materials by following our instruction.

B. Single-Skill Benchmark Experiment Details

For the single-skill benchmark, we need to define an initial state for each skill evaluation: we randomly sample one starting state of the skill from a dataset and fix this state for the entire evaluation. For the medium initialization randomness level, we add different noises to this pre-defined state for skill evaluation rollouts. To prevent an agent from repeating the same episodes
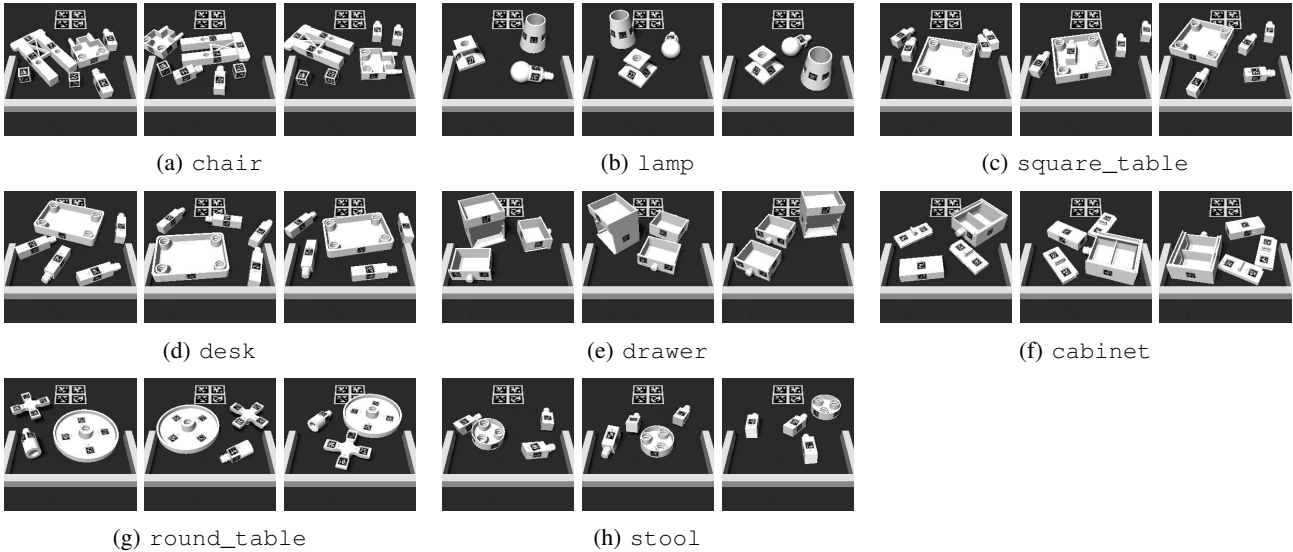
(a) chair   (b) lamp   (c) square_table

(d) desk   (e) drawer   (f) cabinet

(g) round_table   (h) stool

Fig. 15: **Three pre-defined initial states for evaluation in the high task initialization randomness level**.

during evaluation, we also randomize the end-effector pose ($\pm 0.5\,\mathrm{cm}$, $\pm 5°$ and $\pm 5\,\mathrm{cm}$, $\pm 15°$ for the low and medium randomness levels, respectively).

C. Full-Assembly Benchmark Experiment Details

Three initialization randomness levels, namely low, medium, and high, introduce the different levels of perturbation on the initial poses of furniture parts, as illustrated in Section V. The task initialization GUI tool helps configure the environment to align with the specified poses, as illustrated in Section IV-B.

For the high randomness level, we collected data from completely random initial configurations. However, for consistent evaluation results in the high randomness level, we use three distinct predetermined poses as initialization states, as shown in Figure 15.

APPENDIX D
TRAINING DETAILS

The observation space of our environment consists of the front-view and wrist-view camera inputs, and proprioceptive robot states (end-effector position, orientation, velocity, and gripper width). For efficient visual policy learning, we use the pre-trained visual encoder, R3M [41] and VIP [36], which is trained with large-scale egocentric human videos. We concatenate 2048-D feature vector extracted from each camera image with proprioceptive states and feed this into 3-layer MLP consisting of (512, 256, 256) hidden units for the policy network. Hyperparameters for our baseline implementations are listed in Table VI.

Behavioral Cloning (BC [48]). We train the policy using a batch of size 64 for 50 epochs. The policy is optimized using the Adam optimizer [26] with a learning rate of 0.0003.

Implicit Q Learning (IQL [28]). We use the official JAX [8] implementation of IQL with minor changes to use R3M [41] and VIP [36] features. The policy is trained with sparse rewards, where rewards in each demonstration are manually marked as $+1$ when each furniture part is assembled.

TABLE VI: Hyperparameters for BC and IQL.

Hyperparameter	BC	IQL
Learning rate	3e-4	3e-4
Learning rate decay	Step decay	Cosine decay
Learning rate decay factor	0.5	-
# Mini-batches	64	256
Policy # Hidden units	(512, 256, 256)	(512, 256, 256)
Q/V # Hidden units	-	(512, 256, 256)
Activation	ReLU	ReLU
Image encoder	R3M, VIP, ResNet-18	R3M, VIP
Discount factor (γ)	-	0.996
Expectile (τ)	-	0.8
Inverse temperature (β)	-	0.5

APPENDIX E
DATASET DETAILS

A. Data Collection
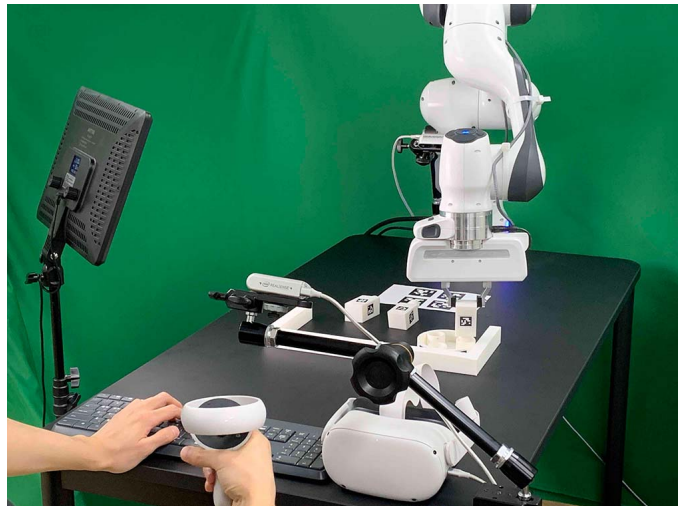
Fig. 16: **Teleoperation setup.** Our system supports Oculus Quest 2 controller and keyboard to control a 7-DoF robotic arm for demonstration collections.

We collected 219 hours of demonstration data over eight furniture models from two human operators. For each of low and medium initialization levels, we collected 100 demonstrations for `desk, round_table, stool, chair`; 150 demonstrations for `cabinet, lamp, square_table`; and 250 demonstrations for `drawer`. For the high initialization level, we collected 50 demonstrations for each furniture model.

Each timestep of a demonstration consists of RGB-D frames from three cameras, robot action, reward, and AprilTag poses for all furniture parts. In the case of low and medium randomness, a phase completion mask is provided. In addition, demonstrations include metadata, such as furniture model id, errors during data collection, and episode success. The statistics of the collected data can be found in Table II.

To cover diverse configurations, we use a single light panel with various color temperatures and vary their positions and directions over time. We also randomize the front-view camera pose every episode. We plan to extend the diversity of data by collecting demonstrations from multiple different laboratories. The teleoperation setup is illustrated in Figure 16.

B. Data Annotation

During data collection, the human operators corrected wrong reward signals from the automatic reward function, possibly caused by a detection error or occlusion. In addition, the number of completed phases in the low and medium randomness levels are manually annotated. The phase of high randomness is not annotated since the assembly processes are inconsistent across the trajectories depending on the initialization.

Our benchmark features 8 different furniture models, each of which is modeled after an existing piece of IKEA furniture, as shown in Figure 17. Due to the feasibility of a single robotic arm, we modify some furniture pieces so that a robot can assemble them with one hand. We manually 3D modeled all furniture using AUTODESK FUSION 360.

(a) IKEA INGOLF `chair`   (b) IKEA LAMPAN `lamp`   (c) IKEA LACK `square_table`

(d) IKEA LAGKAPTEN `desk`   (e) IKEA GURSKEN `drawer`   (f) IKEA HAVSTA `cabinet`

(g) IKEA INGATORP `round_table`   (h) IKEA KYRRE `stool`

Fig. 17: **Furniture** 3**D models.** IKEA model furniture (left), 3D furniture model (middle), and 3D printed furniture model (right).

Each furniture model introduces unique interactions and different levels of challenges. Figure 18 illustrates detailed assembly processes, rewards, and phases for all furniture models. The summaries of the assembly processes and challenges are explained below:

- `lamp`: The robot needs to screw in a light bulb and then place a lamp hood on top of the bulb. The robot should perform sophisticated grasping since the bulb can be easily slipped when grasping and screwing due to the rounded shape.
- `square_table`: The robot needs to screw four table legs to a tabletop. As more legs are attached, they are more likely to collide with assembled table legs. Thus, careful control to avoid collisions is necessary to solve the task.
- `desk`: This task is similar to `square_table`; but the legs are larger and longer, making it possibly more challenging as the robot needs to more precise positioning and orienting its hand before grasping.
- `drawer`: The robot needs to insert and push two drawer boxes into a drawer. The careful alignment of the rails on the drawer is necessary to push the boxes successfully.
- `cabinet`: The robot must first insert two doors into the poles on each side of the body. Next, it must lock the top, so the doors do not slide out. This task requires careful control to align the doors and slide into the pole. Moreover, diverse skills such as flipping the cabinet body, and screwing the top are also needed to accomplish the task.
- `round_table`: The robot should assemble one rounded tabletop, rounded leg, and cross-shaped table base. The robot should handle an easily movable round leg and cross-shaped table base, which requires finding a careful grasping point.
- `stool`: The task consists of one round seat and three tilted legs. Since the legs are not straight, the screwing requires a good grasping pose to assemble the parts robustly. Moreover, finding a collision-free path is necessary since the legs are close to each other when they are assembled.
- `chair`: The task consists of one chair seat, chair back, table legs, and chair nuts (6 parts total). This task requires an accurate and complex interaction with many parts (e.g., screwing, insertion) and diverse strategies in every assembly stage.
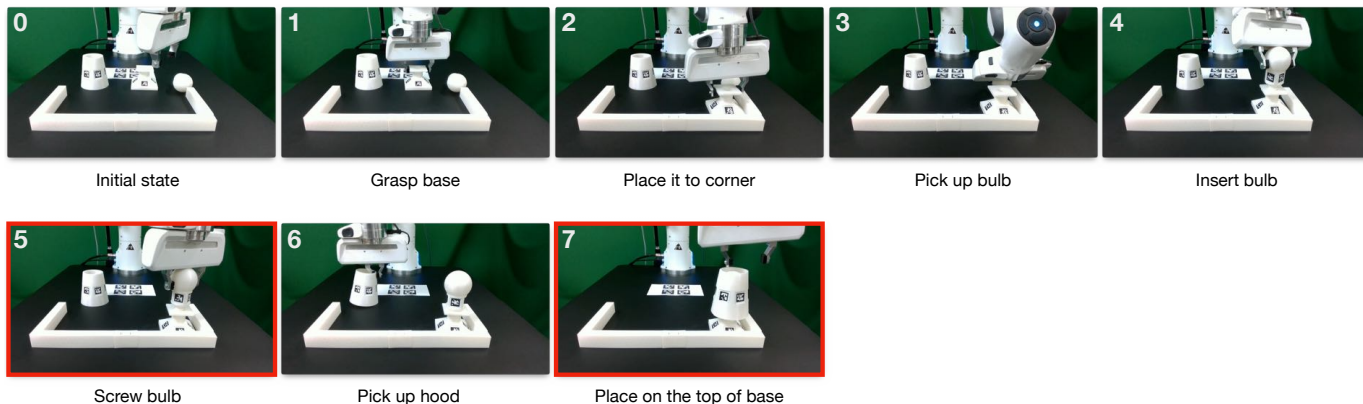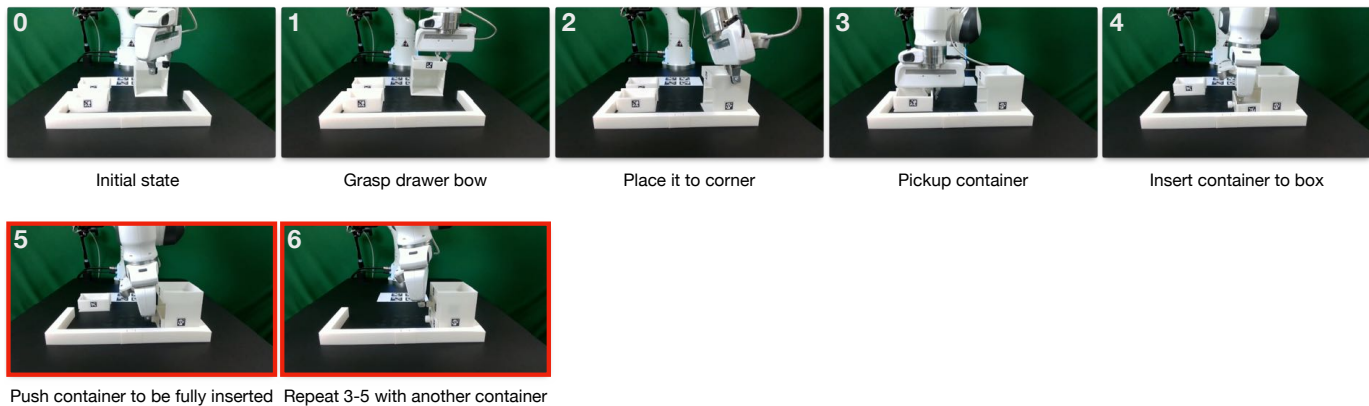
(a) Assembly procedure of `lamp`
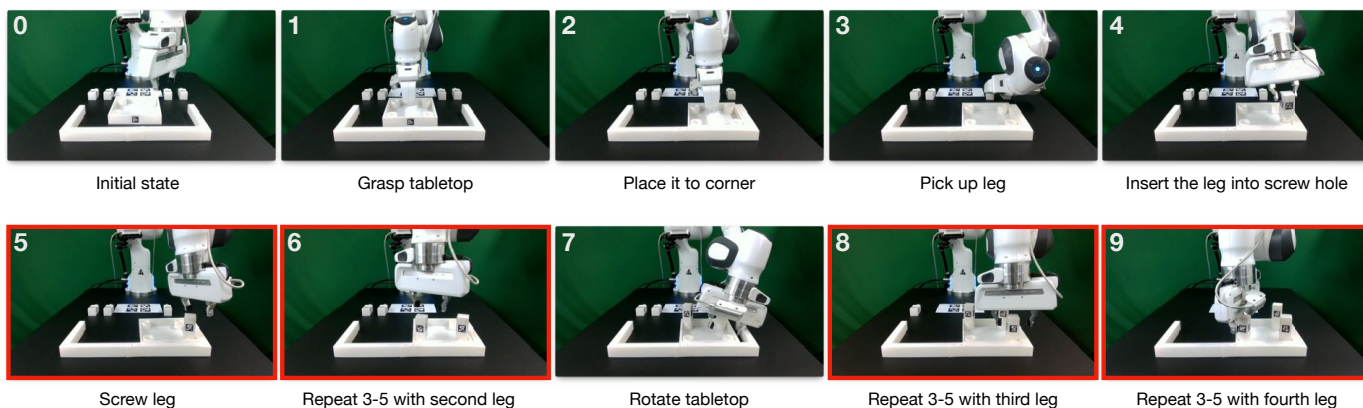
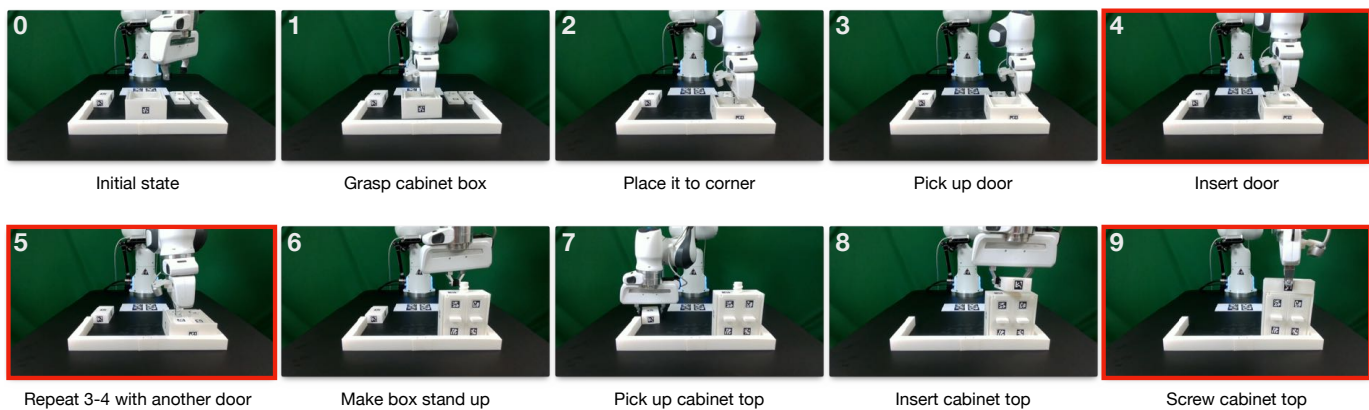(b) Assembly procedure of `square_table`
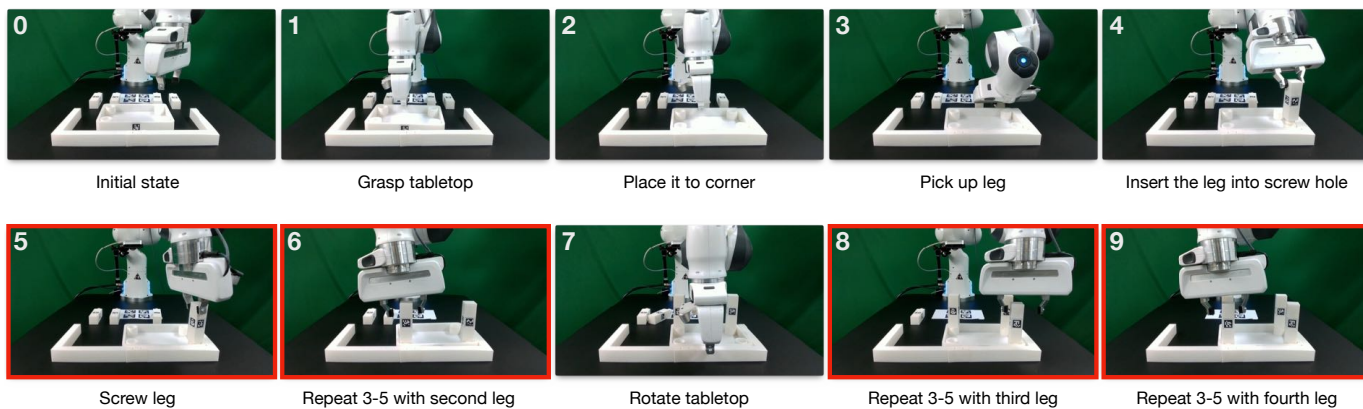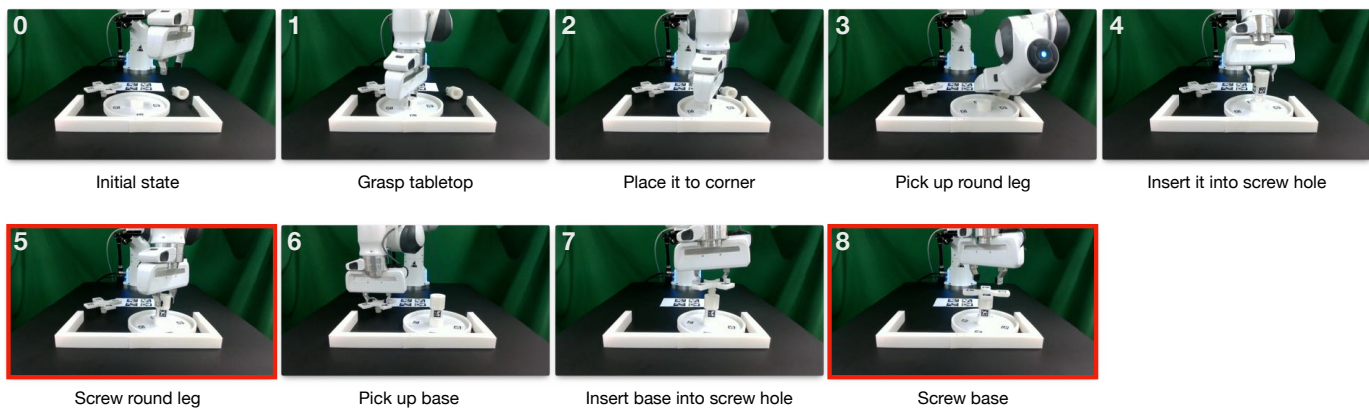

(c) Assembly procedure of `desk`

Fig. 18: **Assembly procedures**: (a) `lamp` (b) `square_table` (c) `desk`. Numbers on the top left corners specify the phase, and red boxes indicate when a reward signal is provided.

Initial state — Grasp drawer bow — Place it to corner — Pickup container — Insert container to box

Push container to be fully inserted — Repeat 3-5 with another container

(d) Assembly procedure of `drawer`

Initial state — Grasp cabinet box — Place it to corner — Pick up door — Insert door

Repeat 3-4 with another door — Make box stand up — Pick up cabinet top — Insert cabinet top — Screw cabinet top

(e) Assembly procedure of `cabinet`

Initial state — Grasp tabletop — Place it to corner — Pick up round leg — Insert it into screw hole

Screw round leg — Pick up base — Insert base into screw hole — Screw base

(f) Assembly procedure of `round_table`

Fig. 18: **Assembly procedures**: (d) `drawer` (e) `cabinet` (f) `round_table`. Numbers on the top left corners specify the phase, and red boxes indicate when a reward signal is provided.
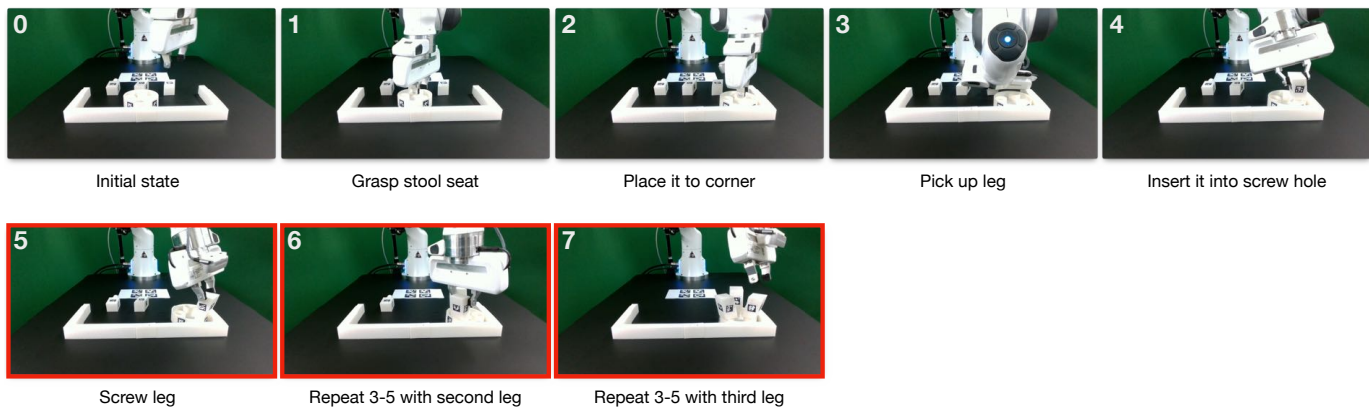
(g) Assembly procedure of `stool`
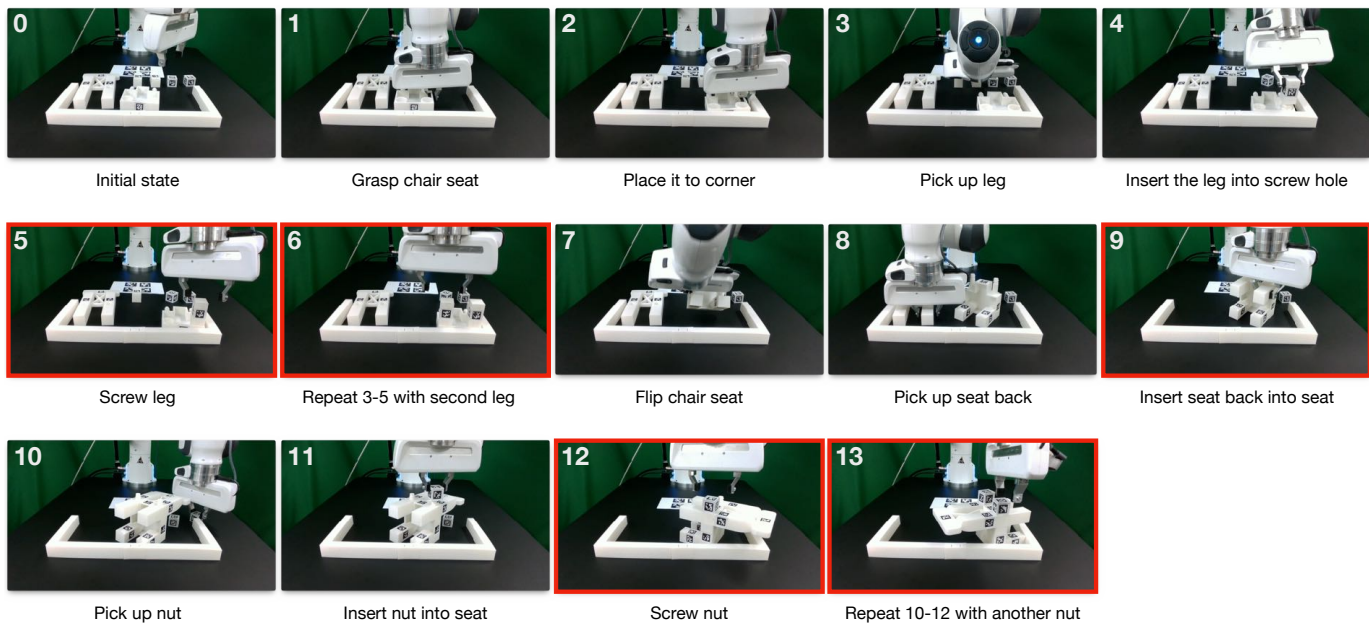
(h) Assembly procedure of `chair`

Fig. 18: **Assembly procedures**: (g) `stool` (h) `chair`. Numbers on the top left corners specify the phase, and red boxes indicate when a reward signal is provided.

The blueprints of furniture models are shown in Figure 19. Some small or thin parts are designed to be thicker than 3 cm to afford space for an AprilTag marker. The largest dimension is smaller than 21 cm, so that our furniture models are compatible with most 3D printers. The potential grasping regions are at most 6 cm (lamp) wide, which is smaller than the open gripper width (7.0 cm for lamp and 6.5 cm for other furniture). In our experiments, the 3D models are printed using a FLASHFORGE GUIDER IIS 3D printer and white PLA filaments. Printing a small furniture model takes approximately 10 hours, while large furniture models, such as desk, need two passes of 3D printing, which takes about 24 hours.

(a) lamp   (b) square_table   (c) drawer

(d) cabinet   (e) round_table   (f) desk

(g) stool   (h) chair
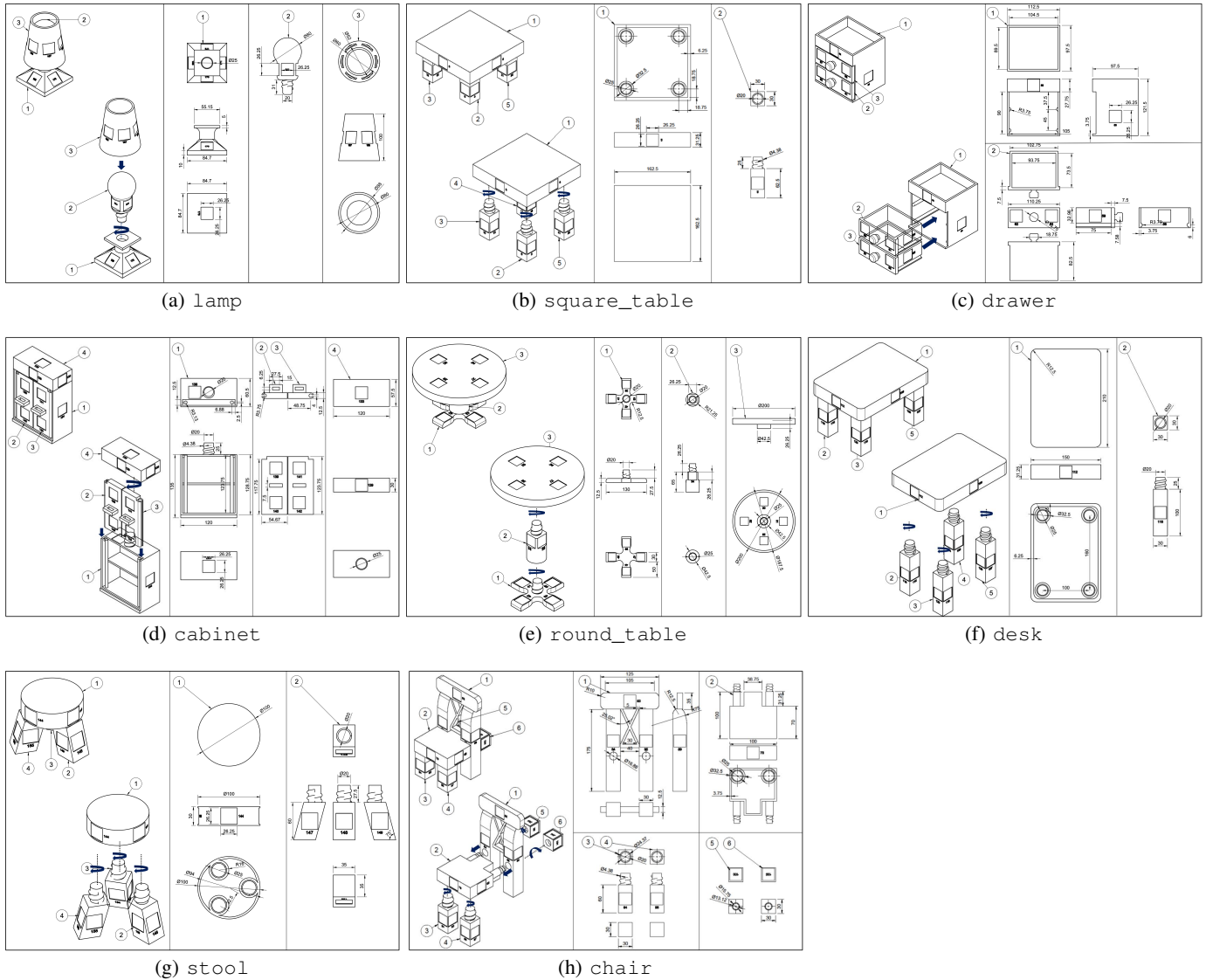
Fig. 19: **Blueprints of furniture models.** The leftmost column shows the final configuration (top) and how furniture parts are assembled (bottom). The rest of the columns illustrate the dimensions of all furniture parts.

APPENDIX G

QUALITATIVE RESULTS

In Figure 20, we present the qualitative results of the evaluated methods, BC and IQL, with their best-performing trials on the low or medium randomness levels. The trained agents are generally capable of completing the first pick and place phases but struggle with more challenging tasks, such as insertion. More videos can be found in our website
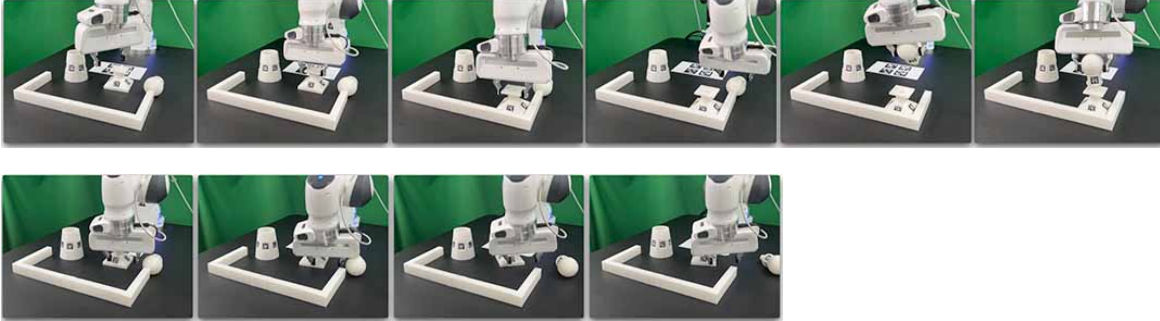
(a) **Qualitative results on `lamp`.** (top) IQL successfully inserts the bulb into the lamp base, but it stops moving afterward. (bottom) BC struggles to grasp the lamp base due to small grasping points.
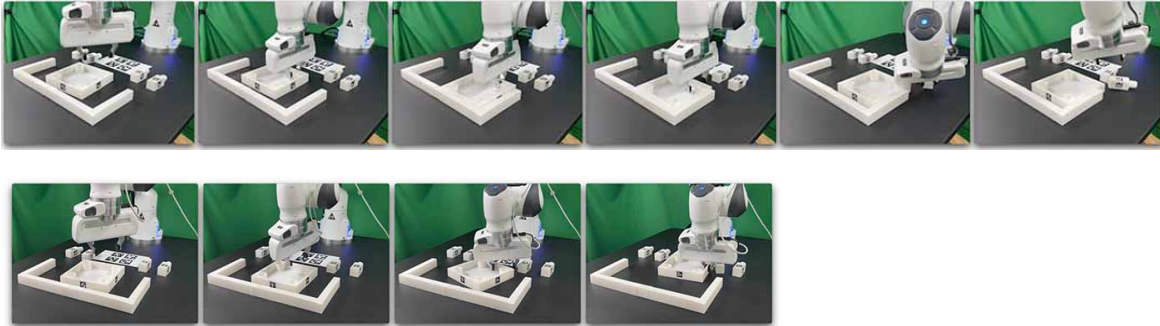
(b) **Qualitative results on `square_table`.** (top) IQL places the tabletop correctly, picks up the leg, but fails the insertion. (bottom) BC grasps the tabletop but fails to place it to the corner.
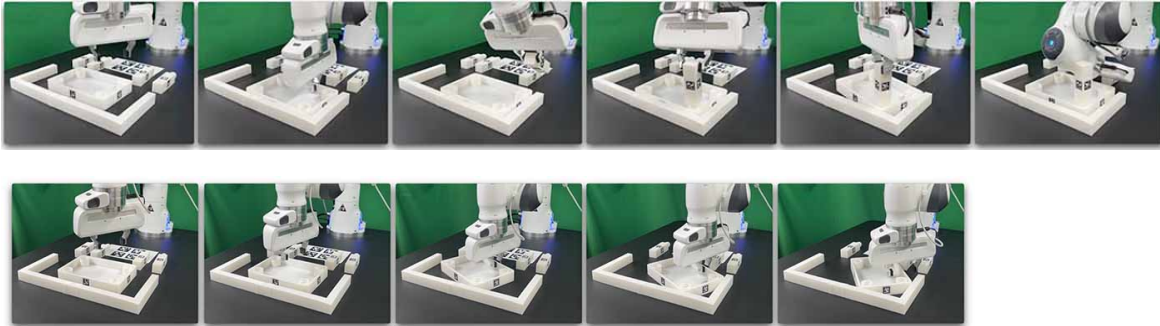
(c) **Qualitative results on `desk`.** (top) IQL screws one leg and then grasps the other. (bottom) BC places the desk top in the corner but stops the motion after releasing the gripper.

Fig. 20: **Sequences of policy evaluation**: (a-b) lamp (c-d) square_table (e-f) desk
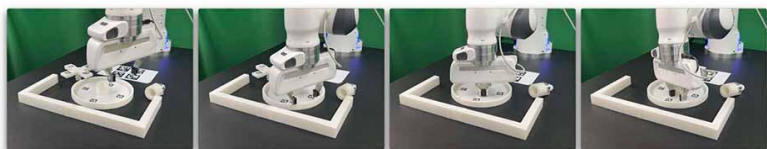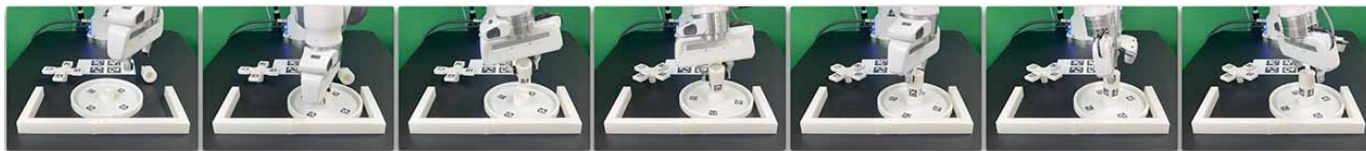
(d) **Qualitative results on `round_table`.** (top) IQL repeats the screwing even after the screw is tightened. (bottom) BC fails to grasp the round table top due to the small grasping region.
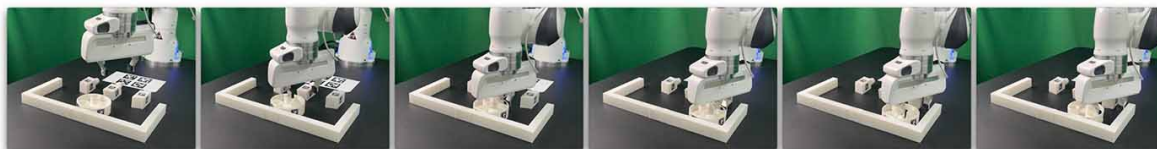
(e) **Qualitative results on `stool`.** IQL tries to screw the leg but fails due to the tilted shape. (bottom) BC pushes the seat to the corner but stops moving.
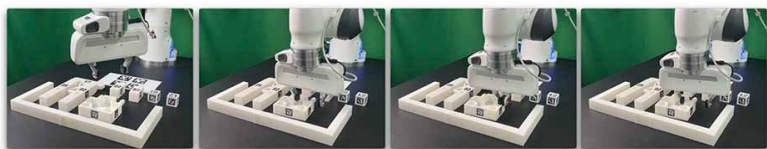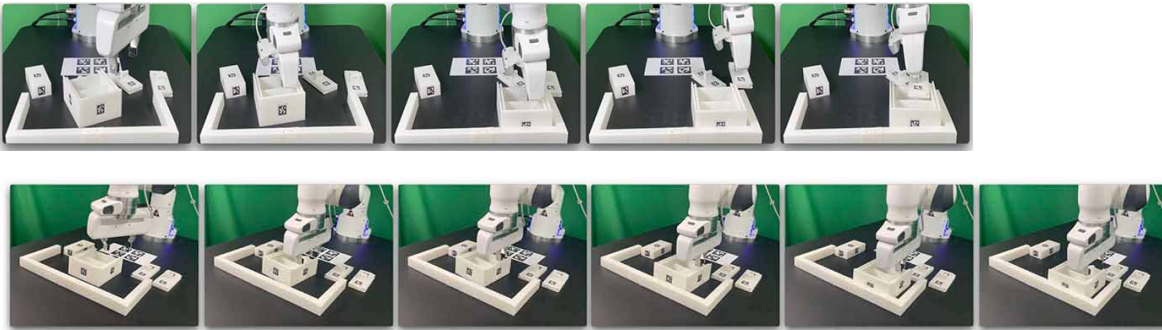

(f) **Qualitative results on `chair`.** (top) IQL insertion fails due to misalignment of the hole. (bottom) BC fails to align the chair seat to the corner.

Fig. 20: **Sequences of policy evaluation**: (g-h) `round_table` (i-j) `stool` (k-l) `chair`
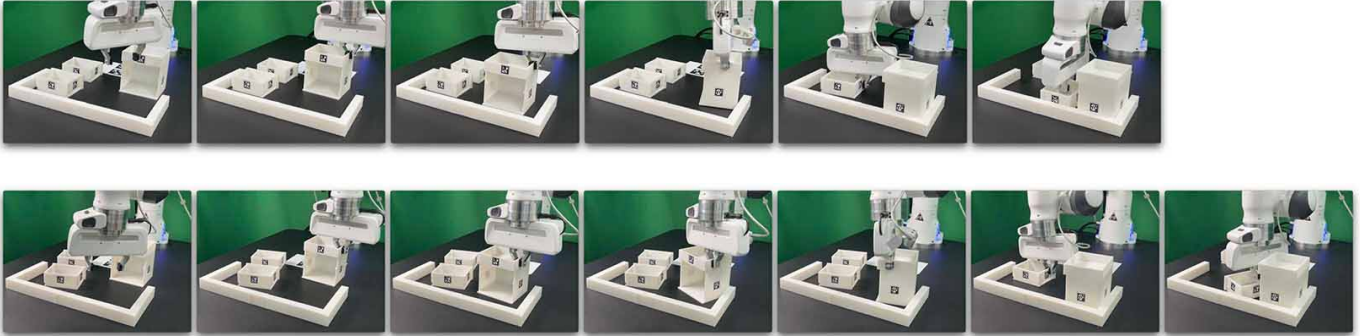
(g) **Qualitative results on `drawer`.** (top) IQL fails on inserting the drawer box. (bottom) BC fails on inserting the drawer box.

(h) **Qualitative results on `cabinet`.** (top) IQL inaccurately inserts the door door. (bottom) BC halts after placing the body in the corner.

Fig. 20: **Sequences of policy evaluation**: (a-b) `lamp` (c-d) `square_table` (e-f) `desk` (g-h) `round_table` (i-j) `stool` (k-l) `chair` (m-n) `drawer` (o-p) `cabinet`. The trajectories are selected with the best-performing rollout in the low or medium randomness level.

This section elaborates on how to reproduce our FurnitureBench step by step (online version).

Please read and follow this instruction thoroughly. Examining the photos and text thoroughly is crucial for creating a reproducible environment. Attention to detail plays a vital role in ensuring precision in this process.

A. Environment Overview
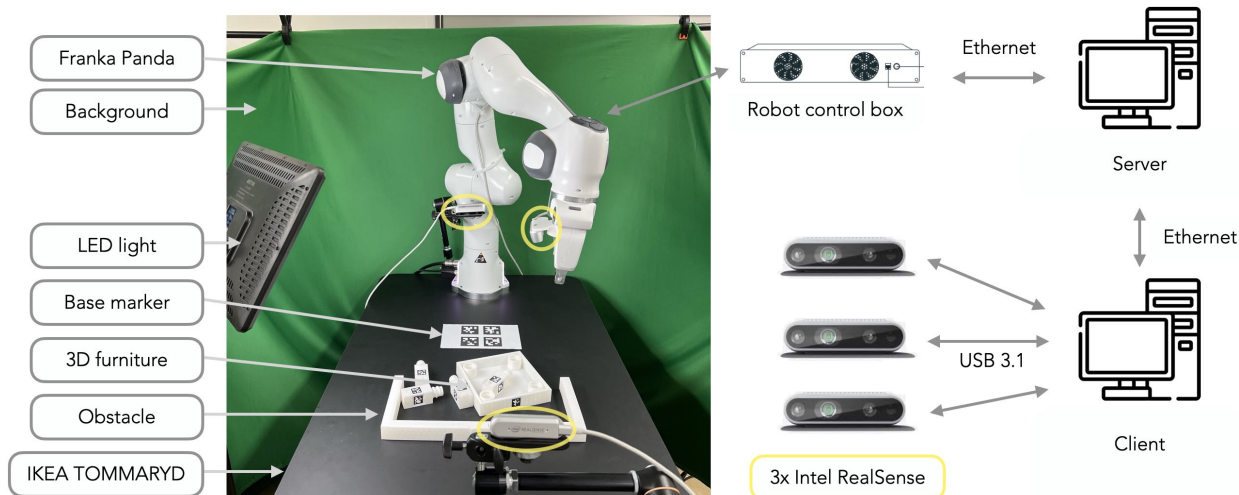
Fig. 21: **FurnitureBench environment.** We illustrate our environment (left) and their connections to server and client computers (right). The client handles I/O, motion planning, and policy inference while the server manages real-time robot control.

B. Mount Robot on Table

The first step is mounting the robot on the table. To place the robot precisely, follow the step-by-step instructions:

1) As shown in Figure 22a, attach marking tapes to the robot body to specify its center. Make sure that the tape's left edge is in the center of the triangle sticker. The tape must be attached straight from the front view.
2) Align one ruler so that $0\,\text{cm}$ is at the left edge of the table and extend it straight, as shown in Figure 22b.
3) Place the robot center (indicated by the left edge of the tape) at point $34.5\,\text{cm}$ of the ruler, as shown in Figure 22c.
4) Make sure the robot is tightly attached to the side of the table, with no room left between it and the table. To double-check, make sure that both rear support pads are closely pressed against the edge of the table, as shown in Figure 22d.
5) Firmly attach the robot to the table by tightly screwing the robot mount, as shown in Figure 22e.
6) Remove the affixed tape from the robot once this step is completed.

Important Checklist:
- Make sure the robot is installed at $34.5\,\text{cm}$ off from the left edge of the table.
- The robot should be tightly attached to the table **without margin**.
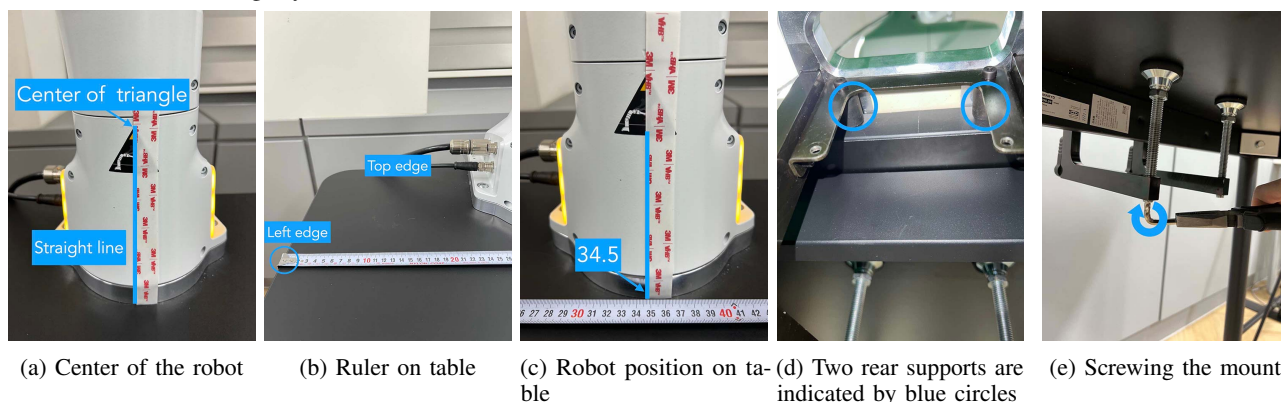- The robot mount is tightly screwed.

(a) Center of the robot   (b) Ruler on table   (c) Robot position on table   (d) Two rear supports are indicated by blue circles   (e) Screwing the mount

Fig. 22: **Robot placement.**

C. Install Background

For consistent background across different lab environments, cover the background with a green backdrop.

1) Clamp the left side of the backdrop, as shown in Figure 24a. Make sure to leave some extra cloth to cover the left side.
2) Similarly, clamp the right side of the backdrop, as shown in Figure 24b.
3) Place a tripod next to the table, and hang the left side of the backdrop to the tripod, as shown in Figure 24c and Figure 24d.
4) Repeat this process for the right side.
5) The final background should look like Figure 23.
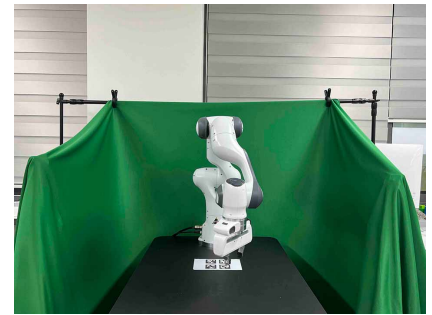
Fig. 23: **Installed Background.**

Important Checklist:

- Make sure there are **minimum wrinkles and shadows** on the cloth.
- Green cloth fully covers the narrow side of the table.
- The cloth covers the left and right edges of the table (at least $1/3$) so that it fully covers the front camera view.
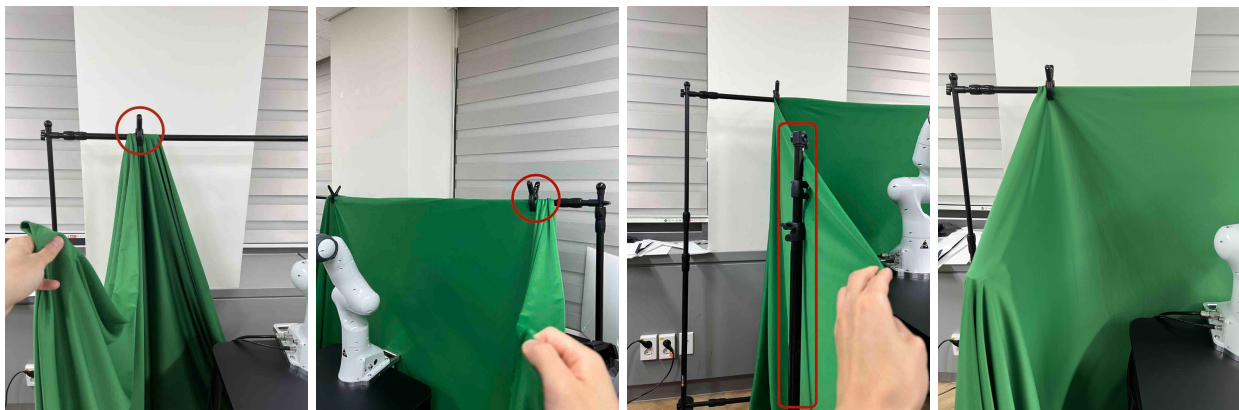
(a) Background left clamp   (b) Background right clamp   (c) Left pole   (d) Left pole covered

Fig. 24: **Background installation.**

D. Place Base AprilTag

The base AprilTag defines the world coordinate system; therefore, the camera will be set relative to this base tag. The position and angle of the base tag are **critical for reproducibility**; thus, the placement of the base tag on the table should be very precise. Be cautious when attaching the AprilTag, as it can easily be attached with tilted angles. Ensure that both the rulers and AprilTag are properly aligned and straight.

1) Align tape ruler so that $0\,\mathrm{cm}$ is at the left of the table and plastic ruler so that $0\,\mathrm{cm}$ is at the top edge of the table, as illustrated in Figure 26a. Make sure the two rulers are perpendicular.
2) Place the center of the base tag at $24.5\,\mathrm{cm}$ horizontally and $37.5\,\mathrm{cm}$ vertically, as shown in Figure 26b, in a correct orientation of the patterns (Figure 26c).
3) Use double-sided tape to affix the base tag. Note that wrinkled paper can cause unreliable detection. Ensure the paper remains flat by attaching it with double-sided tape in all four corners.
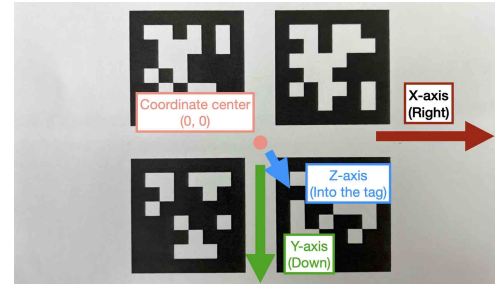

Fig. 25: **Base AprilTag coordinate system.** The arrow indicates a positive direction. The coordinate system follows a right-handed convention for rotation.
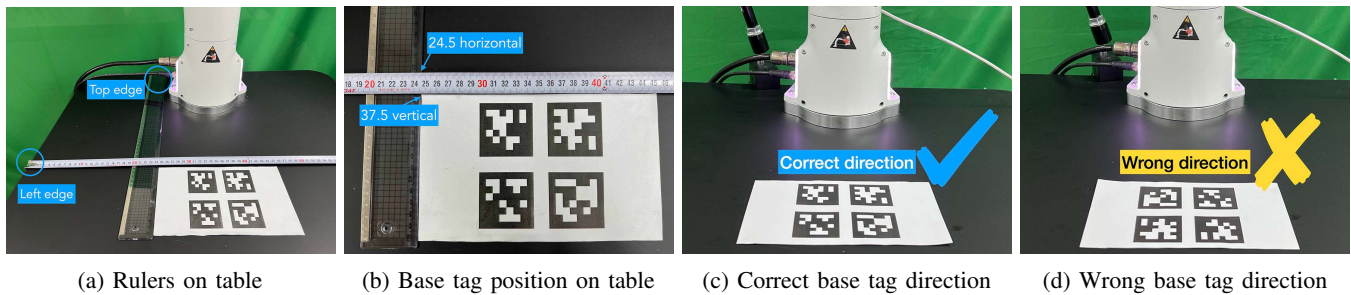
Important Checklist:

- Double-check the base AprilTag in the exact position (like, less than $2\,\mathrm{mm}$ error).
- The base AprilTag is firmly attached flat without wrinkles.
- Check the pattern of the base tag to ensure its correct direction.

| (a) Rulers on table | (b) Base tag position on table | (c) Correct base tag direction | (d) Wrong base tag direction |

Fig. 26: **Base AprilTag.**

E. Install Front and Rear Cameras

Our system requires three cameras: front, rear, and wrist cameras. Prior to installation, determine the specific camera to be used for each view, and write down the serial numbers of the cameras for wrist, front, and rear cameras, as they will be required for subsequent steps. The serial number is written on the bottom of the camera, as shown in Figure 27.

First, install the front and rear cameras. You can utilize any camera mount product for the front camera mount if they can match the same camera view with our original setup. However, we highly recommend users use a camera mount from either Ulanzi or Manfrotto, both of which we have confirmed to be reliable. We explain how to install the front camera using these two camera mounts, although you can further adjust it during fine-grained calibration in Section H-J.

Fig. 27: **Camera serial number** written on the bottom of the camera.

- **Option 1, Ulanzi:** Clamp the front camera mount to the right side of the table, as shown in 28a. Position the camera mount 8 cm away from the table edge, as shown in 28b. While measuring the distance, ensure the camera mount's base is firmly attached, as illustrated in 28c.
- **Option 2, Manfrotto:** Clamp the front camera mount to the right side of the table. The camera bracket needs to be affixed using the left hole and the locking wheel should be oriented outward, as shown in Figure 28d. Position the camera mount 7 cm away from the table edge, as shown in Figure 28e. Arrange the deeper section to face the inside to provide better flexibility in camera movement. During the distance measurement, make sure that the camera mount's base is firmly attached and valves are securely fastened, as shown in Figure 28f.
- Place the camera approximately in the center (horizontally) of the table and orient it to face the base AprilTag. You will fine-tune its pose in a later section.
- Connect the front camera to **client** computer using a USB **3.1** cable.
- Clamp the rear camera mount next to the robot base, as shown in Figure 28g. Plug USB **3.1** cable. Utilize a cable tie to fasten the pair of cables from the robot and the single cable from the camera. Ensure a sufficient gap between the camera mount and the robot to avoid any collision.

(a) Ulanzi position   (b) Ulanzi distance   (c) Ulanzi attacment

(g) Rear camera installation

(d) Manfrotto position   (e) Manfrotto distance   (f) Manfrotto attachment
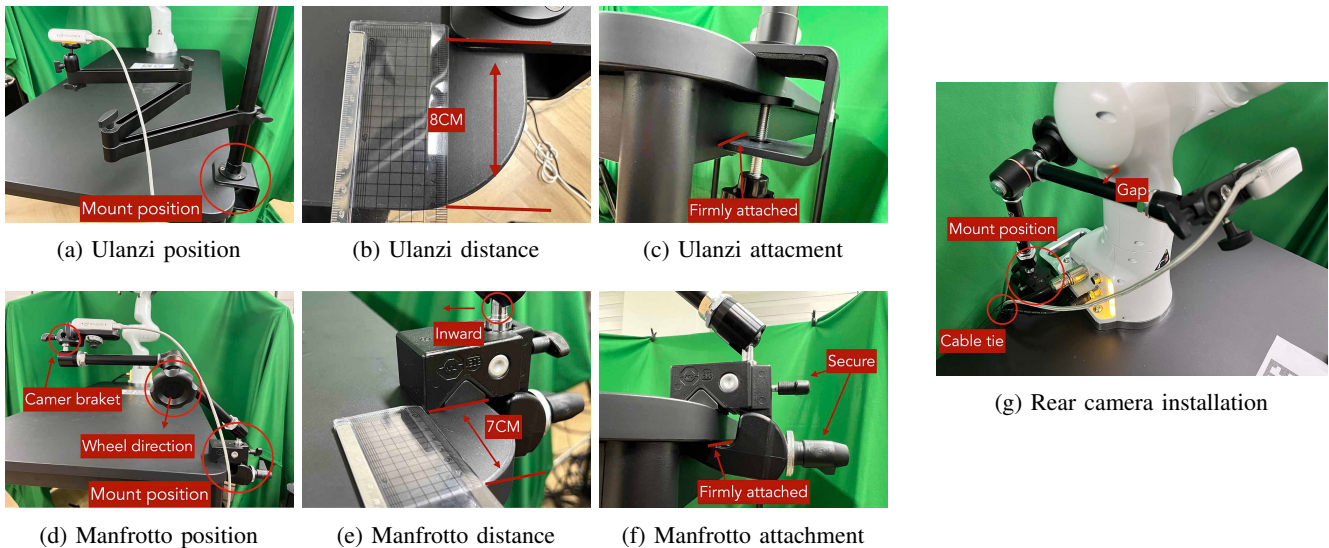
Fig. 28: **Front and rear camera installation** with Ulanzi and Manfrotto camera mounts.

F. Install Wrist Camera

Now, we install the wrist camera as follows:

1) Install the wrist camera on the robot wrist using the 3D printed camera mount. Take note of the direction in which the RGB camera should face: it should be aimed toward the gripper's tip, as shown in Figure 29b. The camera should be positioned on the rear side of the end-effector. Take a look at Figure 29c and Figure 30 to gain a clear understanding of its placement.
2) Connect the wrist camera to **client** computer using a USB **3.1** cable.
3) Fasten the cable to the robot arm with three cable ties, as shown in Figure 30. Ensure to provide additional slack in the cable, allowing the robot to move without any tension from the cable. Trim the surplus length from the cable ties to ensure no extra material remains.

Important Checklist:

- Ensure the direction of the wrist camera is correctly set; the camera is positioned on the end-effector's back side, and the cable is plugged to the left when viewed from the rear.
- Firmly attach the camera and camera mount to the robot by tightening the screws.
- Three cable ties are fastened as shown in Figure 30.
- The cable has additional slack.
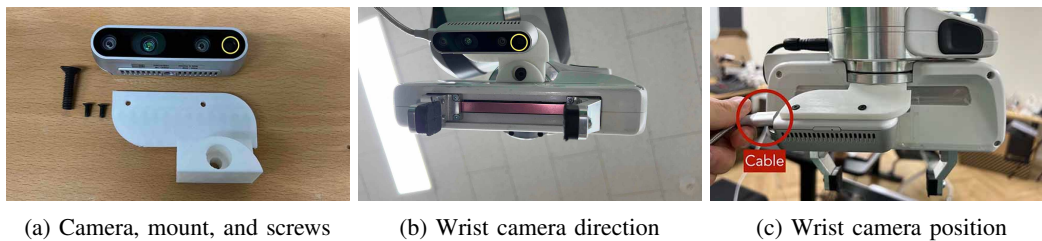- The surplus length from the cable ties is trimmed.

(a) Camera, mount, and screws   (b) Wrist camera direction   (c) Wrist camera position

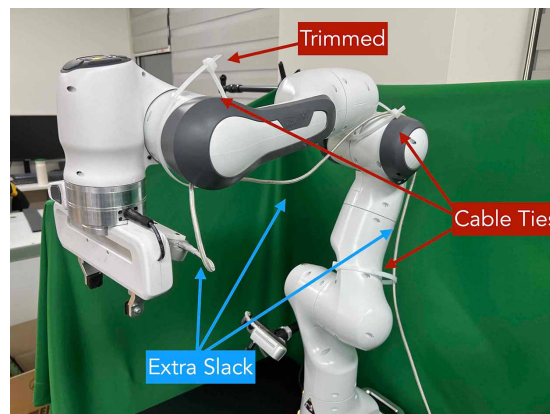Fig. 29: **Wrist camera installation.** The yellow circles represent the RGB camera.

Fig. 30: **Cable organization.**

G. Install Software

Skip this section if you already installed the software.

Before moving on to the following installation steps, install our software stacks on server and client computers. More detailed instructions can be found in our online documentation.

Client computer:

1) Install Franka Control Interface (FCI) version 4.2.2.
2) Install nvidia-docker2.
3) Clone our code and pull our docker image:

```
1  git clone https://github.com/clvrai/furniture-bench.git
2  docker pull furniturebench/client-gpu:latest
```

4) Set environment variables:

```
1  # Set path to the furniture-bench repo, e.g., /home/<username>/furniture-bench
2  export FURNITURE_BENCH=</path/to/furniture-bench>
```

5) Run client docker container:

```
1  ./launch_client.sh --gpu --pulled
```

Server computer:

1) Install real-time kernel.
2) Clone our code and pull our docker image:

```
1  git clone https://github.com/clvrai/furniture-bench.git
2  docker pull furniturebench/server:latest
```

3) Set environment variables:

```
1  # Set path to the furniture-bench repo, e.g., /home/<username>/furniture-bench
2  export FURNITURE_BENCH=</path/to/furniture-bench>
```

H. Set Up Connection

Server, client, and robot communicate through a local Ethernet network, as shown in Figure 31. To establish connections to server and cameras, the following environment variables need to be set in client container:

```
1  export SERVER_IP=<IP of the server> # e.g. 192.168.0.138
2  export CAM_WRIST_SERIAL=<serial number of the wrist camera>
3  export CAM_FRONT_SERIAL=<serial number of the front camera>
4  export CAM_REAR_SERIAL=<serial number of the rear camera>
```

Ensure that all the cameras are correctly installed and appropriately connected. Execute the following command and confirm the items in the checklist.

```
1  python furniture_bench/scripts/run_cam_april.py
```
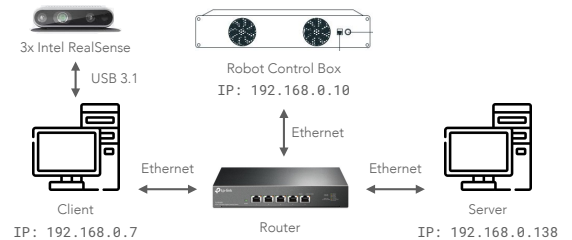

Fig. 31: **Example network setup.** The IP addresses need to be adjusted according to your local network.

Fig. 32: **Camera observations.**

Important Checklist:

- Ensure that the camera displays the wrist, front, and rear views in left-to-right order, as shown in Figure 32.
- The wrist camera view must observe both gripper tips as shown in the left figure of Figure 32.
- The rear camera should be able to detect the two markers present on the base tag, as shown in the right image of Figure 32.

I. Launch and reset the robot

In this section, we provide a guideline for operating and moving the robot to the predetermined position. To operate the robot, unlock the robot, activate FCI (Franka Control Interface), and launch a server-side daemon as explained below:

1) Access the control interface website. You should be able to see a screen like in Figure 33.
2) Unlock the robot in the Franka Emika web interface, as shown in Figure 33a
3) Release the activation button (Figure 33b). The light on the robot should turn blue after releasing the button.
4) Activate FCI in the web interface, as shown in Figure 33c.

Then, launch a server-side daemon.

1) In the server computer, launch the server Docker image with:

```
/launch_server.sh --pulled
```

2) On server docker container, set the environment variable for server IP by `export ROBOT_IP=<robot_ip>`. For example, `export ROBOT_IP=192.168.0.10`
3) In a *"server"* terminal, launch server daemon with executing:

```
/furniture-bench/launch_daemon.sh
```

If everything is installed and configured correctly, it will alternate between opening and closing the gripper. This is the only program needed to be run on the server. Keep it running while executing commands in the *"client-gpu"* terminal.

Finally, execute the following in a *"client-gpu"* terminal and see the robot moves to the reset pose.
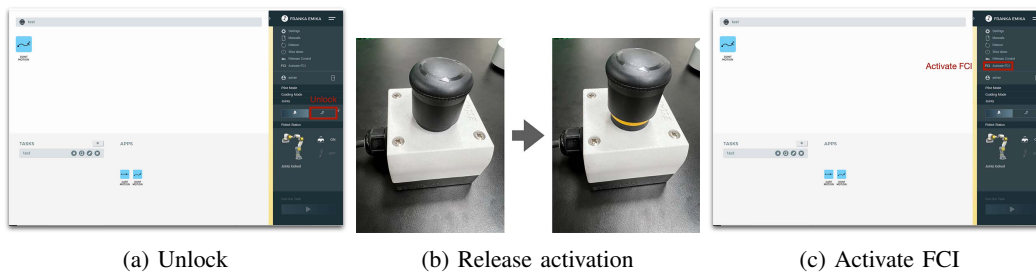
```
python furniture_bench/scripts/reset.py
```


(a) Unlock   (b) Release activation   (c) Activate FCI

Fig. 33: **Activate Franka Panda controller.**

J. Fine-tune Front Camera Pose

We provide a GUI tool to help calibrate the front camera pose with the pre-recorded view overlaid on top of the current camera view, as shown in Figure 34. The calibration can be achieved by matching the numbers and images shown in our calibration tool.

In the GUI tool, the image from the current view is displayed as a solid layer, while the reference image you need to match appears transparent. The number indicates the deviation of the current camera poses from the desired pose. The red texts indicate that the deviation exceeds the threshold (± 0.004 for the position (pos), ± 0.8 for the rotation (rot)), whereas green texts represent that it is within acceptable the boundary. Refer to Figure 25 for a better understanding of the coordinate system to adjust the camera pose.

1) First, run the following command to move the robot up to prevent it from blocking the camera's view.

```
python furniture_bench/scripts/move_up.py
```
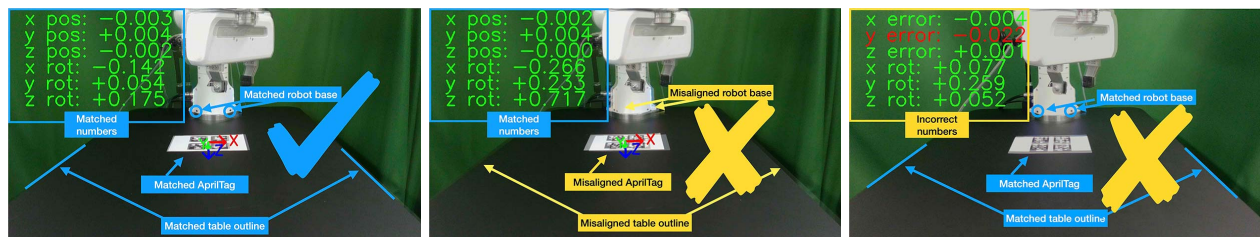
2) Run the camera calibration tool:

```
python furniture_bench/scripts/calibration.py --target setup_front
```

3) Adjust the camera to match **both** **images** and **numbers**.

Here, we list **tips** to simplify the process of matching the camera pose.

- For the Ulanzi camera mount, first adjust its height to match the z position and then fasten it in place.
- When dealing with the Manfrotto camera mount, prioritize matching all settings except for the x position, given that it can be independently modified using the camera bracket.
- In the beginning, ignore the numbers and focus on aligning the table outline and robot base (using the two holes in the robot base as reference points). Take a look at how the matched image looks like in Figure 34a
- Iterative adjust position and rotation to match the alignment and numbers. Based on our experience, it was simpler first to align the position and then adjust the rotation minutely for best alignment.

(a) Correct numbers and images   (b) Correct numbers, incorrect images   (c) Correct images, incorrect numbers

Fig. 34: **Camera calibration.** The numbers in the image indicate the deviation of the camera pose from desired camera pose. The green and red text indicates whether the camera pose is within the threshold or not.

Important Checklist:

- All numbers on the screen should turn green.
- The boundary of the table and the base AprilTag must be aligned with the pre-recorded image.
- The position of the robot base (i.e., two holes) should exactly match the pre-recorded image.

K. Install Obstacle

The 3D-printed obstacle can be attached to the table using double-sided rubber tape. The exact pose of the obstacle can be viewed using our calibration tool, as shown in Figure 35.

1) Install the obstacle with the guidance of the provided GUI tool:

```
python furniture_bench/scripts/calibration.py --target obstacle
```

2) Attach the obstacle to the table while aligning it with the pre-recorded obstacle pose.
3) Affix the obstacle with double-sided rubber tape, as shown in Figure 36.
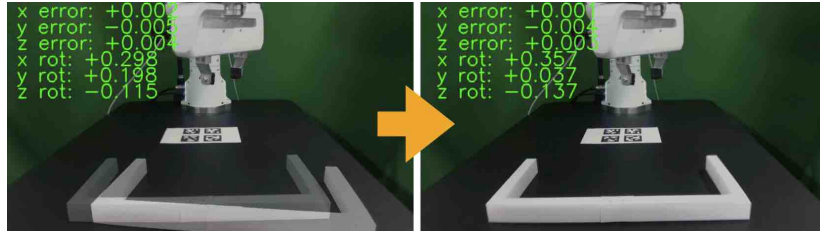
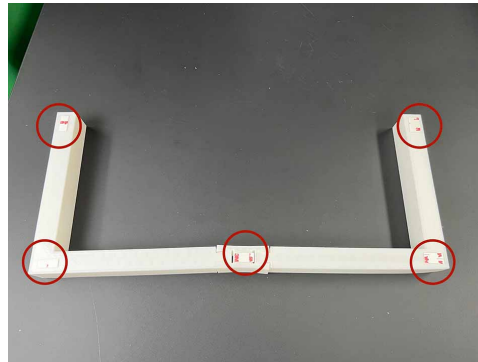Fig. 35: **Obstacle installation guidance tool.**

Fig. 36: **Affix obstacle.** The red circles represent where to attach the double-sided rubber tape. Make sure the obstacle does not move when pushed.

Important Checklist:
- Adjust the obstacle to identically match the transparent one in the GUI tool, as shown in Figure 35 on the right. There should be no discrepancy.
- Firmly attach the obstacle using double-sided rubber tape to prevent it from moving when pushed.

L. Set Up Light

During the data collection process, we randomize the light temperature between $4600\,\mathrm{K}$-$6000\,\mathrm{K}$ as well as the intensity, position, and direction of the light. On the other hand, during the evaluation process, it is essential to maintain lighting conditions as similar as possible. In order to accomplish this, the light should be placed on the left side of the table as shown in Figure 21. Furthermore, the temperature range of $4600\,\mathrm{K}$ to $6000\,\mathrm{K}$ and the brightness range of $500\,\mathrm{lm}$ to $1000\,\mathrm{lm}$ should be set for the lighting panel.

M. Test the Environment

To confirm that the environment setup has been done correctly, conduct a test run using a pre-trained policy for the `one_leg` assembly task. The `one_leg` assembly consists of phases: (1) pick up the tabletop, (2) push to the corner, (3) pick up the leg, (4) insert the leg, and (5) screw the leg. The assessment outcomes can be contrasted with the initial environment depicted in Figure 5. The pre-trained policy is expected to complete over 3 phases on average (such as picking up the leg and occasionally inserting it), achieving a 15-30% success rate in the entire `one_leg` assembly task.

1) Before evaluation, double-check the camera calibration using the following script. All the numbers should be green and the robot base, obstacle, and base tag should be aligned accurately:

```
python furniture_bench/scripts/calibration.py --target one_leg
```

2) Confirm the above
3) Green backdrop cloth has minimum wrinkles.
4) Wipe three camera lenses using a lens cloth, as they may be blurry from fingerprint smudges.
5) Install requirements for the evaluation:

```
pip install -r implicit_q_learning/requirements.txt
pip install -e r3m
pip install -e vip
```

6) Place the furniture components randomly within the workspace, as shown in Figure 37.
7) Evaluate the pre-trained policy using the following script:

```
./evaluate.sh --low --one_leg
```

8) The above command will show a GUI tool and prompt to indicate where furniture parts should be positioned. Initialize the furniture parts, as shown in Figure 38a. The screen will prompt "initialization done" when everything is correctly aligned, as shown in Figure 38b.
9) Once the initialization is done, press "Enter" to execute the policy. Make sure there is nothing but furniture parts in the workspace.
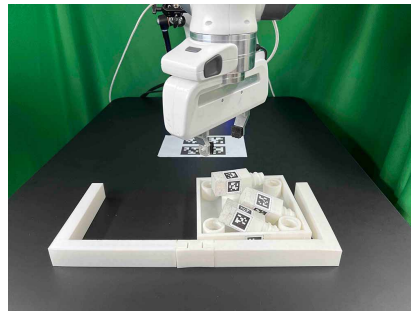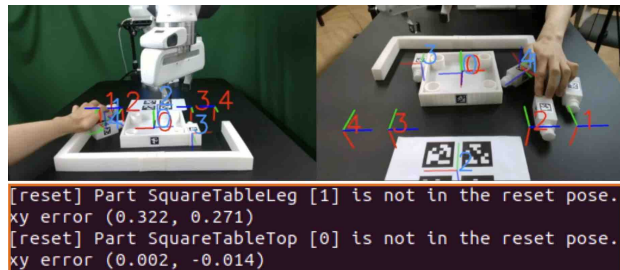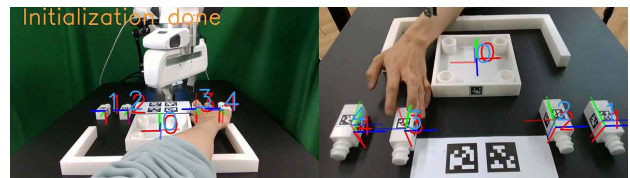

Fig. 37: **Random furniture placement before accurate task initialization.**

(a) GUI tool and prompt indicate where to place each part.

(b) Once "Initialization done" is prompted, press "Enter" to proceed and execute the policy.

Fig. 38: **Task initialization process using the GUI tool.**