# Solving the Real Robot Challenge Using Deep Reinforcement Learning

Robert McCarthy[1]*, Francisco Roldan Sanchez[2,3], Qiang Wang[1], David Cordova Bulens[1], Kevin McGuinness[2,3], Noel O'Connor[2,3], and Stephen Redmond[1,3]

[1] University College Dublin, Ireland
[2] Dublin City University, Ireland
[3] Insight SFI Research Centre for Data Analytics, Ireland

**Abstract.** This paper details our winning submission to Phase 1 of the 2021 Real Robot Challenge[4]; a challenge in which a three-fingered robot must carry a cube along specified goal trajectories. To solve Phase 1, we use a pure reinforcement learning approach which requires minimal expert knowledge of the robotic system or of robotic grasping in general. A sparse, goal-based reward is employed in conjunction with Hindsight Experience Replay to teach the control policy to move the cube to the desired x and y coordinates. Simultaneously, a dense distance-based reward is employed to teach the policy to lift the cube to the desired z coordinate. The policy is trained in simulation with domain randomisation before being transferred to the real robot for evaluation. Although performance tends to worsen after this transfer, our best policy can successfully lift the real cube along goal trajectories via an effective pinching grasp. Our approach[5] outperforms all other submissions, including those leveraging more traditional robotic control techniques, and is the first pure learning-based method to solve this challenge.

**Keywords:** Robotic Manipulation · Deep Reinforcement Learning · Real Robot Challenge.

## 1 Real Robot Challenge

Dexterous robotic manipulation is applicable in various industrial and domestic settings. However, current state-of-the-art robotic control strategies generally struggle in unstructured tasks which require high degrees of dexterity. Data-driven learning methods are promising for these challenging manipulation tasks, yet related research has been limited by the costly nature of real-robot experimentation. In light of these issues, the Real Robot Challenge (RRC) [1] aims to advance the state-of-the-art in robotic manipulation by providing participants

---

[4] https://real-robot-challenge.com       *robert.mccarthy@ucdconnect.ie

[5] Code: https://github.com/RobertMcCarthy97/rrc_phase1. Videos: https://www.youtube.com/playlist?list=PLLJoWXUn8XplFszi16-VZMTDBhMQFuc5o.

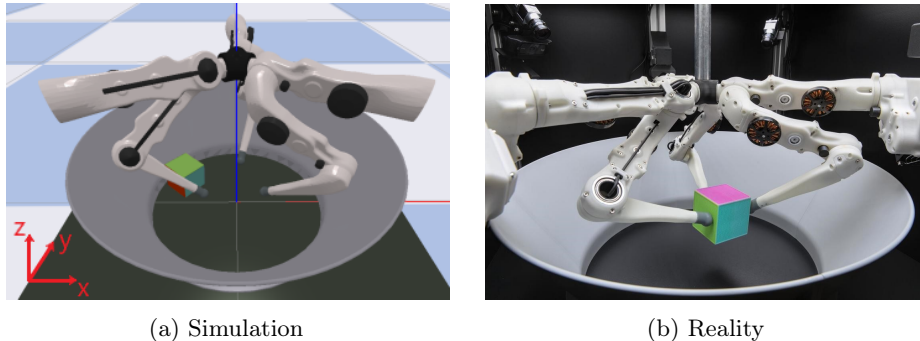(a) Simulation                    (b) Reality

Fig. 1: The simulated and real *'Move Cube on Trajectory'* Trifinger robotic environments. The task is to bring the cube to specified 3-D goal coordinates, along a goal trajectory.

with remote access to well-maintained robotic platforms, allowing for cheap and easy real-robot experimentation. To further support easy experimentation, users are also provided with a simulated version of the robotic setup (see Figure 1).

The 2021 RRC consists of an initial qualifying Pre-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[4]. This paper focuses solely on our approach to Phase 1 of the competition.

In Phase 1, participants are tasked with solving the challenging *'Move Cube on Trajectory'* task. In this task, a cube must be carried along a goal trajectory (which specifies the coordinates at which the cube should be positioned at each time-step) using the provided TriFinger robotic platform [2]. For final Phase 1 evaluation, participants submit their developed control policy and receive a score based on how closely it can follow several randomly sampled goal trajectories.

*'Move Cube on Trajectory'* requires a dexterous policy that can adapt to the various goal and cube positions encountered during an evaluation episode. Last year (2020), the winning solutions to this task consisted of structured policies which relied heavily on inductive biases and task specific engineering [3,4]. We take an alternative approach, formulating the task as a pure reinforcement learning (RL) problem. We then use RL to learn our control policy entirely in simulation before transferring it to the real robot for final evaluation. Upon this evaluation, our learned policy outperformed all other competing submissions, winning Phase 1 of the 2021 RRC.

## 2   Related Work

### 2.1   Traditional Robotic Manipulation

Traditional robotic manipulation controllers often rely on solving inverse kinematic equations [5]. The goal of this approach is to find the parameters needed to

position the end-effector of a robotic system (gripper, finger tips, etc.) into the desired position and orientation [6]. 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 introduced [7,8]. These primitives can each have a defined cost, allowing the robot to avoid non-smooth or non-desired transitions. Exteroceptive feedback in the form of sensors (RGB cameras, depth/tactile sensors, etc.) is usually employed to help the robot achieve the expected behaviour [9,10].

Most successful approaches in previous editions of the Real Robot Challenge make use of a combination of motion planning and motion primitives. The winning team of the 2020 edition of the challenge [4] used a set of primitives to: (i) align the cube to the target position and orientation while keeping it on the ground, and then (ii) perform grasp planning using a Rapidly-exploring Random Tree (RRT) algorithm [11]. During the grasping planning, they use force control feedback to ensure the finger tips apply enough force to lift the cube. Finally, they improve their policy via (simulated) residual policy learning [12], a technique which uses RL to learn corrective actions added to the output of the original control policy. Contrary to these methods, we use a pure learning-based approach which requires minimal task specific engineering.

## 2.2   Reinforcement Learning for Robotic Manipulation

Deep RL methods promise to allow learning of sophisticated, dexterous robotic manipulation strategies that would otherwise be impossible, or at least very difficult, to hand-engineer. However, the data inefficiency of RL is a major barrier to its application in real-world robotics: real robot data collection is time-consuming and expensive. Thus, much RL research to-date has focused on resolving or by-passing these data-efficiency issues.

Due to their generally improved sample complexity, off-policy RL methods [13,14] are often preferred to on-policy methods [15,16]. Model-based RL methods, which explicitly learn a model of their environment, have been proposed to further improve sample complexity [17,18,19], and have seen success in real robot settings, e.g., with in-hand object manipulation [20]. Offline RL techniques seek to leverage previously collected data to accelerate learning [21], and have learned dexterous real-world skills such as drawer opening [22]. Imitation learning methods provide the policy with expert demonstrations to learn from [23,24], enabling success in real robot tasks such as peg insertion [25]. Finally, simulation-to-real (sim-to-real) transfer methods train a policy quickly and cheaply in simulation before deploying it on the real robot, and have notably been used to solve a Rubik's cube with a robot hand [26]. 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 randomisation [27,28] or domain adaptation [29,30] techniques. Domain randomisation, which has been particularly effective [26], randomises the physics parameters in simulation to learn a robust policy that can adapt to the partially unknown physics of the real system.

Provided with a simulated replica of the real robotic setup, but without access to prior data or expert demonstrations, we use sim-to-real transfer to bypass real-robot RL data-efficiency issues.

## 3    Background

**Goal-based Reinforcement Learning.** We frame the RRC robotic environments as a Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{G}, p, r, \gamma, \rho_0)$. $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{G}$ are the state, action and goal spaces, respectively. The state transition distribution is denoted as $p(s'|s, a)$, the initial state distribution as $\rho_0(s)$, 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: $\pi^* = \mathrm{argmax}_\pi \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, g_t)]$.

**Deep Deterministic Policy Gradients (DDPG).** DDPG [13] is an off-policy RL algorithm which, in the goal-based RL setting, maintains the following neural networks: a policy (actor) $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$, and an action-value function (critic) $Q : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$. The critic is trained to minimise the loss $\mathcal{L}_c = \mathbb{E}(Q(s_t, g_t, a_t) - y_t)^2$, where $y_t = r_t + \gamma Q(s_{t+1}, g_{t+1}, \pi(s_{t+1}, g_{t+1}))$. To stabilize the critics training, the targets $y_t$ are produced using slowly updated polyak-averaged versions of the main networks. The actor is trained to minimise the loss: $\mathcal{L}_a = -\mathbb{E}_s Q(s, g, \pi(s, g))$, 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).

**Hindsight Experience Replay (HER).** HER [31] 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 1). 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 $g$ 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 how to achieve $g'$, thus accelerating its acquisition of skills.

## 4    Methods

We train our control policy in simulation with RL before transferring it to the real robot for evaluation. This allows for quicker and easier data collection versus real robot training. To compensate for modelling errors in the simulator, we randomise the simulation dynamics [28]. DDPG + HER is maintained as the RL algorithm, modified slightly to suit our two-component reward system. We now describe in detail our simulated environment, followed by our learning algorithm.

### 4.1   Simulated Environment

**Actions and Observations.** Pure torque control of the robot arms is employed with an action frequency of 20 Hz (i.e. each time-step in the environment is 0.05 seconds). The robot has three arms, with three motorised joints in each arm; thus the action space is 9-dimensional (and continuous). Observations include: (i) robot joint positions, velocities, and torques; (ii) the provided estimate of the cube's pose (i.e. its estimated position and orientation), along with the difference between the current and previous time-step's pose; and (iii) the goal coordinates at which the cube should currently be placed (i.e. the active goal of the trajectory). In total, the observation space has 44 dimensions.

**Episodes.** In each simulated training episode, the robot begins in its default position and the cube is placed in a uniformly random position on the arena floor. Episodes last for 90 time-steps, with the active goal of the randomly sampled goal trajectory changing every 30 time-steps.

**Domain Randomisation.** To help the learned policy generalize from an inaccurate simulation to the real environment, we used some basic domain randomisation (i.e., physics randomisation) during training[6]. 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) each episode. To account for noisy real-robot actuations and observations, uncorrelated noise is added to actions and observations within simulated episodes.

### 4.2   Learning Algorithm

The goal-based nature of the *'Move Cube on Trajectory'* task makes HER a natural fit; HER has excelled in similar goal-based robotic tasks [31] and obviates the need for complex reward engineering. As such, we use DDPG + HER as our RL algorithm[7]. However, in our early experiments we observed that standard DDPG + HER was slow in learning to lift the cube. To resolve this issue, we alter slightly the HER process and incorporate an additional dense reward which encourages cube-lifting behaviors, as is now described.

**Rewards and HER.** In our approach, the agent receives two reward components: (i) a sparse reward based on the the cube's x-y coordinates, $r_{xy}$, and (ii) a dense reward based on the cube's z coordinate, $r_z$ (the coordinate frame can be seen in Figure 1 (a)).

---

[6] Our domain randomization implementation is based on the benchmark code from the 2020 RRC [3].

[7] Our DDPG + HER implementation is taken from `https://github.com/TianhongDai/hindsight-experience-replay`, and uses hyperparameters largely based on [32].

The sparse x-y reward is calculated as:

$$r_{xy} = \begin{cases} 0 & \text{if} \quad \left\| g'_{xy} - g_{xy} \right\| \leq 2cm \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

where $g'_{xy}$ are the x-y coordinates of the *achieved* goal (the actual x-y coordinates of the cube), and $g_{xy}$ are the x-y coordinates of the *desired* goal.

The dense z reward is defined as:

$$r_z = \begin{cases} -a\left| z_{cube} - z_{goal} \right| & \text{if} \quad z_{cube} < z_{goal} \\[2mm] \dfrac{-a}{2}\left| z_{cube} - z_{goal} \right| & \text{if} \quad z_{cube} > z_{goal} \end{cases} \tag{2}$$

where $z_{cube}$ and $z_{goal}$ are the z-coordinates of the cube and goal, respectively, and $a$ is a parameter which weights $r_z$ relative to $r_{xy}$ (we use $a = 20$).

We only apply HER to the x-y coordinates of the goal; i.e., the x-y coordinates of the goal can be altered in hindsight, but the z coordinate remains unchanged. Thus, our HER altered goals are: $\hat{g} = (g'_x, g'_y, g_z)$, 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 x-y plane of the arena floor, it can then more easily stumble upon actions which lift it. Importantly, the $r_{xy}$ + HER approach requires no complicated reward engineering.
2. $r_z$ aims to explicitly teach the agent to lift the cube by encouraging minimisation 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 x-y coordinates of the goal, our HER altered goals, $\hat{g}$, maintain their original z height. This leaves more room for the agent to be rewarded by $r_z$ for any cube lifting it performs.

**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
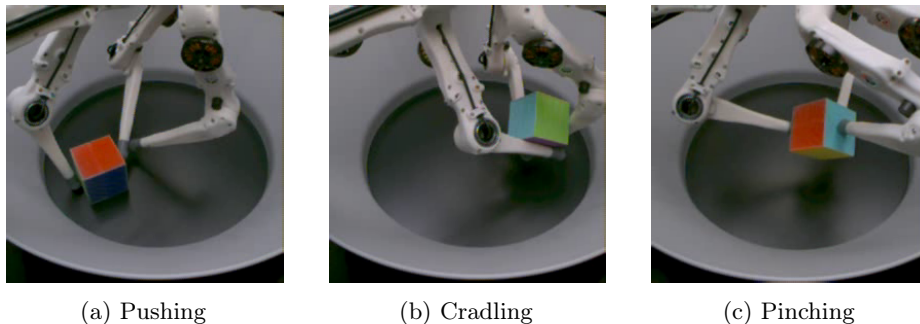
(a) Pushing                (b) Cradling                (c) Pinching

Fig. 2: The various manipulation strategies learned by our approach.

$g_{t+1} = g_t$, even if in reality $g_{t+1}$ was different[8]. Thus, the policy focuses solely on achieving the current active goal and is unconcerned by any future changes in the active goal.

**Exploration vs Exploitation.** We derive our DDPG + HER hyperparameters from Plappert et al. [32], who use a highly 'exploratory' policy when collecting data in the environment: with probability 30% 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 'exploiting' 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 exploiting 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 3).

## 5   Results

### 5.1   Simulation

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 3 compares the training performance of our final algorithm to that of standard HER[9]. Our

---

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

algorithm converges in roughly $\frac{2}{3}$ the time of standard HER, and is markedly improved in the the early stages of training; this allowed us to iteratively develop our approach more quickly. Throughout different training runs, our policies learned several different manipulation strategies, the most distinct of which included: (i) '*pinching*' the cube with two arm tips and supporting it with the third, and (ii) '*cradling*' the cube with all three of its forearms (see Figure 2).
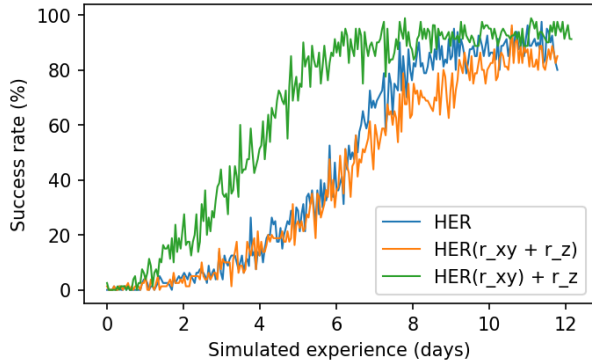


Fig. 3: Success rate vs experience collected during simulated training (1 day $\approx$ 1.7 million environment steps). We compare training with: (i) HER applied to a standard sparse reward (blue), (ii) HER applied to both $r_{xy}$ and $r_z$ (orange), and (iii) our final method where HER is applied to $r_{xy}$ but not to $r_z$. An episode is deemed successful if, when complete, the final goal of the trajectory has been achieved.

Table 1: Self-reported evaluation scores of our learned *pushing*, *cradling*, and *pinching* policies when deployed on the simulated and real robots (mean $\pm$ standard deviation score over 10 episodes). Scores are based on the cumulative position error of the cube during an episode: $score = \sum_{t=0}^{n} -\left(\frac{1}{2}\frac{||\mathbf{e}_{xy}^t||}{d_{xy}} + \frac{1}{2}\frac{|e_z^t|}{d_z}\right)$, where $\mathbf{e}^t = (e_x^t, e_y^t; e_z^t)$ is the error between the cube and goal position at time-step $t$, $d_{xy}$ the arena range on the x-y plane, and $d_z$ the range on the z-axis.

|  | Pushing | Cradling | Pinching |
|---|---|---|---|
| Simulation | -20,399$\pm$3,799 | -6,349$\pm$1,039 | **-6,198**$\pm$1,840 |
| Real robot | -22,137 $\pm$ 3,671 | -14,207 $\pm$ 2,160 | **-11,489** $\pm$ 3,790 |

### 5.2   Real Robot

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 x-y 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 (see `https://real-robot-challenge.com/leaderboard`, username 'thriftysnipe').

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 slide more freely than in simulation. Additionally, on the real robot policies could become stuck with an arm-tip pressing the cube into the wall. As a makeshift solution to this issue, we assumed the policy was stuck whenever the cube had not reached the goal's x-y coordinates for 50 consecutive steps, then uniformly sampled random actions for 7 steps in an attempt to 'free' the policy from its stuck state.

## 6   Discussion

Our relatively simple reinforcement learning approach fully solves the *'Move Cube on Trajectory'* task in simulation. Moreover, our learned policies can successfully implement their sophisticated manipulation strategies on the real robot. Unlike last years benchmark solutions [3], this was achieved with the use of minimal domain-specific knowledge. We outperformed all competing submissions, including those employing more classical robotic control techniques.

Due to the large domain gap, our excellent performances in simulation were not fully matched upon transfer to the real robot. Indeed, the main limitation of our approach was the absence of any training on 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. Similarly, the use of domain adaptation techniques [29,30] could produce a policy more capable of adapting to the real environment. However, ideally the policy could be learned from scratch on the real system; a suitable simulator may not always be available. Although our results in simulation were positive, the algorithm is somewhat sample inefficient, taking roughly 10 million environment steps to converge (equivalent to 6 days of simulated experience). Thus, another important direction for future work would be to reduce sample complexity to increase the feasibility of real robot training; perhaps achievable via a model-based reinforcement learning approach [18,33].

## Acknowledgments

## References

1. Bauer, Stefan, et al. "A Robot Cluster for Reproducible Research in Dexterous Manipulation." arXiv preprint arXiv:2109.10957 (2021).
2. Wüthrich, Manuel, et al. "Trifinger: An open-source robot for learning dexterity." arXiv preprint arXiv:2008.03596 (2020).
3. Funk, Niklas, et al. "Benchmarking Structured Policies and Policy Optimization for Real-World Dexterous Object Manipulation." arXiv preprint arXiv:2105.02087 (2021).
4. Yoneda, Takuma, et al. "Grasp and motion planning for dexterous manipulation for the real robot challenge." arXiv preprint arXiv:2101.02842 (2021).
5. Liu, Rongrong, et al. "Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review." Robotics 10.1 (2021): 22.
6. Wei, Hui, Yijie Bu, and Ziyao Zhu. "Robotic arm controlling based on a spiking neural circuit and synaptic plasticity." Biomedical Signal Processing and Control 55 (2020): 101640.
7. Cohen, Benjamin J., Sachin Chitta, and Maxim Likhachev. "Search-based planning for manipulation with motion primitives." 2010 IEEE International Conference on Robotics and Automation. IEEE, 2010.
8. Stulp, Freek, et al. "Learning motion primitive goals for robust manipulation." 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2011.
9. Montaño, Andrés, and Raúl Suárez. "Manipulation of unknown objects to improve the grasp quality using tactile information." Sensors 18.5 (2018): 1412.
10. Paolo Franceschi and Nicola Castaman "Combining visual and force feedback for the precise robotic manipulation of bulky components", Proc. SPIE 11785, Multimodal Sensing and Artificial Intelligence: Technologies and Applications II, 1178510 (20 June 2021)
11. LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998): 98-11.
12. Silver, Tom, et al. "Residual policy learning." arXiv preprint arXiv:1812.06298 (2018).
13. Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
14. Haarnoja, Tuomas, et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." International conference on machine learning. PMLR, 2018.

15. Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
16. Schulman, John, et al. "Trust region policy optimization." International conference on machine learning. PMLR, 2015.
17. Deisenroth, Marc, and Carl E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." Proceedings of the 28th International Conference on machine learning (ICML-11). 2011.
18. Janner, Michael, et al. "When to trust your model: Model-based policy optimization." arXiv preprint arXiv:1906.08253 (2019).
19. Hafner, Danijar, et al. "Dream to control: Learning behaviors by latent imagination." arXiv preprint arXiv:1912.01603 (2019).
20. Nagabandi, Anusha, et al. "Deep dynamics models for learning dexterous manipulation." Conference on Robot Learning. PMLR, 2020.
21. Levine, Sergey, et al. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems." arXiv preprint arXiv:2005.01643 (2020).
22. Nair, Ashvin, et al. "Accelerating online reinforcement learning with offline datasets." arXiv preprint arXiv:2006.09359 (2020).
23. Pastor, Peter, et al. "Learning and generalization of motor skills by learning from demonstration." 2009 IEEE International Conference on Robotics and Automation. IEEE, 2009.
24. Johns, Edward. "Coarse-to-Fine Imitation Learning: Robot Manipulation from a Single Demonstration." arXiv preprint arXiv:2105.06411 (2021).
25. Vecerik, Mel, et al. "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards." arXiv preprint arXiv:1707.08817 (2017).
26. Akkaya, Ilge, et al. "Solving rubik's cube with a robot hand." arXiv preprint arXiv:1910.07113 (2019).
27. Peng, Xue Bin, et al. "Sim-to-real transfer of robotic control with dynamics randomization." 2018 IEEE international conference on robotics and automation (ICRA). IEEE, 2018.
28. Tobin, Josh, et al. "Domain randomization for transferring deep neural networks from simulation to the real world." 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2017.
29. Arndt, Karol, et al. "Meta reinforcement learning for sim-to-real domain adaptation." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020.
30. Eysenbach, Benjamin, et al. "Off-Dynamics Reinforcement Learning: Training for Transfer with Domain Classifiers." arXiv preprint arXiv:2006.13916 (2020).
31. Andrychowicz, Marcin, et al. "Hindsight experience replay." arXiv preprint arXiv:1707.01495 (2017).
32. Plappert, Matthias, et al. "Multi-goal reinforcement learning: Challenging robotics environments and request for research." arXiv preprint arXiv:1802.09464 (2018).
33. McCarthy, Robert, and Stephen J. Redmond. "Imaginary Hindsight Experience Replay: Curious Model-based Learning for Sparse Reward Tasks." arXiv preprint arXiv:2110.02414 (2021).