Dexterous robotic manipulation using deep reinforcement learning and knowledge transfer for complex sparse reward-based tasks

Qiang Wang | Francisco Roldan Sanchez | Robert McCarthy | David Cordova Bulens | Kevin McGuinness | Noel O’Connor | Manuel Wüthrich | Felix Widmaier | Stefan Bauer | Stephen J. Redmond

1University College Dublin, Dublin, Ireland 
2Dublin City University, Dublin, Ireland 
3Insight SFI Research Centre for Data Analytics, Dublin, Ireland 
4Harvard University, Cambridge, Massachusetts, USA 
5MPI for Intelligent Systems, Tübingen, Germany 
6KTH Stockholm, Stockholm, Sweden

Abstract
This paper describes a deep reinforcement learning (DRL) approach that won Phase 1 of the Real Robot Challenge (RRC) 2021, and then extends this method to a more difficult manipulation task. The RRC consisted of using a TriFinger robot to manipulate a cube along a specified positional trajectory, but with no requirement for the cube to have any specific orientation. We used a relatively simple reward function, a combination of a goal-based sparse reward and a distance reward, in conjunction with Hindsight Experience Replay (HER) to guide the learning of the DRL agent (Deep Deterministic Policy Gradient [DDPG]). Our approach allowed our agents to acquire dexterous robotic manipulation strategies in simulation. These strategies were then deployed on the real robot and outperformed all other competition submissions, including those using more traditional robotic control techniques, in the final evaluation stage of the RRC. Here we extend this method, by modifying the task of Phase 1 of the RRC to require the robot to maintain the cube in a particular orientation, while the cube is moved along the required positional trajectory. The requirement to also orient the cube makes the agent less able to learn the task through blind exploration due to increased problem complexity. To circumvent this issue, we make novel use of a Knowledge Transfer (KT) technique that allows the strategies learned by the agent in the original task (which was agnostic to cube orientation) to be transferred to this task (where orientation matters). KT allowed the agent to learn and perform the extended task in the simulator, which improved the average positional deviation from 0.134 to 0.02 m, and average orientation deviation from 142° to 76° during evaluation. This KT concept shows good generalization properties and could be applied to any actor-critic learning algorithm.

KEYWORDS
deep reinforcement learning, Real Robot Challenge, robotic manipulation, sim-to-real transfer, transfer reinforcement learning
1 INTRODUCTION

Dexterous robotic manipulation is essential in many industrial and domestic settings. Traditional robotic manipulation controllers often rely on solving inverse kinematic equations (Liu et al., 2021). The goal of this approach is to find the robotic joint angle time course to move the end-effector of a robotic system (arm, gripper, fingers, etc.) to a desired pose (Wei et al., 2020). Because the solution to this problem is not unique, motion primitives (i.e., a set of pre-computed movements that a robot can take in a given environment) are typically used (Cohen et al., 2010; Stulp et al., 2011). These primitives can each have a defined cost, allowing the robot to avoid non-smooth or other undesirable transitions. However, these techniques have poor generalization ability and require complex control system structures. More precisely, they require significant bespoke tailoring for each novel manipulation task, leading to high implementation times and costs. Moreover, current state-of-the-art in traditional robotic control strategies generally struggle in unstructured tasks, which require high degrees of dexterity.

Reinforcement learning (RL), a data-driven learning approach in which models are trained by rewarding or punishing an agent that is acting in a particular environment, shows promise (Akkaya et al., 2019) to replace traditional robotic control approaches. The goal of this learning paradigm is to maximize the cumulative sum of scalar or vector reward signals received by the agent in response to the actions that it takes (Sutton & Barto, 1999) so that it learns how to interact appropriately with the environment; that is, it learns how to act in order to maximize the rewards it can receive. However, traditional RL is unable to solve tasks with continuous action and state spaces, due to their high dimensionality. In other words, there are infinite combinations of states and actions in a continuous space, and therefore traditional RL algorithms cannot learn the required mappings between state, action, and reward for these problems. Hence, the field of deep reinforcement learning (DRL) (Mnih et al., 2013) has emerged; a new discipline combining deep learning (DL) (LeCun et al., 2015) and RL, which inherits the capacity to deal with high-dimensional continuous data from DL and decision-making ability from RL. DRL methods have been able to obtain outstanding results in robotic manipulation (Kober et al., 2013; Nguyen & La, 2019).

However, the data inefficiency of DRL is a major barrier to its application in real-world robotics: real robot data collection is time-consuming and expensive. Much DRL research to-date has focused on improving these data-efficiency issues. Due to their generally improved sample complexity, off-policy DRL methods (Haarnoja et al., 2018; Lillicrap et al., 2015) are often preferred to on-policy methods (Schulman et al., 2015, 2017). Model-based DRL methods, which explicitly learn a model of the environment, have been used to further improve sample efficiency (Deisenroth & Rasmussen, 2011; Hafner et al., 2019; Janner et al., 2019), and have seen success in real robot settings (Nagabandi et al., 2020). Moreover, off-line DRL techniques seek to leverage previously collected data to accelerate learning (Levine et al., 2020), and are able to learn dexterous real-world skills such as opening a drawer (Nair et al., 2020). Imitation learning methods provide the policy with expert demonstrations to learn from (Johns, 2021a; Pastor et al., 2009), enabling success in real robot tasks, such as peg insertion (Vecerik et al., 2017). Finally, simulation-to-real (sim-to-real) transfer methods train a policy quickly and cheaply in simulation before deploying it on the real robot, where learning is significantly slower, and have notably been used to solve a Rubik’s cube with a robot hand (Akkaya et al., 2019). To account for simulator modelling errors, and to improve the policies ability to generalize to the real robot, sim-to-real approaches often employ domain randomization (Peng et al., 2018; Tobin et al., 2017) or domain adaptation (Arndt et al., 2020; Eysenbach et al., 2020) techniques. Domain randomization, which has been particularly effective (Akkaya et al., 2019), randomizes the physics parameters in simulation to learn a robust policy that can adapt to the partially unknown physics of the real system.

The costly nature of real-robot experimentation has limited research related to robotic dexterous manipulation. In light of these issues, the Real Robot Challenge (RRC) (Bauer et al., 2021) aims to advance the state-of-the-art in robotic manipulation by providing participants with remote access to a TriFinger robotic platform (see Figure 1b) (Wüthrich et al., 2020), allowing for free and easy real-robot experimentation. To further support ease of experimentation, users are also provided with a simulation of this robotic system (see Figure 1a). Full details can be found in the ‘Protocol’ section of the RRC website.¹

This paper aims to extend the task of Phase 1 of the RRC 2021 to a more difficult task of moving a cube along a 3D positional trajectory while also maintaining a desired orientation. The task of Phase 1 of the RRC 2021 consisted of moving a cube along a defined trajectory. So here we increase the difficulty of this task by requiring the cube orientation to match a target orientation throughout the trajectory. We formulated both tasks as pure RL problems. The movement-related skills of the robot are entirely learned in simulation, hence reducing human involvement compared to traditional real-world learning methods. We chose to use the Deep Deterministic Policy Gradient (DDPG) algorithm, as it was shown to have an excellent performance in handling robotic manipulation tasks with continuous action and observation spaces (Lillicrap et al., 2015). We used a reward function composed of goal-based sparse rewards in conjunction with Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) to teach the control policy to move the cube to the desired xy coordinates (in the first task, controlling cube position) and then also orientation (in the extended task). Simultaneously, a dense distance-based reward is employed to teach the policy to lift the cube to the desired z coordinate. Finally, we use a novel concept that we call knowledge transfer (KT) to facilitate learning during the task requiring cube position and orientation be controlled, which is based on the strategies previously-learned during the task requiring only the cube position be controlled. Compared with other methods used to solve a similar task (Allshire et al., 2021; Schulman et al., 2017), our KT method has the advantage of being easier to implement, and can be extended to all actor-critic RL algorithms.
This paper is structured as follows: Section 2 describes the methodology, including the RRC robotic environment (simulator and real robot) and manipulation tasks, DRL algorithms, and the novel KT approach. Section 3 outlines the results of the conducted experiments, and Section 4 discusses our proposed approaches’ potential for use in DRL for robotic manipulation tasks. Finally, conclusions are drawn on the performance of the proposed approaches in Section 5.

2 | METHODS

2.1 | Real Robot Challenge (RRC)

We were provided with remote access to well-maintained TriFinger robotic platforms (Wüthrich et al., 2020) (see Figure 1b) by the Max Planck Institute for Intelligent Systems (Tübingen/Stuttgart, Germany). To further support ease of experimentation, users are also provided with a simulated version of the robotic setup (see Figure 1a). The 2021 RRC consists of an initial qualifying phase, performed purely in simulation, followed by independent Phases 1 and 2, both performed on the real robot. Full details can be found in the ‘Protocol’ section of the RRC website.

2.1.1 | Phase 1 task: Following a 3D position trajectory

Competition participants are tasked with solving the ‘Move Cube on Trajectory’ task. In this task, a cube must be carried along a goal trajectory, which specifies the Cartesian coordinates at which the cube should be positioned at each time-step. The goals are sampled discretely, and the time interval between the two goals in the training phase can be defined manually. The robot should react as soon as the new goal is received to obtain a higher score. For the final evaluation, multiple previously-unseen goal trajectories must be followed. Each evaluation trajectory consists of 120,000 time-steps, with 10 different coordinates that must be visited by the cube. The structure of the trajectory is such that the next goal (of 10 in total) is introduced at each of the following time-steps, $t \in \{0, 30,000, 40,000, 50,000, \ldots, 110,000\}$. The obtained score, $s_{\text{pos}}$, is defined by a distance-based weighted criterion:

$$s_{\text{pos}} = \frac{1}{2} \left( \frac{\|g'_{xy} - g_{xy}\|}{2d_r} + \frac{\|g'_{z} - g_{z}\|}{d_h} \right),$$

where, $g'_{xy}$ and $g'_{z}$ are the actual $xy$ and $z$ coordinates of the cube, respectively, and $g_{xy}$ and $g_{z}$ are the desired $xy$ and $z$ coordinates of the cube. $d_r$ and $d_h$ are constants representing the radius of the arena floor and the maximum height the fingertips can reach. A higher score, averaged across all tested trajectories, implies a better manipulation performance.
Unlike end-to-end pick-and-place manipulation, the goals are continuously changing in this task. Various action strategies, such as pushing, cradling, or pinching, must be dynamically employed to move the cube in response to the changing positional goals.

2.1.2 Extended task with increased difficulty: Position and orientation trajectory

We extended the task described above by introducing the an orientation trajectory goal in addition to the positional goal; this was inspired by the most challenging task in the RRC 2020, where the cube must not only be moved to the desired position but also aligned with the desired orientation. The task in 2020 is a single end-to-end movement, rather than trajectory. Hence, we integrated position, orientation, and trajectory together to create a more challenging task that we named Move Cube on Trajectory Pro. To evaluate performance during this task we used the criterion used in RRC 2020 (c.f., Equation 2)

\[ s_{\text{pos-ori}} = -\left\| \left( R(g_o) \right)^{-1} R(g_g) \right\| + s_{\text{pos}} \]  

where the \( g_o \) and \( g_g \) are the achieved orientation and the desired orientation, respectively, described using quaternions. \( R \) is the rotation matrix derived from the quaternions representation and \(-1\) is the inverse matrix operator. Note, \( s_{\text{pos}} \) from Equation (1) is incorporated in the calculation of \( s_{\text{pos-ori}} \) to evaluate the cube’s position.

2.2 Simulated environment

2.2.1 Actions and observations

Pure torque control of the robot arms is employed with an update frequency of 20 Hz (i.e., each time-step in the environment is 0.05 s). The robot has three arms, with three motorized joints in each arm; thus, the action space is 9-dimensional (and continuous). Observations include: (i) robot joint positions, velocities, and torques; (ii) cube’s current pose (i.e., its position and orientation, which in simulation is read directly from the simulator with no measurement error, and for the real world arena is estimated using provided computer vision object detection and segmentation methods), along with the difference between the poses at current and previous time-steps; and (iii) the current goal pose. In total, the observation space has 44 dimensions for the Move Cube on Trajectory task, and 48 for the Move Cube on Trajectory Pro.

In the Move Cube on Trajectory task, there are three target variables (three Cartesian position coordinates). In the Move Cube on Trajectory Pro task, there are seven target variables, representing 3D position and an additional four quaternion variables representing 3D orientation.

2.2.2 Episodes

In each simulated training and testing episode, the robot begins in its default position. The simulator instantiates the cube and the arena environment, with the cube’s initial position randomly sampled from the arena floor. Each episode lasts 90 time-steps, and the goal trajectory contains three desired goals, which are randomly sampled from the arena 3D space. The intervals between goals are same; that is, the goal changes every 30 time-steps.

2.2.3 Domain randomization

To help the learned policy generalize from a potentially inaccurate simulation to the real environment, we used some basic domain randomization (DR) techniques (i.e., physics randomization). This includes uniformly sampling, from a specified range, parameters of the simulation physics (e.g., robot mass, restitution, damping, friction; see our code for more details) and cube properties (mass and width) for each episode. To account for noisy real robot actuation and observations, uncorrelated noise is added to actions and observations during simulated episodes.

DR is only applied in the Move Cube on Trajectory task, since it requires sim-to-real transfer. In our final approach, we train the agent initially in the simulated non-DR environment for 300 epochs to allow it to learn the optimal policy quickly and easily. Afterward, the agent is tuned in the simulated DR environment for 100 epochs to enhance its robustness, before deploying to the real TriFinger robot.
2.3 | Learning algorithm

The goal-based nature of the Move Cube on Trajectory and Move Cube on Trajectory Pro tasks makes HER a good choice of learning algorithm; HER has excelled in similar goal-based robotic tasks (Andrychowicz et al., 2017) and obviates the need for complex reward engineering. As such, we combine DDPG and HER to make our RL algorithm. However, in our early experiments we observed that the standard DDPG plus HER learning algorithm was slow in learning to lift the cube. To resolve this issue, we slightly altered the HER process and incorporated an additional dense reward which encourages cube-lifting behaviours. We describe these elements of the learning algorithm below.

2.3.1 | Goal-based reinforcement learning

We frame the RRC robotic environment as a Markov decision process (MDP), defined by the tuple \( (S, A, G, p, r, \gamma, \rho_\phi) \). \( S, A, \) and \( G \) are the state, action and goal spaces, respectively. The state transition distribution is denoted as \( p(s' | s, a) \), the initial state distribution as \( \rho_\phi(s) \), and the reward function as \( r(s, g) \). \( \gamma \in (0, 1) \) discounts future rewards. The goal of the RL agent is to find the optimal policy \( \pi^* \) that maximizes the expected sum of discounted rewards in this MDP over all time, i.e.,

\[
\pi^* = \text{argmax}_\pi \mathbb{E}_tw \sum_{t=0}^\infty \gamma^t r(s_t, g_t).
\]

2.3.2 | Deep Deterministic Policy Gradients (DDPG)

DDPG (Lillicrap et al., 2015) is an off-policy RL algorithm which, in the goal-based RL setting, maintains the following neural networks: a policy (actor) \( \pi: S \times G \rightarrow A \), and an action-value function (critic) \( Q: S \times G \times A \rightarrow \mathbb{R} \). The critic is trained to minimize the loss \( L_c = \mathbb{E}(Q(s_t, g_t) - y_t)^2 \), where

\[
y_t = r_t + \gamma Q(s_{t+1}, g_{t+1}) - r_{t-1}.
\]

To stabilize the critic’s training, the targets \( y_t \) are produced using slowly updated polyak-averaged versions of the main networks. The actor is trained to minimize the loss:

\[
\mathcal{L}_a = -\mathbb{E}_q[Q(s, \pi(s, g))],
\]

where gradients are computed by backpropagating through the combined critic and actor networks. For these updates, the transition tuples \( (s_t, g_t, a_t, r_t, s_{t+1}, g_{t+1}) \) are sampled from a replay buffer which stores previously collected experiences (i.e., off-policy data).

2.3.3 | Hindsight Experience Replay (HER)

HER (Andrychowicz et al., 2017) can be used with any off-policy RL algorithm in goal-based tasks, and is most effective when the reward function is sparse and binary (e.g., Equation 3). To improve learning in the sparse reward setting, HER employs a simple trick when sampling previously collected transitions for policy updates: a proportion of sampled transitions have their goal altered to \( g' \), where \( g' \) is a goal achieved later in the episode. The rewards of these altered transitions are then recalculated with respect to \( g' \), leaving the altered transition tuples as \( (s_t, g_t, a_t, r_t, s_{t+1}, g_{t+1}) \). Even if the original episode was unsuccessful, these altered transitions will teach the agent to achieve \( g' \), thus accelerating its acquisition of skills.

2.3.4 | Rewards and HER

In our approach, the reward function that guides the RL agent’s learning consists of three components: (i) a sparse reward based on the the cube’s \( xy \) coordinates, termed \( r_{xy} \); (ii) a dense reward based on the cube’s \( z \) coordinate, termed \( r_z \) (the coordinate frame can be seen in Figure 1a); and (iii) a sparse reward based on the cube’s 3D orientation (used in the Move Cube on Trajectory Pro task only).

The sparse \( xy \) reward, \( r_{xy} \), is calculated as:

\[
r_{xy} = \begin{cases} 0 & \text{if } \|g'_{xy} - g_{xy}\| \leq 2 \text{cm} \ , \\ -1 & \text{otherwise} \end{cases}
\]

where, \( g'_{xy} \) are the \( xy \) coordinates of the achieved goal (the actual \( xy \) coordinates of the cube), and \( g_{xy} \) are the \( xy \) coordinates of the desired goal.

The dense \( z \) coordinate reward, \( r_z \), is defined as:

\[
r_z = \begin{cases} -a |z_{\text{cube}} - z_{\text{goal}}| & \text{if } z_{\text{cube}} < z_{\text{goal}} \\ -a/2 & \text{if } z_{\text{cube}} > z_{\text{goal}} \end{cases}
\]

where \( a \) is a parameter that determines the reward for lifting the cube.
Thus, the policy focuses solely on achieving the current active goal and is unconcerned with any future changes in the active goal.

We derive our DDPG-HER hyperparameters from Plappert et al. (2018), who use a highly exploratory policy when collecting data in the environment: with 30% probability a random action is sampled (uniformly) from the action space, and when policy actions are chosen, Gaussian noise is applied. This is beneficial for exploration in the early stages of training, however, it can be limiting in the later stages when the policy must be fine-tuned; we found that the exploratory policy repeatedly drops the cube due to the randomly sampled actions and the injected action noise. To resolve this issue, rather than slowly reducing the level of exploration each epoch (which would require a degree of hyperparameter tuning), we make efficient use of evaluation episodes (which are performed by the standard exploitation policy) by adding them to the replay buffer. Thus,

The sparse reward for the orientation is defined as:

$$ r_{ori} = \begin{cases} 0 & \text{if } \| (R(g'_o))^{-1}R(g_o) \| \leq 0.384 \text{rad (i.e., } 22^\circ) \\ -1 & \text{otherwise} \end{cases} $$

(5)

where the $g'_o$ and $g_o$ are the achieved orientation and the desired orientation represented as quaternions. $R$ indicates rotation matrix calculation and $-1$ is the matrix inverse operator. We set the position (2 cm) and orientation (22°) thresholds as per Andrychowicz et al. (2020).

Another pure distance-based reward is also used as a comparator for our position-based reward scheme. This reward is popular in the field because of its clear geometric meaning and its computational efficiency:

$$ r_{ds} = -\| g'_{xyz} - g_{xyz} \| $$

(6)

where, $g'_{xyz}$ and $g_{xyz}$ are the achieved $xyz$ coordinates and the desired $xyz$ coordinates of the cube.

For the position-only Move Cube on Trajectory task, we only apply HER to the $xy$ coordinates of the goal; that is, the $xy$ coordinates of the goal can be altered in hindsight, but the $z$ coordinate remains unchanged. Thus, our HER altered goals are: $\tilde{g} = (g'_{xy}, g_o)$, meaning only $r_{xy}$ is recalculated after HER is applied to a transition sampled during policy updates. This reward system is motivated by the following:

1. Using $r_{xy}$ with HER allows the agent to learn to push the cube around in the early stages of training, even if it cannot yet lift the cube to reach the $z$ coordinate of the goal. As the agent learns to push the cube around in the $xy$ plane of the arena floor, it can then more easily stumble upon actions which lift it. Importantly, the approach of using $r_{xy}$ with HER requires no bespoke reward engineering.
2. $r_z$ aims to explicitly teach the agent to lift the cube by encouraging minimization of the vertical distance between the cube and the goal. It is less punishing when the cube is above the goal, serving to further encourage lifting behaviours.
3. In the early stages of training, the cube mostly remains on the floor. During these stages, most $g'$ sampled by HER will be on the floor. Thus, applying HER to $r_z$ could often lead to the agent being punished for briefly lifting the cube. Since we only apply HER to the $xy$ coordinates of the goal, our HER altered goals, $\tilde{g}$, maintain their original $z$ height. This leaves more room for the agent to be rewarded by $r_z$ for any cube lifting it performs.

In the Move Cube on Trajectory Pro task, the reward of the positional components is same as above. HER is applied on the orientation and thus the HER altered goals are $\tilde{g} = (g'_{xy}, g'_{ori}, g_o)$. Finally all items are summed to form the final reward, $r = r_{xy} + r_z + r_{ori}$.

### 2.3.5 Goal trajectories

In each episode, the agent is faced with multiple goals; it must move the cube from one goal to the next along a given trajectory. To ensure the HER process remains meaningful in these multi-goal episodes, we only sample future achieved goals, $g'$, (to replace $g$) from the period of time in which $g$ was active.

In our implementation, the agent is unaware that it is dealing with trajectories: when updating the policy with transitions $(s_t, g_t, a_t, r_t, s_{t+1}, g_{t+1})$ we always set $g_{t+1} = g_t$, even if in reality $g_{t+1}$ was different. Thus, the policy focuses solely on achieving the current active goal and is unconcerned with any future changes in the active goal.

### 2.3.6 Exploration versus exploitation

We derive our DDPG-HER hyperparameters from Plappert et al. (2018), who use a highly exploratory policy when collecting data in the environment: with 30% probability a random action is sampled (uniformly) from the action space, and when policy actions are chosen, Gaussian noise is applied. This is beneficial for exploration in the early stages of training, however, it can be limiting in the later stages when the policy must be fine-tuned; we found that the exploratory policy repeatedly drops the cube due to the randomly sampled actions and the injected action noise. To resolve this issue, rather than slowly reducing the level of exploration each epoch (which would require a degree of hyperparameter tuning), we make efficient use of evaluation episodes (which are performed by the standard exploitation policy) by adding them to the replay buffer. Thus,
90% of rollouts added to the buffer are collected with the exploratory policy, and the remaining 10% with the exploitation policy. This addition was sufficient to boost final success rates in simulation from 70%–80% to >90% (where ‘success rate’ is equivalent to that seen in Figure 2).

2.4 | Knowledge transfer

The signal from our computationally-simple reward rarely guides the agent to focus on playing with the cube in the early training stage; hence, the agent’s learning relies heavily on pure exploration. However, its arbitrary exploration is unlikely to explore actions that change the cube’s orientation. Unlike the position-only Move Cube on Trajectory task, the orientation exploration in the Move Cube on Trajectory Pro task is comparatively more difficult, requiring the cube to be both lifted and rotated simultaneously. To this end, we introduce our KT approach, which transfers a trained agent’s knowledge (teacher) to another untrained agent (student), or uses the teacher’s knowledge to assist the student with learning. In our case, we train a teacher from the position-only Move Cube on Trajectory task, and train students in the Move Cube on Trajectory Pro task. The teacher’s transferred knowledge increases the likelihood of the student discovering useful actions to control the orientation of the cube through exploring in the early stages of training, instead of exploring all possible actions at random, as the student has been transferred the knowledge required to control the cube’s position; in this way, we expect the agent to learn how to control the cube’s orientation more quickly. We use two strategies to ensure the teacher’s guidance is not too biased toward position-based exploration: (1) increasing the action noise in the early stages of exploration; (2) choose a teacher that has weaker performance in the Move Cube on Trajectory task (i.e., success rate of only 80%).

We have the following three implementations of KT. We name them differently for ease of explanation later.

1. ACTOR-CRITIC: Initialise the student’s actor and critic network weights by loading the teacher’s trained weights.
2. ACTOR: Initialise the student’s actor network weights by loading the teacher’s trained weights and randomly initialize the student’s critic.
3. COLLECT: The student does not inherit any network weights from the teacher. The teacher only helps the student to collect experience in the early stages of training. Both the student’s actor and critic networks are randomly initialized. As the teacher’s knowledge is not updated in the

![Figure 2](image-url)  
**Figure 2** Success rate versus experience collected during simulated training (1 day in simulator, ≈1.7 million environment steps). We took the average over three randomly-selected seeds for this illustration. The solid dotted lines are the trend lines and the faded lines are the raw data. An episode is deemed successful if, when complete, the final goal of the trajectory has been achieved. We compare training with: (i) our final method where HER is applied to \( r_{xy} \) but not to \( r_z \) (red); (ii) HER applied to both \( r_{xy} \) and \( r_z \) (green); (iii) HER applied to a standard sparse reward where \( xyz \) are calculated together (blue); (iv) HER applied to pure distance reward, \( r_{dis} \), where \( xyz \) are calculated together (orange); and (v) pure distance reward, \( r_{dis} \), without HER (purple); (vi) standard sparse reward without HER (black).
new task, the collected experience lacks diversity. Hence, we force the teacher’s participation to decay as training progresses, allowing the student to engage in exploration further, making the experience more beneficial to the student.

4. **SCRATCH**: The attempt to solve the Move Cube on Trajectory Pro task, ‘from scratch’, without using KT. This is used for comparing KT approaches.

### 2.5 | Train, test, and evaluate

The structure of the neural networks and hyperparameters that we used for training are shown in Appendix A. All training and testing were performed on the Sonic high performance cluster at University College Dublin, Ireland. For our training, jobs are randomly assigned to Dell machines configured with R640 2x Intel Xeon Gold 6152 (2.1 GHz, 22 cores) or C6200 V2 2x Intel E5-2660 v2 (2.2 GHz, 10 cores). In each training run, eight RL agents are assigned to eight processors and run in parallel. At the end of each training step, the neural network weights of eight agents are synchronized by averaging. The experiences collected by agents are stored in their respective experience replay buffers and will not be shared. The global success rate and reward are calculated by averaging the local values from eight agents.

- **Training**: For the Move Cube on Trajectory task, each job runs for 300 epochs, overall interacting with the simulated environment for 21.6 million time-steps (2.7 million time-steps for each RL agent). The neural network of each agent is updated 15,000 times and synchronization is performed after each update. For the Move Cube on Trajectory Pro task, the number of training epochs is increased to 500, and other training parameters increased proportionally.

- **Testing**: Testing is performed after each training epoch. Each agent runs for 900 time-steps in the simulator. Experience collected from testing is stored in the replay buffer. The neural network losses, success rates and rewards of eight agents are averaged and saved.

- **Real robot evaluation of Move Cube on Trajectory**: For the Move Cube on Trajectory task performed on the real TriFinger robot, we adopt the evaluation criteria set by the RRC 2021. For each evaluation episode, a random goal trajectory is sampled. Each test run keeps 120,000 time-steps, with 10 different goals (first goal at first time-step, then 30,000 time-steps to the second goal, and 10,000 time-steps for each of the remaining 8 goals after that). The domain gap between simulation and reality was significant, and generally led to inferior scores on the real robot. Policies often struggled to gain control of the real cube which appeared to slip more easily from the fingers than in simulation. Additionally, on the real robot, policies could become stuck with a fingertip pressing the cube into the wall. As a heuristic solution to this issue, we assumed the policy was stuck whenever the cube had not reached the goal’s xy coordinates for 50 consecutive time-steps; then uniformly sampled random actions were taken for 7 time-steps in an attempt to free the policy from its stuck state.

### 3 | RESULTS

#### 3.1 | RRC 2021 phase 1

#### 3.1.1 | Learning outcomes of RL agents

Our method is highly effective in simulation. The algorithm can learn from scratch to proficiently grasp the cube and lift it along goal trajectories. Figure 2 compares the training performance of our final algorithm (red curve) to that of other combinations. Our algorithm appears to converge stably and more quickly than either HER applied to a standard sparse reward or HER applied to both $r_p$ and $r_c$. Furthermore, HER plays a crucial role in the success of our algorithm. The pure DDPG agent finds it difficult to learn from sparse rewards without HER, and the convergence speed is extremely slow for an agent guided by this traditional distance-based reward.

Throughout different training runs, our policies learned several different manipulation strategies, the most distinct of which included: (i) ‘pushing’ the cube on the arena floor; (ii) ‘cradling’ the cube with all three of its forearms (not fingertips); and (iii) ‘pinching’ the cube with two fingertips and supporting it with the third (see Figure 3).

Our final policies transferred to the real robot with reasonable success. Table 1 displays the self-reported scores of our best pinching and cradling policies under RRC Phase 1 evaluation conditions. As a baseline comparison, we trained a simple ‘pushing’ policy which ignores the height component of the goal and simply learns to push the cube along the floor to the goal’s xy coordinates. The pinching policy performed best on the real robot, and is capable of carrying the cube along goal trajectories for extended periods of time, and of recovering the cube when it is dropped. This policy was submitted for the official RRC Phase 1 final evaluation round and obtained the winning score (McCarthy et al., 2021) (see, username ‘thriftysnipe’).
3.1.2 | Agents trained from different random seeds behave differently

We verified that discrepancies exist between agents’ performances when trained from different random seeds (see Table 2) (Henderson et al., 2018); for detailed experimental results before averaging, please see Appendix B: Table B2. In our case, the random seed is used to randomly reset the environment and randomize actions so that agents can explore in the training phase. Interestingly, the best performing agent (Seed 0 in Table 2) in the simulator had a relatively poor performance on the real robot. This agent overfits to the simulator environment, and hence the acquired policy has low generalization capability. The agent trained under (random) Seed 200 (see Table 2) did not obtain the optimal policy and performed the worst on the real robot.

3.1.3 | Domain randomization improves real robot performance

The performance of agents on the real robot significantly improved after tuning using DR (see Table 3), the detailed experimental results can be seen in Tables B1–B3. Among these agents, the agent trained from Seed 123 achieved simulator-level performance. Nonetheless, using DR from the beginning of training is not recommended (at least for our algorithm). In a domain-randomized simulation environment, the agent will be hindered in learning the optimal policy, compared to the policy learned in the standard simulation environment that does not use domain randomization (see Figure 4), and hence will achieve poor performance on the real robots (see Table 3). A demonstration video of this trained network in action can be seen in https://www.youtube.com/watch?v=0Lpod542T9k.

3.2 | Move Cube on Trajectory Pro

Figure 5 compares the training performance of different approaches to learning the Move Cube on Trajectory Pro task, including using our proposed KT scheme compared to not using it. Agents’ overall rewards are decomposed into position and orientation components so that the learning outcomes of the two components can be compared.

Training using the combined DDPG and HER algorithm (not using KT, see SCRATCH curve in Figure 5) completely fails in the Move Cube on Trajectory Pro task, with rewards on all components staying at their small values. In contrast, agents in all three KT methods shows learning trends.
TABLE 2  Evaluated score of RL agents trained under three different random seeds

| Robot (real/sim) | Seed 0 | Seed 123 | Seed 200 | Average over seeds |
|------------------|--------|----------|----------|-------------------|
| Robch1           | -12,175 | -9,308   | -16,266  | -11,864 ± 3181    |
| Robch5           | -11,300 | -8,926   | -13,120  | -6,367 ± 920      |
| Robch6           | -9,913  | -10,017  | -15,750  | -11,864 ± 3181    |
| Sim              | -5,447  | -6,376   | -7,277   | -6,367 ± 920      |
|                 | Avg     | -11,129  | -9,417   | -15,045           |

Note: The RRC organizing institute has a small number of real robots available for remote use, with a small but expected variation in the physical parameters of each arena and the performance of each robot. The evaluations are performed on three of these robots, chosen at random, named Roboch1, Roboch5, and Roboch6, to increase the generalizability of the results. Each evaluation is repeated 15 times (15 goal trajectories) with 120,000 time-steps in each repeat. The results from the simulator are also shown in the Sim subcolumns. The raw results before averaging are shown in Table B2.

TABLE 3  Evaluation rewards of different agents evaluated in simulation and on real robots; three seeds used in this paper

| Seed  | Type                        | Simulation     | Real robot   |
|-------|-----------------------------|----------------|--------------|
| 0     | Scratch(DR)                 | -8808          | -16,685      |
|       | Scratch(NDR)                | -5447          | -11,129      |
|       | Scratch(NDR) + Tune(DR)     | -5825          | -8922        |
| 123   | Scratch(DR)                 | -9873          | -15,726      |
|       | Scratch(NDR)                | -6376          | -9417        |
|       | Scratch(NDR) + Tune(DR)     | -6253          | -7030        |
| 200   | Scratch(DR)                 | -11,779        | -15,398      |
|       | Scratch(NDR)                | -7277          | -15,045      |
|       | Scratch(NDR) + Tune(DR)     | -6147          | -8535        |
| Average over seeds | Scratch(DR)             | -9,913 ± 1866  | -15,936 ± 749 |
|       | Scratch(NDR)                | -6,367 ± 920   | -11,864 ± 3181 |
|       | Scratch(NDR) + Tune(DR)     | -6,075 ± 250   | -8,162 ± 1132 |

Note: Scratch(DR) represents the agent trained from scratch using DR from the beginning of training. Scratch(NDR) is the agent trained from scratch without using DR; Scratch(NDR) + Tune(DR) is the agent firstly trained from scratch without DR and then subsequently tuned using DR in a second training phase. The raw results before averaging are shown in Tables B1–B3.

FIGURE 4  Comparison between agents trained from scratch with domain randomization (DR) and without
Among them, both COLLECT and the SCRATCH use randomly initialized neural networks. However, with the help of the experience collected by the teacher, COLLECT shows a more successful learning performance.

By transferring the teacher’s knowledge of how to perform the position-only task to the ACTOR-CRITIC student, the ACTOR-CRITIC excels in its ability to reach position targets from the beginning of training. This makes it more likely that the ACTOR-CRITIC will discover actions which change the cube’s orientation and are rewarded. All agents learn orientation from scratch, while the ACTOR-CRITIC has a better learning performance than that of COLLECT. However, the COLLECT agent relies strongly on the experience collected by the teacher in the early stages of training, although we set a decay on the teacher’s participation over time. Although the ACTOR agent inherits the knowledge of the teacher’s actor, it performs the worst among all KT strategies. Because the randomly-initialized critic is not compatible with the well-trained actor, and the critic’s feedback cannot meet the actor’s expectations; hence, the actor’s performance worsens.

**FIGURE 5** Learning curves showing rewards achieved by agents using different learning algorithms.
The average deviations between achieved and desired position/orientation over 15 evaluation episodes in the simulator; each episode lasts 120,000 time-steps

|                          | ACTOR-CRITIC | ACTOR | COLLECT | SCRATCH | TEACHER |
|--------------------------|--------------|-------|---------|---------|---------|
| Average position deviation $d_{\text{dev}}$ (m) | 0.023        | 0.066 | 0.031   | 0.134   | 0.024   |
| Average orientation deviation $d_{\text{devangle}}$ (°) | 75.8         | 98.6  | 84.9    | 142.2   | 126.2   |

Note: The deviations are self-customized to see the learning outcomes intuitively. For the position: $d_{\text{dev}} = \| x_{\text{act}} - x_{\text{des}} \|$, which can measure the distance between the actual position of the cube and the goal position. For the orientation: $d_{\text{devangle}} = \| (R(g_{t}))^{-1} R(g_{t}) \|$, which can measure the angle difference between the actual and target orientations. The deviations of all time-steps are accumulated and finally averaged. The raw results before averaging are shown in Table B4.

The evaluation results are shown in Table 4 and the raw results before averaging are shown in Table B4. All three methods enable the student agent to learn the Move Cube on Trajectory Pro task, but behave differently. Among them, the ACTOR-CRITIC agent can achieve comparable performance to the teacher at reaching position targets, and learn to perform well on reaching orientation targets. A demonstration video of this trained network in action can be seen here: https://www.youtube.com/watch?v=GhkCqoMqzU4.

4 | DISCUSSION

Deep reinforcement learning shows promise for solving robotic manipulation tasks requiring dexterity. Compared with traditional robot control methods, it requires less bespoke engineering work and often has better performance, as shown in our results. We have only tried our approach using the DDPG learning algorithm; other later published DRL algorithms, such as SAC (Haarnoja et al., 2018) or TD3 (Fujimoto et al., 2018) may have superior performance, and will be the focus of our future research.

Although DRL approaches for solving robotic manipulation tasks are outstanding, the algorithm is somewhat training and sample inefficient. For example, solving the Move Cube on Trajectory task takes roughly 10 million environment steps to converge (equivalent to 6 days of simulated experience), while a large part of the time is spent in blind exploration of the environment. The KT method we developed can reduce blind exploration in the initial stage and, more importantly, avoid repeated learning of the same knowledge in different experimental trials. It can improve resource utilization efficiency. A future similar, but innovative research direction would be to train a master teacher with broad knowledge in complex environments and use it widely in various downstream tasks (Johns, 2021b). Another important direction for future work would be to increase sample efficiency, perhaps achievable via a model-based reinforcement learning approach (Janner et al., 2019; McCarthy & Redmond, 2021).

The reward function is the driver of DRL: a well-designed and well-shaped reward can improve the quality and speed of DRL (Allshire et al., 2021). A bespoke reward function might limit the utility of a method to a small range of tasks, and is likely costly to develop. In contrast, goal-based sparse rewards can be easily created. Compared to shaped rewards, goal-based sparse rewards have an extremely weak reward signal. The agent can only get the a reward when succeeding in the task; otherwise, it gets punished. However, the probability of success in some tasks is very low, and a dearth of positive feedback hinders improvement in the task. HER can effectively enhance sparse reward signals by modifying the experience data before giving it to the agent. Typically, HER replaces the goal with the posterior state, making the prior-posterior transition highly likely to succeed by the action applied to the prior state. HER has been proven to work wonders (Andrychowicz et al., 2017) in sparse-reward-based tasks, as verified in our case. In addition, our distance-based dense reward can effectively accelerate the agent’s learning in the early stages of training, and its implementation is straightforward. We have only validated the approach of mixing sparse and dense rewards for this specific case so far. Indeed, we know from past experience that this method is not required for some ‘lifting’ tasks, such as OpenAI’s (Brockman et al., 2016) FetchPickAndPlace-v0, in which the robot has a two-finger gripper that can lift the object relatively easily. However, lifting the object in the z-direction using the RRC’s TriFinger robot is much more difficult, requiring dexterous cooperation between three fingers. Therefore, we enhance the z-direction reward signal to accelerate the initial stages of training when random exploration is unlikely to lift the object. However, from our experience we can provide the anecdote that excessive use of dense rewards can diminish the benefit of the HER algorithm, which is designed to work with sparse rewards. Hence our method has an appropriate mix of sparse and dense rewards which balances the benefits of using sparse rewards with HER against having dense rewards to acquire difficult skills earlier in learning. We think the principle of our approach can be generalized, but it is not yet clear how to engineer a reward function with the optimal mix of sparse and dense rewards in general.

Sim-to-real transfer is an essential means of bringing robotic learning into reality. Training an intelligent robotic controller in the real world is expensive and impractical. Take an example of a problem we faced before: we tried to collect the data from the real TriFinger robots to train an agent from scratch; however, a large number of interactions caused a lot of friction between the finger and the cube, resulting in wear and tear, which adhered to the surface of the cube and interfered with the estimation of the cube’s pose by machine vision algorithms. Indeed, the main limitation of our approach was the absence of any real-robot data. It is likely that some fine-tuning of the policy on real data would greatly increase its robustness in the real environment, and developing a technique which could do so efficiently is one direction for future work.
Different from the training of a traditional deep learning model, the training of a deep reinforcement learning model requires a large amount of random exploration. Pseudorandom number sequences are generated to drive this (pseudo)random exploration. However, if the same seed is provided to the pseudorandom number generator for the same experiment, the same sequence of exploratory actions will be performed; in this sense the outcome of the reinforcement learning activity is deterministic. The network learned may always converge on the same solution, irrespective of the initial seed chosen. But empirically this is not the case, and the choice of initial seed does influence the performance of the network that is learned. Changing the seed provided to the pseudorandom number generator would allow the sensitivity of the solution to this initial seed to be evaluated; however, this is obviously a very time-consuming process. In this paper, we evaluate three candidates seeds (arbitrarily, integers 0, 123, and 200) to give an indication of the sensitivity of the training process to seed selection. The results show that the reward score achieved varied by ±27% of the mean across these three seeds. Hence, this shows that when deploying the DRL controller on a robot, the best-performing agent might be selected from a range of policies learned using different seed values.

The development of DRL-driven robotic manipulation is still an unsolved problem, but is currently receiving a lot of attention from the research community. Solving this problem will have a huge impact on industry and society in general. At present, the training of DRL agents is mostly carried out in the simulator, using perfect feedback data which is mostly vision-based. The current capabilities of simulators to render realistic mechanics, including the friction, texture, and deformations at contact interfaces between robotic fingers and the objects they manipulate, are very basic. However, as such simulations improve, and the ability of real world robots to sense these same contact mechanics advances in tandem, we can expect significant improvements in the dexterity of DRL-based robotic manipulation systems. This ability to dexterously interact with and learn from the world is surely the path to developing systems with superior intelligence.

5 | CONCLUSION

Our relatively simple reinforcement learning approach fully solves the Move Cube on Trajectory task in simulation. Unlike the RRC 2020 benchmark solution (Funk et al., 2021), this was achieved with the use of minimal domain-specific knowledge. Reproducing our method is highly feasible, since only one machine with eight processors is required; no GPUs are required. The number of agents can be set to more if an increase in training speed is desirable. Or, the number of agents can be set to less, but sacrificing training speed.

Moreover, our learned policies can be successfully deployed on the real robot through our DR tuning approach, which bridges the large sim-to-real domain gap and allows the achievement of near-simulator-level performance on real robots. In the RRC 2021, which used the Move Cube on Trajectory task, requiring only position targets be reached, we outperformed all competing submissions, including those employing more classical robotic control techniques.

Our novel KT approach can enable an agent to perform more useful interactions with the environment in the early stages of learning, helping to avoid exploration actions that do not result in learning experiences. It allows the complex Move Cube on Trajectory Pro task to be solved efficiently in the simulator. Additionally, KT increases resource utilization efficiency, avoiding repeatedly re-learning similar skills. It may be feasible to deploy this method in any actor-critic RL algorithm, and the implementation method is straightforward.

ACKNOWLEDGEMENTS

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under grant numbers 17/FRL/4832 and SFI/12/RC/2289_P2 and of China Scholarship Council (CSC No. 202006540003). For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission. We acknowledge the Research IT HPC Service at University College Dublin for providing computational facilities and support that contributed to the research results reported in this paper. Open access funding provided by IReL.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ORCID

Qiang Wang https://orcid.org/0000-0002-1445-755X
Robert McCarthy https://orcid.org/0000-0002-2140-6988
David Cordova Bulens https://orcid.org/0000-0003-2466-7179
Kevin McGuinness https://orcid.org/0000-0003-1336-6477
Noel O’Connor https://orcid.org/0000-0002-4033-9135
Felix Widmaier https://orcid.org/0000-0002-1330-3997
Stefan Bauer https://orcid.org/0000-0003-1712-060X
Stephen J. Redmond https://orcid.org/0000-0002-2630-5449
ENDNOTES

1. https://real-robot-challenge.com/2021.

2. https://is.mpg.de/.

3. Agents interact with the environment in (discrete) time-steps, which are incremented after the agent takes an action.

4. https://real-robot-challenge.com/2020.

5. Our domain randomization implementation is based on the benchmark code from the 2020 RRC (Funk et al., 2021).

6. https://github.com/RobertMcCarthy97/rrc_phase1.

7. Our DDPG + HER implementation is taken from https://github.com/TianhongDai/hindsight-experience-replay, and uses hyperparameters largely based on (Plappert et al., 2018).

8. Interestingly, we found that exposing the agent (during updates) to transitions in which \( g_{t-1} \neq g_t \) hurt performance significantly, perhaps due to the extra uncertainty this introduces to the DDPG action-value estimates.

9. The actor is typically a policy function, which outputs actions to interact with the environment. The critic uses approximate architecture and simulations to learn a value function, which is then used to update the actors’ policy parameters (Konda & Tsitsiklis, 1999).

10. These runs did not use domain randomization. Generally we trained from scratch in standard simulation before fine-tuning in a domain-randomized simulation.

11. https://real-robot-challenge.com/2021/#nav-results.

REFERENCES

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paine, A., Plappert, M., Powell, G., Ribas, R., & Schneider, J. (2019). Solving Rubik’s cube with a robot hand. arXiv preprint arXiv:1910.07113.

Allshire, A., Mittal, M., Lodaya, V., Makovychuk, V., Makovychuk, D., Widmaier, F., Wüthrich, M., Bauer, S., Handa, A., & Garg, A. (2021). Transferring dexterous manipulation from GPU simulation to a remote real-world TriFinger. arXiv preprint arXiv:2108.09779.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., & Zaremba, W. (2017). Hindsight experience replay. Advances in Neural Information Processing Systems, 30, 1–11.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., & Schneider, J. (2020). Learning dexterous in-hand manipulation. The International Journal of Robotics Research, 39(1), 3–20.

Arndt, K., Hazara, M., Ghadirzadeh, A., & Kyrki, V. (2020). Meta reinforcement learning for sim-to-real domain adaptation. In 2020 IEEE international conference on robotics and automation (ICRA) (pp. 2725–2731). IEEE.

Bauer, S., Widmaier, F., Wüthrich, M., Funk, N., De Jesus, J. U., Peters, J., Watson, J., Chen, C., Srinivasan, K., Zhang, J., & Zhang, J. (2021). A robot cluster for reproducible research in dexterous manipulation. arXiv preprint arXiv:2109.10957.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540.

Cohen, B. J., Chitta, S., & Likhachev, M. (2010). Search-based planning for manipulation with motion primitives. In 2010 IEEE international conference on robotics and automation. IEEE.

Coumans, E., & Bai, Y. (2016–2021). PyBullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org

Deisenroth, M., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In Proceedings of the 28th international conference on machine learning (ICML-11).

Eysenbach, B., Asawa, S., Chaudhari, S., Levine, S., & Salakhutdinov, R. (2020). Off-dynamics reinforcement learning: Training for transfer with domain classifiers. arXiv preprint arXiv:2006.13916.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In International conference on machine learning (pp. 1587–1596). PMLR.

Funk, N., Schaff, C., Madan, R., Yoneda, T., De Jesus, J. U., Watson, J., Gordon, E. K., Widmaier, F., Bauer, S., Srinivasa, S. S., & Bhattacharjee, T. (2021). Benchmarking structured policies and policy optimization for real-world dexterous object manipulation. IEEE Robotics and Automation Letters, 7(1), 478–485.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905.

Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. arXiv preprint arXiv:1912.01603.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. In Proceedings of the AAAI conference on artificial intelligence (Vol. 32, No. 1).

Janner, M., Fu, J., Zhang, M., & Levine, S. (2019). When to trust your model: Model-based policy optimization. Advances in Neural Information Processing Systems, 32, 1–12.

Johns, E. (2021b). Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In 2021 IEEE international conference on robotics and automation (ICRA) (pp. 4613–4619). IEEE.

Johns, E. (2021a). Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. arXiv preprint arXiv:2105.06411.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11), 1238–1274.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. Advances in Neural Information Processing Systems, 12, 1008–1014.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning.

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
Liu, R., Nageotte, F., Zanne, P., de Mathelin, M., & Dresp-Langley, B. (2021). Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review. Robotics, 10(1), 22.

McCarthy, R., & Redmond, S. J. (2021). Imaginary hindsight experience replay: Curious model-based learning for sparse reward tasks. arXiv preprint arXiv: 2110.02414.

McCarthy, R., Sanchez, F. R., Wang, Q., Bulens, D. C., Mcguinness, K., O’Connor, N., & Redmond, S. J. (2021). Solving the real robot challenge using deep reinforcement learning. arXiv preprint arXiv:2109.15233.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Nagabandi, A., Konolige, K., Levine, S., & Kumar, V. (2020). Deep dynamics models for learning dexterous manipulation. In Conference on robot learning (pp. 1101–1112). PMLR.

Nair, A., Dalal, M., Gupta, A., & Levine, S. (2020). Accelerating online reinforcement learning with offline datasets. arXiv preprint arXiv:2006.09359.

Nguyen, H., & La, H. (2019). Review of deep reinforcement learning for robot manipulation. In 2019 Third IEEE international conference on robotic computing (IRC) (pp. 590–595). IEEE.

Pastor, P., Hoffmann, H., Asfour, T., & Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. In 2009 IEEE international conference on robotics and automation (pp. 763–768). IEEE.

Peng, X. B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In 2018 IEEE international conference on robotics and automation (ICRA) (pp. 3803–3810). IEEE.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., & Kumar, V. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. arXiv preprint arXiv:1802.09464.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In International conference on machine learning (pp. 1889–1897). PMLR.

Schulman, J., Wolski, F., Dhariwal, P.,Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Stulp, F., Theodorou, E.,Kalakrishnan, M.,Pastor, P.,Righetti, L., & Schaal, S. (2011). Learning motion primitive goals for robust manipulation. In 2011 IEEE/RSJ international conference on intelligent robots and systems (pp. 325–331). IEEE.

Sutton, R. S., & Barto, A. G. (1999). Reinforcement learning. Journal of Cognitive Neuroscience, 11(1), 126–134.

Tobin, J., Fong, R.,Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS) (pp. 23–30). IEEE.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pluquet, O., Piot, B.,Heess, N.,Roth, T.,Lampe, T., & Riedmiller, M. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. arXiv preprint arXiv:1707.08817.

Wei, H., Yijie, B., & Zhu, Z. (2020). TriFinger: An open-source robot for learning dexterity. arXiv preprint arXiv:2008.03596.

**AUTHOR BIOGRAPHIES**

**Qiang Wang** is a PhD candidate at University College Dublin in Ireland. He acquired his 4-year Bachelor’s degree accredited jointly by Beijing University of Technology and University College Dublin. His current research interests involve reinforcement learning, machine vision, machine touch, representation learning, and robotics.

**Francisco Roldan Sanchez** is a deep learning engineer from Barcelona, who studied telecommunications engineering at Universitat Politecnica de Catalunya and specialised in computer vision at the Computer Vision Center at Universitat Autonoma de Barcelona. Since 2020, he has been pursuing a PhD at Dublin City University and his main research focus is reinforcement learning in the context of robotic manipulation.

**Robert McCarthy** is a PhD student in the Department of Computer Science at University College London, where he is currently performing robotics and AI research. He holds a BSc (2019) and ME (2021) in Biomedical Engineering from University College Dublin, and has previously worked as a data scientist at CeADAR—Ireland’s Centre for Applied AI (2021–2022).

**David Cordova Bulens** is a Postdoctoral Research Fellow at University College Dublin studying the human sense of touch, tactile sensors, and robotics dexterous manipulation. He obtained a BS degree (2011) in engineering, an MS (2013) in electro-mechanical engineering, and a Ph.D. (2017) in Neurosciences at Université Catholique de Louvain (UCLouvain).

**Kevin Mcguinness** is an Assistant Professor in the School of Electronic Engineering at Dublin City University and SFI Funded Investigator in the Insight SFI Research Centre for Data Analytics. He has a BSc (Hons) in Computer Applications (2005) and a PhD in Computer Vision (2009) from Dublin City University. Since 2009 he has been a postdoctoral researcher at the CLARITY Centre for Sensor Web Technologies, a Research Fellow at the Insight Centre for Data Analytics, and now teaches graduate-level data analysis and machine learning for the School of Electronic Engineering. Kevin has 100+ peer-reviewed publications focused on topics in computer vision, machine learning, and deep learning.
Noel O'Connor is a Full Professor in the School of Electronic Engineering at Dublin City University (DCU) Ireland. He is CEO of the Insight SFI Research Centre for Data Analytics, Ireland's largest SFI-funded Research Centre. The focus of Noel's research is in multi-modal content analysis for applications in security/safety, autonomous vehicles, medical imaging, ambient assisted living, multimedia content-based retrieval, and environmental monitoring. He is an Area Editor for both Signal Processing: Image Communication (Elsevier) and ACM Transactions on Multimedia Computing, Communications, and Applications and as acted as General Chair and Programme Chair for a number of high ranking conferences. Noel collaborates widely with international researchers in Uni. of California, Irvine, Arizona State University, Uni. of Oldenburg, Universitat Politècnica de Catalunya, Uni. of New England (Australia), among others. His commercial experience includes a successful spin-out company and mentorship of two others.

Manuel Wüthrich is a postdoctoral researcher at Harvard University with a focus on computational economics and societal impact of AI. Previously, he worked on a broad range of topics in AI, reinforcement learning, representation learning, control theory, and robotics at the Max-Planck-Institute for Intelligent Systems and ETH Zurich, where he completed his PhD as a fellow of the MPI/ETH for Center for Learning Systems.

Felix Widmaier received the MSc in Computer Science from the University of Tübingen, Germany in 2015. He currently works as a software engineer at the Max Planck Institute for Intelligent Systems in Tübingen, where he develops and maintains the software stack of the TriFinger robots and is a co-organizer of the yearly Real Robot Challenge.

Stefan Bauer is an assistant professor at KTH Stockholm and a CIFAR Azrieli Global Scholar. He obtained his PhD in Computer Science from ETH Zurich and was awarded with the ETH medal for an outstanding doctoral thesis. Before that, he graduated with a BSc and MSc in Mathematics from ETH Zurich and a BSc in Economics and Finance from the University of London. During his studies, he held scholarships from the Swiss and German National Merit Foundation. In 2019, he won the best paper award at the International Conference of Machine Learning (ICML) and in 2020 and 2021, he was one of the lead organisers of the real-robot-challenge.com, a robotics challenge in the cloud.

Stephen J. Redmond is an Associate Professor at the School of Electrical and Electronic Engineering at University College Dublin (UCD). He completed his Bachelor of (Electronic) Engineering at UCD in 2002. He also completed his PhD in biosignal processing at the same institute, in 2006, on the topic of at-home sleep staging. He spent 10 years at the University of New South Wales in Sydney, before returning to UCD in 2018 to lead a Science Foundation Ireland President of Ireland Future Research Leaders Award project on the topic of tactile sensing. He is also a co-founder of the tactile sensor company Contactile: www.contactile.com. He has an interest in the use of signal processing and pattern recognition techniques, combined with novel sensing hardware solutions, in order to bring longitudinal health monitoring into the home. In particular, he has extensive experience monitoring human movement using wearable sensors, and in the robust measurement of physiological signals (ECG, BP & PPG) in unsupervised environments. He also has a research stream on the topic of tactile sensing, spanning from the study of human tactile physiology, through to the development of friction-sensing tactile sensors for robotic grippers. More recently, his group has been interested in the use of deep neural networks for medical image segmentation, and for use with reinforcement learning to enable autonomous dexterous robotic manipulation.

How to cite this article: Wang, Q., Sanchez, F. R., McCarthy, R., Bulens, D. C., McGuinness, K., O'Connor, N., Wüthrich, M., Widmaier, F., Bauer, S., & Redmond, S. J. (2023). Dexterous robotic manipulation using deep reinforcement learning and knowledge transfer for complex sparse reward-based tasks. Expert Systems, 40(6), e13205. https://doi.org/10.1111/exsy.13205
APPENDIX A: NEURAL NETWORK

A.1 Architecture

The code used in our work is mostly from https://github.com/TianhongDai/hindsight-experience-replay, and we have made some edits to adapt it to our environment. The code of Move Cube on Trajectory can be found here https://github.com/RobertMcCarthy97/rrc_phase1, and the code of Move Cube on Trajectory Pro can be found here https://github.com/wq13552463699/rrc_pos_ori. The structures of the actor and critic neural networks are shown in Figure A1.

![Neural Network Diagram](image)

**FIGURE A1** The neural network structure of the actor and critic. The action output by the actor multiplies the max action ratio before applying it to the environment to make sure it cannot exceed the limits of the action space.

A.2 Hyper-parameter

Hyperparameter related to the task/training/testing can be seen in the sub-folder rrc_example_package/her/arguments.py of the GitHub repositories above.

- **Learning rate of the actor**: 0.001
- **Learning rate of the critic**: 0.001
- **Discount factor**: 0.98
- **Batch size**: 256
- **The average coefficient for soft-update of the target networks**: 0.95
- **Replay buffer size**: 1,000,000
- **HER replay strategy**: future
### APPENDIX B: RAW DATA

#### TABLE B1: The evaluated scores of DRL agents trained with different seeds on both simulator and real robot

| Num | Seed 0 | Seed 123 | Seed 200 |
|-----|--------|----------|----------|
|     | Rob1   | Sim      | Rob6     | Sim      | Rob5     | Sim      |
| 1   | −12,115| −8288    | −15,639  | −9200    | −10,120  | −8379    |
| 2   | −15,167| −9430    | −14,262  | −12,781  | −16,159  | −12,284  |
| 3   | −23,378| −8290    | −8257    | −10,560  | −21,487  | −9197    |
| 4   | −21,338| −10,045  | −11,242  | −9759    | −23,810  | −7111    |
| 5   | −25,971| −7998    | −11,947  | −8762    | −10,207  | −9745    |
| 6   | −13,497| −8878    | −18,063  | −8536    | −12,042  | −9183    |
| 7   | −17,905| −9567    | −13,341  | −10,685  | −12,958  | −9670    |
| 8   | −22,253| −9332    | −15,494  | −10,022  | −15,986  | −15,520  |
| 9   | −11,608| −7669    | −12,824  | −7113    | −18,867  | −10,239  |
| 10  | −11,805| −8351    | −21,261  | −15,298  | −17,593  | −8557    |
| 11  | −10,500| −8677    | −20,369  | −9053    | −16,410  | −16,587  |
| 12  | −19,996| −9367    | −23,302  | −9123    | −18,469  | −11,242  |
| 13  | −20,676| −8585    | −18,059  | −8727    | −11,228  | −13,206  |
| 14  | −12,666| −7764    | −18,087  | −8337    | −10,918  | −12,380  |
| 15  | −11,413| −9877    | −13,749  | −10,143  | −14,727  | −23,389  |
| Avg | −16,686| −8808    | −15,726  | −9873    | −15,399  | −11,779  |

Note: All DRL agents are trained with DR from scratch. We did not run multiple times for each agent on the real robot, because agent performances are generally poor. This table is the a part of the raw data of Table 3, before averaging.

#### TABLE B2: The evaluated scores of DRL agents trained with different seeds on both simulator and real robot

| Num | Seed 0 | Seed 123 | Seed 200 |
|-----|--------|----------|----------|
|     | Rob1   | Rob5     | Rob6     | Sim      | Rob1   | Rob5     | Rob6     | Sim      |
| 1   | −9970  | −11,190  | −11,512  | −5635    | −8789  | −7328    | −10,030  | −6579    |
| 2   | −11,412| −9017    | −11,461  | −5029    | −8086  | −10,263  | −13,828  | −6650    |
| 3   | −10,746| −12,084  | −10,900  | −5175    | −11,328 | −9908    | −8270    | −7180    |
| 4   | −7,572 | −15,767  | −11,904  | −5570    | −9,928  | −8228    | −9110    | −6314    |
| 5   | −10,883| −10,430  | −13,682  | −5430    | −13,106 | −8263    | −10,662  | −5746    |
| 6   | −7,896 | −11,427  | −10,371  | −5895    | −11,644 | −10,245  | −6619    | −5932    |
| 7   | −7,942 | −14,505  | −14,028  | −4977    | −6,447  | −11,307  | −12,355  | −6106    |
| 8   | −8,461 | −11,135  | −13,604  | −5698    | −9,180  | −7481    | −11,375  | −6551    |
| 9   | −6,441 | −12,871  | −9,460   | −4921    | −10,841 | −6781    | −7,524   | −6560    |
| 10  | −11,466| −11,620  | −10,030  | −5702    | −4,951  | −8415    | −10,385  | −6257    |
| 11  | −9,513 | −5,192   | −15,004  | −5259    | −9,020  | −8248    | −7,751   | −6055    |
| 12  | −11,928| −10,046  | −11,051  | −6016    | −6,250  | −9,530   | −11,040  | −6722    |
| 13  | −14,798| −11,213  | −12,311  | −5027    | −8,987  | −10,205  | −11,442  | −7148    |
| 14  | −10,501| −12,419  | −16,049  | −5531    | −8,834  | −9,002   | −9,852   | −6101    |
| 15  | −9,178 | −10,589  | −11,256  | −5847    | −12,243 | −8,691   | −16,036  | −5732    |
| Avg | −9,914 | −11,300  | −12,175  | −5448    | −9,309  | −8,926   | −10,017  | −6376    |

Note: All DRL agents are trained from scratch without DR. This table is the raw data of Tables 3 and 2 before averaging.
The evaluated scores of DRL agents trained with different seeds on both simulator and real-robot

| Num | Seed 0 | Seed 123 | Seed 200 |
|-----|--------|----------|----------|
|     | Rob1   | Rob5     | Rob6     | Sim     |
| 1   | -8.787 | -6.947   | -7.396   | -5.257  |
| 2   | -11.681| -6.962   | -10.088  | -6.809  |
| 3   | -6.013 | -8.187   | -6.318   | -5.442  |
| 4   | -10.352| -6.727   | -4.912   | -5.169  |
| 5   | -8.789 | -7.936   | -7.730   | -6.081  |
| 6   | -10.460| -11.045  | -7.351   | -5.908  |
| 7   | -8.279 | -10.296  | -11.356  | -6.426  |
| 8   | -10.261| -9.046   | -6.375   | -6.043  |
| 9   | -6.371 | -11.219  | -6.704   | -4.671  |
| 10  | -10.443| -11.136  | -11.567  | -6.491  |
| 11  | -6.236 | -7.585   | -11.795  | -5.751  |
| 12  | -8.060 | -6.528   | -10.957  | -3.804  |
| 13  | -9.808 | -8.811   | -8.557   | -5.418  |
| 14  | -10.962| -11.804  | -11.804  | -7.607  |
| 15  | -7.681 | -7.249   | -10.592  | -6.500  |
| Avg | -8.946 | -8.945   | -8.876   | -5.825  |

Note: All DRL agents are trained from scratch without DR and then tuned by DR. This table is a part of the raw data of Table 3 before averaging.

The evaluated performance of agent with different KT policies

| Num | ACTOR-CRITIC | ACTOR | COLLECT | SCRATCH | TEACHER |
|-----|--------------|-------|---------|---------|---------|
|     | Pos (m)      | Ori (rad) | Pos (m) | Ori (rad) | Pos (m) | Ori (rad) | Pos (m) | Ori (rad) |
| 1   | 0.0201       | 1.3000  | 0.0698  | 1.7481  | 0.0248  | 1.5535  | 0.1654  | 2.5957  |
| 2   | 0.0253       | 1.2571  | 0.0643  | 1.6544  | 0.0304  | 1.6287  | 0.1143  | 2.4124  |
| 3   | 0.0185       | 1.4060  | 0.0579  | 1.7415  | 0.0301  | 1.4512  | 0.1512  | 2.3510  |
| 4   | 0.0227       | 1.3226  | 0.0797  | 1.8560  | 0.0276  | 1.5742  | 0.1127  | 2.5419  |
| 5   | 0.0221       | 1.2786  | 0.0651  | 1.8221  | 0.0341  | 1.4510  | 0.1208  | 2.2099  |
| 6   | 0.0262       | 1.3984  | 0.0664  | 1.7988  | 0.0271  | 1.4512  | 0.1310  | 2.3118  |
| 7   | 0.0243       | 1.2020  | 0.0691  | 1.6806  | 0.0349  | 1.5679  | 0.1049  | 2.4833  |
| 8   | 0.0198       | 1.1144  | 0.0619  | 1.6527  | 0.0226  | 1.3843  | 0.1732  | 2.3918  |
| 9   | 0.0308       | 1.3189  | 0.0639  | 1.6251  | 0.0297  | 1.4060  | 0.1406  | 2.5777  |
| 10  | 0.0219       | 1.2269  | 0.0611  | 1.6519  | 0.0340  | 1.7080  | 0.1248  | 2.6528  |
| 11  | 0.0225       | 1.7139  | 0.0601  | 1.6495  | 0.0339  | 1.4148  | 0.1215  | 2.5674  |
| 12  | 0.0206       | 1.5068  | 0.0738  | 1.8200  | 0.0320  | 1.4506  | 0.1141  | 2.4468  |
| 13  | 0.0216       | 1.3547  | 0.0694  | 1.7752  | 0.0331  | 1.3885  | 0.1280  | 2.4678  |
| 14  | 0.0255       | 1.2386  | 0.0626  | 1.6965  | 0.0357  | 1.3612  | 0.1348  | 2.5125  |
| 15  | 0.0227       | 1.1980  | 0.0666  | 1.7335  | 0.0287  | 1.4212  | 0.1604  | 2.6438  |
| Avg | 0.0230       | 1.3225  | 0.0661  | 1.7270  | 0.0306  | 1.4808  | 0.1332  | 2.4790  |

Note: Where Pos is the deviation between actual and desired position and Ori is the deviation of the orientation. The deviation is calculated by averaging the accumulated reward over 120,000 time-step.