An Open-Source Multi-Goal Reinforcement Learning Environment for Robotic Manipulation with Pybullet (Supplementary)*

Xintong Yang[1][0000−0002−7612−614X], Ze Ji[1][0000−0002−8968−9902], Jing Wu[2][0000−0001−5123−9861], and Yu-Kun Lai[2][0000−0002−2094−5680]

[1] Centre for Artificial Intelligence, Robotics and Human-Machine Systems (IROHMS), School of Engineering, Cardiff University, Cardiff, UK
{yangx66, jiz1}@cardiff.ac.uk

[2] School of Computer Science and Informatics, Cardiff University, Cardiff, UK
{wuj11, laiy4}@cardiff.ac.uk

1 Multi-step tasks details

Initial state distribution[3]: Fig 1a gives a visualisation. For the block_stack and block_rearrange tasks, blocks' positions are sampled randomly within a square of 0.03 width, centred at the gripper's x-y coordinates (red line). For the chest_push and chest_pick_and_place tasks, they are sampled within a rectangle of length 0.04 and width 0.03, centred at the location 0.05 away from the gripper's x-y coordinates, nearer to the robot base (blue line). The position of the chest is randomly sampled on a 0.03 line, 0.07 away from the robot base (brown line). The initial pose of the gripper tip frame is fixed. For pushing tasks, it is at the centre of the table surface, while for picking tasks, it is 0.075 above the table centre (green line). The difference of the initial gripper tip location is for easier exploration, a design inherited from the OpenAI Gym multi-goal tasks[4].

State representation: All tasks share the same state representation for the Kuka robot and n blocks, $\mathbf{s} = \{\mathbf{s}_{robot}||\mathbf{s}_{block\_1}||...||\mathbf{s}_{block\_n}\}$ ('||' denotes vector concatenation). The robot state includes the Cartesian coordinates and linear velocity of the gripper tip in the world frame, the finger width and velocity in the world frame. That is

$$\mathbf{s}_{robot} = (x_{grip}, y_{grip}, z_{grip}, vx_{grip}, vy_{grip}, vz_{grip}, w_{finger}, v_{finger})$$

The state of each joint is only included if joint space control is used. A block state includes its Cartesian coordinates and Euler orientation angles in the world

* The authors thank the China Scholarship Council (CSC) for financially supporting Xintong Yang in his PhD programme (No. 201908440400).

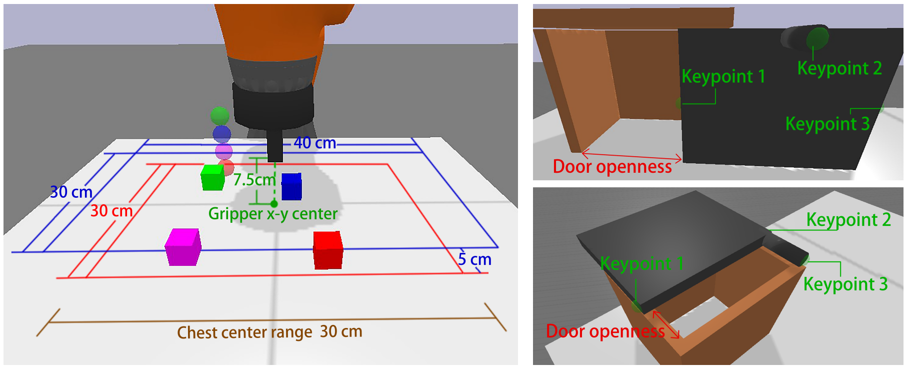
[3] All length values used in the environment are in metres, angles in radians, unless otherwise specified.

[4] This was not specified in the paper, but we found it in the source codes of the package.

frame, its relative position, relative linear velocities and angular velocities with respect to the gripper tip. That is (for the n-th block),

$$\begin{aligned}\mathbf{s}_{block\_n} = (&x_{block\_n}, y_{block\_n}, z_{block\_n}, roll_{block\_n}, pitch_{block\_n}, yaw_{block\_n},\\ &x^{rel}_{block\_n}, y^{rel}_{block\_n}, z^{rel}_{block\_n}, vx^{rel}_{block\_n}, vy^{rel}_{block\_n}, vz^{rel}_{block\_n},\\ &vroll^{rel}_{block\_n}, vpitch^{rel}_{block\_n}, vyaw^{rel}_{block\_n})\end{aligned}$$

For tasks involving a chest, the chest state includes the Cartesian coordinates of the three green keypoints attached on the door and how wide the door is opened (see Fig. 1b).



(a) Initial state distribution

(b) Chest state representation

Fig. 1. (a) Initial state distribution. Red line: initial block position range for tasks without a chest. Blue line: initial block position range for tasks with a chest. Brown line: initial position range for the centre of the chest. Green dot: initial gripper tip position for the BlockRearrange and ChestPush tasks, the point 0.075 above is the initial position for the other two tasks. (b) Chest state representation: the Cartesian coordinates of the three keypoints (green dots) and the width of the door gap (red line). One keypoint is at the end of the door handle, the other two are on the two edges of the door.

Action space: All actions are multi-dimensional and continuous in $[-1, 1]$ as a common definition for continuous control reinforcement learning. For Cartesian space control, an action is mapped to the changes of the gripper tip coordinates in the world frame by multiplying with 0.05[5]. For joint space control, an action is mapped to the changes of the joint angle by multiplying with 0.05. The finger width control dimension is mapped to the range of the finger state, such that -1 corresponds to the fingers being fully opened (symmetric fingers). Only

[5] An action of 1.0 at the x element will move the gripper towards the positive x direction for 0.05.

the ChestPickAndPlace and BlockStack tasks require gripper finger control. In Cartesian space control mode, the gripper tip movement is bounded within a box of length 0.4, width 0.3 and height 0.375, placed on the table surface centre. In joint space control mode, the robot arm movement is bounded by the joint limits.

Goal representation and generation: A goal consists of the Cartesian coordinates of all the n blocks in the world frame. If a chest is involved, the goal includes an extra scalar indicating the largest openness of the door. That is

$$\mathbf{g} = (x_{block\_1}, y_{block\_1}, z_{block\_1}, ..., x_{block\_n}, y_{block\_n}, z_{block\_n})$$

A target is constrained to not overlap with the blocks and other targets given a threshold (0.06 by default).

- BlockRearrange: Target block positions (desired goals) are sampled within the same square (red line in Fig. 1a) in which the initial block positions are sampled.
- ChestPush: Target block positions are fixed at the centre of the chest, on the table surface. With a goal achieving distance threshold of 0.1, this means the task is to push the blocks into a sphere of radius 0.05 centred at the chest centre on the table.
- ChestPickAndPlace: This task is the same as the ChestPush task, except that the robot needs to pick and drop the blocks, rather than push.
- BlockStack: This task first samples a random order in which the blocks need to be stacked, then samples a tower position within the initial block position square (red line in Fig. 1a).

Reward function: Every task comes with a dense and a sparse reward function, based on a desired and an achieved goal. The dense reward function outputs the negative Euclidean distance of the two goals. That is

$$r_{dense} = -||\mathbf{g}_{achieved} - \mathbf{g}_{desired}||_2$$

The sparse reward function outputs 0 when a desired goal is achieved and -1 otherwise. A goal is regarded as achieved when the Euclidean distance of the two goals is smaller than or equal to a given threshold δ, ($\delta = 0.05$ by default). That is

$$r_{sparse} = \begin{cases} 0 & ||\mathbf{g}_{achieved} - \mathbf{g}_{desired}||_2 <= \delta \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

Task horizon: The number of timesteps differs based on the number of blocks involved in a task. Every task has a base number of 50 timesteps, and adding one block increases it by 25. For example, a task of stacking two blocks and a task of pushing one block into a chest both provide a task horizon of 75 timesteps.

2 Curriculum details

The simple curriculum used in (main text) section 3.2 starts by generating easy goals and gradually increases difficulty. It first computes the total number of goals to be generated in the whole training process, which effectively equals the total number of episodes. It then separates the total number of training goals evenly into a number of difficulty levels. The difficulty levels for each task are defined as follows.

- BlockRearrange: The number of levels is equal to the number of blocks. The easiest one is to generate only one random target position, and other target positions are made equal to the blocks' positions. In other words, the curriculum starts by asking the robot to push one block, and gradually increases the number of blocks to push.
- ChestPush: The number of levels is equal to the number of blocks plus one. The easiest level is then to open the door of the chest. The following level starts by pushing one block, and increases to all the blocks.
- ChestPickAndPlace: The same as the ChestPush task, except that the robot is asked to pick and drop the blocks, rather than push.
- BlockStack: The number of levels is equal to the number of blocks. The easiest one is to push or pick-and-place the base block to a random position, and other blocks stay unmoved. Each of the following levels then adds one more block to be stacked.

For each episode, the method samples a level of difficulty for the goal to generate. It maintains a record of the number of generated goals from each level. At the beginning of a training process, the easiest level has a sampling probability of 1.0, and other levels have 0 chances. When the number of generated goals of a level passes half the required number of goals to generate, the method sets the probabilities of this level and its next level to 0.5. When a level finishes generating all the goals, its probability is set to 0 and its next level's probability is set to 1.0 if this next level has not passed half the number of goals to generate. The whole process repeats until the last difficulty level is finished, which means the training process finishes.

To reduce unnecessary training time, the task horizon changes based on the current curriculum level. At the lowest level the task has 50 episode timesteps, and going one level up increases it by 25 timesteps. This means the task horizon increases as the agent is given harder and harder goals to achieve.

This method is totally based on human prior and rather simplistic. As demonstrated by the results (main text section 3.2), it does help the algorithm to learn at the beginning from easier goals, but it consistently fails as goals become more difficult. This indicates that a naive curriculum alone is not enough to achieve such long horizon multi-step manipulation tasks in an extremely sparse reward setting.

3 API details

This section explains the meanings of the arguments of the `make_env(...)` function in more details. Table 1 illustrates these meanings. Table 2 lists all the strings taken by the `task` argument and their corresponding tasks. Code 2 provides an example of setting up camera parameters for rendering observations and goal images.

Table 1. Meanings and data types of the `make_env` function arguments

Argument	Type	Meaning
`task`	String	The name of the task environment to create.
`joint_control`	Boolean	Whether to use joint control actions.
`num_block`	Integer	The number of blocks involved. Only used in multi-step tasks.
`render`	Boolean	Whether to create a GUI window rendering the simulation.
`binary_reward`	Boolean	Whether to use sparse reward signals.
`max_episode_step`	Integer	The number of timesteps of an episode.
`distance_threshold`	Float	The threshold used to determine whether a goal is achieved.
`image_observation`	Boolean	Whether to use images as observations.
`Depth_image`	Boolean	Whether to include depth information in images.
`goal_image`	Boolean	Whether to use images as goals.
`visualize_target`	Boolean	Whether to render a semitransparent sphere at the target position of the manipulated objects.
`camera_setup`	List of Dict	A list of dictionary contains camera parameters, please see Code 2 for an example.
`observation_cam_id`	Integer	The index of the dictionary in the `camera_setup` list used to render observation images.
`goal_cam_id`	Integer	The index of the dictionary in the `camera_setup` list used to render goal images.
`use_curriculum`	Boolean	Whether to use the simple curriculum goal generation method, see section 2. Only used in multi-step tasks.
`num_goal_to_generate`	Integer	The number of goals to be generated in the whole training process, normally equal to the total number of training episodes. Only used in multi-step tasks.

Table 2. Correspondences between the `task` argument and each task

`task=`	task
‘reach’	KukaReach
‘push’	KukaPush
‘slide’	KukaSlide
‘pick_and_place’	KukaPickAndPlace
‘block_stack’	BlockStack
‘block_rearrange’	BlockRearrange
‘chest_pick_and_place’	ChestPickAndPlace
‘chest_push’	ChestPush

Code 2 A list of camera setup dictionary Meaning

```
camera_setup = [
    {
        ’cameraEyePosition’: [-1.0, 0.25, 0.6],       the 3D coordinates of the camera
                                                      frame in the world frame
        ’cameraTargetPosition’: [-0.6, 0.05, 0.2],    the 3D coordinates which the cam-
                                                      era looks at in the world frame
        ’render_width’: 128,                          the width of the rendered image
        ’render_height’: 128                          the height of the rendered image
    },
    {
        ’cameraEyePosition’: [-1.0, -0.25, 0.6],
        ’cameraTargetPosition’: [-0.6, -0.05, 0.2],
        ’render_width’: 128,
        ’render_height’: 128
    }
]
```