



Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

SCHOOL OF ELECTRONIC ENGINEERING

**Sample and time efficient
strategies for off-policy
reinforcement learning in robotic
manipulation tasks**

Francisco Roldan Sanchez, BSc, MSc

A Dissertation submitted in fulfilment of the requirements for
the award of Doctor of Philosophy (PhD)

Supervised by

Prof. Noel E. O'Connor (Dublin City University)
Prof. Kevin McGuinness (Dublin City University)
Prof. Stephen Redmond (University College Dublin)

January 2025

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, and that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Francisco Roldan Sanchez Student number: 20210570

Date: 7th January 2025

Acknowledgements

I would like to express my gratitude to all those who have supported me throughout my PhD journey. This work would not have been possible without the guidance, encouragement, and inspiration from numerous individuals.

First and foremost, I would like to thank my advisors, Noel O'Connor, Stephen Redmond, and Kevin McGuinness, for their invaluable support, patience, and expertise. Their mentorship has been instrumental in shaping my research and academic pursuits, and I am truly grateful for the opportunity to work under their supervision.

A special thanks to my fellow lab members and colleagues, Qiang, Carles, Enric, Eric, Jaime, etc., for their camaraderie, collaboration, and for creating a stimulating research environment.

I am deeply grateful to my family for their unwavering support and encouragement. To my parents, thank you for instilling in me the value of education and hard work, and thank you for your love, understanding, and patience during the demanding phases of my research.

This thesis is a culmination of the efforts and support of many, and I am grateful to each and every one of you.

List of peer-reviewed publications

Publications directly related to the work described in this thesis¹:

- McCarthy, Robert, **Francisco Roldan Sanchez**, Qiang Wang, David Cordova Bulens, Kevin McGuinness, Noel O'Connor, and Stephen J. Redmond. "Solving the real robot challenge using deep reinforcement learning." In 2021 Irish Conference on Artificial Intelligence and Cognitive Science (AICS), 2021.
- Wang, Qiang, **Francisco Roldan Sanchez**, Robert McCarthy, David Cordova Bulens, Kevin McGuinness, Noel O'Connor, Manuel Wüthrich, Felix Widmaier, Stefan Bauer, and Stephen J. Redmond. "Dexterous robotic manipulation using deep reinforcement learning and knowledge transfer for complex sparse reward-based tasks." *Expert Systems* 40, no. 6 (2023): e13205.
- **Sanchez, Francisco Roldan**, Stephen Redmond, Kevin McGuinness, and Noel O'Connor. "Towards advanced robotic manipulation." In 2022 Sixth IEEE International Conference on Robotic Computing (IRC), pp. 302-305. IEEE, 2022.
- **Sanchez, Francisco Roldan**, Qiang Wang, David Cordova Bulens, Kevin McGuinness, Stephen J. Redmond, and Noel E. O'Connor. "Hierarchical reinforcement learning for in-hand robotic manipulation using Davenport chained rotations." In 2023 9th International Conference on Automation, Robotics, and Applications (ICARA), pp. 160-164. IEEE, 2023.
- **Roldan Sanchez, Francisco**, Qiang Wang, David Cordova Bulens, Kevin McGuinness, Stephen Redmond, and Noel O'Connor. "Learning and reusing primitive behaviours to improve Hindsight Experience Replay sample efficiency." In 2024 10th International Conference on Automation, Robotics and Applications (ICARA), pp. 328-333. IEEE, 2024.

Other publications arising from related work:

¹In the case where the author of this thesis is not first author on a publication, the contribution of each author to the publication is explained in the relevant section of the thesis.

-
- Wang, Qiang, Robert McCarthy, David Cordova Bulens, Kevin McGuinness, Noel E. O'Connor, **Francisco Roldán Sánchez**, Nico Gürtler, Felix Widmaier, and Stephen J. Redmond. "Improving Behavioural Cloning with Positive Unlabeled Learning." In 7th Annual Conference on Robot Learning. 2023.
 - Wang, Qiang, Robert McCarthy, David Cordova Bulens, **Francisco Roldan Sanchez**, Kevin McGuinness, Noel E. O'Connor, and Stephen J. Redmond. "Identifying Expert Behavior in Offline Training Datasets Improves Behavioral Cloning of Robotic Manipulation Policies." IEEE Robotics and Automation Letters (2023).
 - Wang, Qiang, Yixin Deng, **Francisco Roldan Sanchez**, Keru Wang, Kevin McGuinness, Noel O'Connor, and Stephen J. Redmond. "Dataset Clustering for Improved Offline Policy Learning." Submitted to ICML 2024. In arXiv preprint arXiv:2402.09550 (2024).

Contents

1	Introduction	1
1.1	Chapter overview	1
1.2	Robotic manipulation and reinforcement learning	2
1.2.1	Sample efficiency	6
1.2.2	Learning time efficiency	7
1.3	Preliminary glossary	8
1.4	Analytical vs Data-driven manipulation	11
1.5	Hypotheses and research questions	13
1.6	Summary of thesis contributions	14
1.7	Thesis structure	15
2	Background	18
2.1	Chapter overview	18
2.2	Introduction to robotic manipulation	19
2.3	Reinforcement learning fundamentals	21
2.3.1	The action-value function	23
2.3.2	Rewards	24
2.3.3	On-policy vs off-policy reinforcement learning	26
2.3.4	Exploration-exploitation trade-off	29
2.4	Off-policy reinforcement learning algorithms	31
2.4.1	Q-learning	31
2.4.2	Deep Q-Networks	32
2.4.3	Soft Actor-Critic	34
2.4.4	Deep Deterministic Policy Gradients	36
2.4.5	Twin-Delayed Deep Deterministic Policy Gradients	37
2.4.6	Summary of algorithms	38
2.5	State of the art review	39
2.5.1	Hindsight Experience Replay	39
2.5.2	Hindsight Experience Replay extensions	42
2.6	Conclusion	48
3	Combination of dense and sparse reward signals	50
3.1	Chapter overview	50
3.2	Motivation	51
3.3	The Real Robot Challenge (RRC)	54
3.3.1	Environment	55
3.3.2	Task definition	56
3.4	Policy learning through reward engineering	57

3.4.1	Network	57
3.4.2	Reward formulation	58
3.4.3	Exploration vs exploitation	60
3.5	Experiments in the simulated environment	61
3.5.1	Experiments formulation	61
3.5.2	Discussion of learning in simulation results	62
3.6	Real robot experiments	64
3.6.1	Manipulation strategies	64
3.6.2	Domain randomisation	67
3.6.3	Discussion of final results and comparison with other teams . .	68
3.7	Applications of our policy	69
3.7.1	Wang <i>et al.</i> 's extension using knowledge transfer	69
3.7.2	Other potential applications	70
3.8	Conclusion	71
3.9	Summary	73
3.9.1	Revisiting the hypothesis	73
3.9.2	Summary of chapter contributions	74
4	Baseline model selection	75
4.1	Chapter overview	75
4.2	Motivation	75
4.3	Environments and tasks	78
4.3.1	Fetch environment	78
4.3.2	Shadow Dexterous Hand environment	83
4.4	Identifying the baseline models	88
4.4.1	Reward systems	88
4.4.2	Policy learning algorithm selection	89
4.5	Conclusion	101
5	Sequential execution of primitive behaviours	103
5.1	Chapter overview	103
5.2	Motivation	104
5.3	Related work	106
5.3.1	Hierarchical control	106
5.3.2	Davenport chained rotations	108
5.4	Curriculum design	113
5.4.1	Obstacle-avoidance manipulation tasks	114
5.4.2	In-hand manipulation tasks	118
5.5	Evaluation	122
5.5.1	Evaluation sets	122
5.5.2	Results	123
5.5.3	Results discussion	124
5.6	Conclusion	130
5.7	Summary	132
5.7.1	Revisiting the hypothesis	132
5.7.2	Summary of chapter contributions	133

6	Dynamically learning and re-using primitive behaviours	134
6.1	Chapter overview	134
6.2	Motivation	135
6.3	Related work	136
6.3.1	Modular Neural Networks	136
6.3.2	Ensemble of policies	137
6.4	Q-switch Mixture of Primitives Hindsight Experience Replay	139
6.4.1	Network	142
6.5	Experiments	142
6.5.1	Environments	142
6.5.2	Experimental setup	143
6.6	Evaluation	146
6.6.1	Fetch environment manipulation tasks	146
6.6.2	In-hand manipulation tasks	161
6.7	Conclusion	168
6.8	Summary	171
6.8.1	Revisiting the hypothesis	171
6.8.2	Summary of chapter contributions	172
7	Conclusions	173
7.1	Chapter overview	173
7.2	Answering the research questions	173
7.3	Summary of thesis contributions	176
7.4	Recommendations for future research	177
7.5	Closing remarks	178
A	Appendix	180
A.1	Reward engineering	180
A.1.1	Grid-search hyperparameter a of Equation 3.3	180
A.1.2	Network hyperparameters	180
A.1.3	Extended domain randomisation results	181
A.2	Sequential execution of primitive behaviours	181
A.2.1	Network Hyperparameters	181
A.3	Automatic use of primitive behaviours	183
A.3.1	Network Hyperparameters	183

List of Figures

1.1	Auto-generated figure by Scopus showing the number of papers tracked by Scopus over the last 40 years that have the terms “reinforcement learning” and “robotics” in the title, abstract or keywords	3
1.2	Auto-generated figure by Scopus showing the number of papers by authors tracked by Scopus over the last 40 years that have the terms “reinforcement learning” and “robotics” in the title, abstract or keywords	4
2.1	Schematic representation of a 7-DoF robotic arm. Image taken from Luo <i>et al.</i> [34]	19
2.2	High-level framework of the reinforcement learning paradigm. At each time-step t , the agent receives an observation of states s_t from the environment, takes an action a_t , and receives a reward r_t	22
2.3	High-level representation of the off-policy reinforcement learning training framework. In order to have a more stable training, several episodes are run using the <i>behaviour policy</i> π_k , and then the <i>target policy</i> π_{k+1} is updated (i.e. the policy that is being evaluated and improved), using the interactions saved in the buffer. In this figure the <i>behaviour policy</i> is the same as the <i>target policy</i> , but this does not necessarily have to be the case.	28
2.4	Visual representation of the HER algorithm. When an episode fails, HER modifies the goal so that it matches with the outcome of the episode. By applying this simple method, the agent is now able to learn from mistakes as well as from successes. Image taken from https://openai.com/research/ingredients-for-robotics-research	41
3.1	Results of Plappert <i>et al.</i> [154] on a pick and place task using DDPG+HER with different reward strategies. The use of sparse rewards and HER is hugely beneficial compared to other approaches. Note that here <i>dense reward</i> refers to a negative Euclidean distance, while <i>sparse reward</i> refers to the same reward system as described in Equation 2.4.	52
3.2	The simulated and real TriFinger robotic environments. The task is to bring the cube to specified 3-D goal coordinates, along a goal trajectory.	54
3.3	Image of a real cube identical to those used during the challenge.	55
3.4	Network architecture of the DDPG agent, consisting of an actor which estimates the action a_t given a state of the environment s_t and a goal g ; and a critic, which evaluates the quality of the taken action a_t	58

3.5	Success rate vs experience collected during simulated training (1 day in simulator, ≈ 1.7 million environment time-steps). We took the average over 3 randomly selected seeds for this illustration. The solid dotted lines are the trend lines and the faded lines are the raw data. We compare the results of learning 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 (blue); (iv) HER applied to pure distance reward, r_{dis} , using a negative Euclidean distance (orange); and (v) Pure distance reward, r_{dis} , without HER (purple); (vi) Standard sparse reward without HER (black).	63
3.6	The various manipulation strategies learned by our team in order to solve the 2021 Real Robot Challenge.	65
3.7	Comparison between agents trained from scratch with DR and without DR in the simulated environment of the TriFinger robot ‘Move Cube on Trajectory’ task.	66
4.1	Visual representation of (a) <i>FetchReach</i> , (b) <i>FetchPush</i> , (c) <i>FetchPickAndPlace</i> , (d) <i>FetchPickObstacle</i> and (e) <i>FetchPickAndThrow</i> tasks.	79
4.2	Visual representation of (a) <i>RotateX</i> , whose goals require to rotate the cube around the x -axis, (b) <i>RotateY</i> , whose goals require to rotate the cube around the y -axis (c) <i>RotateZ</i> , whose goals require to rotate the cube around the z -axis (d) <i>RotateParallel</i> , whose goals consist of a random target rotation around the z -axis and rotations by multiples of $\frac{\pi}{2}$ around the x and y axes and (e) <i>RotateXYZ</i> tasks, whose goals consist of a random target 3D rotation.	83
4.3	Image describing the different joints of the human hand. Image taken from Bullock <i>et al.</i> [184].	86
4.4	Success rate evolution over policy updates of a DDPG + HER agent trained on the <i>FetchReach</i> task of the Fetch environment.	91
4.5	Success rate evolution over policy updates of a DDPG + HER agent trained on the <i>FetchPush</i> task of the Fetch environment.	91
4.6	Success rate evolution over policy updates of a DDPG + HER agent trained on the <i>FetchPickAndPlace</i> task of the Fetch environment.	92
4.7	Success rate evolution over policy updates of the different agents trained on the <i>FetchPickObstacle</i> task of the Fetch environment.	93
4.8	Success rate evolution over policy updates of the different agents trained on the <i>FetchPickAndThrow</i> task of the Fetch environment.	94
4.9	Success rate evolution over policy updates of a DDPG + HER agent trained on the <i>RotateZ</i> task of the Shadow Dexterous Hand environment. In total, the <i>RotateZ</i> task needs 2×10^6 time-steps of simulated experience.	96
4.10	Success rate evolution over policy updates of a DDPG + HER agent trained on the <i>RotateX</i> task of the Shadow Dexterous Hand environment. In total, the <i>RotateX</i> task needs 4×10^6 time-steps of simulated experience.	96

4.11	Success rate evolution over policy updates of a DDPG + HER agent trained on the <i>RotateY</i> task of the Shadow Dexterous Hand environment. In total, the <i>RotateY</i> task needs 4×10^6 time-steps of simulated experience.	97
4.12	Success rate evolution over policy updates of the different agents trained on the <i>RotateParallel</i> task of the Shadow Dexterous Hand environment. In total, the <i>RotateParallel</i> task needs 10^7 time-steps of simulated experience.	98
4.13	Success rate evolution over policy updates of the different agents trained on the <i>RotateXYZ</i> task of the Shadow Dexterous Hand environment. In total, the <i>RotateXYZ</i> task needs 10^7 time-steps of simulated experience.	99
5.1	An example visualisation of a hindsight planner sub-goal decomposition for the Fetch environment and the Shadow Dexterous Hand environments. Image taken from Lai <i>et al.</i> 's paper.	108
5.2	Overview of a decomposition of an arbitrary 3D rotation into a sequence of three extrinsic rotations. In particular, it represents a $(45^\circ, 30^\circ, -60^\circ)$ rotation, using the z - x - z , or 3-1-3, extrinsic rotations configuration. Image taken from DF Malan, 14 September 2004, Euler Angles x-convention, https://en.wikipedia.org/wiki/File:EulerX.png	110
5.3	Visualisation of the rectangular sequential trajectory proposed to solve the <i>FetchPickObstacle</i> task using the policy learned on the <i>FetchPickAndPlace</i> task. WP represent the way-points that the cube must visit along the trajectory.	115
5.4	Visualisation of the triangular sequential trajectory proposed to solve the <i>FetchPickObstacle</i> task using the policy learned on the <i>FetchPickAndPlace</i> task. WP represent the way-points that the cube must visit along the trajectory.	116
5.5	Visualisation of the sequential trajectory proposed to solve the <i>FetchPickAndThrow</i> task using the policy learned on the <i>FetchPickAndPlace</i> task, and opening the gripper as soon as the primitive behaviour reaches the assigned intermediate way-point. Note that only one of the 8 way-points is visited, and the selection of the way-point depends on the target bucket into which the block must be thrown.	117
5.6	Example of a 3D rotation goal decomposition into three Davenport angles. A target 3D rotation (a) is decomposed into three consecutive target rotations around the (b) z , (c) x and again (d) z axes, respectively. Goal decomposition can be understood by looking at the target poses, which are the cubes that are not touching the robotic hand, compared to the starting poses for the rotations, which are shown as the cubes in the robotic hand.	119
6.1	Schematic representation of the behaviour of the QMP algorithm. Image taken from Zhang <i>et al.</i> original paper [249].	139

6.2	Schematic representation of the proposed QMP-HER algorithm. For each time-step during exploration, the candidate action that maximises the estimated Q-value is selected and executed. Actions for each time-step are proposed by each primitive policy as well as the target policy that is being updated each cycle.	140
6.3	Success rate of the target policy $\pi(s, g)$ for each policy update for the experiment described in Section 6.5 on the <i>FetchPush</i> task.	146
6.4	Success rate of the target policy $\pi(s, g)$ for each policy update for the experiment described in Section 6.5 on the <i>FetchPickAndPlace</i> task.	147
6.5	Success rate of the target policy $\pi(s, g)$ for each policy update for the experiment described in Section 6.5 on the <i>FetchPickObstacle</i> task.	148
6.6	Success rate of the target policy $\pi(s, g)$ for each policy update for the experiment described in Section 6.5 on the <i>FetchPickAndThrow</i> task.	149
6.7	Success rate evolution per policy update on the <i>FetchPickAndPlace</i> task using the QMP-HER algorithm with different ϵ values for the ϵ -greedy algorithm.	151
6.8	Visualisation of the average usage over 25 episodes during exploration of the behaviours of the primitive policies and the target policy after 1 policy update and 50 policy updates on the <i>FetchPickObstacle</i> task (simplified to <i>PickObstacle</i> in the plot to shorten the title). In this figure, <i>pick</i> refers to an instance of the policy trained to solve the <i>FetchPickAndPlace</i> task with the objective to lift the block over the obstacle, while <i>place</i> refers to an instance of the policy trained to solve the <i>FetchPickAndPlace</i> task with the goal to place the block at the goal position.	153
6.9	Usage of the target policy $\pi(s, g)$ and the primitive behaviour instances <i>pick</i> and <i>place</i> during the exploration phase after the 1st, the 25th and the 50th policy updates. In this figure, <i>pick</i> refers to an instance of the policy trained to solve the <i>FetchPickAndPlace</i> task with the objective to lift the block over the obstacle, while <i>place</i> refers to an instance of the policy trained to solve the <i>FetchPickAndPlace</i> task with the goal to place the block at the goal position.	154
6.10	Visualisation of the average usage over 25 episodes during exploration of the behaviours of the primitive policies and the target policy after 1 policy update and 50 policy updates on the <i>FetchPickAndThrow</i> task (simplified to <i>PickAndThrow</i> in the plot to shorten the title). In this figure, <i>obstacle</i> refers to an instance of the policy trained to solve the <i>FetchPickObstacle</i> task, setting the objective as the 3D coordinates corresponding to the centre of the target bucket, and <i>pick_centre</i> and <i>pick_goal</i> refer to instances of the policy trained to solve the <i>FetchPickAndPlace</i> task, with the objectives to move the block to a centred elevated position and towards the the 3D coordinates corresponding to the centre of the target bucket, respectively.	155

6.11	Usage of the target policy $\pi(s, g)$ and the primitive behaviour instances <i>obstacle</i> , <i>pick_centre</i> , and <i>pick_goal</i> during the exploration phase after the 1st, the 50th and the 200th policy updates. In this figure, <i>obstacle</i> refers to an instance of the policy trained to solve the <i>FetchPickObstacle</i> task, setting the objective as the 3D coordinates corresponding to the centre of the target bucket, and <i>pick_centre</i> and <i>pick_goal</i> refer to instances of the policy trained to solve the <i>FetchPickAndPlace</i> task, with the objectives to move the block to a centred elevated position and towards the the 3D coordinates corresponding to the centre of the target bucket, respectively.	156
6.12	Effect of having the 7 primitives described in Section 6.5 as part of the QMP (i.e., using instances of the policy learned to solve the <i>FetchReach</i> with slight deviations with respect to the object position, as well as the target goal) to solve the <i>FetchPush</i> task (red line), and only using two instances of the <i>FetchReach</i> task to reach the object position and the target goal as part of the QMP (blue line).	157
6.13	Effect of having the 7 primitives described in Section 6.5 as part of the QMP (i.e., using instances of the policy learned to solve the <i>FetchReach</i> with slight deviations with respect to the object position, as well as the target goal) to solve the <i>FetchPickAndPlace</i> task (red line), and only using two instances of the <i>FetchReach</i> task to reach the object position and the target goal as part of the QMP (blue line).	158
6.14	Effect of having two instances of the policy trained to solve the <i>FetchPickAndPlace</i> task (red line), and a combination of the policy trained to solve the <i>FetchPickAndPlace</i> task and the policy trained in the <i>FetchPush</i> task (red line) as part of the QMP used for the behaviour policy in order to solve the <i>FetchPickObstacle</i> task.	159
6.15	Effect of having only an instance of the policy trained to solve the <i>FetchPickAndPlace</i> task (green line), only an instance of the policy trained to solve the <i>FetchPickObstacle</i> task (blue line) and a combination of both (red line) as part of the QMP used for the behaviour policy in order to solve the <i>FetchPickAndThrow</i> task.	160
6.16	Success rate per policy update of the different policies trained in the different experiments explained in Section 6.5 for the <i>RotateParallel</i> task of the Shadow Dexterous Hand environment.	162
6.17	Comparison of the best policy found by of proposed QMP-HGG and the baseline model based on HGG for the <i>RotateParallel</i> task that was found in Chapter 4.	162
6.18	Success rate per policy update of the different policies trained for the different experiments explained in Section 6.5 for the <i>RotateXYZ</i> task of the Shadow Dexterous Hand environment.	164
6.19	Comparison of the best policy found by of proposed QMP-HGG and the baseline model based on HER for the <i>RotateXYZ</i> task that was found in Chapter 4.	165
6.20	Visualisation of the average usage over 25 episodes during exploration of the behaviours of the primitive policies and the target policy after 1 policy update and 100 policy updates on the <i>RotateParallel</i> task.	167

6.21	Usage of the target policy $\pi(s, g)$ and the primitive behaviour instances z_1 , y_1 , and z_2 during the exploration phase after the 1st, the 250th and the 500th policy updates for the <i>RotateParallel</i> task. In this figure, z_1 and z_2 refer to an instance of the policy trained to solve the <i>RotateZ</i> task with the goal set to be the first and third rotations, respectively, of the z - y - z Davenport decomposition, while y_1 refers to an instance of the policy trained to solve the <i>RotateY</i> task with the goal set to be the second rotation of the z - y - z Davenport decomposition.	168
6.22	Visualisation of the average usage over 25 episodes during exploration of the behaviours of the primitive policies and the target policy after 1 policy update and 100 policy updates on the <i>RotateXYZ</i> task. . . .	169
6.23	Usage of the target policy $\pi(s, g)$ and the primitive behaviours instances z_1 , y_1 and z_2 during the exploration phase after the 1st, the 250th and the 500th policy update for the <i>RotateParallel</i> task. z_1 , y_1 and z_2 represent the same as in Figure 6.21.	170

List of Tables

1.1	Comparison of advantages and disadvantages of analytical and data-driven methods. Analytical control methods do not require powerful hardware as there is no learning and are easy to interpret, however, data-driven methods are generally more adaptable and can be more accurate because they use a learned control strategy.	12
3.1	Self-reported scores for the various manipulation strategies learned on the 2021 Real Robot Challenge. Scores are calculated using Equation 3.1.	65
3.2	Evaluation scores of different policies evaluated in simulation and on real robots; three seeds are used in this experiment. Scratch(DR) represents the policy trained from scratch using DR from the beginning of training. Scratch(NDR) is the policy trained from scratch without using DR; Scratch(NDR) + Tune(DR) is the policy firstly trained from scratch without DR and then subsequently tuned using DR in a second training phase. See Appendix A.1.3 for a more detailed view of the results. Scores are calculated using Equation 3.1.	66
3.3	Final scores of the 2021 Real Robot Challenge on the <i>Move Cube on Trajectory</i> task. Our method obtained the best score and won Phase 1 of the challenge. Scores are calculated using Equation 3.1.	69
3.4	The average deviations between achieved and desired position/orientation over 15 evaluation episodes in the simulator; each episode lasts 120,000 time-steps. For the position: $\mathbf{dev}_{\text{dis}} = \ g'_{xyz} - g_{xyz}\ $, which can measure the distance between the actual position of the cube and the goal position. For the orientation: $\mathbf{dev}_{\text{angle}} = \ (R(g'_o))^{-1}R(g_o)\ $, which can measure the angle difference between the actual and target orientations. The deviations of all time-steps are accumulated and finally averaged.	70
4.1	Overview of the information contained in the observation space of the different tasks of the Fetch environment.	81
4.2	Simplified summary of the selection of baseline learning methods for each of the tasks of the Fetch environment. (*) emphasises that the task is not fully solved, rather, it reaches stagnation at approximately 50% success rate after 275 policy updates.	101
4.3	Simplified summary of the baseline learning method selection for each of the tasks of the Shadow Dexterous Hand environment.	101

5.1	Table of matrix products containing the formulas for extracting the rotation angles of any Davenport chained rotation decomposition configuration.	112
5.2	Alternative test set created to evaluate the method. The table shows how many poses are in the test set that can be solved with either 1, 2 or 3 extrinsic rotations.	123
5.3	Success rates on the test set of 1,000 episodes described in Section 5.5.1 for our method and the policy learned using end-to-end HER and HGG on the <i>FetchPickObstacle</i> . <i>Rectangular traj.</i> and <i>triangular traj.</i> represent the curriculum trajectories described in Section 5.4. The estimated training time of the rectangular and triangular trajectories equals the training time of HER trained to solve the <i>FetchPickAndPlace</i> , because it is the only primitive used to generate these results.	124
5.4	Success rates on the test set of 1,000 episodes described in Section 5.5.1 for our method and the policy learned on the <i>FetchPickAndThrow</i> . We name our method “Sequential execution” and its estimated training time is the same as the policy learned to solve the <i>FetchPickAndPlace</i> task as it is the only primitive used to generate these results.	124
5.5	Success rates on the test set of 1,500 episodes described in Section 5.5.1 for our method and the policy learned on the <i>RotateParallel</i> and <i>RotateXYZ</i> . End-to-end HER and end-to-end HGG have been trained using 10^7 timesteps of simulated experience.	125
5.6	Success rates on the test set of 3,200 episodes described in Table 5.2 for our method and the policy learned on the <i>RotateParallel</i> experiment, depending on how many extrinsic rotations the robot is required to perform. End-to-end HGG was trained using 10^7 time-steps of simulated experience.	125
5.7	Success rates on the test set of 6,600 episodes described in Table 5.2 for our method and the policy learned on the <i>RotateXYZ</i> experiment depending on how many extrinsic rotations the robot is required to perform. End-to-end HER has been trained using 10^7 time-steps of simulated experience.	125
6.1	Summary of results in terms of sample efficiency for the different experiments carried out in the Fetch environment. The table shows the number of policy updates until convergence , understanding convergence in this scenario to mean when the success rate curve plateaus with a policy fully or partially capable of solving the task.	150
A.1	Scores for different values of the hyperparameter a over 10000 episodes. The best score was obtain setting it to 20.	180
A.2	The evaluated scores of DRL agents trained with different seeds on both simulator and real robot. 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.	181

A.3	The evaluated scores of DRL agents trained with different seeds on both simulator and real-robot. All DRL agents are trained from scratch without DR. This table is the raw data of Table 3.2 before averaging.	182
A.4	The evaluated scores of DRL agents trained with different seeds on both simulator and real-robot. 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 3.2 before averaging.	182

List of Acronyms

AI	Artificial Intelligence
DDPG	Deep Deterministic Policy Gradients
DDPG + HER	Deep Deterministic Policy Gradients trained in conjunction with Hindsight Experience Replay
DR	Domain Randomization
ER	Experience Replay
G-HGG	Graph-based Hindsight Goal Generation
HER	Hindsight Experience Replay
HGG	Hindsight Goal Generation
HRL	Hierarchical Reinforcement Learning
MNN	Modular Neural Network
QMP-HER	Q-switch Mixture of Primitives Hindsight Experience Replay
QMP	Q-switch Mixture of Policies
RL	Reinforcement Learning
RRC	Real Robot Challenge
SAC	Soft Actor-Critic
TD3	Twin-Delayed Deep Deterministic Policy Gradients
WP	Way-point

Abstract

In recent years, reinforcement learning algorithms have become very popular in solving robotic manipulation tasks. However, these algorithms usually suffer from sample inefficiency, meaning that they require extensive exploration to find an appropriate control policy capable of solving the robotic manipulation tasks targeted, and, consequently, this sample inefficiency leads to training time inefficiency. The work presented in this thesis aims to solve this problem through three main methods: (i) reward engineering; (ii) a sequential execution of primitive skills; and (iii) an automatic execution of primitive policies. In particular, a novel reward that encourages lifting behaviour is presented and evaluated in the context of the Real Robot Challenge competition problem. A novel learning framework for in-hand object 3D rotations based on sequentially executed primitives is proposed and evaluated in the Shadow Dexterous Hand simulated environment of the OpenAI Gym. Finally, a new algorithm is proposed that is capable of automatically deciding, during exploration, whether or not to use a primitive behaviour, with this approach evaluated in different tasks for different manipulators available in the OpenAI Gym.

Chapter 1

Introduction

1.1 Chapter overview

This chapter serves as an introduction to the thesis, highlighting some of the challenges that are tackled throughout the document and presenting the research questions and hypotheses investigated in this work. In particular, Section 1.2 defines the term robotic manipulation and explains how it relates to reinforcement learning, highlighting the two main problems within this field that are addressed throughout the thesis (i.e., sample and time efficiency in RL-based methods for robotic manipulation tasks). Section 1.4 provides a high-level comparison of the advantages and disadvantages of using analytical versus data-driven control algorithms, and justifies the use of data-driven techniques as explored in this thesis. Section 1.5 presents the different research questions and hypotheses that are explored in the thesis, while Section 1.6 presents the different contributions to the field. Lastly, Section 1.7 explains the structure of the remainder of the dissertation.

1.2 Robotic manipulation and reinforcement learning

Robotic manipulation refers to the ability of robotic systems to interact with and manipulate objects within their environment using mechanical actuators and control algorithms [1]. This fundamental capability allows robots to perform a wide range of tasks, such as grasping, lifting, moving, and placing objects with precision and dexterity. Robotic manipulation plays a vital role in various industries, including manufacturing, healthcare, agriculture, and logistics, where automation of tasks can improve productivity, consistency, and safety [2].

In robotic manipulation, objects are manipulated by robot manipulators, which are often robotic arms equipped with end-effectors, such as grippers or tools. These manipulators use a combination of sensory feedback, control algorithms, and mechanical design to perform intricate manipulation tasks effectively [3], [4]. The ability to manipulate objects accurately is essential for robots to perform tasks that require fine motor skills, coordination, and adaptability in dynamic environments. Robotic manipulation encompasses a wide range of activities, not only the traditional simple pick-and-place tasks, but also complex tasks such as assembly operations or delicate object manipulation.

Reinforcement learning (RL) is a type of machine learning that involves an agent learning to make decisions by receiving feedback from its environment in the form of rewards [5]. The agent interacts with its environment, taking actions based on its current state, to maximise the cumulative reward it receives over time. This technology has become a very popular choice amongst researchers in order to solve robotic control tasks (Figure 1.1 shows the trend, while Figure 1.2 shows the most prolific researchers on this topic).

Even though RL gained popularity in the field of artificial intelligence (AI) in the late 20th century, particularly in the 1990s and early 2000s, it was not until the introduction of neural networks in these frameworks that RL became particularly

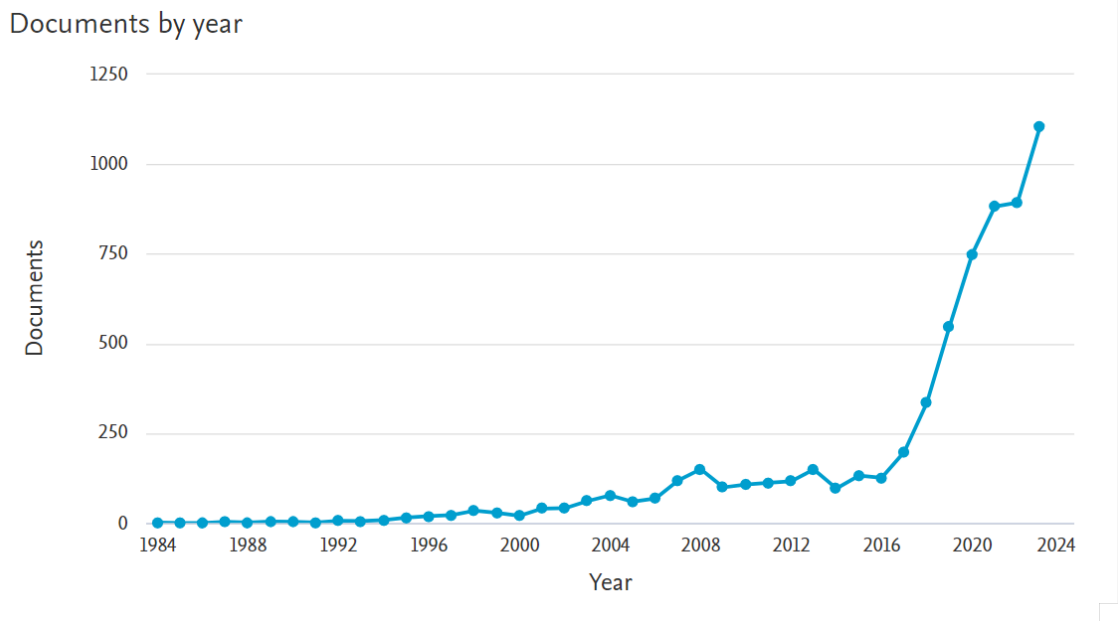


Figure 1.1: Auto-generated figure by Scopus showing the number of papers tracked by Scopus over the last 40 years that have the terms “reinforcement learning” and “robotics” in the title, abstract or keywords .

popular. A major reason for this popularisation was the success of RL in various applications, such as game playing (e.g., backgammon or Go) [6], [7], robotics [8], and autonomous systems [9], [10], which showcased the potential of this approach in solving real-world problems.

Agents in RL refer to the entities that interact with the environment. In the case of robotic manipulation, the robot itself acts as the agent. States represent the different configurations or conditions of the environment that the agent perceives (e.g. in the robotics context, that could include joint configuration information, object position and velocities, etc). Actions are the decisions or movements that the agent can take in a given state to move to the next state. Rewards are feedback signals that the agent receives based on its actions, indicating how well it is performing with respect to the successful completion of the desired task [11].

The goal of RL in robotic manipulation is to train a robot to perform specific tasks or achieve certain objectives by learning from its interactions with the environment. By maximising the cumulative rewards obtained over multiple actions, the robot can learn effective strategies to manipulate objects, navigate its surroundings,

Documents by author

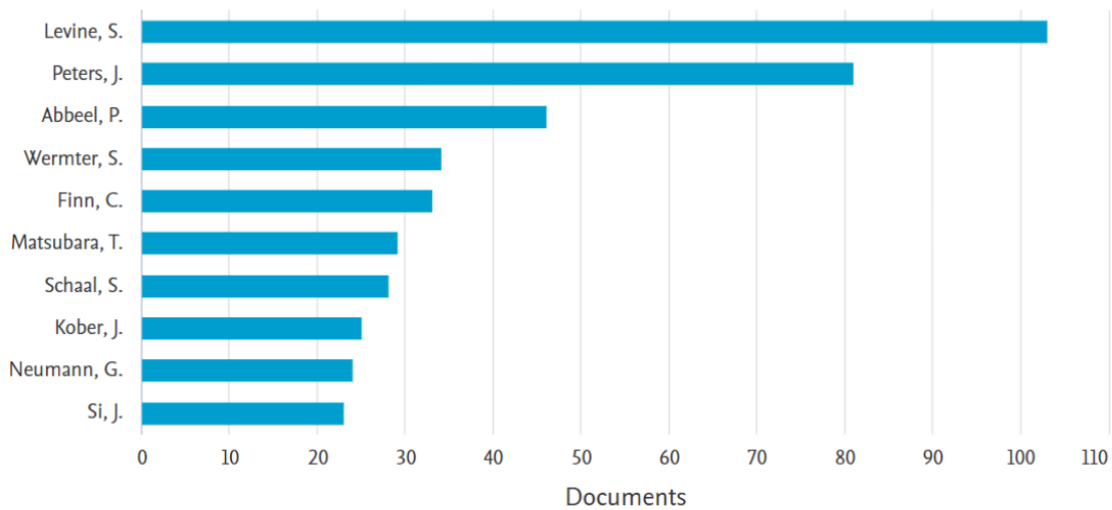


Figure 1.2: Auto-generated figure by Scopus showing the number of papers by authors tracked by Scopus over the last 40 years that have the terms “reinforcement learning” and “robotics” in the title, abstract or keywords .

or autonomously perform other desired tasks [12]–[15].

Assessing the performance of RL algorithms in robotic manipulation tasks involves utilising various evaluation metrics to measure the effectiveness and efficiency of the learning process. These metrics provide valuable insights into the algorithm’s performance and help researchers and developers optimise their approaches [16]–[18].

- **Success rate** is a fundamental evaluation metric that measures the percentage of successful task completions by the robot using the learned policy. In robotic manipulation tasks, the success rate indicates the algorithm’s ability to manipulate objects accurately and achieve desired goals. A high success rate demonstrates the effectiveness of the RL algorithm in learning suitable manipulation strategies.
- **Task completion time** is a critical metric that quantifies the time taken by the robot to successfully complete a manipulation task. A shorter task completion time signifies the competence of the learned policy in controlling the robot’s actions and accomplishing tasks swiftly. Minimising task comple-

tion time is crucial in real-world applications where speed and efficiency are paramount (e.g. search and rescue or surgical robots).

- **Total learning time** measures the duration required to train the RL algorithm to perform specific robotic manipulation tasks. Shorter training times indicate faster learning and shorter time to deployment of a learned policy in practical applications. Efficient learning algorithms with shorter training times can expedite the development and deployment of robotic manipulation systems. This metric is highly dependent on the hardware used, as it is naturally reduced by using more powerful processors for learning. For this reason, when comparing the total training time of two algorithms it is important to train both of them using the same hardware resources.
- **Sample efficiency** assesses how effectively the RL algorithm utilises available data samples to learn the manipulation task. Enhanced sample efficiency signifies the algorithm's ability to learn efficiently with fewer samples, making it more practical and resource-efficient. Improving sample efficiency is essential for reducing training time and the computational resources needed for training robotic manipulation systems.

By considering these metrics, researchers and developers can evaluate the performance of RL algorithms in robotic manipulation tasks and make informed decisions for further enhancements and optimisations. There are three main reasons why these 4 metrics are widely used:

1. **Practicality:** Success rate, completion time, and sample efficiency are straightforward to measure and interpret.
2. **Universality:** They apply across a wide range of robotic tasks, from manipulation to navigation.
3. **Benchmarking:** They allow for direct comparisons between methods and systems in the research literature.

Other metrics, instead, might be useful depending on the task that is being solved. For example, **accuracy** is particularly relevant in perception tasks such as object recognition [19], and it is often indirectly measured through other metrics like task success rate. Other metrics that might be useful depending on the context include **robustness** (often underrepresented because robustness testing across diverse settings is resource-intensive); or **stability** measures, which are usually not provided because it is implicit in tasks and may require specialized metrics for deeper analysis (e.g. Lyapunov stability criteria) [20].

This thesis focuses mainly on sample and time efficiency for the following reasons:

- **Cost of data collection:** In robotic manipulation, both real-world and simulated data collection are expensive and time-consuming. Therefore, **sample efficiency** learning effectively with fewer interactions is critical.
- **High dimensionality of robotic tasks:** robotic manipulation involves high-dimensional state and action spaces (e.g., multi-joint arms with force and torque feedback). This increases the computational burden of training RL policies, making **time efficiency** an important research objective.
- **Core limitations of reinforcement learning in robotics:** robotic manipulation tasks often require solving complex control problems, which take significant computation time to optimise. Addressing **time and sample efficiency** aligns with reducing this bottleneck.

1.2.1 Sample efficiency

In the field of robotic manipulation, sample efficiency plays a crucial role in enhancing the overall performance and effectiveness of robotic systems. Sample efficiency refers to the ability of a RL agent to learn a policy that achieves its objectives with minimal samples or interactions with the environment. It consists of optimising the learning process to make the most out of limited data, therefore reducing the time and resources required for learning a policy.

One of the key reasons why sample efficiency is of paramount importance in robotic manipulation is the complex and dynamic nature of real-world environments. Robots must be able to adapt and learn quickly in order to perform tasks effectively. By using sample-efficient learning methods, robots can more easily acquire the necessary skills and knowledge from a limited number of exploratory interactions, potentially making it feasible for them to operate in diverse and unpredictable environments.

Moreover, sample efficiency directly impacts the deployment and scalability of robotic systems. Typically, reinforcement learning approaches that rely on extensive data collection and training are time-consuming and resource-intensive. In contrast, a sample-efficient algorithm would enable robots to learn from a smaller number of samples, accelerating the learning process, which would make robotic manipulation more accessible and practical in academia and various industries, ranging from manufacturing and logistics to healthcare and agriculture.

In summary, the objective of finding sample-efficient RL methods is of utmost importance in the field of robotic manipulation. By making use of sample-efficient methods, robots can quickly acquire the necessary skills, adapt to dynamic environments, and reduce the time and resources required for system development. This will lead to increased rates of deployment of robotic manipulator systems in real-world applications. Sample efficiency will undoubtedly play a fundamental role in shaping the future of robotic manipulation research as research in this area continues to advance at pace,.

1.2.2 Learning time efficiency

Total training time also plays a crucial role in the development and deployment of robotic manipulation systems. The efficiency and effectiveness of learning algorithms directly impact the speed at which robots can learn and adapt to new tasks, ultimately influencing their overall manipulation performance. Minimising the total training time is essential for enhancing the practicality and scalability of robotic

manipulation solutions in various real-world scenarios.

Reducing the total learning time of robotic manipulation algorithms can lead to faster implementation of learned policies. In time-sensitive environments where quick decision-making and execution are critical, such as surgical robotics, shorter training times allow the engineer to adapt to challenges rapidly and handle tasks with precision. This agility in learning fast can significantly improve the productivity and responsiveness of robotic systems in dynamic environments where the robot needs to complete multiple tasks.

Similar to the need for sample-efficient algorithms, shorter total training times contribute to cost-effectiveness in the development and deployment of robotic manipulation solutions. For example, by requiring fewer computational resources and less time for training, organisations can streamline the production process, reduce operational costs, and therefore accelerate the integration of robotic systems into various industries.

In conclusion, the importance of total training time in robotic manipulation cannot be overstated. Researchers and developers can create agile and adaptive robotic systems capable of mastering complex manipulation tasks quickly and efficiently by optimising training algorithms to minimise training times. This kind of efficient training processes not only drive advancements in robotics, but will also pave the way for broader integration of robotics in diverse fields, revolutionising the way tasks will be automated and performed.

1.3 Preliminary glossary

This section provides an overview of key concepts needed to better understand the following sections of this chapter. Note that the definitions here are intentionally broad, with further details provided in later chapters.

- **Environment:** In robotic manipulation and reinforcement learning, the environment is the system or space where an agent interacts to perform tasks,

encompassing the physical or simulated world, objects, sensors, and dynamics that define the outcomes of the agent's actions.

- **Action:** an operation or movement performed by a robot at a given time step, defined within the context of its control system. Actions could include joint movements, gripper commands, or changes in torque/force.
- **Reward Function:** a mathematical representation of the task goal in reinforcement learning, mapping the current state and action to a numerical value that indicates how desirable the outcome is. The science that designs this kind of function is typically called **reward engineering**.
- **Dense Reward:** a reward signal that provides feedback at every time step, offering detailed guidance to the learning agent on how well it is performing.
- **Sparse Reward:** a reward signal that provides feedback only at specific milestones or upon task completion, often requiring the agent to explore extensively to discover successful strategies.
- **State:** a representation of the robot's and the environment's current conditions, including positions, velocities, object poses, and sensor readings.
- **Policy:** a mapping from states to actions, defining the behaviour of the agent in reinforcement learning. Policies can be deterministic (fixed action for a state) or stochastic (probabilistic).
- **Trajectory:** a sequence of states and actions taken by the robot during the execution of a task, used to evaluate and train reinforcement learning algorithms.
- **Primitive Skill:** a low-level control policy or behaviour that performs a simple, foundational task (e.g., grasping, pushing, or rotating). These skills are often combined to achieve more complex goals.

- **Sequential Execution:** the process of executing multiple actions or primitive skills in a defined sequence to accomplish a complex task. Following a hierarchy can be sample efficient, especially in complex tasks that require extensive exploration by the robot.
- **TriFinger Robot:** a robotic system designed for object manipulation tasks, consisting of three multi-jointed fingers with tactile sensing capabilities. Commonly used in research for its modularity and dexterity [21].
- **Tactile Sensor:** a sensor that provides feedback on physical contact, including pressure, force, and surface texture, crucial for tasks involving precise manipulation or interaction with objects.
- **Object Manipulation Strategy:** a structured approach or policy for controlling a robot to manipulate objects effectively, typically involving planning, control, and learning components.
- **In-Hand Rotations:** a manipulation task in which an object is rotated or reoriented within the grasp of a robotic hand or fingers. This task requires precise control and coordination of forces and often involves tactile feedback for stabilization.
- **Off-Policy Reinforcement Learning:** a reinforcement learning paradigm where the policy being optimised (target policy) is different from the policy used to generate the data (behaviour policy).
- **On-Policy Reinforcement Learning:** a reinforcement learning paradigm where the policy being optimised is the same as the policy used to generate the data.
- **OpenAI Gym Environment:** a standardised toolkit for developing and comparing reinforcement learning algorithms. It provides pre-defined environments, including physics simulations, to test learning methods [22].

- **Exploration vs. Exploitation:** the trade-off in reinforcement learning between exploring new actions to discover their outcomes (exploration) and using known actions to maximise rewards (exploitation).

1.4 Analytical vs Data-driven manipulation

Data-driven control techniques such as RL-based techniques present several advantages over traditional analytical robotic manipulation techniques, which are based on engineered mathematical models, making data-driven control a more effective technology for training robots to perform tasks autonomously.

Firstly, RL is very adaptable to complex and dynamic environments [23]. Traditional robotic manipulation techniques often rely on pre-programmed algorithms that struggle to handle variations in the environment. In contrast, RL allows robots to learn from experience, enabling them to effectively adapt to new and changing conditions.

This in turn means that this learning paradigm promotes continuous improvement through iterative learning [24]. In contrast to traditional analytical techniques that require manual tuning and adjustments [25], RL algorithms can continuously update their strategies based on feedback received from the environment. This iterative learning process enables robots to enhance their performance over time without human intervention.

Lastly, RL enables robots to explore and discover optimal solutions independently [26], [27]. Traditional robotic manipulation techniques are limited by the knowledge and experience of their programmers, often leading to sub-optimal solutions. In contrast, a data-driven approach such as RL empowers robots to explore the consequences of different actions and strategies, discovering novel approaches that may outperform traditional techniques without relying on human intervention in the learning process.

Nonetheless, analytical methods in robotic manipulation offer several advantages compared to purely data-driven techniques, particularly in scenarios where precise

	Analytical control	Data-driven control
Exploration required	Minimal	Extensive
Success rate	Limited	Potentially excellent
Control algorithm	Hand-engineered	Learned
Learning time	Fast	Usually slow
Adaptability	Limited/low	High
Interpretability	High	Low

Table 1.1: Comparison of advantages and disadvantages of analytical and data-driven methods. Analytical control methods do not require powerful hardware as there is no learning and are easy to interpret, however, data-driven methods are generally more adaptable and can be more accurate because they use a learned control strategy.

control and understanding of the underlying physical principles are paramount. One key advantage of analytical methods is their ability to provide deterministic solutions based on mathematical models and an accurate description of the physical principles governing the environment, allowing for robust and predictable behaviour in manipulation tasks [28], [29]. Using analytical models, engineers can design control strategies that ensure precise manipulation of objects without the need for extensive amounts of training data, making these methods well-suited for tasks requiring high accuracy and repeatability.

Another advantage of analytical methods in robotic manipulation is their interpretability and transparency [30], [31]. Analytical models provide clear insights into the decision-making process of the robot, enabling developers to understand and debug the underlying algorithms effectively. This transparency is essential for ensuring the safety and reliability of robotic systems, as developers can trace the behaviour of the robot back to the fundamental principles and make adjustments as needed. In safety-critical applications, such as surgical robotics or manufacturing, the interpretability of analytical methods enhances confidence in the system's performance and facilitates troubleshooting in case of errors [32].

In conclusion, both data-driven and analytical methods have advantages and disadvantages, and the choice of method depends on the application that is being tackled and the availability of hardware with sufficient computational power. Table 1.1 summarises the advantages and disadvantages of the two approaches, and

shows how despite the huge benefits of data-driven approaches, the most important being the great performance and adaptability of data-driven methods, these techniques still have drawbacks related to the lack of interpretability of the models learned and the hugely resource-intensive nature of those methods due the immense data collection required to train such methods. In this thesis, the main focus is on addressing the latter problem by developing more sample and time-efficient RL methods that are capable of solving complex robotic manipulation tasks by making the most of a small number of interactions with the environment.

1.5 Hypotheses and research questions

In this thesis, two main problems associated with the use of RL algorithms to solve robotic manipulation tasks are addressed: sample and training time efficiency. To this end, a number of hypotheses are proposed and investigated in order to guide the development of this thesis towards exploring methodologies that improve the efficiency of current robotic manipulators.

The following research questions have guided the work documented in this thesis and shaped the exploration of the proposed hypotheses:

- (i) **Research Question 1:** How important is reward engineering in obtaining sample and time efficiency when training a RL-based algorithm in lifting an object?
- (ii) **Research Question 2:** How can we make use of basic primitive skills in a sequential manner to solve more complex tasks? In which scenarios would this approach be most useful?
- (iii) **Research Question 3:** How can we reuse primitive skills that are easy to learn in order to accelerate the learning of current state-of-the-art (SOTA) methods in robotic manipulation? Are there strategies that accelerate current SOTA without adding computational complexity?

Answering these research questions allows us to comprehensively explore the following hypotheses:

- (i) **Hypothesis 1:** By carefully designing a reward that encourages a particular desired behaviour, standard RL algorithms can learn more efficiently to solve robotic manipulation tasks that involve the lifting of an object, leading to more sample-efficient methods and, thus, to lower training times.
- (ii) **Hypothesis 2:** A major reason why RL sometimes struggles with sample and time efficiency is because robotic manipulation tasks are too complex to learn from scratch (we refer as learning from scratch to initialising all model parameters randomly and then optimising them using a chosen optimisation algorithm). Instead, we hypothesise that by dividing complex tasks into smaller simpler tasks, that can be learned quickly and then be executed sequentially, the original complex manipulation task can be effectively solved using few interactions with the environment. If this hypothesis is supported, this approach will result in more modular learning, which not only ensures a more sample and time efficiency method, but also a more interpretable algorithm.
- (iii) **Hypothesis 3:** The use of fixed set of basic primitive policies limits the sample efficiency and the performance of robotic manipulation to the expertise of these primitive behaviours. However, we hypothesise that if an algorithm can decide during exploration between when to use one of the existing primitive policies and when to learn a new skill, robots will become more adaptable to dynamic environments and will learn to perform more complex tasks more quickly.

1.6 Summary of thesis contributions

Throughout this thesis, a number of contributions to the field are presented. These are summarised in the following list:

- (i) **Chapter 3:**

- (a) **A novel reward system is proposed that combines sparse and dense components in order to encourage a lifting behaviour for a robotic manipulator.** This algorithm can be used with any off-policy RL technique and **sets a new state of the art** on the ‘*Move Cube on Trajectory*’ task of the TriFinger robot.

(ii) **Chapter 5:**

- (a) **A novel learning framework for in-hand object 3D rotations based on sequential execution of primitives learned by any off-policy RL method is proposed.** This framework operates using the Davenport chained rotations decomposition, a method widely used in the field of aeronautics, and it is more sample efficient than end-to-end learning methods when limiting the number of interactions with the environment to reduce training time.

(iii) **Chapter 6:**

- (a) **A novel algorithm called Q-Switch Mixture of Primitives Hind-sight Experience Replay (QMP-HER) that improves the sample efficiency in complex manipulation tasks by making use of primitive behaviours trained in simpler environments is presented.** This algorithm can work in conjunction with any off-policy RL algorithm.

1.7 Thesis structure

The remainder of the thesis has the following structure:

- **Chapter 2** provides a comprehensive overview of the main background concepts and principles used in the rest of the thesis. It gives an overview of the context of the thesis (i.e., robotic manipulation), and then explains the fundamental concepts of RL, as well as a justification of some of the decisions

made for the development of the thesis. Furthermore, a state-of-the-art review is provided, where some of the most popular sample efficient methods are explained, highlighting their main limitations. This is done while making a connection between the hypotheses raised in section 1.5 and the limitations highlighted throughout the chapter.

- **Chapter 3** introduces a novel reward system that intrinsically motivates the robot to reproduce a lifting behaviour. This is done in the context of the Real Robot Challenge (RRC) [33] competition, which uses a TriFinger robot [21]. It provides a detailed explanation of our winning algorithm, as well as other applications in which our method has been used or could be used in the future.

This chapter addresses Research Question 1.

- **Chapter 4** is a connecting chapter that does not provide any novel algorithmic design contribution to the field nor try to answer any of the research questions formulated in Section 1.5. However, it is important for understanding the contributions of the thesis as it provides a justification for the use of the OpenAI Gym environment [22] throughout the rest of the thesis, as well as providing a new benchmark study in terms of sample and time efficiency for different off-policy RL algorithms using that environment, which is used as baseline for comparison against the performance of algorithms proposed later in the thesis.

- **Chapter 5** presents a sample-efficient approach based on a sequential execution of RL-based primitive behaviours. The results obtained for the algorithm are analysed in terms of performance, sample and time efficiency. **This chapter addresses Research Question 2.**

- **Chapter 6** proposes a new algorithm that improves sample efficiency in complex manipulation tasks by automatically using primitive behaviours only when their use is optimal. It provides a deep analysis of the results obtained, including an analysis of how the primitives used affect the learning and how

the algorithm uses these primitives. **This chapter addresses Research Question 3.**

- **Chapter 7** presents our conclusions, identifies potential future areas of research, summarises the research presented in this thesis, and re-visits the hypotheses and research questions addressed in this thesis.

Chapter 2

Background

2.1 Chapter overview

This chapter provides a comprehensive overview of the main background concepts and principles used in the following chapters. Note that the selected content is high-level in nature and more extensive explanations of relevant related work can also be found in the later chapters. Section 2.2 introduces the essentials for understanding robotic manipulation tasks, including types of robotic manipulators and a description of the manipulator’s main parts. Section 2.3 provides an introduction to some basic RL concepts and Section 2.4 introduces various algorithms along with some key concepts important for understanding the research reported in this thesis. Furthermore, this section also provides a justification for the selection of an off-policy algorithm for our research. Then, a state of the art review is provided in Section 2.5, explaining the best methods that improve performance and sample efficiency in robotic manipulation tasks whilst highlighting their main limitations. Finally, conclusions are presented in Section 2.6, where a connection is made between the limitations of the state of the art and the research questions and hypotheses raised in Chapter 1.

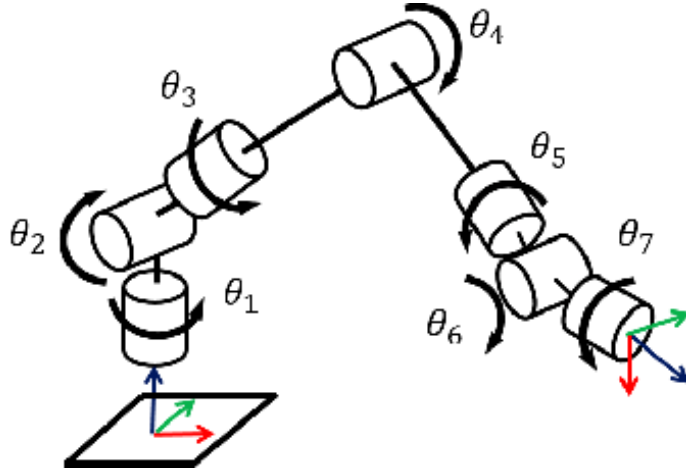


Figure 2.1: Schematic representation of a 7-DoF robotic arm. Image taken from Luo *et al.* [34]

2.2 Introduction to robotic manipulation

A robotic manipulator is typically an actuated mechanical linkage consisting of rigid links (although not always), often implemented as a six-axis arm, which is designed for manipulating and handling objects in various applications, such as manufacturing, assembly, and logistics. These robotic systems consist of linked segments and joints that work together to perform precise and controlled movements for tasks like picking, lifting, placing, and positioning objects with accuracy and repeatability.

Articulated robotic arms are amongst the most popular robotic manipulators. This type of manipulator consists of multiple joints (also called axes) that allow for rotational movements, mimicking the flexibility and dexterity of a human arm making them suitable for tasks that require reaching objects in different locations and orientations [34].

Usually, these kinds of robotic manipulators are differentiated by the number of degrees of freedom (DoF) they possess: the more degrees of freedom a robotic manipulator has, the more dexterous, versatile, and adaptable it can be, but it is also more difficult to effectively control. In nearly all cases, the number of degrees of freedom is equal to the number of joints [35]. Figure 2.1 shows an example of a schematic representation of a 7-DoF robotic arm.

An end-effector is an accessory device that is attached at the end of the kinematic chain, and it is the component that actually interacts with the environment and performs object manipulation. The simplest end-effector is the parallel-jaw gripper, which is capable of opening and closing to grip objects, but the end-effector can have many different designs; from grippers of multiple fingers, to actual mechanical tools such as a screwdriver or a brush, to anthropomorphic hands. The different joints of the robot, the base of the robot, and the end-effector are connected by what are called links or bars.

Traditional approaches for manipulation of objects using robotic arms involve mathematical modelling and analysis of the kinematics and dynamics of robotic manipulators to derive closed-form solutions for control and motion planning tasks [36]–[38]. These approaches usually rely on precise geometric and algebraic representations of the robot’s structure and behaviour to determine the relationship between joint space and task space variables, enabling efficient computation of trajectory planning, inverse kinematics, and dynamic control strategies. However, implementing these systems to achieve dexterity and adaptability is not easy (i.e. deciding the trajectories or destinations to achieve successful manipulation behaviours), and researchers and engineers usually need to invest a lot of time in the design of the control algorithm.

A popular way to tackle robotic manipulation tasks by traditional, analytical methods is by sub-dividing tasks or skills into simpler primitive behaviours that, when sequentially executed, reproduce the original task or skill [39]–[44]. These approaches are typically known as Behaviour Trees (BT), a mechanism that has gained a lot of popularity due to its effective representation of complex robotic behaviours and decision-making processes in a hierarchical and modular structure [45].

BTs organise robotic behaviours into a hierarchical structure consisting of nodes that represent specific actions, conditions, or control logic. This modular design allows for easy integration, modification, and even reuse of individual behavioural components, enabling robust and scalable control architectures for robotic manip-

ulators. For example, with BTs it is possible to gradually add new behaviours, tasks, or functionalities into a manipulation control framework without disrupting the existing control architecture.

In Chapter 1, the following hypothesis was stated:

- **Hypothesis 2:** *A major reason why RL sometimes struggle with sample and time efficiency is because robotic manipulation tasks are too complex to learn from scratch efficiently. Instead, by dividing complex tasks into smaller simpler tasks that are easy and fast to learn, then the resulting policies of these sub-tasks can be sequentially executed managing to solve the original more complex task. This can result in a modular learning, which could not only give a more sample and time efficiency method, but also a more interpretable algorithm.*

Essentially, here we are hypothesising that the combined use of RL algorithms and BTs that follow a hierarchy can be more sample efficient than a fully end-to-end learning approach, especially in complex tasks that require extensive exploration by the robot.

2.3 Reinforcement learning fundamentals

RL is an area of AI in which control models are trained by rewarding or punishing $(r_0, r_1, \dots)$ an agent that is taking actions $(a_0, a_1, \dots)$ in a certain environment [46]. The goal of this learning paradigm is to maximise a long-term reward so that the agent learns how to interact appropriately with the environment. Unlike supervised learning, where ground truth answers are needed to train a model and there is no interaction with any environment, in RL, the model learns by trial and error, and the inputs that the control model receives are directly affected by the actions taken by the agent.

This learning approach can also be understood as a Markov Decision Process (MDP) [47]. Representing the RL paradigm as a MDP is useful because MDPs provide a formal framework for modelling decision-making processes in dynamic and

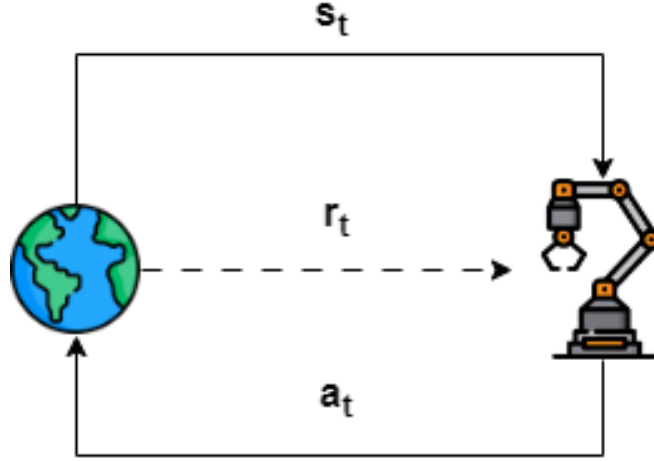


Figure 2.2: High-level framework of the reinforcement learning paradigm. At each time-step t , the agent receives an observation of states s_t from the environment, takes an action a_t , and receives a reward r_t .

uncertain environments, making them well-suited for understanding and solving RL-based problems. Furthermore, by assuming the Markov property (i.e., actions and values are assumed to be a function only of the current state), MDPs ensure that the decision-making process is memory-less and that the agent can learn an optimal policy without needing to consider the entire history of states and actions.

This framework is represented in Figure 2.2. The set of states of the MDP is denoted as $\mathcal{S}$, while the set of actions and rewards are denoted as $\mathcal{A}$ and $\mathcal{R}$, respectively. Episodes are defined as sequences of interactions between an agent and the environment, each sequence starting with the agent in a random state from $\mathcal{S}$, and eventually reaching a terminal state after taking a series of actions. For each time-step in each episode, the agent performs an action based on the current state, and receives a reward, following a trajectory $\mathcal{T} = \{(s_1, a_1, r_1), (s_2, a_2, r_2), \dots, (s_T, a_T, r_T)\}$. The dynamics of the MDP are represented by the probability of transition from one state to another, $p(s_t|a_{t-1}, s_{t-1})$, while the agent policy is the probability of the agent taking an action given a particular state, $\pi(a|s)$. The total cumulative reward, R_T , corresponds to the addition of all rewards $r(s_t)$ along a particular episode, while the discounted return is a way to calculate the total reward that an agent can expect to receive over a sequence of actions, taking into account the time value of rewards

$V_\pi(s) = r_1 + \gamma^1 r_2 + \dots + \gamma^{T-1} r_T$, with $0 < \gamma \leq 1$. By discounting future rewards, the agent can prioritise short-term gains while still considering long-term consequences. This term is usually known in RL as the state-value function $V_\pi(s)$ and the agent's goal is to find the policy $\pi = \pi^*(s)$ that maximises $V_\pi(s)$ [48]:

$$\Pi_T : \pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right] \quad (2.1)$$

In robotic manipulation, however, tasks are usually defined by goals. In goal-based RL, the MDP framework is the same, except now there is a goal space, denoted as $\mathcal{G}$, and the rewards $r(s_t, g)$ not only depend on the state but also on a goal that is sampled from $\mathcal{G}$ each episode. A goal is the objective that the agent must complete in a particular episode, and it usually changes between episodes. This means that the state-value function, $V_\pi(s, g)$, is also dependent on the goal for the episode, and therefore the agent's goal is to find the policy $\pi = \pi^*(s, g)$ that maximises $V_\pi(s, g)$:

$$\Pi_T : \pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, g) \right] \quad (2.2)$$

2.3.1 The action-value function

As seen in the previous section, the value function in RL evaluates the expected cumulative reward an agent can achieve by following a specific policy in a given state (or state-action pair, as will be seen in this section). The value function is a key component in an RL algorithm, as it guides the agent's decision-making process by estimating the long-term desirability of different states or actions based on the rewards they are expected to produce.

There are two primary types of value functions used in RL [49]:

1. **The state-value function**, denoted as $V_\pi(s)$, estimates the expected cumulative reward that an agent can achieve from a specific state s while following a given policy $\pi(s)$. It represents the desirability of being in a particular state and serves as a measure of how appropriate it is for the agent to be in that

state under the current policy. The state-value function helps the agent assess which states are more favourable or valuable in terms of obtaining long-term rewards.

2. The **action-value function**, denoted as $Q_\pi(s, a)$, estimates the expected cumulative reward that an agent can achieve by taking a specific action a in a particular state s and then following a given policy $\pi(s)$. The action-value function is often considered to give a more detailed assessment than the state-value function of the desirability of choosing a particular action in a specific state, as it provides information about the value of taking a specific action in a given state, rather than just the value of being in that state; and thus, it helps the agent select the best action to maximise long-term rewards. An intuitive way to understand this function is as a metric that quantifies the quality of an action a for a given state s .

The action-value function, $Q_\pi(s, a)$, is also known as the Q-function, and it is represented by the Bellman equation [50]:

$$Q_\pi(s, a) = \mathbb{E}_\pi [r(s_t, a) + \gamma \max(Q_\pi(s_{t+1}, a'))] \quad (2.3)$$

where $Q_\pi(s, a)$ is the action-value function, $r(s_t, a)$ is the reward received after taking action a in state s_t , γ is the discount factor that determines the importance of future rewards, and $\max(Q_\pi(s_{t+1}, a'))$ represents the maximum expected cumulative reward for the next state s_{t+1} and all possible actions a' that can be taken from s_{t+1} .

2.3.2 Rewards

As can be seen in Equation 2.3, the action-value function directly depends on the reward definition given for the problem that is being solved. In other words, an inappropriate reward definition would result in a Q-function that is unable to capture the essence of the task, and thus would not be able to accurately quantify how good an action is for a given state. There are two main reward systems that are commonly

used in RL:

1. **Dense rewards** provide feedback to the agent at each time step, offering continuous feedback based on the agent's actions. These kind of rewards are useful for providing immediate feedback and helping the agent learn more quickly by receiving feedback frequently throughout the learning process and measure how close the agent is to solving the task. Typical dense rewards used in robotic manipulation include the distance between the robot's end-effector and the goal position or object [51], differences between end-effector or object poses [52], dense metrics that capture the stability of the robot's grasp [53], and more.
2. **Sparse rewards** are less frequent compared to dense rewards and are only provided when the agent accomplishes specific goals or achieves certain milestones in the environment. Sparse rewards can make learning more challenging for the agent as it must learn to associate these occasional rewards with the correct actions or states, but they are particularly effective in challenging environments where immediate feedback may be scarce or difficult to accurately define, as is the case in many robotic manipulation tasks. The use of sparse rewards has become very popular since the emergence of Hindsight Experience Replay (HER) [54], a method that overcomes many of the challenges associated with dealing with sparse rewards that have been mentioned in this section; to learn more about this algorithm, see Section 2.5.1. These rewards are usually defined as:

$$r_t = \begin{cases} 0 & \text{if task is completed} \\ -1 & \text{otherwise} \end{cases} \quad (2.4)$$

where r_t is the reward given to the agent at time-step t .

Rewards can also be categorised according to whether they are explicitly obtained from the environment or not:

1. **Extrinsic rewards** are directly provided by the environment based on the result of the agent's actions (e.g. points, money, etc). These rewards serve as the primary source of feedback for the agent and drive its learning process towards achieving the desired goals [55], [56].
2. **Intrinsic rewards** are generated by the agents themselves based on their internal state and their own model of the environment. They can also act as auxiliary rewards that are not directly related to the task but can help the agent explore the environment, learn new behaviours, provide security, or in general overcome challenges more effectively. Intrinsic rewards supplement the extrinsic rewards and can enhance the agent's learning, for example by promoting exploration and curiosity [57].

In Chapter 1, one of the hypotheses is related to intrinsic rewards. In particular, Hypothesis 1 states:

- **Hypothesis 1:** *By carefully designing a reward that encourages a particular desired behaviour, standard RL algorithms can learn more efficiently to solve robotic manipulation tasks that involve the lifting of an object, leading to more sample-efficient methods and, thus, to lower training times.*

In other words, our objective is to design a new reward function that improves the sample and time efficiency of state of-the-art RL algorithms by intrinsically motivating the agent to execute as soon as possible the most universal behaviour in the context of a robotic manipulation task: the lifting behaviour (all robotic manipulation tasks require the robot to lift or sustain the manipulated object).

2.3.3 On-policy vs off-policy reinforcement learning

Distinguishing between on-policy (e.g., Policy Gradient methods [58] or Trust Region Policy Optimisation [59]) and off-policy RL (e.g., Q-learning [60] or Deep Q-Networks [61]) is crucial for understanding how agents learn and improve their

decision-making processes in MDP-based environments [62], [63]. The primary differences between on-policy and off-policy RL are listed as follows:

Learning framework

- **On-policy:** In on-policy RL, the agent evaluates and updates its policy while directly interacting with the environment based on the current policy. The agent learns from its own experiences under the current policy, requiring it to balance exploration and exploitation within the same policy framework.
- **Off-policy:** In off-policy RL, the agent learns the value of different actions or policies based on data generated by following a *behaviour policy* that can be different from the current policy (in this context usually called *target policy*) that is being evaluated and improved (see Figure 2.3). This means that off-policy methods can separate exploration from exploitation by following a different policy for exploration and updating a target policy independently. This decoupling enables more flexibility in exploring different strategies.

Policy adaptability and stability

- **On-Policy:** On-policy methods are more prone to policy instability because the agent learns and updates its policy based on its ongoing interactions with the environment. Therefore, a policy learned using on-policy RL can exhibit abrupt error variability between episodes, making these algorithms perform poorly in environments with complex goal and action spaces. On the other hand, the ability of on-policy algorithms to update the policy based on real-time interactions allows the agent to quickly adapt to changes and uncertainties in the environment, making them suitable for environments where the characteristics and dynamics of the environment change frequently or continuously.
- **Off-Policy:** Off-policy learning exhibits more stability as it considers more

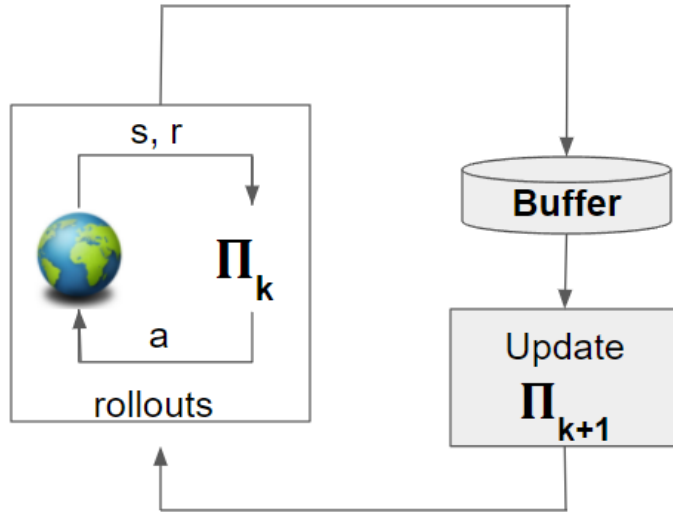


Figure 2.3: High-level representation of the off-policy reinforcement learning training framework. In order to have a more stable training, several episodes are run using the *behaviour policy* π_k , and then the *target policy* π_{k+1} is updated (i.e. the policy that is being evaluated and improved), using the interactions saved in the buffer. In this figure the *behaviour policy* is the same as the *target policy*, but this does not necessarily have to be the case.

interactions with the environment before each policy update. Contrary to on-policy learning, with off-policy RL a considerable number of rollouts (i.e. interactions with the environment) using the same behaviour policy are generated and saved in a buffer, and then used to update the policy, making the learning more stable in stationary environments where the task is fixed. This is a similar idea to the use of batches of data in deep learning as a trade-off between the accuracy of the model and faster training times [64]. For this reason, off-policy RL is sometimes incorrectly referred to as batch-RL (or offline RL), while in fact they are different learning frameworks [65]. The difference between off-policy RL and batch-RL is that the latter involves training an RL agent using a fixed dataset of historical data collected from interactions with the environment, while off-policy RL keeps updating the buffer by interacting with the environment while the algorithm learns [66].

Sample efficiency

- **On-policy:** On-policy methods are known to be relatively less sample-efficient overall in environments with complex action and goal spaces, particularly those environments that have continuous action spaces, as a consequence of relying on a single policy for both exploration and exploitation. However, for environments with discrete action spaces, on-policy algorithms can be beneficial in terms of sample efficiency, as these tasks are simpler to control.
- **Off-policy:** Off-policy methods can potentially be more sample efficient by reusing past experiences, in particular in fields such as robotic manipulation, which often involves high-dimensional continuous state and action spaces, making the learning process resource-intensive. Off-policy algorithms' use of past data leads to faster learning and improved performance with fewer interactions with the environment in this context.

Overall, the advantages of sample efficiency, exploration flexibility, policy stability, and data reuse make off-policy RL algorithms more suited than their on-policy RL counterparts for addressing the complexities and requirements of robotic manipulation tasks.

2.3.4 Exploration-exploitation trade-off

The exploration-exploitation trade-off is a fundamental concept in RL that refers to the balance between exploring unknown actions or states to discover potentially better strategies (*exploration*) and exploiting known information to maximise immediate rewards (*exploitation*). The trade-off relates to the fact that there is risk in exploring new actions or states rather than opting for highly likely short-term gains.

On the one hand, exploration involves taking novel actions or visiting unexplored states to gather information about the environment and potential rewards. This is essential for discovering new strategies, learning the dynamics of the environment, and avoiding premature convergence to sub-optimal solutions. Agents explore by

trying out different actions, and seeking long-term benefits even if this strategy incurs short-term costs.

Alternatively, exploitation involves leveraging known information or past experiences to maximise immediate rewards and making decisions based on the current best value-estimates. Agents exploit this by choosing actions that have historically resulted in high rewards or outcomes, aiming to exploit the best-known strategies to maximise short-term gains.

Finding the right trade-off between exploration and exploitation is vital for reinforcement learning agents to achieve a balance between discovering new knowledge and effectively exploiting existing information. Agents need to sufficiently explore to learn about the environment and avoid getting stuck in sub-optimal solutions while still exploiting learned strategies to maximise rewards and performance.

There are many different strategies that perform this trade-off [67]–[69]. In the context of off-policy RL for robotic manipulation, one of the most popular techniques is called the epsilon-greedy algorithm (or ϵ -greedy for short) [70]. It is a simple and intuitive method that involves choosing between exploration and exploitation based on a selected probability value (ϵ). In particular, the epsilon-greedy strategy works as follows:

- **Exploration** (with probability $p = \epsilon$): the agent samples and selects a random action with a probability of ϵ , from the action space $\mathcal{A}$, enabling exploration of unseen actions or states to gather information about the environment and potential rewards.
- **Exploitation** (with probability $p = 1 - \epsilon$): With a probability of $1 - \epsilon$, the agent selects the action that is believed at that moment to be the best, based on the learned knowledge and past experiences (i.e., the action that maximises the state-value $V_\pi(s)$ or action-value $Q_\pi(s, a)$ function), aiming to maximise immediate rewards through exploitation.

The versatility, simplicity, and effectiveness in managing the exploration-exploitation trade-off of the epsilon-greedy algorithm make it a popular choice

Algorithm 1 Q-learning algorithm

Given: Q-table $\mathbb{Q}$

```
1: Initialise  $\mathbb{Q}$ 
2: for episode =1 to  $M$  do
3:   Sample an initial state  $s_0$ 
4:   for  $t=0$  to  $T-1$  do
5:     Sample action  $a_t$  that maximises  $Q_\pi(s_t, a_t)$  using  $\epsilon$ -greedy.
6:     Execute action  $a_t$  and observe state  $s_{t+1}$ 
7:     Update  $\mathbb{Q}$  using Equation 2.5
8:   end for
9: end for
```

amongst researchers and developers for off-policy RL techniques in the context of robotic manipulation applications [71]–[76]. For this reason, this algorithm is used throughout the development of the research reported in this thesis.

2.4 Off-policy reinforcement learning algorithms

This section aims to introduce different off-policy RL algorithms to provide a general understanding of the most popular methods. Different algorithms are presented across Section 2.4.1 to Section 2.4.5. However, in the remainder of this thesis, only one of these algorithms is used: the Deep Deterministic Policy Gradients (DDPG) algorithm [77]. A justification for this decision is provided in Section 2.4.6.

2.4.1 Q-learning

Q-learning is a popular off-policy RL algorithm used to train an agent to make optimal decisions in an MDP-based environment [60]. When training an agent using Q-learning, a Q-table of state-action pairs must be preset by initial exploration, along with the state-action pair Q-value. The Q-value is a metric that captures the quality of the transition, which is represented by a state s_t , an action a_t , the future state s_{t+1} reached by taking the action, and a reward r_t . In other words, it defines how good or bad an action a_t is that is taken for a particular state s_t , when following a policy $\pi(s, g)$.

In particular, the Q-value for a state is the expected discounted cumulative

reward given the state and the action, which during exploration is chosen either randomly or based on the maximum Q-value. During training, as the environment is further explored, the Q-table is updated re-calculating these Q-values, which are updated based on the observed rewards using the solution of the Bellman equation [78]:

$$Q_{new}^{\pi} = Q(s_t, a) + \alpha \left[r(s_t, a) + \gamma \max_{a'} (Q(s_{t+1}, a')) - Q(s_t, a) \right], \quad (2.5)$$

where $Q(s_t, a)$ denotes for the current Q-value, α is the learning rate, $r(s_t, a)$ is the reward for taking action a when in state s_t , γ is the discount factor, determining the importance of future rewards, and $\max(Q(s_{t+1}, a'))$ is the maximum expected reward in the new state given all possible actions a' that can be taken in the next state s_{t+1} .

The Q-learning algorithm is summarised in Algorithm 1. Q-learning is a popular algorithm in robotic manipulation because it is a model-free RL algorithm that can learn optimal policies simply via trial and error, which makes it well-suited for tasks where the environment is unknown or complex, as is the case in robotic manipulation tasks [79]–[82].

2.4.2 Deep Q-Networks

Despite the popularity of Q-learning, it is evident that this learning framework is clearly limited by its sample inefficiency, as it requires very extensive exploration in environments where either the goal space, the action space, and/or the observation space are complex. Furthermore, the impracticality of representing the Q-function as a table containing the Q-values of each combination of state-action-goal not only harms the ability of the agent to learn an appropriate policy with few interactions with the environment, but also limits its applicability to the real world, as in most cases it is infeasible to allow such levels of exploration because the real-world consists of continuous states that cannot be accurately represented in a table.

Algorithm 2 Deep Q-learning with Experience Replay

Given: Q-table $\mathbb{Q}$

Replay buffer $\mathcal{R}$

```

1: Initialise  $\mathbb{Q}$ 
2: for episode = 1 to  $M$  do
3:   Sample an initial state  $s_0$  and a goal  $g$ 
4:   for  $t = 0$  to  $T - 1$  do
5:     Sample action  $a$  that maximises  $Q_\pi(s_t, a)$  using  $\epsilon$ -greedy
6:     Execute action  $a$  and observe state  $s_{t+1}$ 
7:     Store transition  $(a, s_t, s_{t+1}, g, r_t)$  in replay buffer  $\mathcal{R}$ 
8:     Sample random mini-batch of transitions from  $\mathcal{R}$ 
9:     Using the sampled mini-batch, update network parameters using the loss
        function defined in Equation 2.6
10:   end for
11: end for

```

Instead, Mnih *et al.* proposed the Deep Q-Networks (DQN), introducing neural networks in order to learn an approximation for these Q-values [61]. In their work, they not only proposed the use of neural networks to address these issues associated with the use of the Q-learning algorithm, but they also proposed a technique called Experience Replay (ER), which makes use of a buffer of collected samples to make the network updates more stable by using uncorrelated transitions in a batch update, as has been previously shown in Figure 2.3 and outlined in Algorithm 2. The introduction of ER is also very significant in terms of sample efficiency, as the algorithm now permits reusing each transition in different updates.

To approximate the Q-values, the DQN makes use of the following loss function:

$$\mathcal{L}(\theta_i) = \mathbb{E}[(y_i - Q_\pi(s_t, a; \theta_i)] \quad (2.6)$$

where θ_i represent the parameters of the neural network at the iteration step i , y_i represents the temporal difference (TD) target defined as $y_i = R(s_t, a) + \gamma \max_{a'}(Q(s_{t+1}, a'; \theta_i))$, and the difference $y_i - Q_\pi(s_t, a; \theta_i)$ is called the TD error [83].

The architecture of DQNs is very simple: a simple Convolutional Neural Network (CNN) [84] with three hidden layers, the output of the network being a dense layer with as many neurons as actions in the action space $\mathcal{A}$, representing the Q-value associated to each of these actions given the state, which is given to the neural

network as the input (together with the goal for goal-based environments). This makes DQNs well-suited for agents with discrete action spaces, but limits the use of this algorithm for environments where the agent utilises a continuous action space, as is usually the case in robotic manipulation tasks.

In particular, DQNs have been shown to be especially effective in learning complex, non-linear relationships in high-dimensional input spaces, which can be beneficial for robotic manipulation tasks that require processing of sensory data from RGB cameras or other sensors [85]–[87].

2.4.3 Soft Actor-Critic

The Soft Actor-Critic (SAC) algorithm is a state-of-the-art off-policy RL algorithm that combines the advantages of actor-critic methods with entropy regularisation to achieve robust and efficient policy learning [88]. SAC is designed to address challenges such as exploration-exploitation trade-offs, sample efficiency, and learning stability, making it well-suited for a variety of tasks, including those with continuous action spaces and complex environments, such as robotic manipulation tasks [89].

Actor-critic methods use a framework that combines the strengths of two neural network components: the actor and the critic [90]. This architecture is designed to address the challenges of both policy optimisation and value estimation in the context of RL tasks. These two networks have the following roles:

- The **actor** network is responsible for learning the policy, which determines the agent’s actions based on the observed state and episode goal. This network outputs a probability distribution over the actions, allowing the agent to select actions that maximise expected rewards. It is also usually used to explore the action space and update the policy based on the feedback received from the environment.
- The **critic** component is responsible for estimating the value $Q_\pi(s, a)$ associated with taking a specific action in a given state. This network provides

feedback on the goodness of actions (the higher the estimated reward for taking an action is, the better that action is) taken by the actor, helping the agent evaluate its decision-making and learn from outcomes of its actions. By approximating the value function, the critic aids in assessing the quality of the agent’s policy and guiding policy updates.

This actor-critic architecture combines the benefits of policy-based (actor) and value-based (critic) methods, offering a more stable and efficient approach to reinforcement learning. By leveraging the actor to explore the action space and the critic to evaluate the quality of the actions, this framework enables the agent to learn more robust policies that balance exploration and exploitation effectively.

In the SAC algorithm, the objective is to find the optimal policy that maximises the expected long-term reward and long-term entropy. In other words, the agent tries to simultaneously succeed at the task at hand while also acting as randomly as possible:

$$J(\pi) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a) + \alpha H(\pi(s)) \right] \quad (2.7)$$

where α is a “temperature” parameter that determines the importance of the entropy (randomness of action) term relative to the reward.

The training of the critic network is carried out by minimising the squared difference between the predicted Q value and reward plus the discounted expectation of the action-value :

$$\mathcal{L}(\theta) = \mathbb{E} [(Q_{\theta}(s_t, a_t) - y_t)^2] \quad (2.8)$$

with $y_t = r_t + \gamma(1 - d_t)Q_{\theta}(s_{t+1}, \pi_{\phi}(s_{t+1}))$, and d_t representing a binary variable that indicates whether the episode has terminated at time step t . Specifically, d_t takes on a value of 1 if the episode has terminated at time-step t (i.e., the agent has reached a terminal state), and 0 otherwise. Alternatively, the actor network is trained using the equation that follows:

$$\nabla_{\phi} J(\phi) = \mathbb{E} [\nabla_{\phi} \log \pi_{\phi}(s_t) \cdot \alpha \log \pi_{\phi}(s_t) - Q_{\theta}(s_t, a_t)] \quad (2.9)$$

where ϕ represents the parameters of the actor network, and θ represents the parameters of the critic network.

The ability of SAC to balance exploration and exploitation while promoting diverse and efficient policy learning means SAC is a powerful tool for training agents in complex and uncertain environments, making it one of the most popular algorithm choices in recent years for robotic manipulation tasks [91]–[95].

2.4.4 Deep Deterministic Policy Gradients

The Deep Deterministic Policy Gradients (DDPG) algorithm is another state-of-the-art model-free off-policy RL algorithm that combines the ideas of deep Q-learning and deterministic policy gradients by implementing an actor-critic architecture [77]. DDPG is specifically designed for continuous action spaces, making it well-suited for tasks such as robotic manipulation.

Similarly to SAC, in DDPG, two neural networks are used: an actor network that learns the policy (i.e., the optimal action to take in a given state) and a critic network that learns the Q-value function (i.e., the expected cumulative reward for taking a specific action in a given state). The actor network is trained to maximise the expected return by updating its parameters in the direction that increases the Q-value estimated by the critic network, while the critic network is trained using the TD error between the predicted Q-value and the actual reward received.

In the DDPG algorithm, both the actor and the critic networks are updated using the concept of deterministic policy gradients. Particularly, the critic network is updated using the Bellman equation to minimise the TD error between the estimated Q-values and the target Q-values, as done in SAC (see Equation 2.8). This network is trained by minimising the loss function using stochastic gradient descent or a similar optimisation algorithm [96].

The actor network is updated using the deterministic policy gradient to maximise

the expected return by adjusting the policy parameters in the direction that increases the Q-values. It uses the following equation:

$$\nabla \phi J \approx \mathbb{E}[\nabla_a Q_\theta(s, \pi_\phi(s)) \nabla_\phi \pi_\phi(s)] \quad (2.10)$$

where J is the expected return, $\pi_\phi(s)$ is the current policy determined by the actor network, $Q_\theta(s, \pi_\phi(s))$ is the estimation of the Q-value done by the critic network, ϕ represents the parameters of the actor network, and θ represents the parameters of the critic network.

Overall, DDPG is a powerful algorithm for learning continuous action and state space control policies in high-dimensional state spaces, and has been successfully applied to a wide range of robotic manipulation tasks, including real-world scenarios [71], [97]–[101].

2.4.5 Twin-Delayed Deep Deterministic Policy Gradients

The Twin-Delayed Deep Deterministic Policy Gradients (TD3) algorithm is an extension of the DDPG algorithm, and as such it is also a model-free off-policy RL algorithm designed for continuous action spaces [102]. This algorithm was proposed to address some of the limitations of DDPG, such as overestimation of Q-values and sensitivity to hyperparameters.

The problem of overestimation of Q-values occurs when the estimated Q-values in an RL algorithm are consistently larger than the true values [103], [104]. This can lead to sub-optimal policies being learned and slower convergence to the optimal policy. Overestimation of Q-values can be particularly problematic in algorithms using action-value functions, where the Q-value is used to estimate the value of an action, such as the algorithms presented throughout this entire Section.

In TD3, two Q-value networks (also known as twin Q-networks) are used to reduce overestimation bias. Additionally, TD3 introduces a target policy smoothing technique and delayed policy updates to stabilise the training and improve the performance of the model. By using these modifications, TD3 aims to achieve bet-

ter sample efficiency and robustness compared to DDPG in tasks with continuous high-dimensional state and action spaces.

When training the critic networks, both critics are updated independently to mitigate overestimation bias and enhance the stability of Q-value estimates, and the target Q-values for the critic networks are calculated using the minimum value between the two critics to address potential overestimation. Both networks are updated by minimising the temporal difference error between the predicted Q-values and the target Q-values obtained from the twin networks, as done with the critic network of a regular DDPG (see Equation 2.8).

The actor network is updated by maximising the Q-values obtained from the twin critic networks through the deterministic policy gradient. Furthermore, a target policy smoothing technique is applied during the actor update by adding Gaussian noise to the target policy, which promotes smoother policy updates and reduces fluctuations during training. It is also updated less frequently than the critic networks with the aim of stabilising the learning.

Overall, the combination of improved stability, reduced overestimation bias, and efficient handling of high-dimensional continuous action and state spaces has contributed to the increase of popularity of the TD3 algorithm in the realm of robotic manipulation over the past three years [105]–[109].

2.4.6 Summary of algorithms

Throughout the development of this thesis, only one of the off-policy RL algorithms presented in this Section is used: the DDPG algorithm. There are several reasons for this.

One primary reason is the simplicity and ease of implementation of DDPG compared to more complex algorithms like SAC or TD3. DDPG has been widely used in the literature and has a well-established code base compared to SAC and TD3 (especially when this research began, at the beginning of 2020), and many of the solutions in the existing literature use DDPG as their baseline algorithm (see Section 2.5 for

more details). Because RL is very sensitive to the hyperparameter selection, and we wanted to avoid as much as possible the need to tune hyperparameters (because it is a resource-intensive process), DDPG was the most sensible algorithm selection in that context. Despite this, if the thesis was to begin in 2024, when the code base of SAC and TD3 has been sufficiently developed to be stable, and when it would be less difficult to identify appropriate hyperparameters (because other people have already discovered them), these algorithms would be given more serious consideration for inclusion in this work.

Besides this, one of the main concerns of the general public about the introduction of RL-based robotic manipulators into the real world is the uncertainty associated with artificial intelligence (AI) control algorithms. DDPG is an off-policy RL algorithm that employs a deterministic policy, which is particularly useful in robotic manipulation tasks where precise and deterministic actions are often required for safety reasons. On the other hand, algorithms like SAC introduce stochasticity through various methods (i.e., by maximising the entropy and introducing a smoothing policy technique), which adds to the inherent uncertainty that most AI models generally have.

2.5 State of the art review

This section reviews the state of the art for sample, time, and performance efficiency. In particular, Sub-section 2.5.1 introduces the Hindsight Experience Replay (HER) strategy, a pivotal algorithm for this thesis [54]. Section 2.5.2 introduces several extensions of that algorithm, including hindsight goal generation, hindsight sampling strategies, and hindsight action selection methods (or model-based methods).

2.5.1 Hindsight Experience Replay

Hindsight Experience Replay (HER) is a technique used in RL, particularly in the context of robotic manipulation and goal-based RL tasks, to improve learning ef-

iciency and tackle the sparse reward problem [54]. HER operates by re-framing failure as success, enabling the agent to learn from its past experiences more effectively.

The sparse reward problem, which has already been introduced in Section 2.3.2, is a common challenge in RL where the agent receives very few or infrequent rewards during its interactions with the environment. In tasks that use sparse rewards, the agent may only receive feedback (i.e., rewards) sporadically, making it challenging to learn an effective policy that leads to desirable outcomes.

The main idea of HER is to generate additional learning experiences from unsuccessful episodes by saving these episodes in the replay buffer and changing the goal for those episodes when they are replayed. Instead of considering only the original goal that was not achieved, the agent redefines the goal to include the achieved state for that episode (see Figure 2.4). By doing so, the agent can extract valuable learning signals from what was previously considered a failure, and therefore the learning gets accelerated, effectively mitigating the impact of sparse rewards. These altered goals are usually called *hindsight goals*.

When training any off-policy RL-based agent, it samples experiences from the replay buffer, which under the HER logic also includes hindsight goals. By incorporating these hindsight transitions during training, the agent can not only learn from achieving the primary goal but also from achieving intermediate goals along the trajectory, if said intermediate goals belong to the goal space $\mathcal{G}$. This makes off-policy RL algorithms learn more effectively making use of all experiences to update and improve its policy.

In particular, the authors of the paper used DDPG as their base off-policy RL algorithm, and most of the extensions, that will be exposed in Section 2.5.2, also make use of DDPG as their baseline. This is another reason why DDPG was selected as the baseline algorithm in this thesis.

In conclusion, the ability of HER to address the sparse reward problem, improve learning efficiency, and integrate well with existing off-policy RL frameworks makes

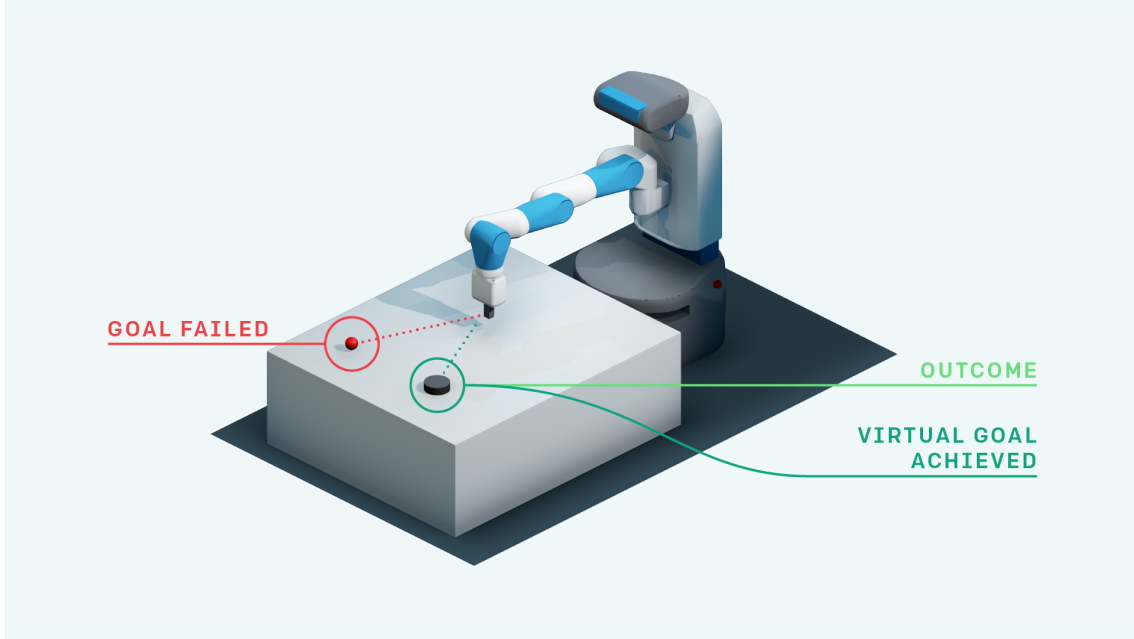


Figure 2.4: Visual representation of the HER algorithm. When an episode fails, HER modifies the goal so that it matches with the outcome of the episode. By applying this simple method, the agent is now able to learn from mistakes as well as from successes. Image taken from <https://openai.com/research/ingredients-for-robotics-research>

it a highly popular and effective technique in the field of robotic manipulation. Its impact on accelerating the learning of dexterous manipulation skills by robotic agents has solidified its position as a go-to method for researchers in the field [110]–[114].

Despite the benefits of employing HER, there are some risks associated with its usage. Firstly, by learning from hindsight experiences, there is a risk that the agent may overfit to specific scenarios or sequences of actions, leading to sub-optimal performance in complex environments. Therefore, selecting appropriate hindsight goals for the agent to learn becomes paramount and it can be challenging depending on the task at hand, as it requires a good understanding of the task and environment to ensure that the goals are meaningful, informative and sufficiently diverse.

This can be particularly difficult in tasks with complex high-dimensional action, state and/or goal spaces; as HER usually requires large quantities of simulated experience to find a policy capable of solving complex tasks in such environments, which usually leads to very large training times. In other words, a standard HER is

usually sample inefficient when dealing with complex environments.

Furthermore, because HER is a model-free algorithm, it has no knowledge of the dynamics of the environment and, therefore, learning manipulation tasks from scratch becomes very challenging.

2.5.2 Hindsight Experience Replay extensions

A number of variations of HER have been proposed by the research community in recent years. In this section, these variations are categorised according to the course of action taken by the algorithm variant:

- **Hindsight Goal Generation:** This variation focuses on how goals are generated for the replay buffer, which directly impacts the learning process.
- **Hindsight sampling strategies:** This variation looks at different ways to sample transitions from the replay buffer, which also directly impacts the quality of the learned policy.
- **Hindsight action selection:** This variation focuses on finding methods for selecting actions during the replay process leveraging the exploration-exploitation decoupling mechanism of off-policy RL.

Usually, most methods that fall into one of these categories can be used in conjunction with methods of other categories. In this section, some of the most popular and successful methods are introduced.

Hindsight Goal Generation

The Hindsight Goal Generation (HGG) mechanism was proposed by Ren *et al.* [115] in order to prevent the selected hindsight goal distribution from differing from the original goal distribution. This is particularly useful for tasks where the goal distribution does not match the state distribution, as most hindsight goals usually fall near the initial state, especially in the first interactions with the environment. For

example, in a task where a robotic arm has to pick an object and throw it to a place out of reach of the robotic arm, the only hindsight goals that are useful for solving the task would be those where the object lands outside of the reach of the robotic arm, but instead most of the hindsight goals at the beginning of the exploration are more likely to be nearby the original position of the object.

The main idea behind HGG is to create intermediate alternative goals that are in-between the initial state and the goal and that are easy to achieve in the short term and are also likely to ease the task of reaching the final goal in the long term. To do that, they approximate the value function of the actual goal distribution by a lower bound that decomposes into two terms: a value function based on a hindsight goal distribution and the Wasserstein distance between the two distributions [116]. Then, they introduce an efficient discrete Wasserstein Barycenter solver (i.e. a method that finds a central distribution that best represents a collection of distributions) to generate a set of hindsight goals that optimises the lower bound. Once this set of hindsight goals have been generated, they are used for exploration.

In optimisation problems, the lower bound represents the best possible value that can be obtained for the given constraints and conditions (i.e., in our context, the goal space and the initial state space distributions). By focusing on optimising the lower bound, one aims to enhance the overall performance or efficiency of an algorithm, ensuring that it operates as effectively as possible within the specified limitations.

A popular variant of HGG is the Graph-based Hindsight Goal Generation (G-HGG), which is a very useful algorithm for environments that have obstacles and turns the robotic manipulation task into a “*maze-solving*” robotic manipulation task [117]. In the G-HGG mechanism, the Euclidean distance is replaced by an obstacle-avoiding graph, which is a discrete representation of the environment’s 3D space, and the intermediate hindsight goals are chosen based on the shortest distances following the graph.

Another approach inspired by this is the Hindsight Planner [118]. In their ap-

proach, they make use of a Long Short-Term Memory (LSTM) network [119] in order to plan the intermediate goals by making this LSTM predict the next intermediate goal such that the last predicted goal by the LSTM is the episode’s goal, and then using standard DDPG + HER as the learning algorithm.

GoalGAN is another method that is typically used, which consists of a two-stage end-to-end trajectory prediction method that separates the prediction task into goal position estimation and routing [120]. The goal position estimation Generator-Discriminator architecture of GoalGAN explicitly estimates the probability distribution of future goal positions, and thus enables the sampling of intermediate goals that can be used for exploration.

All these approaches improve the sample efficiency of DDPG + HER, but at the cost of increased computational complexity. This translates into longer training times depending on the environment and the task, which effectively counteracts the benefits of sample efficiency improvements in these scenarios, as there is no point in learning with fewer interactions if this learning takes more time.

Hindsight sampling strategies

Deciding which hindsight goals to sample and which transitions to use for updating the policy is vital when training a HER-based algorithm. Some extensions of HER that focus on improving the sampling strategy include:

- **Prioritised Hindsight Experience Replay:** This evolution assigns priorities to transitions in the replay buffer based on their potential for learning and improvement. By sampling transitions with higher priorities more frequently, the algorithm can focus on experiences that are likely to lead to significant learning progress [121]–[123].
- **Curiosity-Driven Hindsight Experience Replay:** This evolution incorporates curiosity-driven exploration into the sampling strategy, encouraging the agent to seek out novel and informative experiences. By prioritising transitions

that lead to unexpected or interesting outcomes, the algorithm can enhance its learning process and discover new strategies for achieving goals [124]–[127].

- **Diversity-Driven Hindsight Experience Replay:** This evolution emphasises sampling diverse transitions from the replay buffer to ensure that the agent explores a wide range of scenarios and learns robust policies. By promoting diversity in the sampled experiences, the algorithm can avoid overfitting to specific situations and improve its performance in varied environments [128]–[130].

One of the most popular of these techniques is the Energy-Based Hindsight Experience Prioritisation (EBP-HER), which is a diversity-driven algorithm [129]. In EBP-HER, a trajectory energy function takes the potential, kinetic, and rotational energy into consideration defined as the sum of the transition energy of the target object over the trajectory (see Equation 2.15), and episodes that have high trajectory energy are prioritised while sampling from the buffer. The transition energy is defined as the total energy increase from the previous state s_{t-1} to the current state s_t .

In particular, the potential energy is calculated as:

$$E_p(s_t) = mgz_t \quad (2.11)$$

where $E_p(s_t)$ is the potential energy at the state s_t , m is the mass of the object, $g \approx 9.81m/s^2$ is the gravity of earth, and z_t is the altitude of the object at time-step t .

The kinetic energy is defined as:

$$E_k(s_t) = \frac{m}{2}[v_{x,t}^2 + v_{y,t}^2 + v_{z,t}^2] \quad (2.12)$$

where $E_k(s_t)$ is the kinetic energy at the state s_t , m is the mass of the object and $v_{x,t} \approx (x_t - x_{t-1})/\Delta t$, $v_{y,t} \approx (y_t - y_{t-1})/\Delta t$, $v_{z,t} \approx (z_t - z_{t-1})/\Delta t$, where Δt is the time-step interval, are the velocities along each of the axes.

The rotational energy is calculated as:

$$E_r(s_t) = \frac{1}{2}[I_x\omega_{x,t} + I_y\omega_{y,t} + I_z\omega_{z,t}] \quad (2.13)$$

where $E_r(s_t)$ is the rotational energy at the state s_t , I_x, I_y , and I_z are the moment of inertia around each axis of rotation (these are set to constants equal to 1), and $\omega_{x,t} \approx (\phi_t - \phi_{t-1})/\Delta t$, $\omega_{y,t} \approx (\theta_t - \theta_{t-1})/\Delta t$, and $\omega_{z,t} \approx (\psi_t - \psi_{t-1})/\Delta t$ are the angular velocities for an object pose represented as an Euler orientation $p = (\phi, \theta, \psi)$.

The total energy is defined as $E(s_t) = E_p(s_t) + E_k(s_t) + E_r(s_t)$, and the energy of a state transition and the trajectory energy are defined as follows:

$$E_{tran}(s_{t-1}, s_t) = \min(E(s_t), E_{tran}^{max}) \quad (2.14)$$

$$E_{traj}(\tau) = E_{traj}(s_1, s_2, \dots, s_T) = \sum_{t=1}^T E_{tran}(s_{t-1}, s_t) \quad (2.15)$$

where $E_{tran}(s_{t-1}, s_t)$ is the energy of a state transition, and E_{tran}^{max} is the predefined maximal transition energy value. According to the authors, the transition energy is clipped to a threshold to mitigate the influence of some particularly large-energy transitions. During sampling from the replay buffer, the agent uses the trajectory energy values directly as the probability for sampling (i.e., the value of the trajectory energy over the sum of the energy of all the trajectories of the buffer as done in Equation 2.16). In other words, the high energy trajectories have higher priorities to be replayed:

$$p(\tau_i) = \frac{E_{traj}(\tau_i)}{\sum_{n=1}^N E_{traj}(\tau_n)} \quad (2.16)$$

There are many different recent works in the literature that use this method, not only in robotic manipulation tasks, but in a wide range of applications [131]–[135]. For all the research in this thesis, the EBP-HER strategy is used to sample the transitions used to update the target policy. However, for simplicity, we will

denotes these methods as simply “DDPG + HER” or “DDPG + HGG”.

Hindsight action selection

Hindsight action selection algorithms are model-based off-policy RL algorithms that leverage a learned model of the environment to simulate possible trajectories and outcomes, allowing the agent to plan and make decisions while exploring the environment based on this prior knowledge rather than solely relying on the environment interactions. By integrating model-based planning with off-policy learning paradigms, model-based off-policy RL offers a powerful framework for agents to efficiently explore the environment, leverage simulated experiences for decision-making, and enhance learning and decision-making capabilities in complex tasks.

Some common algorithms that focus on improving the action selection mechanism during exploration include tree-based planning algorithms, such as Monte Carlo Tree Search (MCTS), which, by iteratively expanding a search tree and selecting promising action sequences, makes informed decisions based on the model’s predictions. [136]–[139]. Other approaches make use of Model Predictive Control (MPC) algorithms, which utilise a learned dynamics model of the environment to predict future states and rewards, optimising a sequence of actions to maximise a predefined objective [140]–[142].

Model-based off-policy RL algorithms offer various benefits, including improved sample efficiency, the ability to plan ahead, and enhanced decision-making. However, they also come with several drawbacks that can limit their practical application in certain scenarios. First, model-based algorithms rely heavily on the learned dynamics model, meaning that if the model fails to accurately capture the true dynamics of the environment, errors in predictions can propagate and lead to sub-optimal decision-making. Furthermore, learned models can introduce bias in the agent’s decision-making process, particularly in environments with complex or high-dimensional state spaces.

Another key consideration about model-based algorithms is the increased com-

putational complexity associated with them, which can hinder real-time decision-making and scalability, as well as the total training time. Moreover, balancing exploration and exploitation is challenging in model-based off-policy RL, as agents must decide between leveraging the learned model for planning and interacting with the environment to gather valuable novel experiences.

In Chapter 1, the following hypothesis is stated in relation to this:

- **Hypothesis 3:** *The use of fixed set of basic primitive policies limits the sample efficiency and the performance of robotic manipulation to the expertise of these primitive behaviours. However, we hypothesise that if an algorithm can decide during exploration between when to use one of the existing primitive policies and when to learn a new skill, robots will become more adaptable to dynamic environments and will learn to perform more complex tasks more quickly.*

This is very much related to hindsight action selection algorithms, as we are essentially hypothesising that the use of a learned model that leverages knowledge from primitive behaviours can make for a more effective action selection algorithm that would lead to faster policy improvement and higher adaptability to new tasks.

2.6 Conclusion

As has been seen in this section, sample efficiency remains a major challenge in RL-based robotic manipulation solutions, and most algorithms that address sample efficiency do it at the cost of increased computational complexity, extending the training time.

As such, in this thesis, three different strategies are explored, which are related to the hypotheses and research questions already stated in Chapter 1. First, we explore the use of reward engineering, and in particular of intrinsic rewards, because the use of these rewards does not introduce much computational complexity and, at the same time, can guide the agent towards more efficient exploration during training.

Secondly, we evaluate the sequential use of learned primitive policies, which do not require extensive environment exploration to be learned, in order to solve more complex robotic manipulation tasks, inspired by the typical BT algorithm used in traditional robotic manipulation.

Lastly, we propose a novel algorithm that automatically decides when to use the primitive behaviours and when to instead use the learned target policy during exploration, aided by the learned action-value function, with the aim to find a more sample and time efficient mechanism to learn complex manipulation tasks in complex environments (e.g., an environment that contains obstacles that must be avoided).

As justified in Section 2.4.6, this thesis focuses on DDPG trained in conjunction with HER (DDPG + HER), and other extensions of this algorithm such as HGG or G-HGG, as a methodological framework within which to complete this work. Furthermore, the EBP-HER strategy is applied in all chapters, but for simplicity, we term the method “HER”.

Chapter 3

Combination of dense and sparse reward signals

3.1 Chapter overview

This chapter addresses one of the main challenges facing robotic manipulation learning algorithms: learning to lift an object in a sample efficient fashion. In particular, we aim to answer the following research question:

- **Research Question 1:** How important is reward engineering in achieving sample and time efficiency when training a RL-based algorithm to lift an object?

To do this, we study the effect in terms of sample efficiency and performance that reward engineering has on a Deep Deterministic Policy Gradients (DDPG) [77] and Hindsight Experience Replay (HER) [54] based agent learning to manipulate a cube. In particular, we analyse the effect of adding a dense reward component that encourages a lifting behaviour by penalising the agent more severely when the manipulated cube is below the target position.¹ The work described in this chapter

¹The work described in this chapter was carried out as part of our participation in the Real Robot Challenge 2021, and done in collaboration with fellow students Robert McCarthy and Qiang Wang. Specifically, Robert and I took the leadership while the challenge was taking place, both of us running experiments at the same time due to the large training times required, while Qiang took the lead on expanding the results of our method after the competition was finished and extended the work by solving a more advanced task using our final model as the baseline.

was published at the Irish Conference on Artificial Intelligence and Cognitive Science (AICS) 2021 [143] and in the Expert Systems journal [144]. Section 3.2 identifies the motivations for using reward engineering and reviews some examples from the literature, Section 3.3 introduces the Real Robot Challenge (RRC) 2021 [33], while Section 3.4 details the proposed method, including the network design, the novel reward system and other implementation details related to the learning algorithm. In Section 3.5 the results obtained in the simulated environment are presented, while the final results obtained with the real robot as well as a comparison with the solution proposed by other teams can be found in Section 3.6. Section 3.7 discusses different applications of our method that other works have developed, and Section 3.8 presents the conclusions of the chapter. Finally, a summary of the contributions can be found in section 3.9.2.

3.2 Motivation

There are a number of design choices that must be made before training an RL-based agent. Some of them, such as the structure of the observation and action spaces are usually constrained by the nature of the environment, as well as the task in hand (e.g., observation and action spaces will be different depending on whether the agent is a robotic arm with a parallel-jaw gripper or a robotic hand). Alternatively, the reward design is a fully open choice, meaning that many different reward systems can be used with success in order to solve the same task. As a result, reward engineering (or reward shaping) is a topic that has attracted a lot of interest since the emergence of RL [145]–[148].

Reward engineering is a crucial aspect for improving sample efficiency in RL [149]. By carefully designing and engineering the reward function, a learning agent can be guided towards desired behaviours and outcomes. A well-defined reward function provides clear signals to the agent about what actions are more desirable and helps the agent to explore the environment more efficiently by focusing on learning the most relevant information [150].

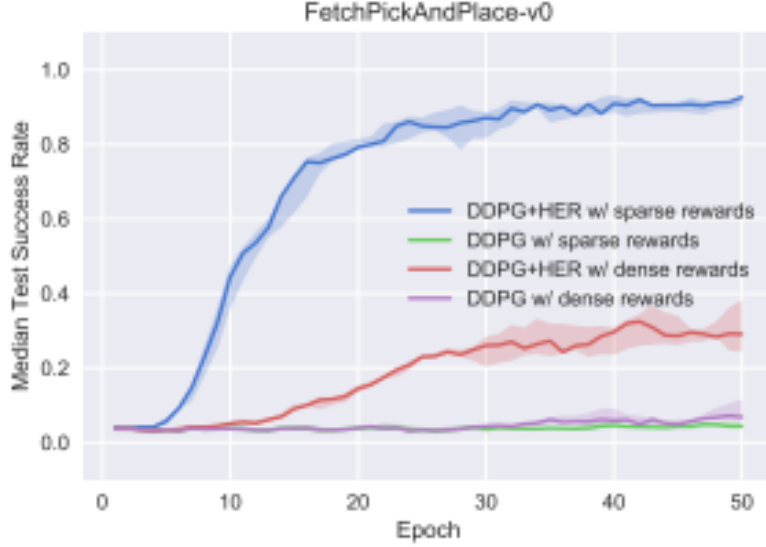


Figure 3.1: Results of Plappert *et al.* [154] on a pick and place task using DDPG+HER with different reward strategies. The use of sparse rewards and HER is hugely beneficial compared to other approaches. Note that here *dense reward* refers to a negative Euclidean distance, while *sparse reward* refers to the same reward system as described in Equation 2.4.

Exploring different reward systems can provide alternative perspectives on the task at hand, allowing us to capture different aspects of the desired behaviour [151]. This exploration can help us understand the trade-offs between different reward structures and their impact on the learning process, as well as identify potential unintended consequences that may arise from certain reward designs.

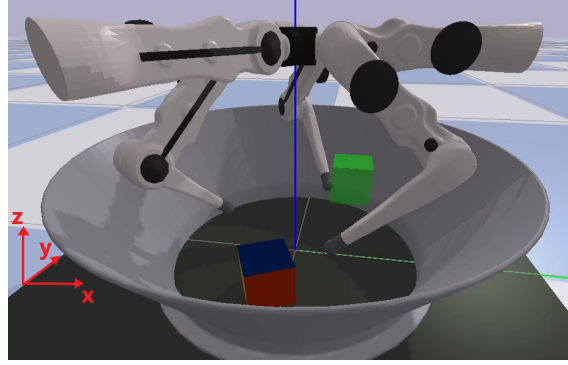
Furthermore, reward engineering allows us to address some of the weaknesses of sparse reward systems, where the agent receives feedback only at certain moments while exploring the environment. By carefully designing the reward function, it is possible to provide more informative rewards that bridge the gap between the sparse feedback and the actions taken by the agent [152], [153].

The use of reward engineering has been extensively investigated for various manipulation tasks with different robotic manipulators in recent years, particularly to solve the basic yet difficult challenges of (i) learning to grasp objects [155], [156] and (ii) protecting the hardware from damage [157]. In this chapter, we focus on the first of these two challenges, learning to hold an object. Plappert *et al.* [154] reported the use of HER in combination with different reward strategies, which proved that the

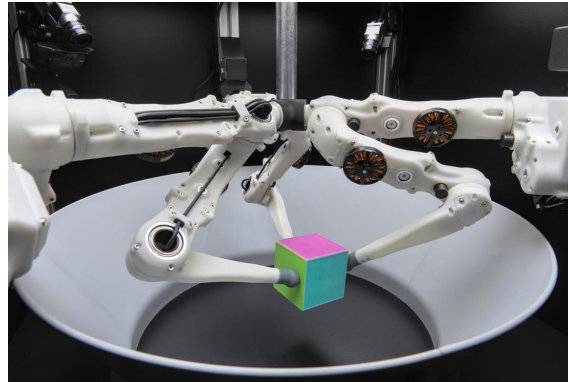
use of sparse reward systems in conjunction with HER was the superior method of those tested, as it is shown in Figure 3.1. Other examples include Kim *et al.* [158], who proposed a combination of sparse rewards with different weights by dividing the problem into three sub-tasks: two reaching tasks (i.e., approaching the object and reaching the goal position) and one grasping task; or Luo *et al.* [156], who designed a reward system to solve a lifting task, beginning their experiments using the dense reward and switching to the sparse reward system after a certain number of epochs, which is a method that they called *Dense2Sparse*.

The latter is an example of a reward shaping method that modifies the reward system mid-training. There are other examples that follow this approach: in the case of Luo *et al.*, this was done by introducing yet another hyperparameter, which complicates even further the applicability of the algorithm; this value will change depending on the environment and task. Other reward shaping mechanisms that follow this strategy make use of learned weights applied to different rewards that are used together to provide feedback to the agent, and some others even directly depend on external feedback [159]–[161]. However, these methods increase the algorithm’s computational complexity, extending the training time required to train those models, as either the weights need to be learned or the external feedback needs to be processed and integrated into the algorithm.

In this chapter, a novel reward system that encourages the agent to quickly learn a block lifting behaviour is proposed, that does not need to change mid-training. The proposed reward system combines the benefits of both sparse and dense reward systems, and it is evaluated within the context of the RRC 2021 competition, a challenge featured in the NeurIPS Competition Track [33]. This reward system constitutes, to the best knowledge of the author, the state-of-the-art on the ‘*Move Cube on Trajectory*’ task of the TriFinger robot [21] and improves the sample efficiency with respect to a DDPG-agent trained using HER with sparse rewards.



(a) Simulation



(b) Real Robot

Figure 3.2: The simulated and real TriFinger robotic environments. The task is to bring the cube to specified 3-D goal coordinates, along a goal trajectory.

3.3 The Real Robot Challenge (RRC)

Simulators, particularly in the context of robotic learning, are not perfect models of reality. There are multiple factors that make real world scenarios more challenging than a simulated environment. For example, sensors produce noisy data, or robots suffer from vibrations; both are real-world phenomena that are typically not represented in simulators. Robotic manipulation in the real world is still, therefore, very challenging. To direct attention towards real-world performance, the robotics community has been very active in organising competitions like the Real Robot Challenge (RRC) [33].

The RRC is a competition organised by the Max Planck Institute for Intelligent Systems. The 2021 edition consisted of a three-fingered robot that had to carry a cube along specified positional goal trajectories. The organisers of the challenge provided a simulator for the robot, along with remote access to real robots so that

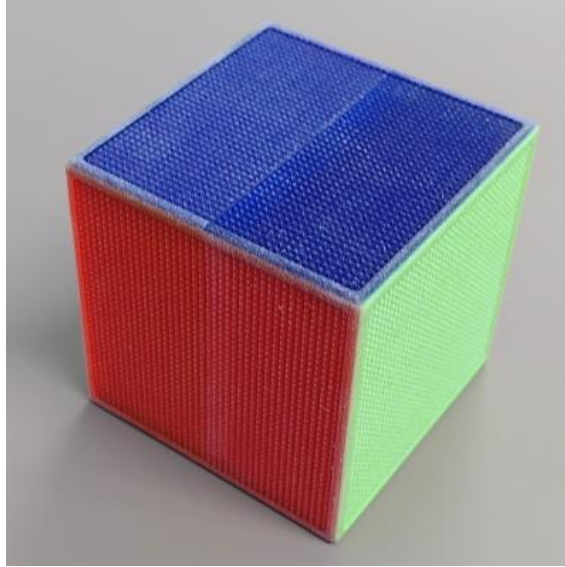


Figure 3.3: Image of a real cube identical to those used during the challenge.

participants could run their experiments on a real setup.

3.3.1 Environment

TriFinger robot and cube

The TriFinger robot is a robotic system developed by the Robotics and Perception Group at the Max Planck Institute for Intelligent Systems [21]. It is a dexterous robotic robot with three fingers, each equipped with three joints, allowing for the performance of a wide range of grasping and manipulation tasks. The TriFinger robot is designed to perform various tasks, such as object manipulation, grasping, and manipulation of small objects.

In the 2021 edition of the RRC competition, the cube used had a width of 6.5 cm and a weight of 94 g, and its surface texture was rough, to facilitate grasping behaviours. Each side had a different colour, to help with camera-based object tracking, which consists of a colour segmentation method already provided by the challenge organisers (see Figure 3.3).

Observation space

In total, the observation space has 44 dimensions. These include:

1. Robot joint positions, velocities, and torques;
2. The cube’s current pose (i.e., its position and orientation, which in simulation is read directly from the simulator with no measurement error, but for the real world arena is estimated using the camera-based object detection and segmentation methods provided by the challenge organisers), along with the difference between the poses at current and previous time-steps;
3. Current goal coordinates.

Action space

Pure torque control of the robot arms is employed with an update frequency of 20 Hz (i.e., each time-step in the environment equates to 0.05 s). The robot has three arms, each with three motorised joints; thus, the action space is 9-dimensional and continuous.

3.3.2 Task definition

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, $t \in \{0, 30000, 40000, 50000, \dots, 110000\}$. The obtained score, $score_{pos}$ is defined by a distance-based weighted criterion designed

by the organizers of the RRC [33]:

$$score_{pos} = -\frac{1}{2} \left(\frac{\|s_{xy} - g_{xy}\|}{2d_r} + \frac{\|s_z - g_z\|}{d_h} \right) \quad (3.1)$$

where s_{xy} and s_z are the observed xy and z coordinates of the cube, respectively, and g_{xy} and g_z are the goal 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, and a faster manipulation as well (the score is calculated for each time-step, which means that the faster $\|s_{xy} - g_{xy}\|$ and $\|s_z - g_z\|$ get to 0 the higher the global score).

It is important to note that this metric is only used for evaluation by the organisers of the RRC. In contrast, to evaluate the performance of our method, we use the success rate as the main metric. An episode is deemed successful if, when complete, the final goal of the trajectory has been achieved. By using the success rate metric we can also analyse the sample efficiency of our proposed method, which is the main interest of this thesis.

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.

3.4 Policy learning through reward engineering

3.4.1 Network

The goal-based nature of the proposed task makes HER a natural choice of learning algorithm, as this algorithm has excelled in similar goal-based robotic tasks and, theoretically, obviates the need for complex reward engineering [162]–[165]. As such, we implemented a DDPG agent to be trained in conjunction with HER. The actor

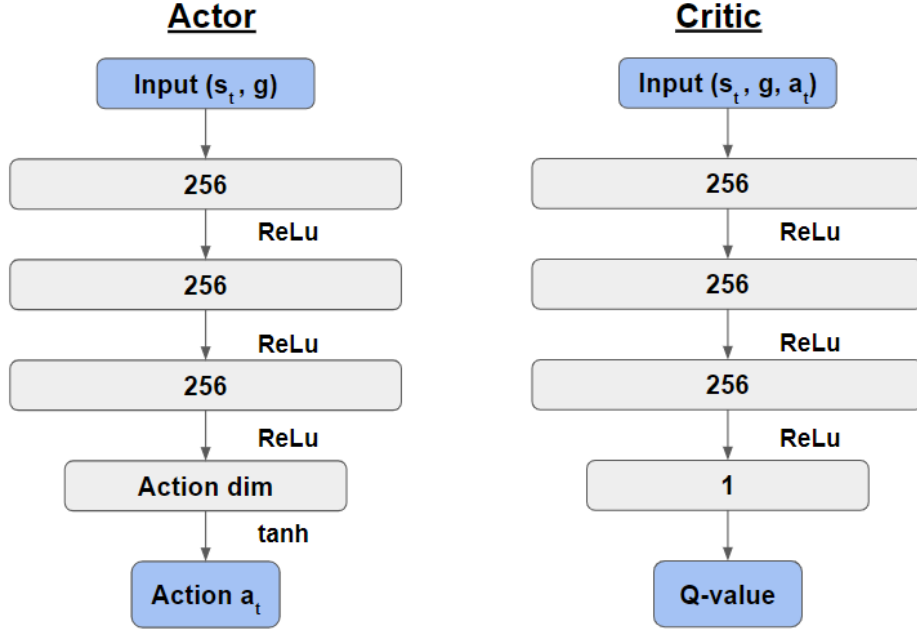


Figure 3.4: Network architecture of the DDPG agent, consisting of an actor which estimates the action a_t given a state of the environment s_t and a goal g ; and a critic, which evaluates the quality of the taken action a_t .

and the critic networks of the DDPG agent have both three hidden fully-connected layers of 256 dimensions, as proposed by Plappert *et al.* to solve other robotic manipulation tasks [154] (see Figure 3.4). Details about the hyperparameter settings of the network are included in Appendix A.1.2 for reproducibility and repeatability purposes.

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 actuality g_{t+1} was different². Thus, the policy focuses solely on achieving the current active goal and is unconcerned by any future changes in the active goal.

3.4.2 Reward formulation

We used DDPG + HER making use of a simple sparse reward that checked whether the cube was in the goal position or not; but in our early experiments, we observed

²Interestingly, we found that exposing the agent (during updates) to transitions in which $g_{t+1} \neq g_t$ significantly decreased performance, perhaps due to the extra uncertainty this introduced to the DDPG action-value estimates.

that this standard approach was slow to learn to lift the cube. To solve this, we alter the reward system incorporating an additional dense reward that encourages the robot to lift the cube.

In our approach, the agent receives two reward components: a sparse reward based on the cube's xy coordinates, r_{xy} , and a dense reward based on the cube's z coordinate, r_z (the coordinate frame can be seen in Figure 3.2 (a)).

The sparse xy reward is calculated as:

$$r_{xy} = \begin{cases} 0 & \text{if } \|g'_{xy} - g_{xy}\| \leq 2 \text{ cm} \\ -1 & \text{otherwise} \end{cases} \quad (3.2)$$

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 agent receives a negative feedback when the cube is farther than 2 cm, as this distance is widely used by the community with this robot and this cube, which means that our results will be easier to compare [166]. The dense z reward is defined as:

$$r_z = \begin{cases} -a|z_{cube} - z_{goal}| & \text{if } z_{cube} < z_{goal} \\ \frac{-a}{2}|z_{cube} - z_{goal}| & \text{if } z_{cube} > z_{goal} \end{cases} \quad (3.3)$$

where z_{cube} and z_{goal} are the z -coordinates of the cube and goal, respectively, and a is a parameter that weights r_z (we use $a = 20$, see Appendix A.1.1).

Note that both r_{xy} and r_z are in different scales. To address this issue, for each action, the agent receives the addition of both reward components normalised between -1 and 0.

In our final model, we only applied HER to the xy coordinates of the goal; i.e., the xy coordinates of the goal can be altered in hindsight, but the z coordinate remained unchanged. Thus, our HER altered goals are: $\hat{g} = (g'_x, g'_y, g_z)$, meaning only r_{xy} was recalculated after HER was applied to a sampled transition during

policy updates. This reward system is motivated by the following considerations:

1. Using r_{xy} with HER allows the agent to learn to push the cube 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 the xy -plane of the arena floor, it can then more easily stumble upon actions which lift it. Importantly, the $r_{xy} + \text{HER}$ 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 learning, the cube mostly remains on the floor. Consequently, during these stages most g' sampled by HER will be on the floor. Thus, applying HER to r_z here 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 $\hat{g}$ maintain their original z height. This leads to opportunities for the agent to be rewarded by r_z for performing cube-lifting.

3.4.3 Exploration vs exploitation

We derive our DDPG + HER hyperparameters from Plappert *et al.* [154], who use a highly ‘exploratory’ policy when collecting data in the environment; they assign probability of 30% to choosing random actions, sampled uniformly from the action space (i.e., they use the ϵ -greedy algorithm with 0.3 probability of selecting a random action), 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, as we found that the exploratory policy repeatedly drops the cube due to the randomly sampled actions and injected action noise.

To resolve this issue, rather than slowly reducing the level of exploration each epoch by adapting the epsilon value, which would require adding yet another hyperparameter that must be tuned, we make efficient use of simulated evaluation episodes (performed by the standard ‘exploiting’ policy and used to measure the performance of each policy update) by additionally adding them to the replay buffer. In this setup, 90% of rollouts added to the buffer are collected with the exploratory policy, and the remaining 10% with the exploiting policy.

3.5 Experiments in the simulated environment

3.5.1 Experiments formulation

When running experiments in the simulated environment, the main objective is to find the best policy to solve the task in order to potentially deploy it on the real robot. In our context, there are a number of reward combinations and configurations that we can employ: using purely sparse rewards, purely dense rewards or a combination of both (i.e., our method), and whether to use HER on all the components of the reward or only with certain components.

In particular, we use the following notation to express different experiments:

- **HER(Sparse(XY))+Dense(Z)**: DDPG agent trained using the reward system explained in Section 3.4.2, applying HER logic only to the xy -component of the achieved position (i.e., the sparse component).
- **HER(Sparse(XY) + Dense(Z))**: DDPG agent trained using the above-mentioned reward system, applying HER logic to all (xyz) positional components.
- **Sparse(XYZ)**: DDPG agent trained using a sparse reward system, where the agent receives -1 when the task is not finished and 0 when it is completed. It is considered to be finished when the cube position is less than 2 cm away from

the target goal, as this is considered a small enough distance to deem the task complete: $\|g'_{xyz} - g_{xyz}\| \leq 2 \text{ cm.}$

- **HER(Sparse(XYZ))**: DDPG agent trained using the above-mentioned sparse reward and applying HER logic. This is the vanilla DDPG+HER.
- **Dis(XYZ)**: DDPG agent trained using a dense reward, defined as a negative Euclidean distance (i.e., the closer the cube is to the goal position, the higher the reward is). This is the same as $Dense(XYZ)$, but as it is the actual distance of the object to its goal location we decided to give the more explicit name $Dis(XYZ)$
- **HER(Dis(XYZ))**: DDPG agent trained using the above-mentioned dense reward and applying HER logic.

where $Dense(\cdot)$ represents a negative Euclidean distance along/in the axis/plane, and $Sparse(\cdot)$ represents the reward system stated in Equation 2.4.

In order to properly evaluate the performance of these reward systems, rather than examining the reward values, we make use of the *success rate*. The success rate is defined as the percentage of goals that were accomplished over all of the test rollout episodes.

3.5.2 Discussion of learning in simulation results

As mentioned in Section 3.4.2, training standard DDPG in conjunction with HER with a simple sparse reward system (i.e., $HER(Sparse(XYZ))$) was found to be very slow, which was very problematic in the context of the RRC due to the strict competition entry submission deadlines. As can be seen in Figure 3.5, when training DDPG + HER with this reward system the agent had to interact with the environment for a total of at least 6 simulated days ³ to achieve competitive results (i.e., around a 70% success rate).

³A day in simulation is not equal to a real day, due to the use of multiple workers to generate data from the simulator

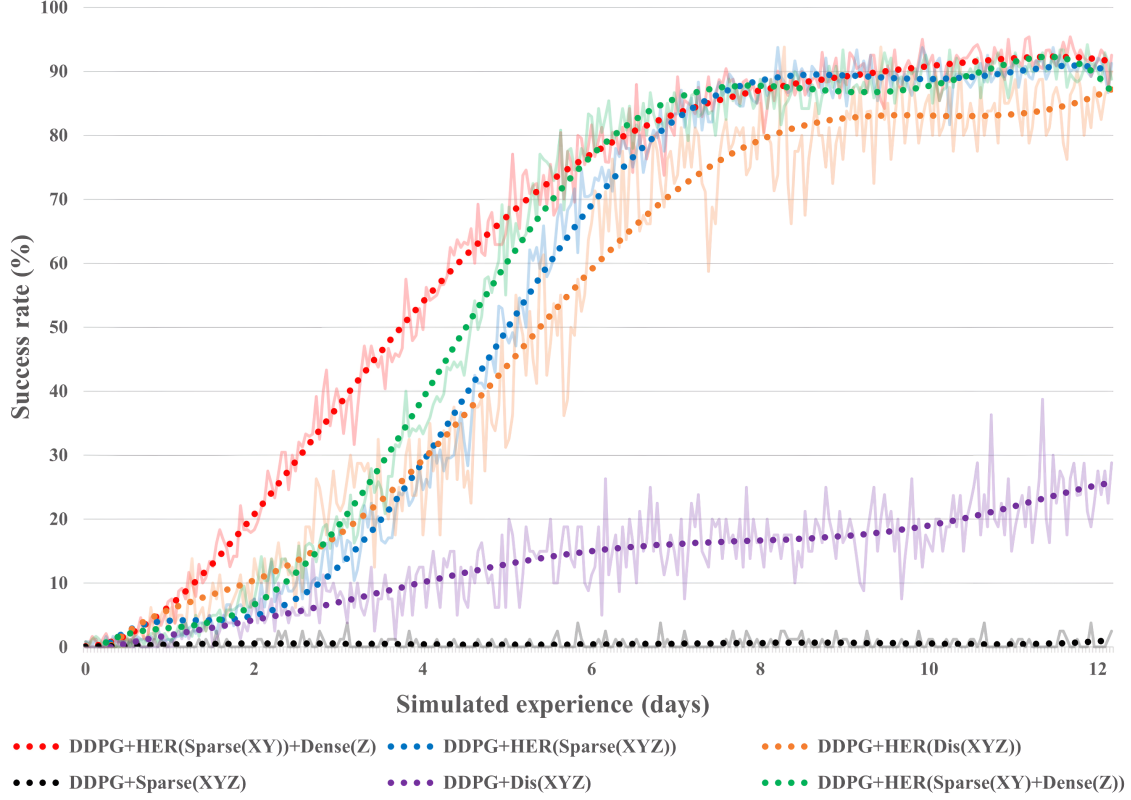


Figure 3.5: Success rate vs experience collected during simulated training (1 day in simulator, ≈ 1.7 million environment time-steps). We took the average over 3 randomly selected seeds for this illustration. The solid dotted lines are the trend lines and the faded lines are the raw data. We compare the results of learning 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 (blue); (iv) HER applied to pure distance reward, r_{dis} , using a negative Euclidean distance (orange); and (v) Pure distance reward, r_{dis} , without HER (purple); (vi) Standard sparse reward without HER (black).

From the results, it is evident that the use of HER is necessary to deal with this kind of task being performed with this complex manipulator, as the models that did not use the HER algorithm were unable to achieve competitive results. In the case of the sparse reward, it was not able to extract any useful policy, while in the case of the distance-based dense reward, the learning pace was extremely slow. On the other hand, all the models that used HER were able to learn policies that were capable of solving the task. However, those using our reward system (i.e., **HER(Sparse(XY))+Dense(Z)** and **HER(Sparse(XY) + Dense(Z))**) were able to learn faster, as both of them reach an 80% success rate before standard HER, as can be seen in Figure 3.5, regardless of the reward type used.

Not only that, but these two policies manage to achieve a success rate of 30% in 2-3 days, while standard HER takes 4 days. This means that when using our proposed reward, a lifting behaviour is learned faster and the agent benefits from this new skill in the early stages of learning. In particular, the policy learned by applying the HER logic only to the position in the xy -plane (i.e., **HER(Sparse(XY))+Dense(Z)**) is learned faster; especially in the first interactions with the environment when compared to the rate of learning of other policies (i.e., it is more sample efficient). This is because in those first interactions the agent is unable to lift the cube, and if we apply HER to the whole goal space (e.g., **HER(Sparse(XY) + Dense(Z))** or **HER(Sparse(XYZ))**) we create many valid goals that also do not lift the cube, and therefore we first learn a push policy; which would counter the effect of using the dense reward r_z we defined, whose intended purpose is to encourage a lifting behaviour by penalising more severely when the cube is below the target position.

Furthermore, the **HER(Sparse(XY))+Dense(Z)** reward system configuration eventually achieves a slightly better performance value at the later stages of the training, surpassing a 90% success rate. This is the reason why we decided to use this reward system and deploy this learned policy on the real robot.

3.6 Real robot experiments

3.6.1 Manipulation strategies

By making the agent focus on specific parts of the observation and using force feedback obtained from tactile sensors embedded in the finger tips, we trained three different strategies that could solve, to a certain degree, the task (see Figure 3.6). These are the following:

1. *Pushing*: By ignoring the positional error along the z -axis of the world frame, we managed to train a policy that learned a pushing behaviour.
2. *Cradling*: By also taking into account the error in the z -direction, but inci-

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

Table 3.1: Self-reported scores for the various manipulation strategies learned on the 2021 Real Robot Challenge. Scores are calculated using Equation 3.1.

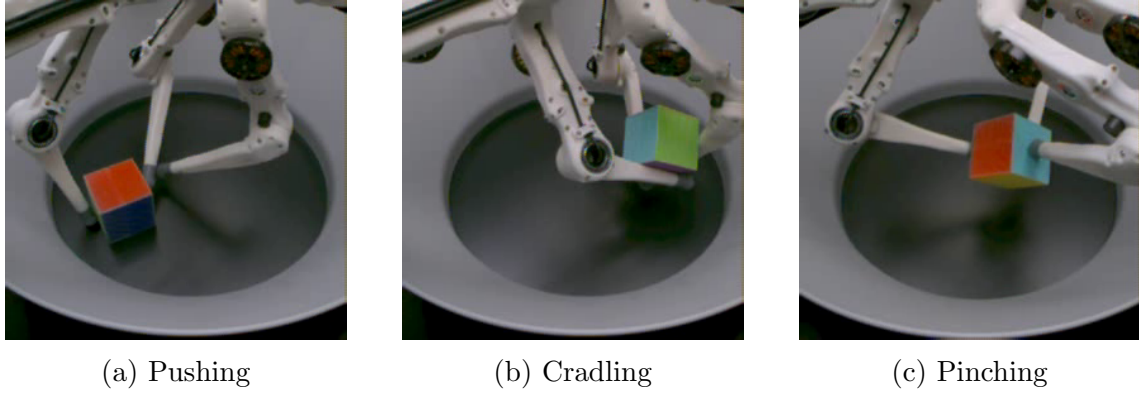


Figure 3.6: The various manipulation strategies learned by our team in order to solve the 2021 Real Robot Challenge.

dentally not securing the grasp in the fingertips of the TriFinger robot, we managed to train a policy that learned a cradling behaviour.

3. *Pinching*; By making use of force feedback from tactile sensors to help secure the grasp, adding its output information to the state, we managed to train a policy that learned a pinching behaviour.

The self-reported evaluation scores of each of these strategies can be seen in Table 3.1 (the scores are calculated using Equation 3.1). As expected, the best policy of all in the real robot is the pinching strategy, which secures the grasp through force feedback, and was the selected strategy for the challenge submission. Despite that, it is worth noting that only the learning of the pinching policy represented a significant improvement on the real robot, as in the simulated environment the cradling and the pinching policies, despite their difference in behaviour, had very similar performances.

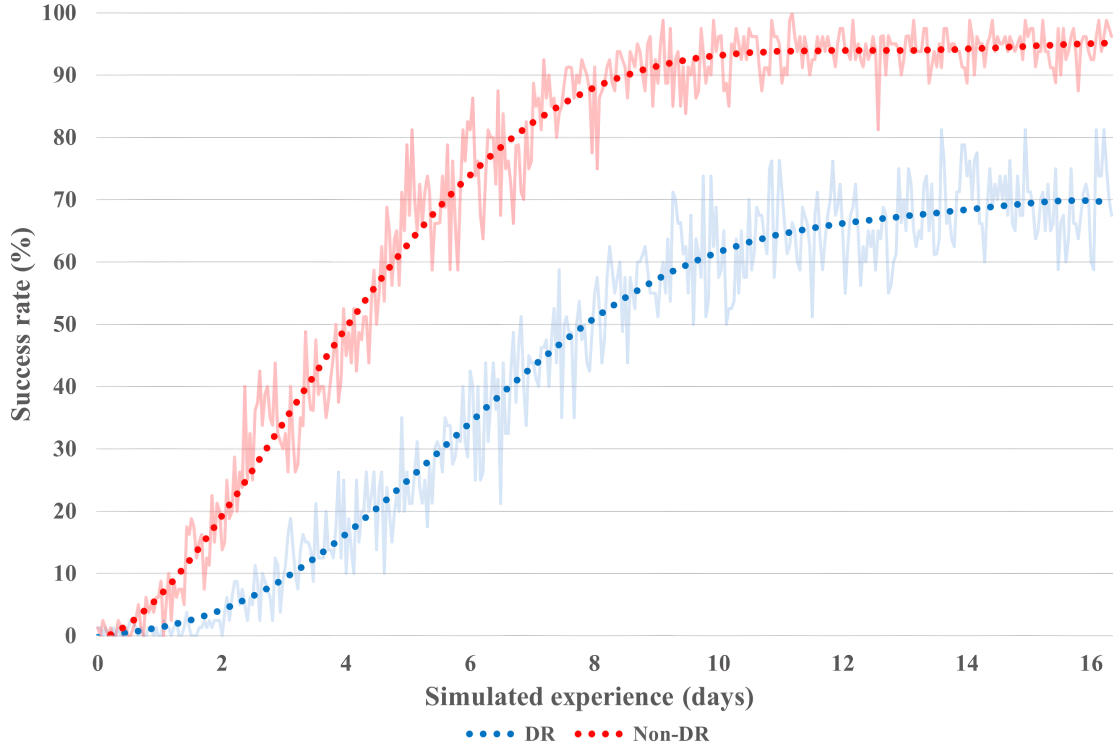


Figure 3.7: Comparison between agents trained from scratch with DR and without DR in the simulated environment of the TriFinger robot ‘Move Cube on Trajectory’ task.

Seed	Method	Simulation	Real robot
0	Scratch(DR)	-8808	-16685
	Scratch(NDR)	-5447	-11129
	Scratch(NDR) + Tune(DR)	-5825	-8922
123	Scratch(DR)	-9873	-15726
	Scratch(NDR)	-6376	-9417
	Scratch(NDR) + Tune(DR)	-6253	-7030
200	Scratch(DR)	-11779	-15398
	Scratch(NDR)	-7277	-15045
	Scratch(NDR) + Tune(DR)	-6147	-8535
Average	Scratch(DR)	-9913 $\pm$ 1866	-15936 $\pm$ 749
	Scratch(NDR)	-6367 $\pm$ 920	-11864 $\pm$ 3181
	Scratch(NDR) + Tune(DR)	-6075 $\pm$ 250	-8162 $\pm$ 1132

Table 3.2: Evaluation scores of different policies evaluated in simulation and on real robots; three seeds are used in this experiment. Scratch(DR) represents the policy trained from scratch using DR from the beginning of training. Scratch(NDR) is the policy trained from scratch without using DR; Scratch(NDR) + Tune(DR) is the policy firstly trained from scratch without DR and then subsequently tuned using DR in a second training phase. See Appendix A.1.3 for a more detailed view of the results. Scores are calculated using Equation 3.1.

3.6.2 Domain randomisation

To help the learned policy generalise from a potentially inaccurate simulation to the real environment, we used some simple domain randomisation (DR) techniques; our domain randomisation implementation is based on the benchmark code from the 2020 RRC [166]. This includes uniformly sampling, from a specified range, parameters of the physics simulation (e.g., robot mass, restitution, damping, friction; and cube properties (mass and width) for each episode. Note that in order to account for noisy real robot actuation and observations, uncorrelated noise is added to actions and observations during simulated episodes [167].

However, the use of DR from the beginning of the training hinders the performance of the learned policy (see Figure 3.7), as adding uncertainty (i.e., noise) to the observations of a non-expert agent simply makes the task more complicated. Despite this, DR is very useful for fine-tuning policies that are capable of solving the task in simulation, as can be seen in Table 3.2, where the agents that were fine-tuned after having learned from scratch without DR performed better on the real robot than those trained from scratch with and without DR.

Therefore, this policy was transferred to the real robot and our submission outperformed all competitors’ submissions and won this first phase of the challenge, without requiring any kind of additional domain adaptation or fine-tuning of the policy after deployment on real robot data.

It is important to note that when transferring our policies to the real robot, they could become stuck with a finger-tip pressing the cube into the wall. As a solution to this issue, we assumed the policy was stuck whenever the cube had not reached the goal’s xy -coordinates after 50 consecutive steps, and then uniformly sampled random actions were selected for 7 time-steps in an attempt to ‘free’ the policy from its stuck state by observing a new state.

3.6.3 Discussion of final results and comparison with other teams

While there were six different teams participating in the challenge, in this section only the methods used by the two of the more successful teams are summarised, and later compared with our method using the final scores of the challenge.

Madan *et al.* extended the Cartesian Position Control with triangulated grasp (CPC-TG) [168] approach from RRC 2020 [166], which used a solution based on position-guided motion primitives for the fingertips, by interpolating trajectories between the goal and cube positions for smoother motions. Instead, Yao *et al.* [169] used a mixed approach by making use of a RL technique to find the best contact points for the robot fingers and then used classical control techniques, as in [168].

The final scores of the challenge are shown in Table 3.3. Our method obtained the best score of all the methods presented on the *Move Cube on Trajectory* task and won the challenge⁴. Despite that, our method presented some drawbacks with respect to Madan *et al.*, as their grasp on the real robot seemed to be more secure, letting the cube fall fewer times. However, in the event of the cube falling, our method is much more versatile to adapt and can quickly recover, while when using Madan’s method the robot must re-start its loop and move the cube to an acceptable initial configuration for their algorithm to recover.

Moreover, by applying smooth motions as per Madan *et al.*, the robot’s actions are slower than when using our policy, as our model applied no restrictions on the action selection. Even though in theory this could translate into a slightly higher grasp security, the final results prove that a pure reinforcement learning approach is able to achieve grasp security while also performing the task faster, which leads to our model obtaining a higher final score in the challenge.

Furthermore, our method was faster to move the cube around the arena space, as the network is trained using a reward system that rewards finishing the task quickly, as for each time-step that the robot has not successfully solved the task it receives

⁴<https://real-robot-challenge.com/2021#nav-results>

Position	Method	Final Score
1	Ours	-11586
2	Madan <i>et al.</i> [168]	-14714
3	Yao <i>et al.</i> [169]	-29333

Table 3.3: Final scores of the 2021 Real Robot Challenge on the *Move Cube on Trajectory* task. Our method obtained the best score and won Phase 1 of the challenge. Scores are calculated using Equation 3.1.

a penalty.

3.7 Applications of our policy

3.7.1 Wang *et al.*’s extension using knowledge transfer

One of the uses of our policy has been to act as an expert teacher policy for exploration in order to solve tasks of even greater difficulty. Particularly, Wang *et al.* proposed a novel knowledge transfer (KT) technique where they used our learned policy on the *Move Cube on Trajectory* task as a starting point for both the critic and the actor networks of a new agent in order to train a DDPG agent in the *Move Cube on Trajectory Pro* task [144]. In this new advanced task, the cube must not only be moved to an arbitrary goal position but also aligned with a goal orientation.

Three different KT structures were explored:

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 initialise 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 initialised.

	ACTOR-CRITIC	ACTOR	COLLECT	SCRATCH	TEACHER
Average pos. dev _{dis} (mm)	23	66	31	134	24
Average ori. dev _{angle} (°)	75.8	98.6	84.9	142.2	126.2

Table 3.4: The average deviations between achieved and desired position/orientation over 15 evaluation episodes in the simulator; each episode lasts 120,000 time-steps. For the position: $\mathbf{dev}_{\text{dis}} = \|g'_{xyz} - g_{xyz}\|$, which can measure the distance between the actual position of the cube and the goal position. For the orientation: $\mathbf{dev}_{\text{angle}} = \|(R(g'_o))^{-1}R(g_o)\|$, which can measure the angle difference between the actual and target orientations. The deviations of all time-steps are accumulated and finally averaged.

The results of these experiments are shown in Table 3.4, which shows the average standard deviation for the position: $\mathbf{dev}_{\text{dis}} = \|g'_{xyz} - g_{xyz}\|$, which measures the distance between the actual position of the cube and the goal position; and for the orientation: $\mathbf{dev}_{\text{angle}} = \|(R(g'_o))^{-1}R(g_o)\|$, which measures the angle difference between the actual and target orientations. Table 3.4 shows that the **ACTOR-CRITIC** configuration obtains the lower position and orientation deviation, and improves the results obtained by the RRC 2020 edition winners [170]. In Table 3.4, apart from the KT configurations explained above, there are two other learning strategies used for comparison: **SCRATCH** and **TEACHER**. The **SCRATCH** strategy is the standard DDPG + HER without making use of any kind of KT, while the **TEACHER** strategy shows the score obtained in the ‘Move Cube on Trajectory Pro’ task by the policy trained on the simpler ‘Move Cube on Trajectory’ task (i.e., the final model we submitted to the challenge).

3.7.2 Other potential applications

Up until now, there has been one main application of our policy and our method. However, there are many different potential applications where this policy could be used effectively in order to solve dexterous robotic manipulation tasks apart from using the policy learned on the ‘Move Cube on Trajectory’ task employing a KT technique.

The most obvious application of our method is its use to train any kind of robotic manipulator that has the aim of lifting an object. By adding an intrinsic motivation

in the form of a reward, we presented a method that is very easily adaptable to any RL-based environment and that could fit into many different existing works [171]–[173], possibly improving their performance and/or their sample efficiency.

One of the main weaknesses of our algorithm was that the behaviour of the policy learned using the simulated environment could not be fully replicated when deployed on the real robot. As such, another alternative could include the creation of a trajectories dataset for the ‘*Move Cube on Trajectory*’ task using our policy, so that this behaviour could be improved by offline policy optimisation [174], [175], as similarly done in the 2022 edition of the RRC, where they created a dataset of expert and mid-expert policies (i.e., policies that are capable of solving the proposed tasks with different levels of expertise) on different tasks using the Proximal Policy Optimization (PPO) algorithm [176], and participants were asked to replicate or even improve the behaviour of the policy by analysing the dataset [177].

3.8 Conclusion

In this chapter, a novel reward system that combines dense and sparse components has been proposed and evaluated in the context of the Real Robot Challenge (RRC) 2021, where it sets a new state of the art performance on the ‘*Move Cube on Trajectory*’ task. Unlike the other solutions submitted to the challenge by other teams [168]–[170], both in the 2020 and the 2021 edition of the competition, our model makes use of minimal domain-specific knowledge, employing instead a fully end-to-end RL approach.

Our results prove that dividing the reward into a sparse component for the xy -coordinate space and a dense component for the z coordinate that encourages a lifting behaviour improves the sample performance of a DDPG-based agent trained in conjunction with HER on a cube manipulation task, being able to learn to lift the cube to the correct position faster than using the standard sparse reward setting. This happens because when training an RL-based agent from scratch using sparse rewards, the first interactions of the agent will typically be failed exploratory actions,

where the cube stays on the arena floor, therefore, creating no hindsight input for goals that require the robot to lift the cube. With our method, because it gets greater penalisation when the cube is below the target position, the lifting behaviour is learned earlier and, consequently, more hindsight goals with $z > 0$ are generated by HER.

Different manipulation strategies were explored, with the pinching policy being the most efficient, by making use of force feedback obtained from tactile sensors embedded in the finger tips of the TriFinger robot. This policy was later fine-tuned in simulation with DR, for each episode slightly altering the parameters of the physics simulation and the cube properties. Due to the large domain gap, our excellent performances in simulation were not fully realised 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 further 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 [178], [179] could produce a policy more capable of adapting to the real environment than our domain randomisation approach.

This policy was also used by Wang *et al.* in their KT technique, managing to solve the more advanced ‘*Move Cube on Trajectory Pro*’ task of the RRC while avoiding the blind exploration during early exploratory interactions (i.e., avoiding actions that do not lead to useful learning experiences or repeatedly re-learning similar skills) by initialising their actor and critic networks with our policy’s weights [144].

Despite this, the total number of environment interactions that a DDPG agent trained with HER needs in order to achieve a good performance in complex environments is very large, regardless of the reward system it is using; between 7-10 days of simulated experience. This is likely to be a consequence of not adding prior knowledge about the domain or the task in hand, and a major drawback of any end-to-end RL-based solution that deals with complex environments; that is, en-

vironments with large observation/action spaces or goals that are inherently very difficult to achieve).

In the next chapters, we abandon the idea of end-to-end RL with minimal or no domain knowledge by making the agent learn simple tasks (i.e., primitive skills or behaviours) that can be learned quickly (both in terms of sample and time efficiency) and then later executed sequentially to solve multiple different robotic manipulation tasks of greater difficulty. In other words, we move from model-free off-policy RL approaches to model-based off-policy RL.

Furthermore, we move from the RRC environment, the use of which depends on the availability of the robots (e.g., as soon as the competition finished, we could no longer run additional experiments) and the availability of the organisers from the Max Planck Institute for Intelligent Systems (if algorithms were to be tested on the real robot, they had to set-up these experiments on our behalf). Instead, we start using an open-source ‘gym’ simulated environment designed by OpenAI, as it is widely used by the research community, meaning most of the newest models that exist are evaluated on this environment. This also means that the results obtained can be more easily compared to other existing RL methods.

3.9 Summary

3.9.1 Revisiting the hypothesis

In Chapter 1, the following hypothesis was stated:

- **Hypothesis 1:** *By carefully designing a reward that encourages a particular desired behaviour, standard RL algorithms can learn more efficiently to solve robotic manipulation tasks that involve the lifting of an object, leading to more sample-efficient methods and, thus, to lower training times.*

As can be seen throughout this chapter, this was proved to be the case. In other words, **Hypothesis 1 has been supported by the results**, at least in the context

where it has been evaluated, as it reaches 80% of success rate using 2 fewer days of simulated experience as has been seen in Section 3.5. Considering the results of our experiments in this chapter, it is easy to appreciate that our reward system started learning faster in the initial interactions with the environment, meaning it learned a lifting behaviour more rapidly than the other reward systems evaluated. Despite this, our results also showed that standard off-policy RL algorithms are resource-intensive, requiring days of exploration by interacting with the environment. In other words, the real bottleneck for sample efficiency seems to be making efficient use of exploration, rather than designing a thoroughly engineered reward.

3.9.2 Summary of chapter contributions

The contributions to the field made in this chapter can be summarised as:

1. **A novel reward system is presented that combines both sparse and dense components in order to encourage the learning of a lifting behaviour for a robotic manipulator.** This algorithm can be used with any off-policy RL technique and defines the new state-of-the-art performance on the ‘*Move Cube on Trajectory*’ task for the TriFinger robot.
2. **We developed three different cube manipulation strategies** that can be used with the TriFinger robot, **by making use of force feedback** obtained from tactile sensors allocated on the finger tips of the robot.

Chapter 4

Baseline model selection

4.1 Chapter overview

The aim of this chapter is to justify the change of environment for the remainder of this thesis, moving from the RRC environment to the OpenAI Gym [22], as well as to identify baseline models that can operate in two of OpenAI Gym environments: the Fetch environment [154] and the Shadow Dexterous Hand environment [180]. As such, this chapter does not present any novel contribution, rather it serves the purpose of establishing the basis for experimentation for the rest of the thesis, as the conclusions of this chapter are repeatedly used when comparing the different methods proposed throughout this thesis. In particular, Section 4.2 explains the motivations and justifies the change of environment, while Section 4.3 explains the above-mentioned environments. Section 4.4 describes the experiments used for algorithm selection, and finally, the conclusions are presented in Section 4.5.

4.2 Motivation

A key aspect in research is reproducibility and comparability of the algorithms designed [181]. Reproducibility refers to whether the results obtained in a research study can be independently verified and validated by other researchers. Providing detailed documentation of the experimental setup, algorithms, and data used,

enables others to replicate experiments and confirm the validity of findings, which promotes transparency and trust within the scientific community, allowing for the advancement of knowledge and the building of further research upon existing work.

Comparability allows researchers to evaluate and compare different algorithms in a more standardised manner. By using common benchmarks, datasets, or evaluation metrics, researchers can objectively evaluate the performance of their methods and compare them with existing approaches. This facilitates the identification of strengths, weaknesses, and areas for improvement, leading to more rapid advancements in the field.

Moreover, the reproducibility and comparability principles foster collaboration and knowledge sharing amongst researchers in the field [182]. When works can be easily reproduced and compared, it becomes more straightforward for researchers to build upon each other’s work, validate or refine existing methods, and collectively address challenges in the field. This collaborative environment accelerates progress and promotes the development of more robust and effective solutions.

Because of this, at the end of 2021 OpenAI decided to acquire (and open-source from the beginning of 2022) a very popular physics engine that provides accurate and efficient simulation of robotic systems, named Mujoco [183]. The acquisition and open-sourcing of Mujoco by OpenAI was very significant for the robotics research community, and in particular for the development of this thesis, for several reasons.

Firstly, by acquiring Mujoco, OpenAI gained control over this powerful tool, ensuring its availability and development for the research community and its full integration with their OpenAI Gym framework, which contains several different robotic environments and tasks (including non-manipulation tasks) [22].

Secondly, this open-sourcing enlarged the community of researchers working with that environment, meaning that by using the OpenAI gym, we can benefit from the collective knowledge and expertise of the community, as well as gain access to a wide range of resources, tutorials, and support. Not only that, but most RL-based algorithms for robotic manipulation are implemented in these environment tasks,

making the reproducibility and comparability in robotic research more straightforward, as the community now has a common benchmark framework by using the OpenAI Gym environment tasks.

Finally, and probably the most important factor in the context of this thesis, due to the type of hardware set-up available (i.e., Dockerised machines¹), since Mujoco was open-sourced it is compatible with virtualised environments, like the Docker machines used in our research group, meaning we could integrate this technology easily into our computing set-up.

For all these reasons, we decided to stop using the TriFinger simulator from the RRC, as it is not a widely used robotic system by the robotics research community. As such, the comparison of robotic manipulation algorithms in that environment is more difficult, as it requires not only training the different existing methods we would like to evaluate in the TriFinger environment, but also adapting these methods to the environment structure, which does not fully follow the OpenAI Gym standards. Instead, by using the environments found on the OpenAI Gym, we can more easily evaluate different existing algorithms from the literature, as well as compare our proposed algorithms against them.

In the following sections, two environments of the OpenAI gym are introduced along with the different tasks that we explore, and then different experiments are run in order to find the most sample and time efficient method for each of those tasks. The results obtained in this section are used throughout the remainder of the thesis as baselines in order to benchmark the results of our newly proposed algorithms. Furthermore, an analysis is provided on which models could serve as primitive behaviours that could be used to more efficiently solve other more complex tasks, as already introduced in Section 3.8.

¹<https://www.docker.com/>

4.3 Environments and tasks

4.3.1 Fetch environment

In this chapter, we explore some of the block manipulation tasks proposed by OpenAI for the Fetch robotics arm [154], which is a 7-degrees of freedom robotic arm that uses a parallel-jaw gripper, as well as two more tasks proposed by Bing *et al.* [117]. These are summarised as follows, in order of increasing complexity:

- *FetchReach*: the goal is to move the parallel-jaw gripper to a position in the 3D space. The gripper remains closed.
- *FetchPush*: the goal is to move a block to a point in the 3D space that is over the table. The gripper remains closed.
- *FetchPickAndPlace*: the goal is to move a block to a point in the 3D space. Gripper controls are activated.
- *FetchPickObstacle*: the goal is to move a block to a point in the 3D space with an obstacle between the initial and the target position. Gripper controls are activated.
- *FetchPickAndThrow*: the goal is to lift a cube and throw it into one of eight boxes that are out of reach for the robotic arm. Gripper controls are activated.

Episode structure

Episodes in this environment have a duration of 50 time-steps (i.e., the robot must solve the different tasks using at the most 50 actions). For all tasks, episodes always start using the same robotic arm configuration, placing the gripper in the following Cartesian coordinates $(x, y, z) = [1.3419, 0.7491, 0.555]$ meters, while its orientation in quaternions is set to be $(w, x, y, z) = [1.0, 0.0, 1.0, 0.0]$, the only exception being the *FetchPickObstacle* task, which places the gripper at a slightly higher position ($z = 0.655$ m) in order to avoid colliding with the obstacle, which is placed in the

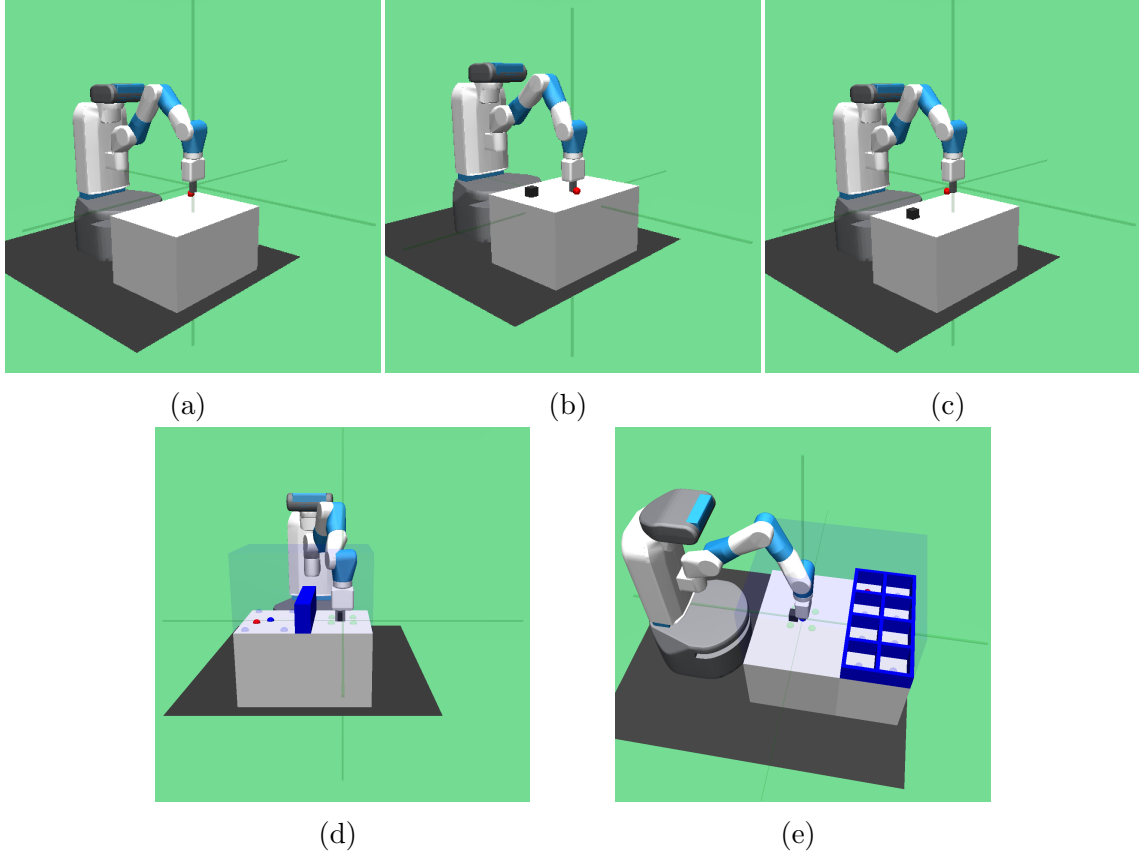


Figure 4.1: Visual representation of (a) *FetchReach*, (b) *FetchPush*, (c) *FetchPickAndPlace*, (d) *FetchPickObstacle* and (e) *FetchPickAndThrow* tasks.

middle of the table. The rest of the robot joints are set by inverse kinematics, and the robot’s base is fixed throughout the whole episode at $(x, y, z) = [0.405, 0.48, 0]$ m.

In the case where the task at hand involves the manipulation of a block (i.e., all tasks except the *FetchReach* task), said block has an initial height of $z = 0.42$ m in order to ensure the cube is on top of the working table (the table also has a height of 0.42 m).

For the *FetchPush* and *FetchPickAndPlace* tasks, the initial x and y coordinates of the block are obtained by sampling from a uniform distribution with a range of $[-0.15, 0.15]$ m and adding the value as an offset to the initial gripper configuration. In order to prevent episodes with very simple initial configurations, random xy couples are sampled until the 2-dimensional Euclidean distance from the gripper to the block is greater than 0.1 m. Likewise, goal positions are sampled using uniform

distributions, adding the sampled value as an offset to the initial coordinates of the gripper configuration. In the case of the xy -plane, using a uniform distribution with a range of $[-0.15, 0.15]$ m for each coordinate, setting the z -coordinate to be the same as the table for the *FetchPush* task ($z = 0.42$ m). For the *FetchPickAndPlace* and *FetchReach* tasks, as goals can be in the air, it generates the goal of the z -coordinate by adding an offset obtained by sampling from a uniform distribution with a range of $[0, 0.45]$ m to the z position where the block is initially located ($z = 0.42$ m).

The initial block position and target goals are set differently for the *FetchPickObstacle* and *FetchPickAndThrow* tasks, both with respect to each other and to the rest of the tasks. This is because the presence of obstacles implies a set of constraints for the block's initial configuration. For instance, for the *FetchPickObstacle* task it would make no sense to have both the block and the goal on the same side of the wall, as essentially there would be no obstacle for the robot to deal with, or for the *FetchPickAndThrow* task it would make no sense to sample a goal that is outside of the region of the boxes, as otherwise there would be no need to throw the block.

For the *FetchPickObstacle* task, the obstacle is placed in the middle of the table at the following Cartesian coordinates $(x, y, z) = [1.3419, 0.7491, 0.42]$ m. The obstacle is 0.4 m in length across the x -axis, 0.1 m wide across the y -axis, and 0.2 m in height. Because this obstacle configuration creates a collision with the initial end-effector configuration used for the other tasks, the gripper is instead initialised at a slightly higher position ($z = 0.62$ m). The y -coordinates for the goal and initial positions of the block are generated by sampling from a uniform distribution over the range $[0.05, 0.15]$ m and adding this value to the initial obstacle y -position. In the case of the generation of goals, the sampled value is multiplied by -1 in order to translate the goal toward the other side of the obstacle. The x -coordinates for the goal and initial position are generated as in the other tasks: using uniform distributions and adding the sampled value as an offset to the initial coordinates gripper configuration with a range of $[-0.15, 0.15]$ m.

Environment	<i>FetchReach</i>	<i>FetchPush</i>	<i>FetchPickAndPlace</i>	<i>FetchPickObstacle</i>	<i>FetchPickAndThrow</i>
Gripper position	✓	✓	✓	✓	✓
Block position	✗	✓	✓	✓	✓
Rel. block position	✗	✓	✓	✓	✓
Gripper joint displ.	✓	✓	✓	✓	✓
Rel. block linear vel	✗	✓	✓	✓	✓
Block angular vel.	✗	✓	✓	✓	✓
Gripper linear vel.	✓	✓	✓	✓	✓
Gripper fingers linear vel.	✓	✓	✓	✓	✓
Obstacles	✗	✗	✗	✓	✓
Goal space	Continuous	Continuous	Continuous	Continuous	Discrete

Table 4.1: Overview of the information contained in the observation space of the different tasks of the Fetch environment.

The *FetchPickAndThrow* task operates on a larger table so that the boxes can be placed just out of reach of the robotic arm. Each box has a size of $0.15 \times 0.15 \times 0.2$ m³, and is placed in two rows (i.e., 4 boxes per each row), the first row of boxes starting at $x = 1.725$ m and the second row starting at $x = 1.875$ m. In this environment, each box is assigned an index (e.g., block in the right of the first row = 1) and one index is randomly selected using a discrete uniform distribution. The initial block position is very similar to the other simpler tasks, but in this case, reducing the range of the uniform distribution from which the offsets are sampled. Instead, it uses a uniform distribution with a range of $[-0.06, 0.06]$ m in order to avoid having the block very close to the margins of the table at the beginning of an episode.

Action space

The Fetch environment makes use of Cartesian control, adding an extra dimension in order to control the parallel-jaw gripper. This means that the action space is 4-dimensional, the first three elements stating the displacement of the end-effector in the x , y , and z directions respectively, and the last one controlling the opening and closing of the parallel-jaw gripper.

Observation space

The observation space is composed differently depending on the nature of the task: tasks that do not interact with objects in the environment have a 10-dimensional ob-

servation space, while tasks that have to manipulate the block have 22-dimensional observation spaces. In fact, because we are dealing with goal-based tasks, both the achieved object/gripper position/orientation and the desired object/gripper position/orientation are also included in the observation, meaning that the spaces end up being 16-dimensional and 28-dimensional.

In particular, the observation space includes the following information about the kinematics of the block object and the end-effector (see Table 6.1 for a summary):

- End-effector position in global coordinates (x, y, z) .
- Block position in global coordinates (x, y, z) . This information is not included nor relevant for the *FetchReach* task.
- Relative block position with respect to the end-effector $(x_{block} - x_{gripper}, y_{block} - y_{gripper}, z_{block} - z_{gripper})$. This information is also not included for the *FetchReach* task.
- Joint displacements of the right and left finger gripper.
- Relative block linear velocity for each axis with respect to the gripper. This information is not included for the *FetchReach* task.
- Block angular velocity along each axis of the world frame. This information is not included for the *FetchReach* task.
- End-effector linear velocity along each axis of the world frame.
- Right and left gripper finger linear velocities.
- Target goal block position (end-effector position in the case of the *FetchReach* task).
- Current block position (end-effector position in the case of the *FetchReach* task).

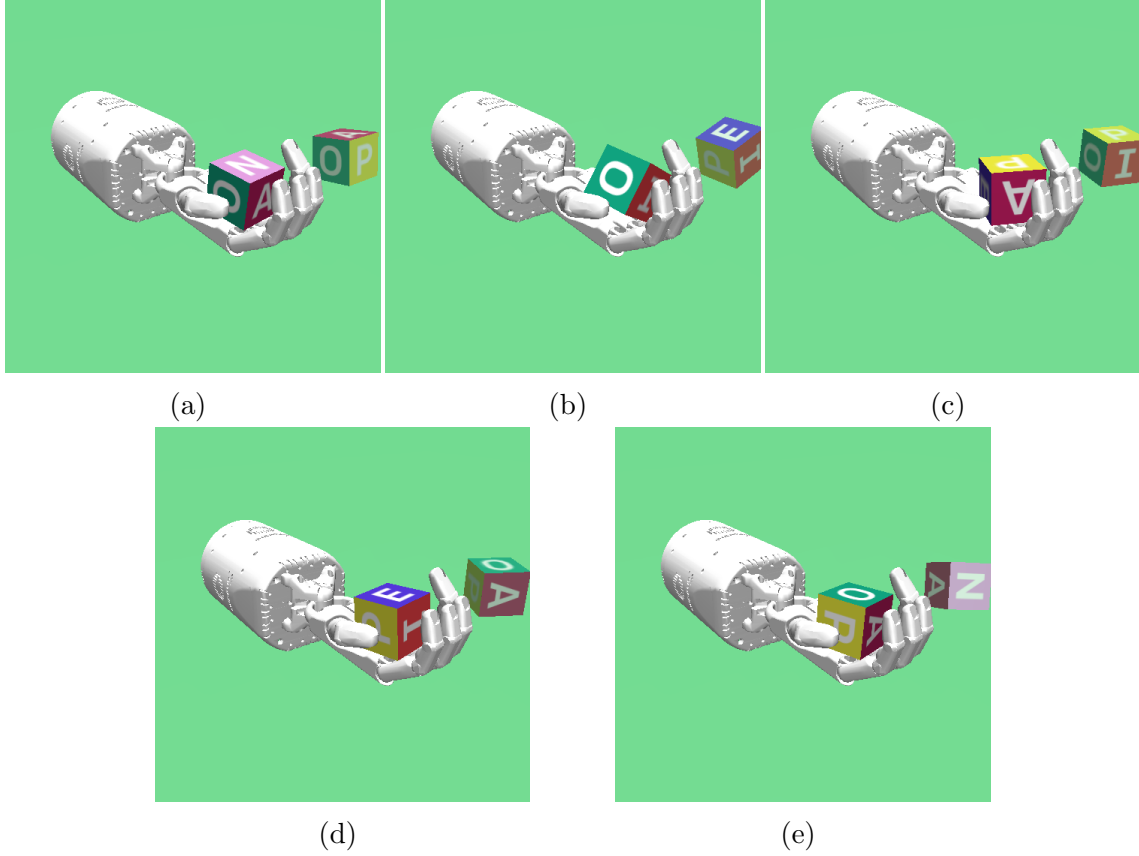


Figure 4.2: Visual representation of (a) *RotateX*, whose goals require to rotate the cube around the x -axis, (b) *RotateY*, whose goals require to rotate the cube around the y -axis (c) *RotateZ*, whose goals require to rotate the cube around the z -axis (d) *RotateParallel*, whose goals consist of a random target rotation around the z -axis and rotations by multiples of $\frac{\pi}{2}$ around the x and y axes and (e) *RotateXYZ* tasks, whose goals consist of a random target 3D rotation.

4.3.2 Shadow Dexterous Hand environment

The in-hand manipulation tasks of the OpenAI gym are based on the Shadow Dexterous Hand, an anthropomorphic robotic hand that has 24 hinge joints, but only 20 of them can act independently, meaning the remaining 4 are coupled joints that depend on the motion of the other joints [180]. In this chapter, we explore three different cube manipulation tasks of increasing complexity that are included in this environment:

- *RotateZ*: goal consists of a random target rotation around the z -axis of the reference frame and position is ignored.
- *RotateParallel*: goal consists of a random target rotation around the z -axis

and rotations by multiples of $\frac{\pi}{2}$ around the x and y axes, with position being ignored. In other words, any side of the cube can end up flat on the hand, but with an arbitrary pose around the z -axis.

- *RotateXYZ*: goal consists of a random target 3D rotation with the target position ignored.

In order to learn rotations around the x and y axes, two different new tasks based on the *RotateZ* task were implemented for the hand environment, following OpenAI Gym’s guidelines²:

- *RotateX*: goal consists of a random target rotation around the x -axis of the reference world frame and again position is ignored.
- *RotateY*: goal consists of a random target rotation around the y -axis of the reference frame, with the block’s position ignored;

Episode structure

Episodes for all tasks in this environment are set to have a length of 100 time-steps; meaning the robot can take at most 100 actions in order to solve the task. For all tasks, the robotic hand starts with the same joint initial configuration, setting those joints to their origin position with zero displacement. The initial cube position is fixed for all tasks as well, this being at the following Cartesian coordinates $(x, y, z) = (1, 0.87, 0.2)$ m, which corresponds to the centre point of the hand, ensuring that this way the block starts at a static position and resting on top of the palm of the hand. The initial position of the cube is not randomised, because the tasks explored in this environment do not take into account the positioning of the cube, only the orientation.

To generate the initial orientation of the cube, it is firstly initialised to a reference orientation $(w, x, y, z) = (1, 0, 0, 0)$ (in quaternion space), and an orientation offset that is sampled from a uniform distribution with a range of $[-\pi, \pi]$ radians is

²New implemented tasks can be found at https://github.com/franroldans/custom_gym

added to an axis (or various axes, depending on the task) by multiplying the initial orientation with the sampled offset expressed as a quaternion.

In the case of the *RotateParallel* task, these offsets are selected from the set $\{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\}$ radians for the x and y axes, while for the z -axis it is selected from a uniform distribution with a range of $[-\pi, \pi]$ radians. The simple *RotateX*, *RotateY*, and *RotateZ* tasks follow the same strategy to initialise the cube orientation as the *RotateParallel* task, because this ensures the cube is in different configurations (i.e., the different faces of the cube are facing different directions) while still ensuring that the cube is beginning the episode in a stable position with no tilted cube configurations. However, for the *RotateXYZ* task, offsets for all axes are selected from a uniform distribution with a range of $[-\pi, \pi]$ radians.

A similar procedure is carried out in order to sample the target orientation goals, but adding the sampled orientation offsets to the initial cube orientation instead of adding it to the reference orientation $(w, x, y, z) = (1, 0, 0, 0)$. In order to generate the goals for the *RotateX*, *RotateY*, and *RotateZ* tasks, a rotation is sampled from a uniform distribution with a range of $[-\pi, \pi]$ radians and executed only around the x , y and z axes, respectively (i.e., the *RotateX* only applies the sampled rotation around the x -axis to the initial orientation, the y and z remain the same as in the initial cube orientation configuration). For the *RotateParallel* task, an offset for the x and y axes is selected from the set those offsets are selected from the set $\{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\}$ radians and a random rotation sampled from a uniform distribution with a range of $[-\pi, \pi]$ radians is applied around the x -axis. For the *RotateXYZ*, a random rotation around all axes is selected from a uniform distribution with a range of $[-\pi, \pi]$ radians.

Action space

The action space is 20-dimensional and continuous, which corresponds to each of the joints that can execute actions independently from the other joints. The control actions are set to be absolute angular positions of the different independent joints. More specifically, the action space controls the following (for a better understanding

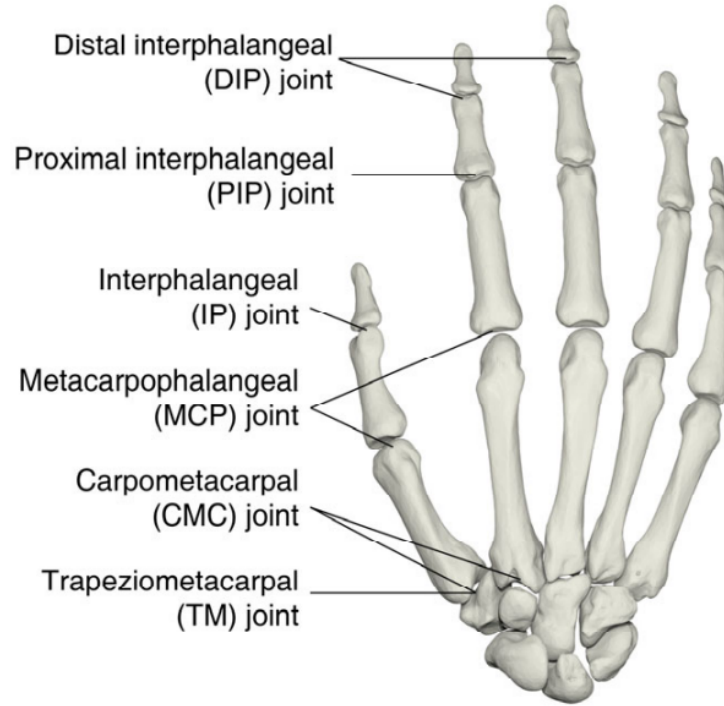


Figure 4.3: Image describing the different joints of the human hand. Image taken from Bullock *et al.* [184].

of the nomenclature of the hand joints, it is recommended to use Figure 4.3, which is obtained from Bullock *et al.* [184]):

- **Wrist:** Angular positions of the wrist describing the radial deviation (in the horizontal plane) and the flexion/extension (in the vertical plane) of the wrist.
- **Forefinger:** Horizontal angular position of the MCP joint, vertical angular position of the MCP joint, and angular position of the PIP joint.
- **Middle finger:** Horizontal angular position of the MCP joint, vertical angular position of the MCP joint, and angular position of the PIP joint.
- **Ring finger:** Horizontal angular position of the MCP joint, vertical angular position of the MCP joint, and angular position of the PIP joint.
- **Little finger:** Angular position of the PIP joint, angular position of the CMC joint, horizontal angular position of the MCP joint, vertical angular position of the MCP joint, and angular position of the PIP joint.

- **Thumb finger:** Horizontal angular position of the CMC joint, vertical Angular position of the CMC joint, horizontal angular position of the MCP joint, vertical angular position of the MCP joint, and angular position of the IP joint.

Observation space

Observations in this environment are 61-dimensional, but because these tasks are goal-based, information about the desired and the achieved goal is concatenated to the observation, meaning it ends up being 69-dimensional (orientations are expressed as quaternions). Observations in all of these tasks include positions and velocities of the robot joints, as well as information about the object being manipulated. In particular, the following information is included in the environment observations of all tasks:

- **Wrist:** Angular position and velocity of the horizontal and vertical joints.
- **Forefinger:** Horizontal and vertical angular position of the MCP joint, angular position and velocity of the PIP joint, and angular position and velocity of the DIP joint.
- **Middle finger:** Horizontal and vertical angular position and velocity of the MCP joint, angular position and velocity of the PIP joint, and angular position and velocity of the DIP joint.
- **Ring finger:** Horizontal and vertical angular position and velocity of the MCP joint, angular position and velocity of the PIP joint, and angular position and velocity of the DIP joint.
- **Little finger:** Angular position and velocity of the CMC joint, horizontal and vertical angular position and velocity of the MCP joint, angular position and velocity of the PIP joint, and angular position and velocity of the DIP joint

- **Thumb finger:** Horizontal and vertical angular position and velocity of the CMC joint, horizontal and vertical angular position and velocity of the MCP joint, and angular position and velocity of the IP joint.
- **Cube:** Linear velocity in x, y, z directions, angular velocity in x, y, z axes, position coordinates (x, y, z) , and orientation expressed in quaternions (w, x, y, z) .
- **Desired Goal:** Target orientation expressed in quaternions (w, x, y, z) .

4.4 Identifying the baseline models

The aim of this section is to evaluate in these environments the most sample and time efficient methods of the literature. This is done in order to find the best existing method from the state-of-the-art in solving the tasks previously explained, which is used in later chapters as the basis for comparison with our proposed solutions. This means that methods that might find a more expert policy in the long run, but at the cost of time or sample efficiency, might be discarded if there is another method that achieves to solve the task faster, even if it is marginally less reliable.

4.4.1 Reward systems

For both environments and all tasks, we use a sparse reward setting. At each time-step, the agent gets a reward of 0 if the task is completed, and -1 otherwise, meaning the total episode reward ranges from $[0, 50]$ or $[0, 100]$ depending on the episode duration set for the task. We decide not to use the reward system designed in Chapter 3 to encourage a lifting behaviour because it only makes sense for one task whose goals can be in the air (i.e., the *FetchPickAndPlace* task), which is simple enough to be solved using a sparse reward setting obtaining perfect or nearly perfect performance in terms of success rate with few environment interactions (see Subsection 4.4.2 for more information on this). Therefore, for the sake of simplicity, we use a sparse reward setting for all tasks.

Fetch environment

In order to determine whether the block (or end-effector in the case of the *FetchReach* task) is in the intended position, an Euclidean norm (also called the L2 norm) between the goal and the achieved block position (or end-effector) is computed, considering the task to be successfully solved when such norm is lower than 0.05 m, as this is considered a small enough distance, as expressed in the following equation:

$$r_\theta = \begin{cases} 0 & \text{if } \|g_{goal} - g_{block}\| \leq 0.05 \text{ m} \\ -1 & \text{otherwise} \end{cases} \quad (4.1)$$

where $\|g_{goal} - g_{block}\|$ is the Euclidean norm between the goal position for the episode g_{goal} , and the current achieved position of the block g_{block} .

Shadow Dexterous Hand environment

In order to determine whether a rotation has been completed successfully, a quaternion difference between the desired and the achieved poses is computed and the task is considered to be solved successfully when this difference is less than 0.1 radians ($\approx 5.73^\circ$), as we consider this a small enough orientation difference, as expressed in the following equation:

$$r_\theta = \begin{cases} 0 & \text{if } \arccos(2\langle q_1, q_2 \rangle^2 - 1) \leq 0.1 \text{ rad} \\ -1 & \text{otherwise} \end{cases} \quad (4.2)$$

where $\langle q_1, q_2 \rangle$ represents the inner product of quaternions q_1 and q_2 .

4.4.2 Policy learning algorithm selection

We begin the experimentation by making use of the same network that we used for Chapter 3 as a baseline model, using the same hyperparameter configuration (see Figure 3.4 and Appendix A.1.2). However, as can be seen throughout this subsection, simple DDPG + HER is not always sufficient to learn a policy that can solve

the tasks described in Section 4.3. As such, other variations of the HER algorithm are also evaluated, both in terms of performance and sample efficiency, in order to choose the most appropriate algorithm for each of the tasks explored. In particular, for those tasks where HER was not able to learn a policy capable of solving the task, Hindsight Goal Generation (HGG) [115] and Graph-based Hindsight Goal Generation (G-HGG) [117] are also explored and evaluated to become the baselines models. Models are trained using an Intel Core i7-6850K CPU @ 3.60GHz and no data generation parallelisation in order to simulate a worse-case scenario in terms of the availability of hardware resources.

Fetch environment experiments

The goal of this section is to identify the best baseline model that will be used to train the low-level primitive behaviours that will later be sequentially used to solve other more complex tasks. Because HGG and G-HGG add computational complexity, due to the extra data processing that is required in order to properly use these methods, for all those tasks that DDPG + HER is capable of successfully solving we keep DDPG + HER as baseline as it is the simplest and the most computational efficient option (i.e., the fastest to be trained).

Therefore, we first train a DDPG + HER agent on the most simple tasks of the Fetch environment (i.e., the *FetchReach*, *FetchPush*, and *FetchPickAndPlace* tasks) in order to evaluate if the use of a more advanced variation of HER is necessary for those tasks. As expected, for those simple tasks the agent was able to learn a policy capable of successfully solving any kind of goal, as can be seen in Figures 4.4, 4.5 and 4.6. Figure 4.6 also includes the result of HER using the reward setting proposed in Chapter 3, as the *FetchPickAndPlace* task is the only task that benefits from this reward because the goal position of the manipulated object is on the air, contrary to the rest of tasks which is on ground.

In particular, for the dummy task *FetchReach*, thanks to the strengths of the HER algorithm in altering the goals of saved episodes to match the achieved goal,

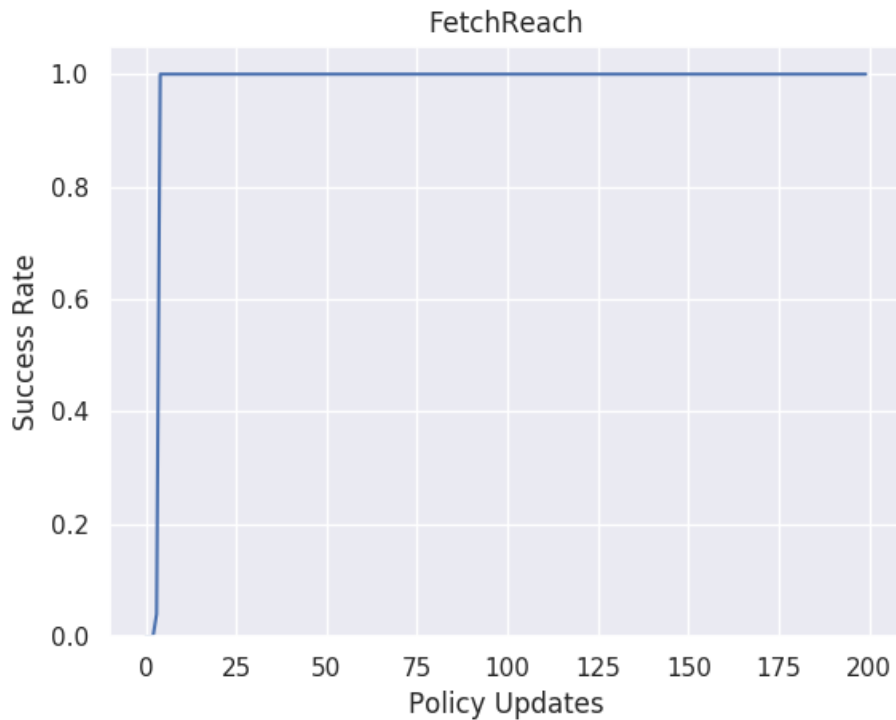


Figure 4.4: Success rate evolution over policy updates of a DDPG + HER agent trained on the *FetchReach* task of the Fetch environment.

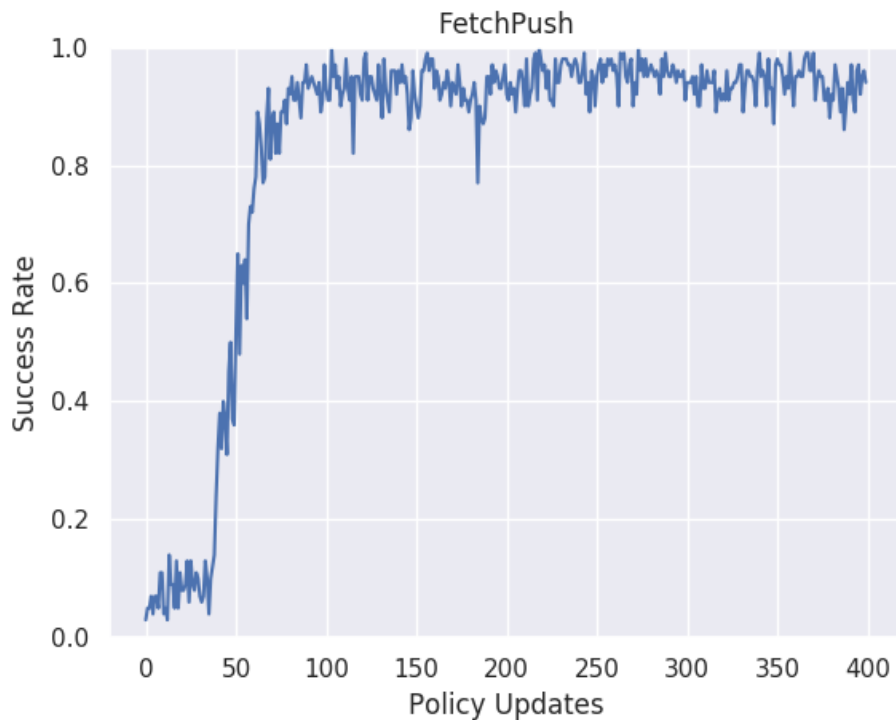


Figure 4.5: Success rate evolution over policy updates of a DDPG + HER agent trained on the *FetchPush* task of the Fetch environment.

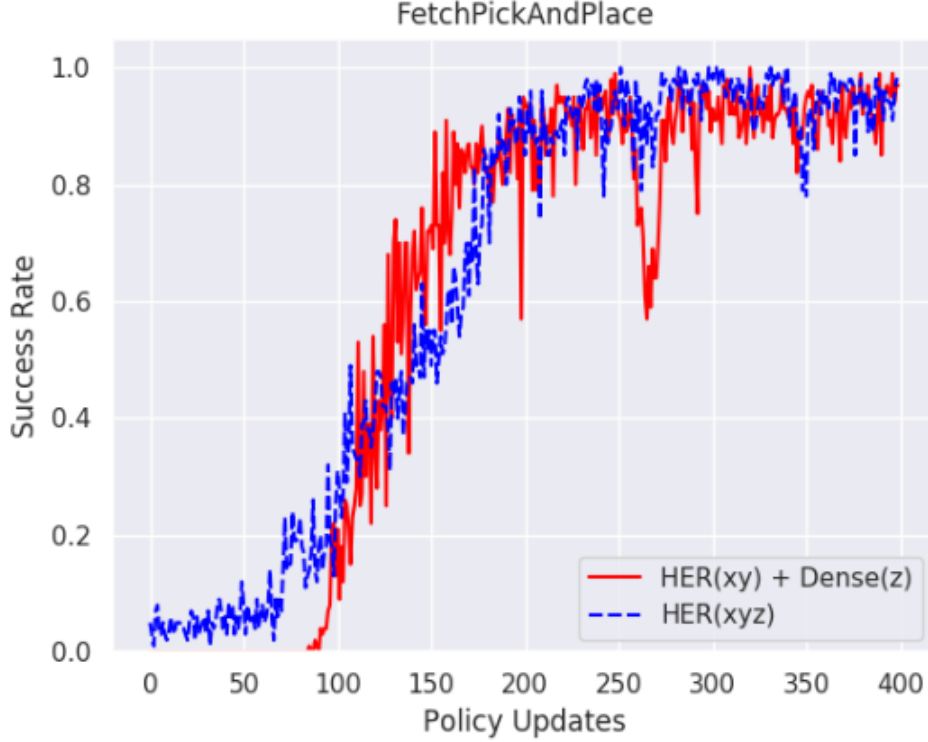


Figure 4.6: Success rate evolution over policy updates of a DDPG + HER agent trained on the *FetchPickAndPlace* task of the Fetch environment.

it manages to reach a perfect score in 5 only policy updates. On the other hand, for the *FetchPush* and *FetchPickAndPlace* tasks, the algorithm required a more extensive exploration in order to find a policy that could perform the manipulation objective, taking around 70 policy updates to get to an average of 90% success rate in the case of the *FetchPush* task, and around 180 updates in the case of the *FetchPickAndPlace* task, regardless of the reward system used. Because there is a very minimal difference in performance between both rewards in this environment, for the rest of the thesis we keep using sparse rewards because they are easier to interpret.

Furthermore, the training time for those tasks was also very low: the average time per policy update of the agent trained to solve the *FetchReach* task is 25.47 seconds while on the *FetchPush* and *FetchPickAndPlace* tasks it is 29.23 seconds, meaning it is possible to train all of these tasks in less than two days. Note that this is more time than the simple addition of $25.47 \text{ sec} \times 200 + 29.23 \text{ sec} \times 400 \times 2$. This

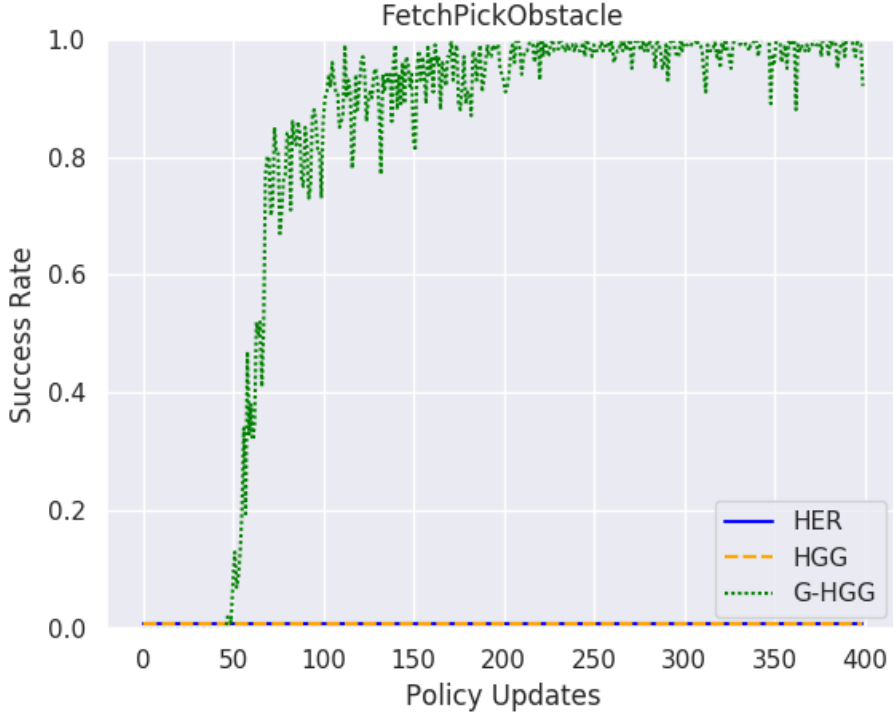


Figure 4.7: Success rate evolution over policy updates of the different agents trained on the *FetchPickObstacle* task of the Fetch environment.

is because besides the training time per policy update, the time needed to generate the episodes in order to evaluate the policy (i.e., to calculate the success rate) during training must also be included in the sum. This makes these policies suitable to be used as primitive behaviours in order to sequentially execute more complex tasks, as these policies represent basic manipulation skills which are easy to learn quickly.

However, when learning in environments where there are obstacles that must be avoided (i.e., the *FetchPickObstacle* and *FetchPickAndThrow* tasks), HER completely fails to learn any useful policy, as can be seen in Figures 4.7 and 4.8. Nonetheless, it is not the only algorithm that fails in these tasks, as HGG is also unable to learn anything useful.

Instead, it was necessary to use a G-HGG-based agent, which replaces the Euclidean distance metric typically used to evaluate goal-based tasks with a graph-based distance metric that adapts to the task at hand, making the hard task of avoiding an obstacle easier for the agent.

In particular, this DDPG + G-HGG agent manages to successfully solve the

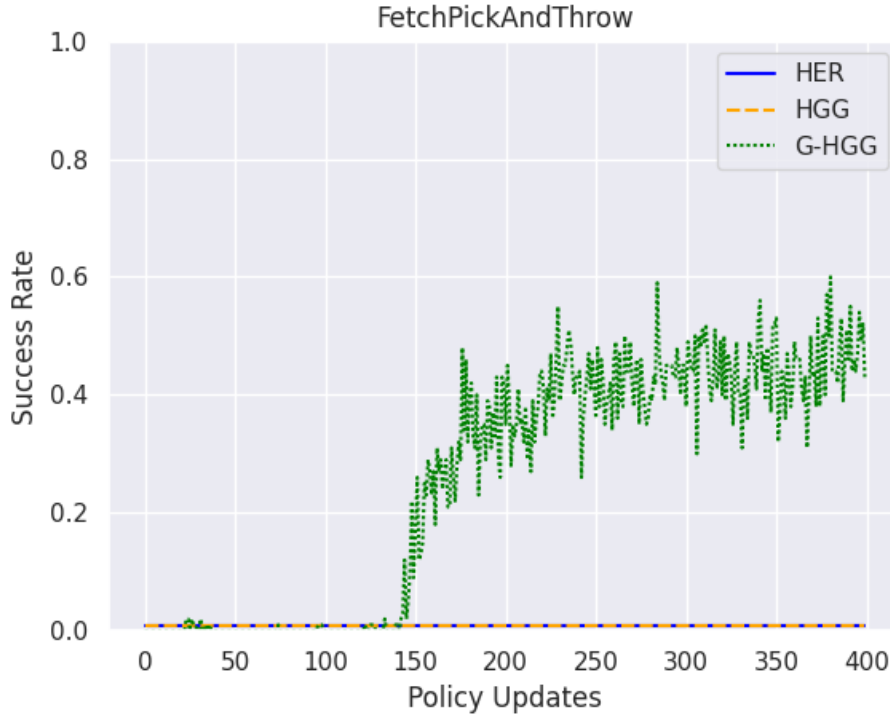


Figure 4.8: Success rate evolution over policy updates of the different agents trained on the *FetchPickAndThrow* task of the Fetch environment.

FetchPickObstacle task in approximately 175 updates, and takes 117 seconds per policy update to train. This is a considerable time, but still manageable because the whole training process lasts approximately one full day. However, the training of the *FetchPickAndPlace* takes less time and by sequentially executing this primitive using multiple different intermediate steps, this task could be solved in a more sample-efficient manner.

On the other hand, the G-HGG agent was only able to partially solve the *Fetch-PickAndThrow* task, starting its learning after 150 policy updates and stagnating at a 50% success rate after approximately 300 policy updates. More significantly, the average time per policy update for the agent in this task is 396.5 seconds, which makes the training of this environment extend to about 44 hours of training; if also considering the time needed to generate episodes for testing during the training phase, this time extends to almost 4 days.

After analysing the behaviour of the learned policy on the *FetchPickAndThrow* task, the robot is observed to be capable of throwing the block to the buckets of

the first row nearest the robot base, but always fails when aiming for the second row buckets that are further away. This happens because during exploration the block almost never ends up in one of the second row buckets, as the action sequence required to do it is too complex, making it impossible to generate hindsight goals in these cases and, on the other hand, exploiting the goals corresponding to the buckets of the first row.

Because G-HGG is a resource-intensive algorithm, it is likely there are more time and sample efficient solutions for both the *FetchPickObstacle* and the *Fetch-PickAndThrow* tasks. For example, the policy learned on the *FetchPickAndPlace* task could be sequentially used by following a series of hand-engineered 3D waypoints that define a trajectory and be able to execute those tasks successfully in a structured, hierarchical fashion.

Shadow Dexterous Hand environment experiments

Just as for the Fetch environment experiments, if DDPG + HER manages to learn a proficient policy in the tasks of the Shadow Dexterous Hand environment, DDPG + HER will be the selected baseline because it is the simplest and the least resource intensive. As such, we first train a DDPG + HER agent on the most simple tasks of the environment (i.e., the *RotateZ*, *RotateX*, and *RotateY* tasks) to evaluate the need to use other more sophisticated variations of HER.

For all these tasks, most times DDPG + HER was able to find a policy that can solve the task, as can be observed in Figures 4.9, 4.10 and 4.11. Furthermore, the time required to train each of these tasks is not excessive. In the case of the *RotateZ* task, DDPG + HER manages to converge in approximately 200 policy updates (i.e., 90% of accuracy on average over several policy updates), while for the *RotateX* and *RotateY* tasks it takes approximately 500 policy updates, each policy update taking on average 96.56 seconds (i.e., approximately one day of training).

The reason why learning the *RotateZ* task takes fewer policy updates to converge is that any rotation around the z -axis involves a tilted cube pose (see Figure 4.2 (c)),

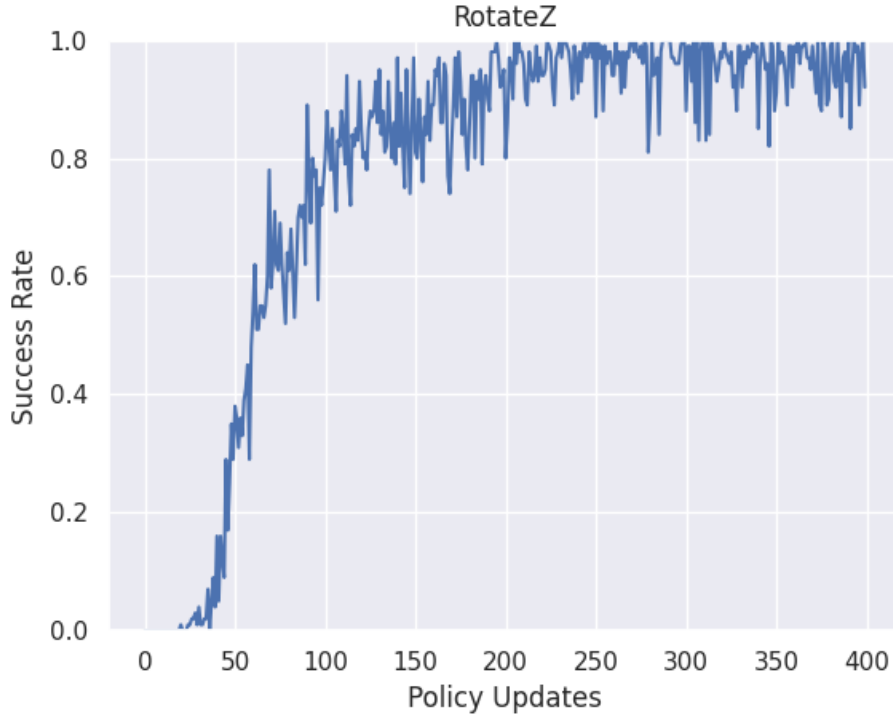


Figure 4.9: Success rate evolution over policy updates of a DDPG + HER agent trained on the *RotateZ* task of the Shadow Dexterous Hand environment. In total, the *RotateZ* task needs 2×10^6 time-steps of simulated experience.

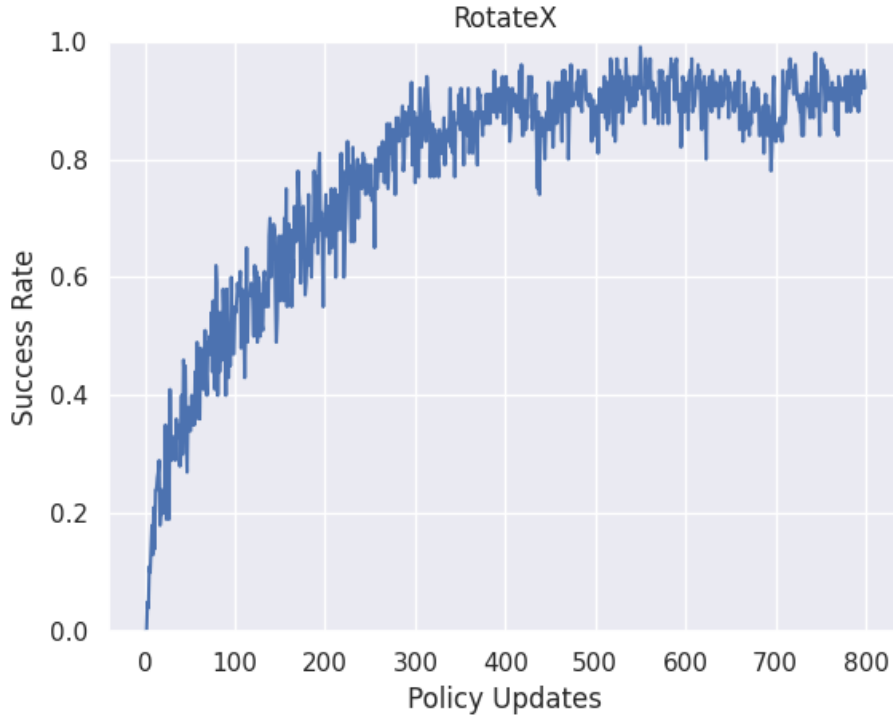


Figure 4.10: Success rate evolution over policy updates of a DDPG + HER agent trained on the *RotateX* task of the Shadow Dexterous Hand environment. In total, the *RotateX* task needs 4×10^6 time-steps of simulated experience.



Figure 4.11: Success rate evolution over policy updates of a DDPG + HER agent trained on the *RotateY* task of the Shadow Dexterous Hand environment. In total, the *RotateY* task needs 4×10^6 time-steps of simulated experience.



Figure 4.12: Success rate evolution over policy updates of the different agents trained on the *RotateParallel* task of the Shadow Dexterous Hand environment. In total, the *RotateParallel* task needs 10^7 time-steps of simulated experience.

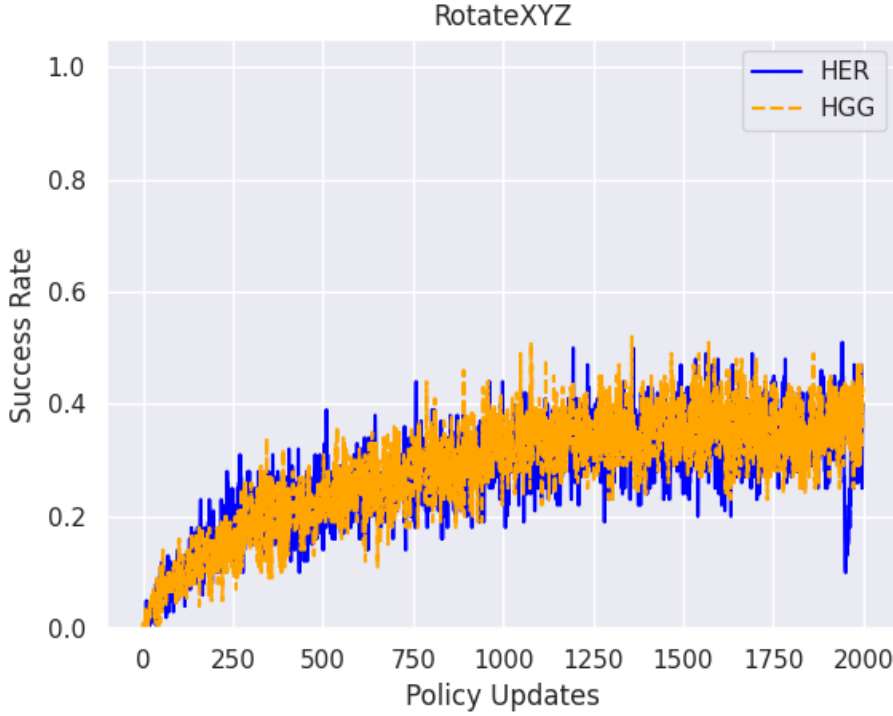


Figure 4.13: Success rate evolution over policy updates of the different agents trained on the *RotateXYZ* task of the Shadow Dexterous Hand environment. In total, the *RotateXYZ* task needs 10^7 time-steps of simulated experience.

meaning the rotation can be done by having the cube in a stable position with one of its faces resting on the palm of the robotic hand. This makes this manipulation task easier, as there are fewer episodes during the exploration phase where the cube falls from the hand, which generates uninformative experiences from which HER can not generate hindsight goals.

Because of the high success rate and training time efficiency, the resulting policies that were capable of solving these tasks are ideal candidates to become primitive policies with the aim of performing arbitrary 3D rotations as a sequence of uniaxial rotations around the various axes.

However, the success of DDPG + HER on the *RotateParallel* and *RotateXYZ* tasks is nowhere near the level of success achieved in the simpler tasks: the policy learned on the *RotateParallel* task only manages to solve the task with a 15% success rate after 2000 policy updates, while in the *RotateXYZ* task it achieves approximately a 40% success rate after the same number of policy updates, with each

policy update taking an average of 99.85 seconds (see Figures 4.12 and 4.13).

On the other hand, the DDPG + HGG agent was able to learn a more proficient policy on the *RotateParallel* task, being able to reach approximately a 50% average success rate in approximately 1750 policy updates. The policy learned on the *RotateXYZ* task, however, had a very similar learning rate and performance as the DDPG + HER agent, being able to reach a 40% success rate by the end of the training, with the only difference being that the error variance across policy updates is lower for the DDPG + HGG agent when compared to the DDPG + HER agent. Each policy update for the HGG takes 179.73 seconds on average.

Perhaps the most interesting finding from the results of these experiments is that HGG is effective for the *RotateParallel* task, as it obtains a greater success rate than HER, but it does not achieve any meaningful improvement in solving the *RotateXYZ* task.

This happens because of the logic that uses HER and the fact that in the *RotateParallel* task the y and x components of the goal space are sampled from the discrete space, $\{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\}$ radians, differently from the *RotateXYZ* task where all components are continuous and random. This translates into lots of achieved poses during exploration that are not valid for the task when using HER because those poses do not belong within the goal space and thus, hindering the pace of learning as the hindsight experience becomes uninformative. Instead, on the *RotateXYZ* task, the cube goal poses are completely random, meaning that any achieved cube pose (except those where the cube is falling from the hand) is a potential target pose for a hindsight scenario. In other words, for the *RotateXYZ* task there will always be a hindsight outcome that achieves the maximum reward regardless of the final state of the cube, as long as the cube stays in the hand.

Therefore, after analysing the results obtained by these algorithms, DDPG + HER is used as a baseline for the *RotateXYZ* task because there are very few differences in terms of performance between HER and HGG, while HGG substantially increases computational complexity; and DDPG + HGG is used as a

Task	Reach	Push	PickAndPlace	PickObstacle	PickAndThrow
Selected method	HER	HER	HER	G-HGG	G-HGG
Success rate	100%	98.17%	96.64%	99.23%	48.73%
Updates to converge	5	70	180	175	400*
Time per policy update (s)	25.47	29.23	29.23	117	396.5
Approx. total training time	≈ 4 hours	≈ 8 hours	≈ 8 hours	≈ 1 day	≈ 3.5 days

Table 4.2: Simplified summary of the selection of baseline learning methods for each of the tasks of the Fetch environment. (*) emphasises that the task is not fully solved, rather, it reaches stagnation at approximately 50% success rate after 275 policy updates.

Task	RotateZ	RotateX	RotateY	RotateParallel	RotateXYZ
Selected method	HER	HER	HER	HGG	HER
Success rate	97.74%	94.95%	96.17%	53.04%	40.56%
Updates to converge	200	500	500	1750	2000
Time per policy update (s)	96.56	96.56	96.56	179.73	99.85
Approx. total training time	≈ 14 hours	≈ 1 day	≈ 1 day	≈ 5 days	≈ 3 days

Table 4.3: Simplified summary of the baseline learning method selection for each of the tasks of the Shadow Dexterous Hand environment.

baseline model for the *RotateParallel* task because the increase of success rate of HGG is significant enough to accept the higher computational cost that comes with using said algorithm.

4.5 Conclusion

As can be seen throughout this Chapter, the selection of the learning algorithm depends on the task that is being solved. In particular, a DDPG + HER agent was selected as a baseline model for the *FetchReach*, *FetchPush*, *FetchPickAndPlace*, *RotateZ*, *RotateX*, *RotateY*, and *RotateXYZ* tasks. However, a DDPG + HGG agent was selected as a baseline model for the *RotateParallel* task, and a DDPG + G-HGG agent was chosen for the obstacle-avoiding tasks *FetchPickObstacle* and *FetchPickAndThrow*. Tables 4.2 and 4.3 summarise the algorithms chosen for each task along with different metrics captured during the training process.

Even though the selected baseline models show a strong performance, they do so at the cost of high training times and poor sample efficiency, as the number of updates they require to reach convergence tend to be large. In other words, there is room for improvement regarding the ability of these methods to learn fast using

few interactions with the environment in an efficient manner.

In the following chapter, the policies learned on the most simple tasks of the Fetch environment (i.e., *FetchReach*, *FetchPush* and *FetchPickAndPlace*) and the Shadow Dexterous Hand environment (i.e. *RotateZ*, *RotateX*, and *RotateY*) are sequentially used by following a hand-engineered curriculum design with the aim of improving these results in terms of sample and time efficiency, as well as in terms of performance (i.e., success rate).

Chapter 5

Sequential execution of primitive behaviours

5.1 Chapter overview

In Chapter 3, reward engineering is explored as a means of improving the sample efficiency of off-policy RL algorithms. However, as pointed out in that chapter, the main bottleneck of these algorithms in terms of sample efficiency is not in the design of the reward, but rather in the huge amount of exploration these algorithms require to solve complex tasks, as well as the lack of domain knowledge usually employed to train them. In this chapter, we aim to answer the following research question:

- **Research Question 2:** How can we make use of basic primitive skills in a sequential manner to solve more complex tasks? In which scenarios would this approach be most useful?

In particular, this chapter investigates the effect, in terms of sample efficiency, time consumption, and task success rate, of a novel learning framework based on a sequential execution of learned primitive policies, instead of learning to solve manipulation tasks in an end-to-end fashion. This approach is evaluated using two different robotic manipulators that are included in the OpenAI Gym [22]. The work described in this chapter has been accepted for publication at the 2023 9th Inter-

national Conference on Automation, Robotics and Applications (ICARA) and was awarded with the best session presentation [185]¹. Section 5.2 identifies the motivations for using a sequence of executed primitives, as well as the benefits that these provide compared to end-to-end RL-based approaches. Section 5.3 presents some relevant related work for the tasks at hand in this chapter that has not been previously introduced in Chapter 2. Section 5.4 explains the method and the results are presented and discussed in Section 5.5. Finally, Section 5.6 presents the conclusions and Section 5.7 summarises the main contributions of this chapter.

5.2 Motivation

A primary reason why it takes so long to train state-of-the-art reinforcement learning techniques in manipulation tasks is that they are very complex to solve without domain knowledge [186]–[188]. However, most manipulation tasks can be decomposed into a number of easier tasks that a robot can learn more easily with much less training or for which an engineer may even be able to design a solution [189]–[192]. For example, robots trained with RL can easily learn how to reach a point in space, or how to lift an object, when these tasks are trained independently. However, if the robot needs to learn from scratch how to both reach for an object and then lift it, the training time required increases in a nonlinear manner, meaning it often requires more simulated experience overall to learn these skills.

Training time becomes an important consideration for tasks that have high degrees of complexity, like the block manipulation tasks in OpenAI’s Shadow Dexterous Hand environment environment [22]. The most popular method for learning in this kind of goal-based environment is DDPG [77] trained in conjunction with HER [54]. DDPG + HER is capable of successfully solving most tasks implemented in this environment, but the amount of simulated experience it requires for training is immense, exceeding 38×10^7 time-steps in the most complex tasks; and even then it is not able to discover an optimal policy that is able to succeed for all target

¹<https://icara.us/2023.html>

goals [154].

Traditional robotic manipulation systems usually define a set of non-primitive and primitive tasks that can be organised in a hierarchy [193]–[195]. Non-primitive actions are a composition of primitive actions performed in a particular order, where a primitive action is one which cannot or should not be further divided into a sequence of simpler actions. Therefore, in order to reproduce a non-primitive action, a robot should sequentially perform a set of primitive actions [196]–[198]. We refer to Hierarchical Reinforcement Learning (HRL) as those reinforcement learning methods that decompose a problem into a hierarchy of *sub-problems*, such that solving the original problem requires the sequential execution of the solutions to these *sub-problems* as if they were primitive actions [199]–[201].

In order to demonstrate the benefit of using domain knowledge to decompose a manipulation problem into primitive sub-problems, in this chapter we propose to decompose the goals into simpler sub-goals that can be learned more easily by standard HER + DDPG using sparse rewards, to then sequentially execute those learned policies in order to solve more complex tasks. We evaluate this approach using the two robotic manipulators from the OpenAI Gym, introduced in the previous chapter: the Fetch environment [154] and the Shadow Dexterous Hand environment [180], as these are the most popular environments at the moment, as explained in the previous chapter.

Because these robots perform manipulation tasks of different natures, the goal decomposition must be performed using different strategies (one is meant to grasp an object and manipulate it, while the other is an in-hand manipulation of the object): (i) for the Fetch environment we make use of simple behaviours, such as reaching a point in space, and pushing and picking an object in order to perform obstacle-avoidance manipulation tasks, and (ii) for the Shadow Dexterous Hand environment we propose the use of Davenport angles [202] to decompose 3D rotations of a cube into a sequential chain of 1D rotations around different orthogonal axes.

By independently learning these primitives and then sequentially applying a

defined method, we investigate if this methodology will accelerate the learning of the overall task goal when compared to training using the standard learning method that uses HER.

5.3 Related work

5.3.1 Hierarchical control

Unlike model-free off-policy RL algorithms such as HER, which allow agents to learn dexterous manipulation skills using minimal and even no domain knowledge, traditional robotic manipulation controllers have typically made use of a set of simple manipulation abilities normally named *primitive actions* that, when sequentially executed, can solve more complex tasks [203]–[205].

There are multiple examples in the literature employing sequential execution of primitive policies in order to perform a lifting behaviour and carry out a manipulation task with it (e.g., pick and place task) [206]–[209]. For example, Skoglund *et al.* define a set of dynamic motion primitives (DMP), manually engineered, including some that move the end-effector to different places of the space, others that copy pre-defined trajectories and other action-based primitives such as grasping or releasing an object [210]. More recently, Marzari *et al.* define a sub-set of three sub-tasks to solve a pick and place task (approach, manipulate, and retract), which are trained using RL, and later using a High Level Choreographer (HLC) network that learns to sequentially execute those sub-tasks to output the intended behaviours. This network is typically called an orchestrator network because of the purpose of the network) [211].

For the Fetch environment tasks, we employ a similar technique as Marzari *et al.*, as we train a DDPG + HER based agent to solve simple tasks such as reaching a point with the end-effector, and pushing or picking a block, and then sequentially execute them in order to solve other more complex tasks that require avoiding obstacles. However, unlike Marzari *et al.*, we apply a pre-defined curriculum

eliminating the computational cost that comes with the training of an additional orchestrator network.

In the context of in-hand robotic manipulation, primitives are usually defined depending on the type of action they produce. For example, Li *et al.* define three different skills (reposing, sliding, and flipping), and a mid-level controller plans how to sequentially apply them to reach a particular cube pose for a three-fingered robotic hand [212]. Bhatt *et al.* use a similar method but define those primitives as shift, pivot, twist, and finger gait, determining how the robot fingers behave for each of the primitives [213]. Instead, Abondance *et al.* decompose their manipulation task goal into a sequence of two translations across two different axes and a rotation around one axis, making the execution of the intended task more organised and predictable [214].

Many different recent works make use of a learned hierarchical decomposition in order to solve tasks of higher complexity [201], [215]–[218]. Some examples include the Relay Hindsight Experience Replay (RHER) [219], which generates intermediate random motion sub-goals, and then uses the policies learned to solve these sub-goals, which they call *teacher policies*, with probability α during exploration to guide the learning of another agent, evaluated using the Fetch environment tasks. Similarly, Lai *et al.* propose the Hindsight Planner [118], a method that makes use of a LSTM in order to plan the intermediate goals, and evaluate their method using the Shadow Dexterous Hand environment tasks.

Even though it has been proved that these more recent methods, outlined in the previous paragraph, have better performances than the standard DDPG + HER algorithm in terms of task success rate, they do not necessarily provide intermediate goals that make sense to solve the task, as the intermediate goals proposed are generated by using history trajectories with the aim to learn the structure of feasible intermediate goals (see Figure 5.1). Furthermore, they require the use of additional reward systems and networks to generate and follow the planned sub-goals, which increases the computational cost during the learning phase of those models.

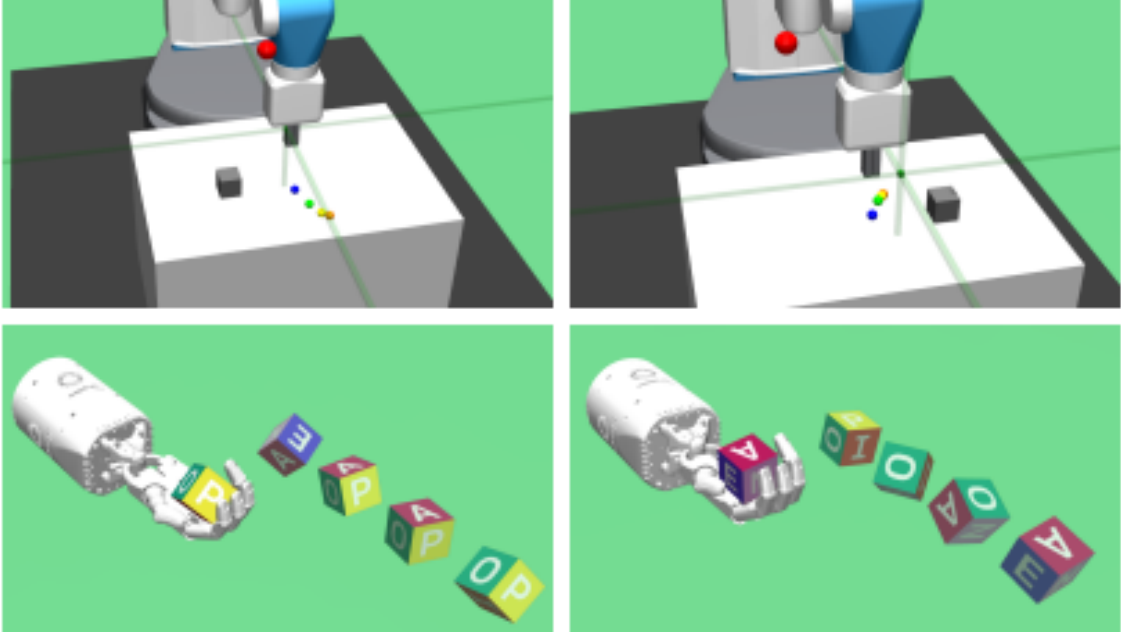


Figure 5.1: An example visualisation of a hindsight planner sub-goal decomposition for the Fetch environment and the Shadow Dexterous Hand environments. Image taken from Lai *et al.*'s paper.

For the Shadow Dexterous Hand environment tasks, instead of focusing on specific hand movements, and similarly to Abondance *et al.* (but adapted to our problem), in this chapter we propose to decompose the target pose of the cube into a sequence of simpler rotations that are easier to learn by an agent using a model-free off-policy method such as DDPG + HER, and sequentially execute only two primitive skills to successfully complete the more complex overall 3D rotation.

5.3.2 Davenport chained rotations

Complex tasks are often defined by complex goals. For example, in the case of the block manipulation tasks of the OpenAI's Gym Shadow Dexterous Hand environment, this complexity appears when the goal the robot needs to achieve consists of implementing arbitrary 3D rotations around all axes. However, any 3D rotation can be decomposed into three sequential rotations around the orthogonal axes of a known reference frame [202] (see Figure 5.2). These rotations can be performed around the axes of a fixed system of reference (extrinsic rotations) or around the

axes of the rotating system of reference (intrinsic rotations), but only if the second axis of rotation is orthogonal to the plane containing the first and third axes [220].

There is no unique way to decompose a 3D rotation into three consecutive rotations around extrinsic axes. The same rotation can be achieved by decomposing a 3D rotation into elementary rotations around z - x - z , x - y - x , y - z - y , z - y - z , x - z - x or y - x - y (if also considering intrinsic rotations, more configurations must be added: x - y - z , x - z - y , y - x - z , y - z - x , z - x - y , z - y - x).

The choices of axes where one axis is repeated are called symmetric Euler angles or *classic Euler Angles*, while those where all three axes are used and none is repeated, are called asymmetric Euler angles or *Tait-Bryan* angles. The decision on which framework to choose depends on the application. For example, in the world of aeronautics the use of the 3-2-1 Euler angles intrinsic rotation decomposition is widely used to represent the aircraft attitude [221]–[223], or in the field of astrodynamics the 3-1-3 extrinsic rotation configuration can be used to describe orbits [224], [225]. Because the reference frame is fixed in the Gym Shadow Dexterous Hand environment tasks, the decomposed sequence of rotations applied to the cube should be extrinsic (i.e., it must use a symmetric Euler angles framework for the rotation decomposition).

In the same way that any orientation can be achieved by composing three elemental rotations starting from a known orientation, any rotation matrix $\mathbf{R}$ can be decomposed as a product of three elemental rotation matrices. For example:

$$\mathbf{R} = \mathbf{X}(\alpha)\mathbf{Y}(\beta)\mathbf{Z}(\gamma) \quad (5.1)$$

is the decomposition of the rotation matrix $\mathbf{R}$ into three rotations, where it first executes a rotation of α around the x -axis, then a rotation of β around the y -axis and finally a rotation of γ around the z -axis. In this example, $\mathbf{X}(\alpha)$, $\mathbf{Y}(\beta)$, $\mathbf{Z}(\gamma)$ are rotation matrices that correspond to elemental rotations around the x , y , and z axes:



Figure 5.2: Overview of a decomposition of an arbitrary 3D rotation into a sequence of three extrinsic rotations. In particular, it represents a $(45^\circ, 30^\circ, -60^\circ)$ rotation, using the z - x - z , or 3-1-3, extrinsic rotations configuration. Image taken from DF Malan, 14 September 2004, Euler Angles x-convention, <https://en.wikipedia.org/wiki/File:EulerX.png>

$$\mathbf{X}(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \quad (5.2)$$

$$\mathbf{Y}(\beta) = \begin{pmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \quad (5.3)$$

$$\mathbf{Z}(\gamma) = \begin{pmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.4)$$

As such, any given target rotation matrix can be obtained by evaluating the composition of three rotations around the axes. For example, in the situation where we are using a 3-1-3 configuration (or z - x - z), the resulting rotation matrix will be $\mathbf{R} = \mathbf{Z}(\alpha)\mathbf{X}(\beta)\mathbf{Z}(\gamma)$, where $\mathbf{Z}(\alpha)$ and $\mathbf{Z}(\gamma)$ are rotation matrices as described in Equation 5.4 and $\mathbf{X}(\beta)$ is a rotation matrix as described in Equation 5.2. Instead, if we were working using a 2-3-2 configuration (or y - z - y), the resulting rotation matrix will be $\mathbf{R} = \mathbf{Y}(\alpha)\mathbf{Z}(\beta)\mathbf{Y}(\gamma)$, where $\mathbf{Y}(\alpha)$ and $\mathbf{Y}(\gamma)$ are rotation matrices as described in Equation 5.3 and $\mathbf{Z}(\beta)$ is a rotation matrix as described in Equation 5.4.

However, in most robotic manipulation applications, the goal is to reach or to apply a general 3D rotation matrix to an object, meaning that the angles α , β and γ are the unknown variables, not the rotation matrix $\mathbf{R}$. Rothmayr and Hodges presented Table 5.1, which contains the general formulas to find those angles once the rotation decomposition system has been selected and fixed [226]. In the left column, the different rotation matrices $\mathbf{R}$ resulting from the sequential execution of these is shown, assigning an equation to each of the values of the matrix. The right column contains the formulas that are used to obtain the fundamental rotation angles from elements of $\mathbf{R}$. The table uses the following considerations for the sake of simplicity:

	Rotation matrix $\mathbf{R}$	Angle solution
$X_1Z_2X_3 =$	$\begin{bmatrix} c_2 & -c_3s_2 & s_2s_3 \\ c_1s_2 & c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 \\ s_1s_2 & c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$	$\alpha = \arctan(\frac{R_{31}}{R_{21}})$ $\beta = \arccos(R_{11})$ $\gamma = \arctan(\frac{R_{13}}{-R_{12}})$
$X_1Y_2X_3 =$	$\begin{bmatrix} c_2 & s_2s_3 & c_3s_2 \\ s_1s_2 & c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 \\ -c_1s_2 & c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$	$\alpha = \arctan(\frac{R_{21}}{-R_{31}})$ $\beta = \arccos(R_{11})$ $\gamma = \arctan(\frac{R_{12}}{R_{13}})$
$Y_1X_2Y_3 =$	$\begin{bmatrix} c_1c_3 - c_2s_1s_3 & s_1s_2 & c_1s_3 + c_2c_3s_1 \\ s_2s_3 & c_2 & -c_3s_2 \\ -c_3s_1 - c_1c_2s_3 & c_1s_2 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$	$\alpha = \arctan(\frac{R_{12}}{R_{32}})$ $\beta = \arccos(R_{22})$ $\gamma = \arctan(\frac{R_{21}}{-R_{23}})$
$Y_1Z_2Y_3 =$	$\begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1s_2 & c_3s_1 + c_1c_2s_3 \\ c_3s_2 & c_2 & s_2s_3 \\ -c_1s_3 - c_2c_3s_1 & s_1s_2 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$	$\alpha = \arctan(\frac{R_{32}}{-R_{12}})$ $\beta = \arccos(R_{22})$ $\gamma = \arctan(\frac{R_{23}}{R_{21}})$
$Z_1Y_2Z_3 =$	$\begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 & c_1s_2 \\ c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 & s_1s_2 \\ -c_3s_2 & s_2s_3 & c_2 \end{bmatrix}$	$\alpha = \arctan(\frac{R_{23}}{R_{13}})$ $\beta = \arctan(\frac{\sqrt{1-R_{33}^2}}{R_{33}})$ $\gamma = \arctan(\frac{R_{32}}{-R_{31}})$
$Z_1X_2Z_3 =$	$\begin{bmatrix} c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 & s_1s_2 \\ c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 & -c_1s_2 \\ s_2s_3 & c_3s_2 & c_2 \end{bmatrix}$	$\alpha = \arctan(\frac{R_{13}}{-R_{23}})$ $\beta = \arccos(R_{33})$ $\gamma = \arctan(\frac{R_{31}}{R_{32}})$

Table 5.1: Table of matrix products containing the formulas for extracting the rotation angles of any Davenport chained rotation decomposition configuration.

Algorithm 3 Overview of a framework for simplified sequential execution of learned primitive policies

Given: K primitive policies: $\mathbb{P} = \{\hat{\pi}_1(s, g), \hat{\pi}_2(s, g), \dots, \hat{\pi}_K(s, g)\}$.

- 1: Load $\mathbb{P}$.
 - 2: **for** each episode **do**
 - 3: Sample a goal $g \sim \mathcal{G}$ and an initial state, s_0 .
 - 4: Define way-points $\mathbb{W} = \{w_1, w_2, \dots, w_K = g\}$ based on the episode goal g and the initial state s_0 .
 - 5: **for** $k=1$ to K **do**
 - 6: Execute $\hat{\pi}_k(s, w_k)$ until the way-point w_k is successfully reached.
 - 7: **end for**
 - 8: **end for**
-

1. When rotations have a positive angle, those are always to the right, according to the right hand rule (i.e., clockwise looking along the rotation axis, with the vector of the axis pointing away from the observer).
2. 1, 2, 3 represent the angles α , β and γ (i.e., the angles corresponding to the first, second, and third rotation, respectively, around the chosen rotation axes).
3. X, Y, Z are the matrices representing the rotations around the axes x , y and z of the fixed coordinate system of reference.
4. s and c represent sines and cosines (e.g., s_1 represents the sinus of α or c_3 represents the cosines of γ).

Euler angles suffer from the problem of gimbal lock [227], where the representation loses a degree of freedom and it is not possible to determine the first and third angles uniquely. In this case, the third angle is set to zero by design, and we look for the first rotation so that the returned angles still represent the correct 3D rotation.

5.4 Curriculum design

The goal of this section is to describe an algorithm that sequentially executes a set of K primitive behaviours $\mathbb{P} = \{\hat{\pi}_1, \hat{\pi}_2, \dots, \hat{\pi}_K\}$ to visit a set of K way-point goals, $\mathbb{W} = \{w_1, w_2, \dots, w_K\}$, that belong to the goal space $\mathcal{G}$ of the environment and define a target manipulation trajectory, and which always has the last way-point

set to be the same as the sampled goal g for a given episode ($w_K = g \sim \mathcal{G}$) (see Algorithm 3).

At the beginning of each episode, an initial state s_0 and a goal g for the episode must be sampled, and then the set of way-points must be defined. A way-point w_k corresponds to an intermediate goal that the primitive behaviour $\hat{\pi}_K$ (from the set $\mathbb{P}$) can ideally reach. Therefore, the definition of these way-points depends on the task that is being executed: for the Fetch environment, because the goal is expressed as a 3D coordinate, those way-points also correspond to intermediate 3D coordinates; alternatively, for the Shadow Dexterous Hand environment, because the goal is expressed as a quaternion that defines a pose, way-points correspond to intermediate cube poses.

When sequentially executing these primitive behaviours, $\hat{\pi}_2(s, w_2)$ is not executed until $\hat{\pi}_1(s, w_1)$ has successfully reached way-point w_1 , repeating this process until the last way-point w_K , which is the same as the goal g of the episode, is reached. To avoid a scenario where the agent becomes *stuck* (i.e., a policy $\hat{\pi}_K$ unable to reach a way-point w_k , and therefore getting stuck), a maximum number of time-steps to reach each way-point can be set, with the algorithm skipping to following way-point after reaching that threshold. In the following sections a more specific description is provided on how these way-points are set for the Fetch and the Shadow Dexterous hand environment complex tasks (i.e., *FetchPickObstacle*, *FetchPickAndThrow*, *RotateParallel*, and *RotateXYZ*).

5.4.1 Obstacle-avoidance manipulation tasks

We trained DDPG + HER for each of the obstacle-free tasks of the Fetch environment. We hypothesise that training DDPG + HER to learn basic manipulation skills in obstacle-free tasks (e.g., *FetchPickAndPlace* task) and then later using these learned policies in a sequential manner (see Figures 5.3 and 5.4) can achieve better success rates than using HER or G-HGG in an end-to-end manner on the most difficult *FetchPickObstacle* and *FetchPickAndThrow* tasks, or at least have the same

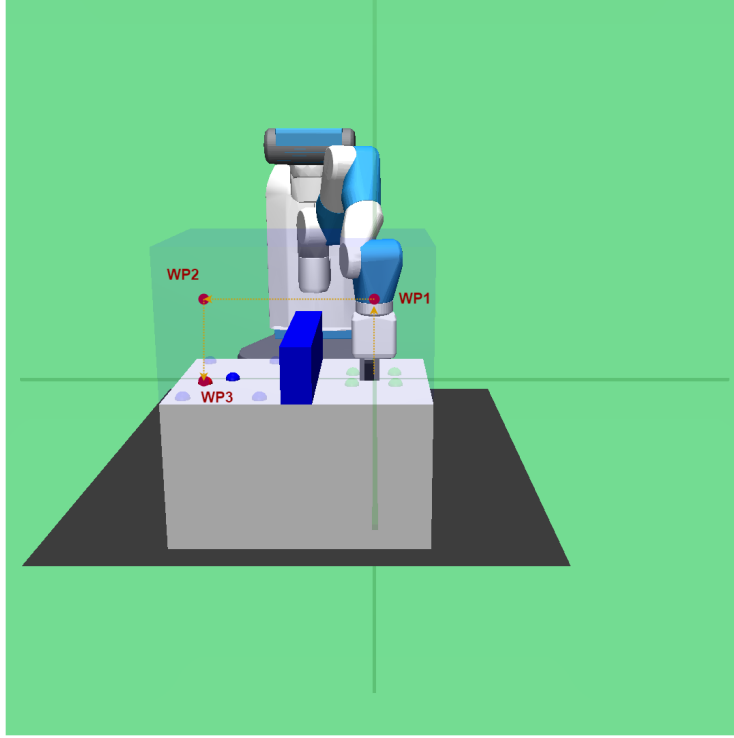


Figure 5.3: Visualisation of the rectangular sequential trajectory proposed to solve the *FetchPickObstacle* task using the policy learned on the *FetchPickAndPlace* task. WP represent the way-points that the cube must visit along the trajectory.

success rate with less overall training time (i.e., learning in a more computationally-efficient fashion).

At the beginning of each episode, an initial state s_0 and a goal g are sampled. In these tasks, goals represent a 3D coordinate that specifies the target position of the cube, while the state s contains the current position of the cube in the 3D space among other information related to the environment.

In order to solve the *FetchPickObstacle* task, two different curricula are proposed by defining a different set of way-points as 3D coordinates that will be reached using the policy learned to solve the *FetchPickAndPlace* task:

- **Rectangular trajectory:** this set-up aims to lift the cube and move it over the obstacle following a rectangular trajectory (see Figure 5.3). To do this, the policy learned to solve the *FetchPickAndPlace* task is used to lift the block up 0.6 meters from where the cube is, high enough to be able to avoid the obstacle (i.e., setting as way-point the 3D coordinate $(x_{cube}, y_{cube}, 0.6)$ m.

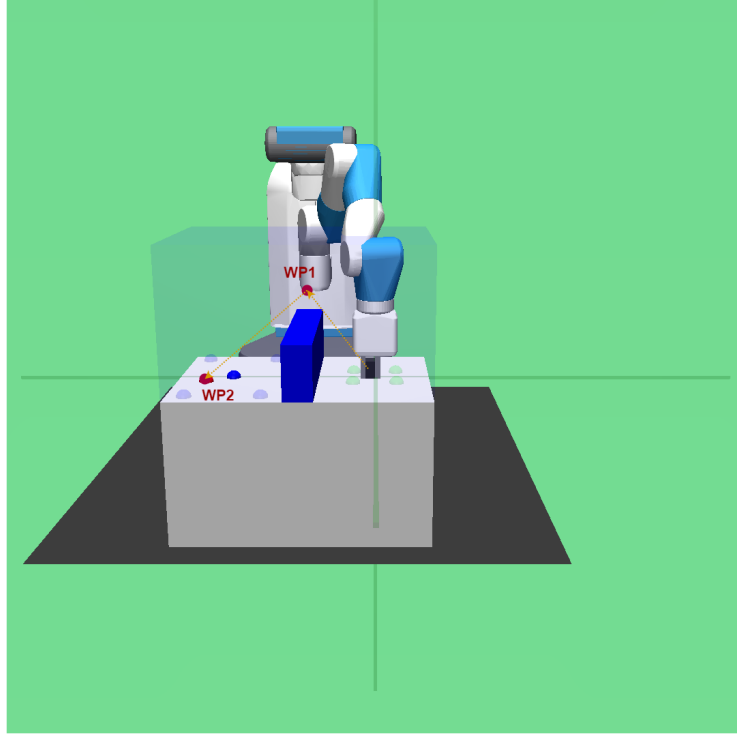


Figure 5.4: Visualisation of the triangular sequential trajectory proposed to solve the *FetchPickObstacle* task using the policy learned on the *FetchPickAndPlace* task. WP represent the way-points that the cube must visit along the trajectory.

Then, the second way-point is set to be directly over the goal position (i.e., the 3D coordinate $(x_{goal}, y_{goal}, 0.6)$ m, and finally the last way-point is set to be the same as the episode goal g .

- **Triangular trajectory:** this set-up aims to lift the cube and move it over the obstacle following a triangular trajectory, with its highest point of the trajectory being precisely when the block is over the obstacle (see Figure 5.4). To do this, the policy learned to solve the *FetchPickAndPlace* task is used to lift the block and move it precisely over the obstacle high enough that the block does not touch said obstacle. In particular, this way-point is set to the 3D coordinate $(0, 0, 0.6)$ m. The last way-point is set to be the same as the episode goal g .

Because there might be times where the last sequence of the execution can not be fully solved using the policy learned to solve the *FetchPickAndPlace* task, the policy learned to solve the *FetchPush* task can be also used if necessary to refine

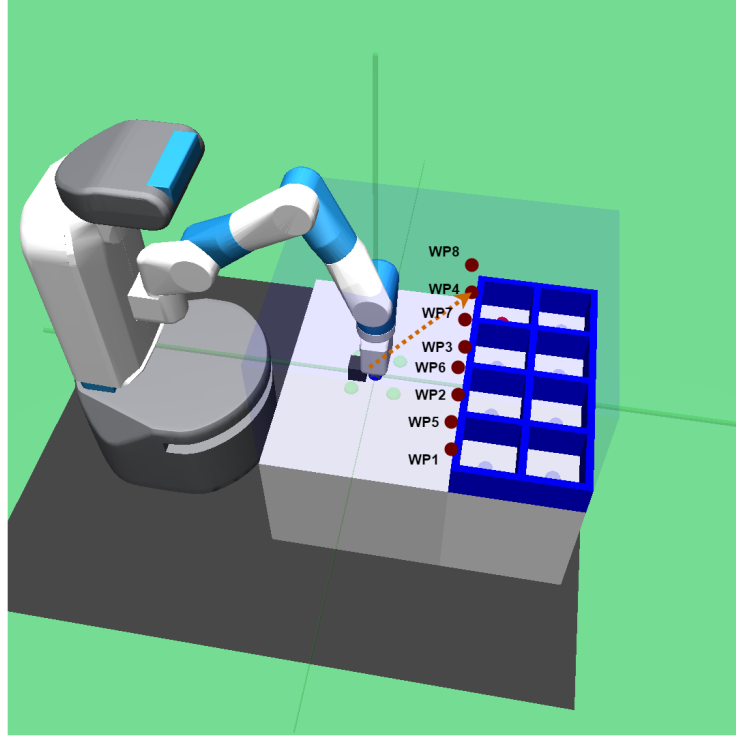


Figure 5.5: Visualisation of the sequential trajectory proposed to solve the *Fetch-PickAndThrow* task using the policy learned on the *FetchPickAndPlace* task, and opening the gripper as soon as the primitive behaviour reaches the assigned intermediate way-point. Note that only one of the 8 way-points is visited, and the selection of the way-point depends on the target bucket into which the block must be thrown.

the block position without the need of lifting it again, regardless of the trajectory set-up chosen.

The *FetchPickAndThrow* task is a much more difficult problem, and in order to successfully solve it the robot must execute behaviours that have not been learned by any of the selected primitives (i.e., opening the gripper with the intention of throwing the cube). Therefore, only one solution based on a sequenced execution of primitive behaviours is proposed (see Figure 5.5).

To attempt to solve this task, a set of 8 different way-points $\mathbb{W} = \{w_1, w_2, \dots, w_8\}$ are heuristically defined, which correspond to intermediate 3D coordinates that are situated over the limit of the buckets of the first row (see Figure 5.5). For each episode, one of these way-points is chosen depending on the sampled goal, and the policy learns to solve the *FetchPickAndPlace* task picks the block and moves it to the selected way-point. Once the block reaches the chosen way-point, the gripper opens

and lets the block go, ideally profiting from the inertia generated by the motion of the robotic arm so that the block ends up falling inside the desired bucket.

As explained in Section 4.3, the goal selection is achieved by sampling from a discrete uniform distribution, and each bucket is assigned an index such that the first bucket on the right side of the first row (from the perspective of the robotic arm) is assigned the index $ID = 1$, the second of the first row is assigned $ID = 2$, and so on; while the first bucket on the right side of the first row is assigned the index $ID = 5$, and so on. As such, each of the way-points is assigned to one individual bucket, and depending on the goal the robot will visit only the way-point associated with the bucket.

The way-points associated to the buckets of the first row (i.e., way-points 1 to 4) are located at the 3D coordinates $(x_{bucket_limit}, y_{bucket_limit}, 0.5)$ m, where $(x_{bucket_limit}$ and y_{bucket_limit} represent the xy position of the wall of the bucket that is closest to the robotic arm; while those associated to the buckets of the second row (i.e., way-points 5 to 8) are located at the 3D coordinates $(x_{bucket_limit}, y_{bucket_limit}, 0.7)$ m, taking x_{bucket_limit} and y_{bucket_limit} values from the bucket of the first row located next to the target bucket (e.g., when the goal is to throw the cube into the bucket with $ID = 5$, it will take the same x and y coordinates as those used for WP1; if the goal is $ID = 6$, it would use the values of WP2, and so on).

5.4.2 In-hand manipulation tasks

We trained DDPG + HER for each of the basic uni-axial rotation tasks: *RotateX*, *RotateY*, and *RotateZ*. For these tasks, episodes have a length of 100 time-steps because this is the default value set in the original OpenAI Gym. This means that, when evaluating the algorithm, the agent will have 300 time-steps in order to solve the task, 100 time-steps for each of the three chained rotations. We hypothesise that training DDPG + HER to learn extrinsic rotations around fixed axes in the world frame of reference and using the learned policies deployed in a sequential manner (see Figure 5.6) can achieve better success rates than using HER or HGG in an

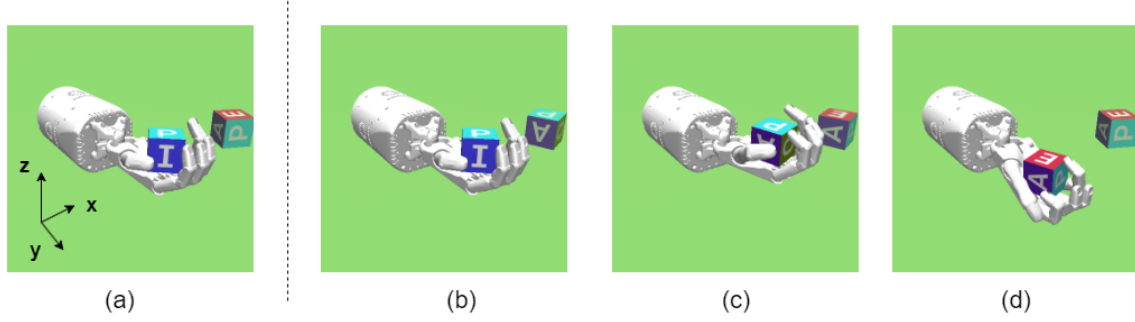


Figure 5.6: Example of a 3D rotation goal decomposition into three Davenport angles. A target 3D rotation (a) is decomposed into three consecutive target rotations around the (b) z , (c) x and again (d) z axes, respectively. Goal decomposition can be understood by looking at the target poses, which are the cubes that are not touching the robotic hand, compared to the starting poses for the rotations, which are shown as the cubes in the robotic hand.

end-to-end fashion on the most complex *RotateParallel* and *RotateXYZ* tasks, if both methods are trained using the same amount of simulated experience.

To do this, we propose to decompose the target rotation into a Davenport decomposition of chained rotations around extrinsic axes, each intermediate rotation being a way-point that the robot must visit in order to fulfil the task. At the beginning of each episode, an initial state s_0 and a goal g for the episode are sampled. This initial state, among other information that can be checked in Section 4.3, contains the current pose of the cube expressed as a quaternion $q = (w, x, y, z)$, while the goal is also expressed as a quaternion $q_g = (w_g, x_g, y_g, z_g)$ that defines the target pose of the cube. In other words, rather than having a goal that expresses the target re-orientation that must be applied (i.e., the rotation matrix), the goals of the episodes define the target cube orientation with respect to a fixed reference axis.

The Davenport decomposition is a method that works using rotation matrices, which means that the rotation matrix $\mathbf{R}$ between the current cube pose $q = (w, x, y, z)$ and the target cube pose $q' = (w', x', y', z')$ must be found. To do so, the following steps are followed:

1. Normalise both quaternions q and q' to ensure they have unit length.
2. Calculate the quaternion conjugate of q , denoted as $q_{conj} = (w, -x, -y, -z)$, by changing the signs of the x , y , and z components.

3. Compute the quaternion product of q' and q_{conj} , denoted as $q_{rot} = q' * q_{conj}$.

The result of this product results in a new quaternion defining the rotation

$$q_{rot} = (w_{rot}, x_{rot}, y_{rot}, z_{rot}).$$

4. Convert rotation quaternion q_{rot} into a rotation matrix using Equation 5.5, where the terms of q_{rot} are expressed as $q_{rot} = (q_0, q_1, q_2, q_3)$ for simplicity. This equation was obtained from [228].

$$\mathbf{R}(Q_{rot}) = \begin{pmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 2(q_0^2 + q_3^2) - 1 \end{pmatrix} \quad (5.5)$$

Once this matrix has been obtained, any Davenport decomposition can be achieved making use of the Table 5.1, taking $\mathbf{R}(q_{rot})$ to be the rotation matrix. The output of any Davenport decomposition is a sequence of three rotations around the axes, $\mathbf{R}(q_{rot}) = \mathbf{R}_1(\alpha)\mathbf{R}_2(\beta)\mathbf{R}_3(\gamma)$, each $\mathbf{R}_1$, $\mathbf{R}_2$, and $\mathbf{R}_3$ being our way-points $\mathbb{W}$ for the manipulation task. However, these are matrices that express a rotation, instead of a cube pose, meaning that these way-points do not belong to the goal space $\mathcal{G}$. Therefore, these way-points must be reset to express cube poses by applying the re-orientation to the cube in a sequential manner. To reorient an object defined by a quaternion $q = (w, x, y, z)$ using a rotation matrix $\mathbf{R}$, the following steps must be followed:

1. Convert the quaternion q to a 3x3 rotation matrix $\mathbf{R}(q)$. This can be done using Equation 5.5.
2. Multiply the rotation matrix $\mathbf{R}(q)$ that represents the cube pose with the corresponding quaternion rotation matrix of the Davenport decomposition $\mathbf{R}_k(\theta)$ (with $k=1, 2, 3$ and $\theta=\alpha, \beta, \gamma$) to obtain the new rotation matrix $\mathbf{R}_{new}$ that represents the pose in matrix format corresponding to the way-point goal.

This can be done by matrix multiplication: $\mathbf{R}_{new} = \mathbf{R}(q)\mathbf{R}_k(\theta)$ and has the following notation for simplification:

$$\mathbf{R}_{new} = \mathbf{R}(q)\mathbf{R}_k(\theta) = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \quad (5.6)$$

3. Calculate the trace of $\mathbf{R}_{new}$, which is the sum of the diagonal elements:

$$\text{tr}(\mathbf{R}_{new}) = r_{11} + r_{22} + r_{33} \quad (5.7)$$

4. Calculate the scalar component w of the quaternion:

$$w_{new} = \frac{\sqrt{1 + \text{tr}(\mathbf{R}_{new})}}{2}. \quad (5.8)$$

5. Calculate the x, y, and z components of the quaternion using the following formulas:

$$x_{new} = \frac{r_{32} - r_{23}}{4w_{new}} \quad y_{new} = \frac{r_{13} - r_{31}}{4w_{new}} \quad z_{new} = \frac{r_{21} - r_{12}}{4w_{new}} \quad (5.9)$$

6. Assemble the new quaternion q_{new} using the calculated components: $q_{new} = (w_{new}, x_{new}, y_{new}, z_{new})$. The new quaternion q_{new} represents the quaternion equivalent of the rotation matrix $\mathbf{R}_{new}$, which at the same time corresponds to the goal orientation of the

Once this process is iteratively applied throughout the whole curriculum, the way-points, $\mathbb{W} = \{w_1, w_2, w_3\}$, represent the intermediate cube target poses calculated by the Davenport decomposition expressed in a quaternion format with respect to a fixed reference coordinate system, exactly the same way episode goals are defined in the Shadow Dexterous Hand environment tasks.

5.5 Evaluation

5.5.1 Evaluation sets

Different strategies are used to generate the dataset depending on the task to be evaluated. This section describes dataset construction for both the Fetch and the Shadow Dexterous environment tasks.

Fetch environment

Generating a test dataset for both of the Fetch environment tasks explored in this Section is very simple. As a matter of fact, the only thing required is to define the total number of episodes used for testing, and the goal generation of each of the tasks described in Section 4.3 will create a sufficiently varied dataset to extract meaningful conclusions. We decide to set this number up to 1,000 episodes for both tasks in order to ensure there is enough variability, even though it is expected to have repeated goals in those episodes. The evaluation is carried out in terms of performance (i.e., task success rate), and time efficiency.

Shadow Dexterous Hand environment

Similarly, the goal generation of each of the tasks of this environment (described in Section 4.3) is sufficient to create a varied dataset for evaluating our method. However, because the goal space is larger, a larger number of episodes is required to ensure the validity of the results. Therefore, for each Davenport rotation decomposition configuration, we generate the test set by running 1,500 different episodes, this time ensuring that there are no repeated initial and target cube poses (i.e., the environment resets until a non-used initial environment configuration is found).

A second dataset is created in order to analyse the performance of the baseline models and our algorithm depending on how many extrinsic rotations must be executed to complete the 3D rotation: this test set of 6,600 different initial and target cube poses for each rotation configuration is created by randomly sampling the goal

Comparable to	# rotations required to solve		
	1	2	3
<i>RotateParallel</i>	200	1000	2000
<i>RotateXYZ</i>	600	2000	4000

Table 5.2: Alternative test set created to evaluate the method. The table shows how many poses are in the test set that can be solved with either 1, 2 or 3 extrinsic rotations.

space of the primitive behaviours (e.g., if evaluating the z - x - z , a random rotation around the z -axis is obtained by sampling a random goal from the *RotateZ* task, and repeating the process). For the creation of this dataset, we do not ensure the rotations in the dataset are unique.

Not all rotations that are comparable to the *RotateXYZ* task can be performed by the policy trained on the *RotateParallel* task, because the goals generated on the *RotateParallel* task only had angles of rotation around the x and y axes contained in the set $\{-\pi, -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\}$ radians. Therefore, when building this second dataset for the *RotateParallel*, this must be taken into account when sampling rotations around the x and y axes. Because the goal space is reduced, 3,200 rotations are sampled into this dataset, but we do not ensure those rotations are unique. As for the Fetch environment tasks, the evaluation is carried out in terms of performance (i.e., success rate), and time efficiency.

5.5.2 Results

The evaluation of the method on the test sets show mixed results. On the one hand, the sequential execution of primitive behaviours did not contribute to improving performance in the environment that required obstacle avoidance (see Tables 5.3 and 5.4), possibly due to the need to learn new behaviours to solve those tasks that were not within the skill set of the chosen set of primitive policies.

On the other hand, decomposing complex 3D rotations into elementary rotations around extrinsic axes can be beneficial when computing resources for training are

Method	Success rate	Estimated training time (hours)
End-to-end HER	0%	≈ 8
End-to-end HGG	0%	≈ 14
End-to-end G-HGG	99.8%	≈ 24
Rectangular traj.	98.5%	≈ 8
Triangular traj.	98.9%	≈ 8

Table 5.3: Success rates on the test set of 1,000 episodes described in Section 5.5.1 for our method and the policy learned using end-to-end HER and HGG on the *FetchPickObstacle*. *Rectangular traj.* and *triangular traj.* represent the curriculum trajectories described in Section 5.4. The estimated training time of the rectangular and triangular trajectories equals the training time of HER trained to solve the *FetchPickAndPlace*, because it is the only primitive used to generate these results.

Method	Success rate	Estimated training time (hours)
End-to-end HER	0%	≈ 16
End-to-end HGG	0%	≈ 72
End-to-end G-HGG	47.6%	≈ 84
Sequential execution	18.4%	≈ 8

Table 5.4: Success rates on the test set of 1,000 episodes described in Section 5.5.1 for our method and the policy learned on the *FetchPickAndThrow*. We name our method “Sequential execution” and its estimated training time is the same as the policy learned to solve the *FetchPickAndPlace* task as it is the only primitive used to generate these results.

limited (see Tables 5.5, 5.6 and 5.7). However, the success rates obtained when applying this approach vary depending on how many elementary extrinsic rotations must be applied to achieve the target 3D rotation.

The next section discusses the results of each of the tasks from both environments, focusing on success rate and sample efficiency, as well as introducing ideas for future work that could improve the results obtained in these environments in particular.

5.5.3 Results discussion

Fetch environment

As Tables 5.3 and 5.4 show, any of the proposed way-point sets were able to surpass the performance of the baseline model policy (i.e., the policy trained using G-HGG). In the *FetchPickObstacle* task, our two proposed trajectories managed to obtain near

Method	RotateParallel	RotateXYZ
End-to-end HER	21.13%	43.20%
End-to-end HGG	48.93%	46.53%
$z - x - z$	53.60%	46.67%
$z - y - z$	54.33%	48.06%
$x - y - x$	49.86%	46.13%
$x - z - x$	50.26%	46.80%
$y - x - y$	48.80%	44.06%
$y - z - y$	51.14%	47.66%

Table 5.5: Success rates on the test set of 1,500 episodes described in Section 5.5.1 for our method and the policy learned on the *RotateParallel* and *RotateXYZ*. End-to-end HER and end-to-end HGG have been trained using 10^7 timesteps of simulated experience.

Method	1 rotation	2 rotations	3 rotations
End-to-end HGG	69.0%	63.40%	47.15%
$z - x - z$	97.5%	88.40%	66.05%
$z - y - z$	97.5%	89.30%	68.20%
$x - y - x$	92.0%	80.10%	62.65%
$x - z - x$	92.0%	83.60%	62.90%
$y - x - y$	94.0%	82.90%	60.65%
$y - z - y$	94.0%	85.20%	62.30%

Table 5.6: Success rates on the test set of 3,200 episodes described in Table 5.2 for our method and the policy learned on the *RotateParallel* experiment, depending on how many extrinsic rotations the robot is required to perform. End-to-end HGG was trained using 10^7 time-steps of simulated experience.

Method	1 rotation	2 rotations	3 rotations
End-to-end HER	57.50%	47.25%	43.85%
$z - x - z$	96.16%	82.21%	51.68%
$z - y - z$	96.16%	88.89%	51.80%
$x - y - x$	93.00%	71.13%	48.75%
$x - z - x$	93.00%	80.35%	49.07%
$y - x - y$	90.50%	73.10%	45.65%
$y - z - y$	90.50%	79.65%	49.42 %

Table 5.7: Success rates on the test set of 6,600 episodes described in Table 5.2 for our method and the policy learned on the *RotateXYZ* experiment depending on how many extrinsic rotations the robot is required to perform. End-to-end HER has been trained using 10^7 time-steps of simulated experience.

perfect success rates, but still fail more often than the end-to-end policy. Despite this, the estimated training time of our method is much lower than the end-to-end approach, as the only primitive that our method uses is the policy trained to solve the *FetchPickAndPlace* task, which only takes approximately 8 hours of training time. However, when adding an instance of the policy trained to solve the *FetchPush* task at the end of the sequenced execution, this performance increases for both of the proposed algorithms to obtain a 99.4% success rate in the case of the rectangular trajectory, and 99.6% in the case of the triangular trajectory. The total estimated training time in this case would be 16 hours, as the time needed to train the policy that solves the *FetchPush* task must also be included.

It is worth noting that despite the fact that DDPG + HER trained on the *FetchPickAndPlace* task is not more sample efficient than DDPG + G-HGG trained on the *FetchPickObstacle* task (one takes 180 policy updates while the other takes 175 approximately, as shown in Table 4.2), the policy updates of G-HGG take a much longer time (one takes 29.23 seconds on average, while the other takes 117 seconds).

Despite the minimal decrease in performance and sample efficiency, the clear superiority in terms of training time efficiency makes the sequential execution of primitive behaviours a better method to deal with a pick and place problem that involves avoiding an obstacle. This is because all the behaviours that the robot must reproduce are present in the set of primitive behaviours that has been previously set. However, if these behaviours were not present in the primitive policies, the results obtained would be worse.

This is exactly the case for the sequential execution of primitive behaviours proposed to solve the *FetchPickAndThrow* task. In order to solve this task, the robot must pick a block and throw it into a bucket that is out of reach of the robotic arm. Because there was no “throw” behaviour in the set of primitives learned, this had to be hard-coded by opening the gripper at specific points of the space, hoping that the cube would fall into the desired bucket. Even though a manual design of this kind

of curriculum would be possible, it would be very time-consuming for the engineer, as every single way-point should be refined to the exact location where opening the gripper would result in the block falling into the desired bucket. Furthermore, even if those way-points were correctly defined with absolute precision, the approach to these way-points would be different depending on the initial position of the block (i.e., if the block is more to the left in the xy space than the chosen way-point, it will make the approach to that point different than if it was more to the right), which results in inconsistent episode outcomes.

While the selection of the set of primitives to solve this task is not perfect, it is not useless either. It is true that in order to properly solve this task, the robot should learn a new behaviour that reproduces the throwing action, but the first half of the problem consists on grasping a block, which is doable using the policy trained to solve the *FetchPickAndPlace* task. A good way to circumvent this problem could be by defining a manual curriculum for the initial time-steps and then letting the RL-based agent learn the new behaviour through trial and error by exploring the environment, making the task easier as it would already start with the block lifted in a stable position.

One more drawback of using a sequence of primitive behaviours is the increase of execution time that comes with this approach. If the way-points are not optimally set, end-to-end learning is likely to find a more optimal and quicker manner to solve the task. As such, it is very difficult to create a sequence of primitives that are executed faster than an end-to-end RL method, and for both tasks the G-HGG can manage to solve the problem with a higher average reward (i.e., it solves the task using fewer actions/environment time-steps on average). This is a general drawback of the method and not the task, meaning that this happens regardless of the task that is being attempted to be solved.

Shadow Dexterous Hand environment

The most notable result of the experiments that can be seen in Tables 5.5, 5.6, and 5.7 is the difference in success rates depending on the dataset used for evaluation. While the best Davenport decomposition configuration on the dataset of 1,500 unique rotations achieves 54.33% success rate on the *RotateParallel* task, and 48.06% on the *RotateXYZ* task, these values increase when using a dataset that contains examples of repeated rotations. This, however, happens by chance, as after running a Two-Sample Proportion test [229] to compare the proportion of success between the two datasets, the p-value was higher than 0.05. However, this does not happen for the policies learned in an end-to-end fashion, as it obtains similar performances in both scenarios, which could imply that while exploring the environment certain rotations are more likely to appear than others. Regardless, the results show a clear superiority in terms of task success rate for the proposed sequential model based on a Davenport decomposition, as it manages to obtain greater success rates than the end-to-end learning baselines for both the *RotateParallel* and the *RotateXYZ* tasks.

While the success rate obtained when requiring only one extrinsic rotation is near perfect, this result does not hold when two or more extrinsic rotations are needed to reach the goal. The reason behind this decrease in performance is because each rotation step expects that the previous rotation goal has been reached perfectly. If the second extrinsic rotation goal is not successfully achieved, the third rotation will automatically fail because the policy employed has been trained to perform rotations around only one axis, but the goal it needs to reach would also require it to correct the previous mistake(s).

There is a possible alternative to overcome this problem: the method of decomposing the rotation could be applied in an iterative manner, with each sequence of three rotations converging on the target goal through repeated attempts. However, doing this would obviously slow down the execution of the manipulation task, which could be a critical consideration depending on the real-world application context for

the manipulation task.

From a qualitative analysis of some of our preliminary results, most of the errors our method produced in the second rotation appeared when dealing with large angle rotations. While the policies learned for the *RotateX* and *RotateY* tasks were very proficient in applying rotations on the range of $[-\frac{\pi}{2}, \frac{\pi}{2}]$ radians, when having to apply rotations in the range of $[-\pi, -\frac{\pi}{2})$ and $(\frac{\pi}{2}, \pi]$ radians, they were not always correctly executed. This was overcome by introducing additional intermediate steps: whenever these policies need to perform a large rotation with a magnitude greater than $\frac{\pi}{2}$ radians, firstly rotate the cube either $\frac{\pi}{2}$ or $-\frac{\pi}{2}$ radians and then apply the remaining smaller rotation to complete the goal. Note that while the results shown in Tables 5.6 and 5.7 include this extra step, the results shown in Table 5.5 do not. This could be another reason why there is a significant difference between the success rates obtained with the datasets that do not ensure rotation uniqueness and the dataset that does ensure this, as the error of the second rotation is carried over to the third extrinsic rotation.

After including this extra intermediate step, the success rate on the test set increased from 67.05% to 88.89% for those 3D rotations that can be decomposed into two extrinsic rotations in the *z-y-z* configuration. Some errors appeared in the third extrinsic rotation, where the agent could not always maintain or deal with tilted cube poses. This happens because the policy trained on the primitive tasks had the cube initialised in a way that it always had one face touching the robotic hand in a stable position. Because the policy never saw this kind of tilted configuration during training, on execution it tends to fail. A solution to this would be introducing these tilted initial configurations and goals into the environments so that the robot sees these configurations during the learning phase. Other solutions could include forcing the agent to learn how to apply rotation vectors instead of reaching a particular pose, making the learning invariant to the initial and target poses; or applying intermediate rotations with a magnitude of $\frac{\pi}{2}$ radians, where the cube would be kept face down for all intermediate rotations until the last rotation, and then apply

the remaining rotation in the last step. However, finding the perfect decomposition by a hand-engineered curriculum will always be a very time-consuming task.

As mentioned in previous Sections, DDPG + HER is able to successfully solve this task when being trained for 38×10^7 time-steps and using 19 CPU cores generating simulated experience in parallel, one worker for each core [22]. Translated to days, this equates to approximately 30 days of training when using an Intel Core i7-6850K CPU @ 3.60GHz. Considering the sensitivity of RL models to hyperparameter selection, this becomes an issue when computing resources are scarce [230]. Instead, our method is trained using the same CPU model with no parallelisation, and the training of each primitive skill takes less than one day (three days if adding the training time of all the primitive skills, or two days if only considering the training time of the primitive skills used to perform one arbitrary 3D rotation, such as z - y - z). In other words, our method is much more sample efficient and time efficient than its end-to-end counterparts.

5.6 Conclusion

Training time is a key consideration for all artificial intelligence research, as often small companies and academia have scarce computing resources. In this chapter, we explore the idea of decomposing complex manipulation tasks into a set of simpler tasks, in the context of a simulated robotic arm and a simulated robotic hand environments, where the skills required to complete the simplified tasks can be learned more quickly.

Through the addition of domain knowledge and the sequential execution of primitive behaviours that reproduce basic manipulation abilities, a robotic arm can avoid obstacles, thus requiring less time overall during the training process. Likewise, a robotic hand can learn how to rotate a cube, with the simulated experience used during training limited to 10^7 time-steps, and obtain greater success rates than the usual end-to-end RL algorithms. However, obtaining the correct primitives is paramount in this context, as there is no additional learning involved in the process.

This means that if, in order to solve a task, the robot must reproduce a behaviour that is not contained in the set of primitive behaviours, the robot is likely to fail to solve the intended task.

Furthermore, when sequentially executing primitive behaviours there is a risk of reaching a *stuck* environment scenario, where one of the intermediate primitives is unable to reach the goal (either because of lack of expertise in the behaviour the policy is capable of producing, or due to an inadequate definition of the intermediate goal), thus blocking the execution of the primitive behaviours that should be subsequently executed. To solve this, a maximum number of time-steps per primitive policy can be set, but there is no guarantee that the following primitive will be able to reach its way-point if the previous primitive failed to do so. In other words, the proposed algorithm is very susceptible to carrying over errors from failed intermediate steps.

Several ideas for future work have been proposed throughout the discussion section, some of them specific to an environment in particular and others more general that could be applied to any task of any environment. These are summarised as follows for the reader's convenience:

- In the context of the Fetch environment tasks, it would be interesting to create an algorithm that starts by using a primitive behaviour policy and, after reaching a particular way-point, let DDPG + HER learn how to finish the execution of the task (e.g., start manipulation problems with the block already grasped);
- The use data augmentation techniques to find policies capable of correcting mistakes made in previous steps should be explored;
- In the context of the Shadow Dexterous Hand environment tasks, it would be interesting to explore iteratively decomposing the target rotation into three new elementary extrinsic rotations and execute again when an attempt fails;
- In the context of the Shadow Dexterous Hand environment tasks, goal and

state exploration could be improved by introducing tilted cube configurations and goals during training in order to make the agent learn these kinds of rotations;

- In the context of the Shadow Dexterous Hand environment tasks, we could encourage the agent to learn how to apply a given goal rotation instead of attempting to reach a particular goal pose. However, as seen in Chapter 3, there are very few benefits to adding intrinsic motivations in the form of reward engineering.

To sum up, sequential execution of primitive behaviours might have better sample efficiency on some occasions (e.g., Shadow Dexterous Hand environment tasks) and it is often a more time-efficient method compared to conventional RL algorithms. However, the performance that this method can achieve is limited by the level of expertise of the primitive policies, meaning that the success rate might be hindered for those tasks requiring behaviours that are not producible by the primitive policies. Furthermore, for all tasks, the execution time of the sequenced execution was longer than the end-to-end solutions, meaning that, depending on the application, the solution generated by this learning method may not be appropriate (e.g., an industrial manufacturing set-up where robots are required to complete tasks as quickly as possible).

5.7 Summary

5.7.1 Revisiting the hypothesis

In Chapter 1, the following hypothesis was stated:

- **Hypothesis 2:** *A major reason why RL sometimes struggles with sample and time efficiency is because robotic manipulation tasks are too complex to learn from scratch (we refer as learning from scratch to initialising all model parameters randomly and then optimising them using a chosen optimisation algorithm).*

Instead, we hypothesise that by dividing complex tasks into smaller simpler tasks, that can be learned quickly and then be executed sequentially, the original complex manipulation task can be effectively solved using few interactions with the environment. If this hypothesis is supported, this approach will result in more modular learning, which not only ensures a more sample and time efficiency method, but also a more interpretable algorithm.

This hypothesis has been supported by the results generated in this chapter, as the sequential execution of was faster to train (about 8 hours) than the end-to-end approaches (24 hours), while achieving both the same performance. However, it could be argued that it can only be partially accepted, as the sequential execution of primitive skills does not always manage to compete in terms of performance with an end-to-end data-driven approach such as DDPG + G-HGG, as the expertise of the model designed directly depends on the expertise encoded in the primitive behaviour policies. This means that, when a new primitive skill is required to complete a task, sequential execution of existing primitive policies will not succeed in completing the task unless this new skill is added to the library of primitives. This idea motivates the work of the next chapter, which aims to dynamically learn and reuse primitive policies in order to solve more complex tasks in a more sample efficient manner.

5.7.2 Summary of chapter contributions

The contributions of this chapter can be summarised as follows:

1. **A novel learning framework for in-hand object 3D rotations based on a sequential execution of primitives learned by any off-policy RL method, which is more sample efficient than learning to solve the task end-to-end.** This framework operates using the Davenport chained rotations decomposition, a method widely used in the field of aeronautics.

Chapter 6

Dynamically learning and re-using primitive behaviours

6.1 Chapter overview

A method that sequentially executes the behaviours of primitive policies is presented in the preceeding chapter. Based on the results obtained it is concluded that using a sequential approach is usually a more sample efficient method in complex environments. However, we acknowledge that the performance of the algorithm is limited by having a fixed set of primitive policies, meaning that the algorithm will fail when a new primitive skill that is not producible by any of the primitive policies is required to complete a task. In contrast, this chapter aims to answer the following research question:

- **Research Question 3:** How can we reuse primitive skills that are easy to learn in order to accelerate the learning of current state-of-the-art (SOTA) methods in robotic manipulation? Are there strategies that accelerate current SOTA without adding computational complexity?

In particular, this chapter explores the idea of dynamically learning and re-using primitive behaviours that produce basic manipulation skills with the aim to facilitate the learning of other more complex manipulation abilities. To do so, a new

algorithm is presented, which we name Q-switch Mixture of Primitives Hindsight Experience Replay (QMP-HER) and evaluate it using the same environments used in the previous chapter. The work presented in this chapter has been accepted for publication at the 2024 10th International Conference on Automation, Robotics and Applications (ICARA) [231]. Section 6.2 describes the motivation for this research, while Section 6.3 presents some related work specific to this chapter. Section 6.4 explains the method and how it is applied to the different environments, Section 6.5 introduces the different experiments explored, and Section 6.6 presents and discusses the results obtained. Finally, Section 6.7 presents the conclusions, and a summary of the chapter is provided in Section 6.8.

6.2 Motivation

Chapter 5 demonstrated some of the benefits, in terms of sample and time efficiency, of decomposing tasks into more simple *sub-goals* or *sub-problems*. However, the method presented in Chapter 5 leaves no room for learning new manipulation abilities, meaning that if the robot requires the execution of a manipulation skill that is not producible by the primitive policies, the robot is likely to fail in solving the task at hand, as was the case of the *FetchPickAndThrow* task of the Fetch environment. In other words, the sequential execution of primitive behaviours is likely to have worse performance than an end-to-end RL-based trained agent, as long as the agent is capable of learning that task, because the end-to-end learning version generates policies that are more tailored and context-specific, even if learning such a policy is inefficient in terms of time and/or resources.

A key reason why performance can worsen when sequentially executing primitive behaviours is because the curriculum applied in the method presented in Chapter 5 is manually designed, meaning that the primitives are only executed when certain pre-scheduled events occur (e.g., the manipulated block reaches a particular way-point in the 3D space). Ideally, an efficient method that makes use of primitive behaviours would make use of the primitive policies whenever its execution is useful

to solve the task the robot is trying to perform, but it should not use them when the execution of these primitives will not lead to a useful environment state, using instead a new learnable policy.

A method with these characteristics would not only make for a more efficient use of the primitive behaviours, but it would also prevent possible mistakes in the curriculum definition performed by the engineer, as the new learnable policy could correct those actions from the primitive behaviours that are not appropriate for finding a solution to the task.

This chapter proposes a novel algorithm that, during exploration, evaluates and selects behaviours generated by policies that are capable of performing simpler primitive skills. In this way, we can accelerate the learning of new more skilful policies capable of performing more difficult manipulation tasks by integrating this method of suggesting and selecting actions using primitive policies into the exploration-exploitation logic that any off-policy RL technique employs. We evaluate our method on four different robotic block manipulation tasks, comparing its success rate, its sample efficiency, and its computation time against an agent trained using the baseline algorithms described in Section 4.4 of Chapter 5.

6.3 Related work

6.3.1 Modular Neural Networks

The ensemble-based approach proposed in this chapter is motivated by the concept of Modular Neural Networks (MNNs) which refers to neural networks that are composed of multiple interconnected modules [232]. A module can be thought of as a self-contained neural network block that performs a specific function or task. These modules can either be executed sequentially [233] or they be combined in different ways to form a larger neural network architecture [234]. As such, each module can have its own set of weights and parameters, and can process different types of data inputs.

The advantages of using modular neural networks include flexibility and improved interpretability. By dividing the neural network into smaller modules, it becomes easier to train each module individually or update specific parts of the network without affecting the entire architecture. Modular networks also make it possible to reuse modules across different tasks or domains, reducing the effort required by the engineer to design new architectures from scratch. Furthermore, modular networks can provide interpretability by allowing researchers to analyse and understand the functionality of each module separately.

MNNs can be used in various applications, such as computer vision [235], [236], natural language processing [237], [238], robotics [233], [239], and RL [240], [241]. By incorporating modular design principles, these networks can offer better performances in complex tasks.

6.3.2 Ensemble of policies

Inspired by the MNN concept of having neural modules that have expertise in specific sub-problems or sub-tasks, the research community has proposed various ensemble of policy algorithms in order to enhance the performance of RL-based agents in complex tasks. An ensemble of policies is a collection of K individual policies $\mathbb{P} = \{\hat{\pi}_1, \hat{\pi}_2, \dots, \hat{\pi}_K\}$ that are used together to solve a task. Each policy in the ensemble $\mathbb{P}$ may have different strengths and weaknesses, and by combining their decisions, the ensemble can benefit from the range and variety of skills possessed by the policies that make up the ensemble, improving the overall performance of the algorithm. In other words, this ensemble approach can help in reducing the inherent uncertainties and risks associated with relying on a single policy.

It is important to note that policies in the ensemble do not necessarily share the data type input, just as for MNNs, meaning the data used by these policies could come from different sources (raw images, text, etc...), or could have different dimensionalities. The only requirement for a policy to be part of the ensemble is that the data input it uses can be obtained from the environment observations, either

directly or by processing and extracting features from the information contained in the environment observation.

There are different strategies to create such ensembles. For example, in [242], the authors define a hierarchical policy, with a high-level task policy that decides which lower-level primitive policy of the ensemble to use, and a low-level parameter policy that determines how to instantiate the chosen primitive, in order to learn how to order the selection of these policies and their parameters over time. In other words, the proposed algorithm learns how to sequentially apply those primitives and the duration of their execution, but as happens for our solution proposed in Chapter 5 the agent has no room to learn new skills; it can only reproduce sequences of primitive behaviours.

Other more recent methods benefit from the estimated ‘Q-values’ in their methods, which estimate the quality of an action a_t for a given state s_t when the Q-function estimation is correctly learned. SUNRISE [243] is an ensemble mechanism that makes use of the Q-value estimation by training N independent actor-critic based agents in a task (note that those actor-critic architectures do not necessarily need to be designed equally, meaning there can be DDPG-based agents, a SAC-based agents, etc.), and then selecting the candidate action of the agent that maximises a *weighted Bellman backup* equation, which is calculated using the mean and variance of the Q-values obtained by each candidate action using the critics of each of the agents of the ensemble:

$$a_t = \arg \max_a \{Q_{mean}(s_t, a) + \lambda Q_{std}(s_t, a)\} \quad (6.1)$$

This idea of filtering actions based on the Q-value has also been used in different imitation learning frameworks and offline RL methods to filter noisy actions or unwanted behaviours, becoming a very popular approach in recent years [244]–[248].

A more recent method is the Q-switch Mixture of Policies (QMP) proposed by Zhang *et al.* [249], which uses a Q-filter in a multi-task setting designed to simultaneously find policies that each solve a different task that share the observation space

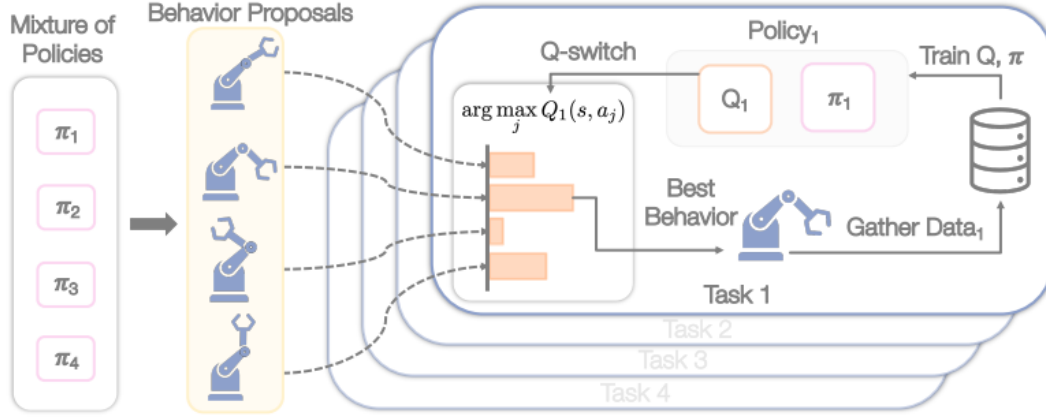


Figure 6.1: Schematic representation of the behaviour of the QMP algorithm. Image taken from Zhang *et al.* original paper [249].

with the other tasks, with the aim of identifying shareable behaviours between policies. During exploration, each of the policies of the ensemble proposes a candidate action, but only the action that maximises the Q-value associated with the target policy for the task being solved is executed (see Figure 6.1).

In this paper, we use a QMP to improve the sample efficiency of HER, but instead of sharing behaviours between policies that are being simultaneously trained in the same environment, we make use of fixed primitive policies that do not necessarily share the observation or/and the goal spaces with the target task (see Figure 6.2).

6.4 Q-switch Mixture of Primitives Hindsight Experience Replay

The focus of this work is to improve the sample and time efficiency of the state-of-the-art off-policy RL models for robotic manipulation tasks by reusing simpler primitive skills that can be more easily learned. These primitive skills are policies obtained by training a DDPG agent using HER in simpler tasks.

We introduce a set of L objectives $\mathbb{O} = \{o_1, o_2, \dots, o_L\}$, which serve as optional intermediate goals for the K primitive policies $\mathbb{P} = \{\hat{\pi}_1(s, g), \hat{\pi}_2(s, g), \dots, \hat{\pi}_K(s, g)\}$. These objectives can be thought of as flexible waypoints (as in Chapter 5) that guide the primitive policy execution.



Figure 6.2: Schematic representation of the proposed QMP-HER algorithm. For each time-step during exploration, the candidate action that maximises the estimated Q-value is selected and executed. Actions for each time-step are proposed by each primitive policy as well as the target policy that is being updated each cycle.

Algorithm 4 Q-switch Mixture of Primitives Hindsight Experience Replay

Given: Off-policy RL agent $\mathbb{A} \equiv \pi(s, g), Q^\pi(s, g, a)$.
 K primitives: $\mathbb{P} = \{\hat{\pi}_1(s, g), \hat{\pi}_2(s, g), \dots, \hat{\pi}_K(s, g)\}$
Replay Buffer $\mathcal{R}$

- 1: Initialise $\mathbb{A}$
- 2: Load $\mathbb{P}$
- 3: Define a set of L objectives $\mathbb{O} = \{o_1, o_2, \dots, o_L\}$ for $\mathbb{P}$
- 4: **for** epoch = 0 to N **do**
- 5: **for** episode = 1 to M **do**
- 6: Sample a goal g and an initial state s_0
- 7: **for** $t = 0$ to $T - 1$ **do**
- 8: **for** $k = 1$ to K **do**
- 9: **for** $l = 1$ to L **do**
- 10: $s', g' \gg$ Adjust state and/or goal based on o_l
- 11: Sample candidate action $a_t^{k,l}$ from $\hat{\pi}_k(s', g')$
- 12: $\mathcal{A}_t \gg$ Store candidate action $a_t^{k,l}$
- 13: **end for**
- 14: **end for**
- 15: Sample candidate action a_t^π from $\pi(s_t, g)$
- 16: $\mathcal{A}_t \gg$ Store candidate action a_t^π
- 17: Select best action a^* using Eq. 6.2, applying ϵ -greedy
- 18: Execute action a^* and observe state s_{t+1}
- 19: $\mathcal{R} \gg$ Store transition $(a^*, s_t, s_{t+1}, g, r_t)$
- 20: **end for**
- 21: **end for**
- 22: Apply HER logic and update $\pi(s, g), Q^\pi(s, g, a)$
- 23: **end for**

At each time-step during exploration, every primitive policy $\hat{\pi}_k$ generates candidate actions for each objective o_l , resulting in a set of candidate actions $\mathcal{A}_t$ that includes:

$$\mathcal{A}_t = \{a_t^\pi, a_t^{1,1}, a_t^{1,2}, \dots, a_t^{K,L}\}.$$

Here, a_t^π is the action proposed by the target policy $\pi(s, g)$. The critic $Q^\pi(s, a)$ evaluates all actions in $\mathcal{A}_t$, and the action with the highest Q-value is selected for execution:

$$a^* = \arg \max_{a \in \mathcal{A}_t} Q^\pi(s_t, g, a). \quad (6.2)$$

During the warm-up phase (prior to learning), Q-values are initially uninformative, so the agent relies on actions from the primitive policies and uses ϵ -greedy exploration (i.e., occasional random actions with probability ϵ). After warm-up, the

target policy $\pi(s, g)$ is also considered in the selection process.

Initially, actions from primitive policies dominate the decision-making process. As the target policy improves over training, its actions achieve higher Q-values, naturally transitioning the agent’s behaviour to that of the original HER algorithm. This progressive shift enables efficient learning while leveraging the guidance of primitive policies during early exploration.

We name this algorithm *Q-switch Mixture of Primitives Hindsight Experience Replay* (QMP-HER); see Algorithm 4 for details.

6.4.1 Network

The QMP-HER algorithm is a method that can be used in conjunction with any off-policy RL algorithm. Furthermore, the proposed QMP could also be trained in conjunction with other more advanced variants of HER, like HGG and G-HGG.

For the experiments presented in Section 6.5, we use the standard DDPG + HER algorithm in conjunction with our QMP variation (which is named DDPG + QMP-HER for simplicity for the remainder of this chapter). The network has the same architecture as the network used in Chapter 5, and has different hyperparameters depending on the task to be solved (see Appendix A.3.1 for more details, or Section 6.5 for a full description).

6.5 Experiments

6.5.1 Environments

Both of the environments explained in Section 4.3 are used throughout the different experiments of this Section. To improve the sample efficiency of DDPG + HER, we aim to make use of the behaviour learned by the baseline models trained on the simplest tasks (see Section 4.4.2). Therefore, the method we present in this chapter cannot be used in the *FetchReach*, *RotateZ*, *RotateX* nor *RotateY* environments, as they are the simplest for each of the environments and, therefore, there are no

primitives that can be used to make the agent learn these tasks faster. However, once these policies have been learned there is some knowledge of the dynamics of the robot that has been acquired by this behaviour policy, and so they can be used to more quickly learn how to perform other more complex tasks.

6.5.2 Experimental setup

Fetch environment

Because the observation space of the *FetchReach* environment is smaller than the observation space of the other three environments, it is necessary to remove the object state information from the observations of these other environments in order to use the policy learned on the *FetchReach* task, and instead use this information as the objective for the policy learned on the *FetchReach* (see Table 6.1) for more information about the Fetch environment tasks observations).

The method proposed in this chapter requires a heuristic definition of the objectives for the primitive policies. These objectives, in the context of the Fetch environment, are defined based on regions of interest in the 3D space relative to the object and the goal position, and might vary depending on the target task. For example, in order to learn a push or lifting policy, a robot can benefit from approaching the object from all possible directions. However, when learning a policy in tasks with physical obstacles, the robot benefits from manipulating the object in an elevated position, avoiding the obstacles.

To solve the *FetchPush* task we define the objectives by setting a number of different goals for the *FetchReach* primitives. For each observation of the *FetchPush* task, we adjust it by removing the object information from the input and set as goals for the *FetchReach* primitive policy the location of the object and the episode goal position, but we also include objectives that point toward the object with slight deviations, so that the agent can choose touching the object from the left, right, front and behind. To solve the *FetchPickAndPlace* task, we make use of the same primitives and objectives as used to solve the *FetchPush* task. For this experiment,

we also use the target policy during warm-up time-steps as it is necessary for gripper state exploration, because the policy trained on the *FetchReach* task does not have the gripper control activated.

FetchPickObstacle is a task of sufficient complexity that HER cannot solve it. However, the task has many similarities with the *FetchPickAndPlace* task and the policy learned in that task can be useful to accelerate the learning pace. We perform two different experiments: the first one being a manually designed curriculum using a triangular trajectory as described in Section 5.4, but limiting its execution to 20 time-steps, and for the remainder of the episode the agent executes the actions predicted by its actor network $\pi(s)$. We call this framework *PickPlace&HER*. In the second experiment, we use our method defining two objectives for the *FetchPickAndPlace* primitive, one that aims to pick the block and move it above the obstacle and a second one aiming for the goal position, which would ideally result in following the same triangular trajectory, but we leave more room for the algorithm to improve the behaviour as it can always choose to use the *target* policy during exploration.

The *FetchPickAndThrow* task is much more challenging than the previously mentioned tasks, as the robot must lift up the cube and throw it into a basket, meaning that the goal space is a discrete set of eight regions instead of the continuous goal space in previous experiments. Similar to the *FetchPickObstacle* task, we believe that this task can be accelerated by starting with the object grasped by the gripper. Therefore, we run two different experiments, a first one running a similar strategy as we defined for the *PickPlace&HER* exploration algorithm, but in this case grasping the object and moving it to a centred and elevated position, then letting the target policy explore (we still call this method *PickPlace&HER* for simplicity, even though different curricula are applied in the first time-steps). The second experiment involves our method using a combination of primitive behaviours: we use the policy learned on the *FetchPickObstacle* task, setting as the objective the same goal that was sampled for that episode (i.e., setting the objectives as the 3D coordinates

corresponding to the center of the target bucket); and the policy learned on the *FetchPickAndPlace* task, setting as objectives a centred and elevated position and the sampled goal for that episode.

To calculate the success rate, we do not apply the QMP-HER logic (behaviour policy). Instead, we only test the actions predicted by the actor of the target policy $\pi(s)$ that is being updated to learn the execution of the task end-to-end.

Shadow Dexterous Hand environment

Two different experiments using a QMP were performed on the *RotateParallel* task. In the first experiment, we decompose the target rotation goal each episode into a chain of three consecutive rotations around the axes (i.e., Davenport chained rotations). Because there is no unique way to perform this decomposition, we choose the configuration $z-y-z$ as it produces the best performance according to the results obtained in Section 5.5. In other words, we use two instances of the primitive policy trained to solve the *RotateZ* task and an instance of the primitive policy trained to solve the *RotateY* task, setting as target goals each of the intermediate rotation targets of the Davenport decomposition.

For the second experiment, we set an instance of the policies trained to solve the *RotateZ*, *RotateX*, and *RotateY* tasks, but this time setting as target goal the same as the sampled goal for the episode. The reasoning behind this experiment is to only let the agent explore the environment when the target goal does not belong to any of the goal spaces of the primitive policies, but instead use the primitive policies whenever the sampled goal for the episode belongs to the goal space of any of the primitive policies. To put it in simpler words, whenever the sampled goal for the episode consists on a rotation around one axis, this goal during exploration will be executed by the primitive policies trained to solve the *RotateZ*, *RotateX*, and *RotateY* tasks, because they will propose the actions that obtain a higher Q-value for that goal.

For the experiments in the Shadow Dexterous Hand environment, we train both

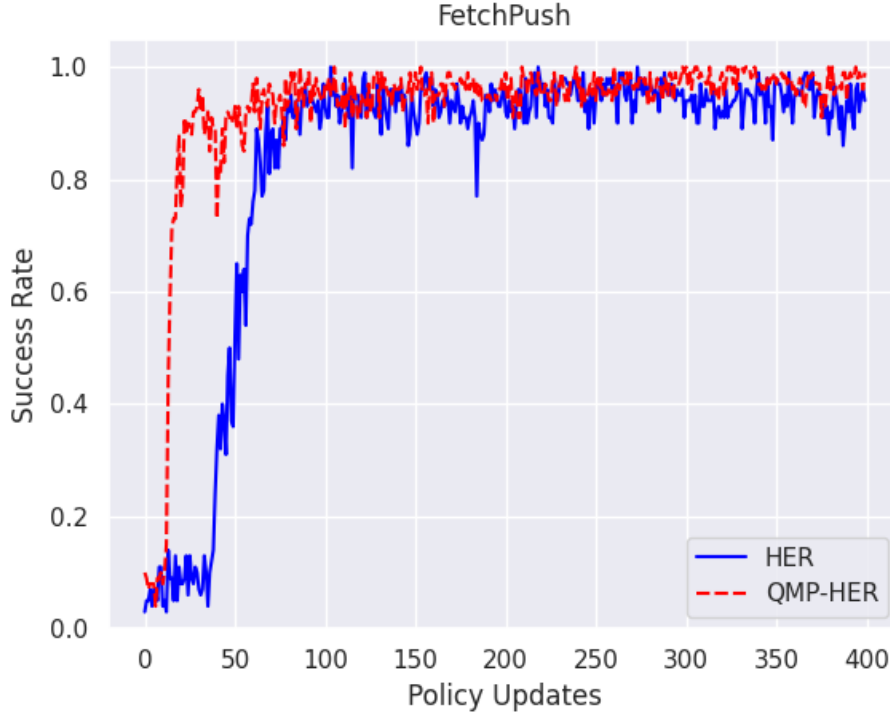


Figure 6.3: Success rate of the target policy $\pi(s, g)$ for each policy update for the experiment described in Section 6.5 on the *FetchPush* task.

DDPG + QMP-HER and DDPG + QMP-HGG in order to evaluate the advantages and disadvantages of using our method combined with both algorithms. In particular, in the case of DDPG + QMP-HER we use the primitives policies explained in the previous two paragraphs, while for the DDPG + QMP-HGG we use an instance of each of the primitive policies (i.e. policies trained on the *RotateZ*, *RotateX*, and *RotateY* tasks) to propose a candidate action along with the target policy for each of the intermediate goals generated by HGG.

6.6 Evaluation

6.6.1 Fetch environment manipulation tasks

The results demonstrate that our method was able to accelerate the learning of HER, as it is capable of reaching a convergence point faster than HER in all tasks of the Fetch environment (see Figures 6.3, 6.4, 6.5, and 6.6).

In the case of the *FetchPush* task, our method is able to obtain a success rate

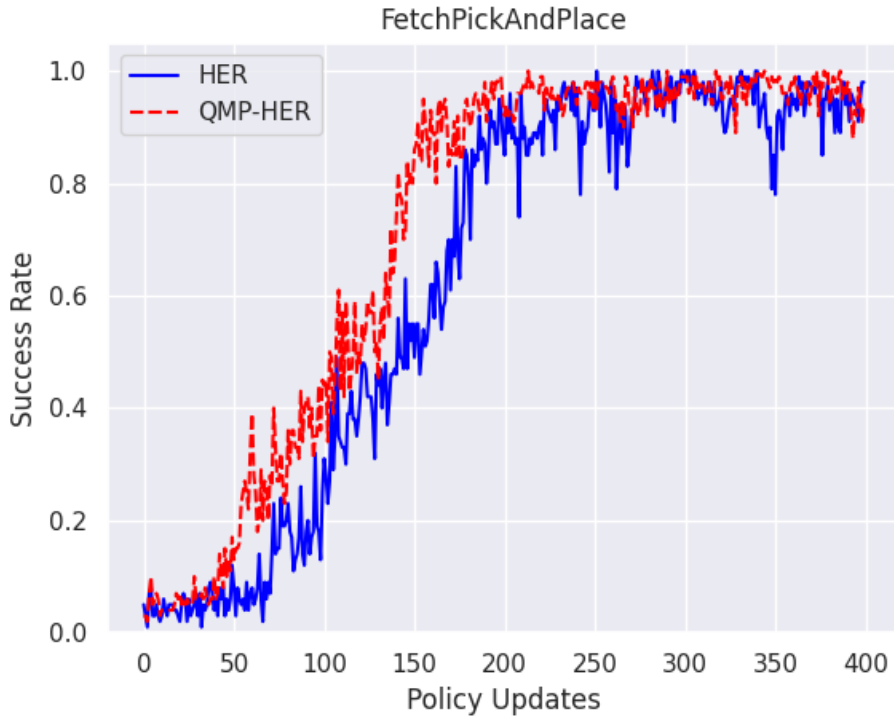


Figure 6.4: Success rate of the target policy $\pi(s, g)$ for each policy update for the experiment described in Section 6.5 on the *FetchPickAndPlace* task.

of 80% in fewer than 20 policy updates, while it takes approximately 60 policy updates for HER to achieve the same performance. Our method is also able to achieve a 95% success rate faster, as it only needs 25 policy updates, while HER requires approximately 70 updates (see Figure 6.3). A similar result is achieved for the *FetchPickAndPlace* task, where the success rate of our method reaches 95% accuracy using approximately 150 policy updates, while HER takes approximately 180 policy updates, proving that our method increases learning sample efficiency (see Figure 6.4).

The results obtained for the different experiments for the *FetchPickObstacle* task can be seen in Fig 6.5. While the task is too difficult for HER, which completely fails, our approach was able to reach an average of 95% success rate in approximately 60 policy updates, compared to the approximately 175 updates needed for G-HGG. Furthermore, the average time per policy update of our model is 86 seconds, while for G-HGG it takes 117 seconds. The *PickPlace^ℰHER* behaviour policy, while still working better than HER, proved to be inferior to the other methods explored,

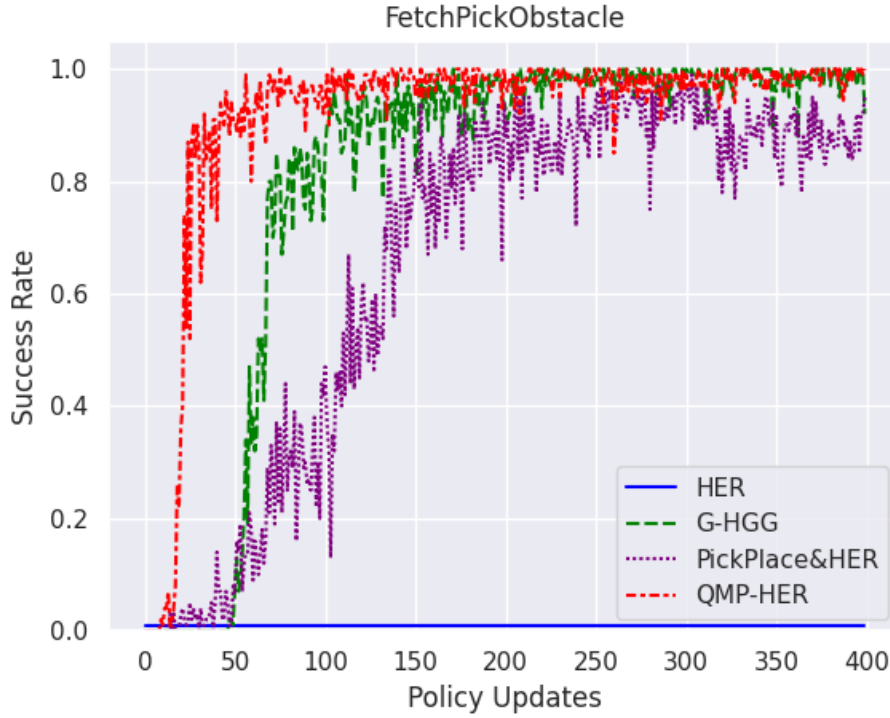


Figure 6.5: Success rate of the target policy $\pi(s, g)$ for each policy update for the experiment described in Section 6.5 on the *FetchPickObstacle* task.

as the learning pace is slower. This is because the *PickPlace&HER* method uses the triangular trajectory as a *behaviour* policy, which is a very proficient policy, leading to very little exploration and a lot of exploitation while interacting with the environment and, therefore, to very few hindsight goals generated due to errors during exploration.

It is also important to note that the policy learned using a QMP-HER did not always end up having a behaviour that reproduces a triangular trajectory as we were expecting based on the objectives set for this experiment. Instead, it learned that the most efficient actions to perform at the beginning of the episode were the actions that would follow the triangular trajectory, but as soon as the cube was over the obstacle the agent learns that the faster and more efficient way to reach the point is to throw the cube. However, this behaviour did not occur for all episodes, as there were times when the behaviour of the policy learned followed the triangular trajectory set by the episode objectives.

For the *FetchPickAndThrow* experiments, we found that our method and G-HGG

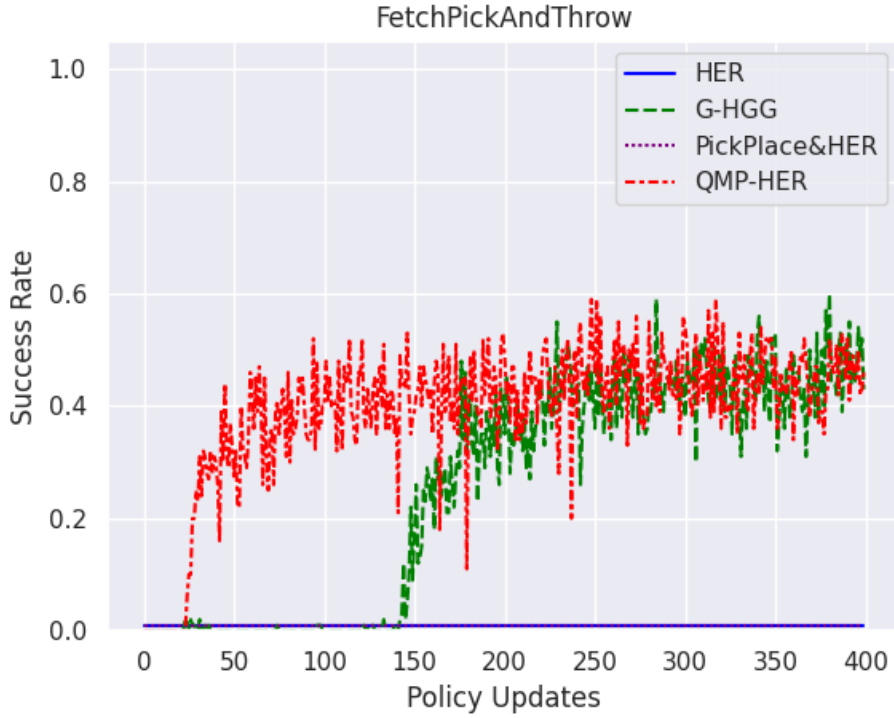


Figure 6.6: Success rate of the target policy $\pi(s, g)$ for each policy update for the experiment described in Section 6.5 on the *FetchPickAndThrow* task.

are able to achieve a 50% success rate, while the other methods are unable to learn any useful policy (see Figure 6.6). After analysing the behaviour of the learned policy, it can be seen that for both methods the robot is observed to be capable throwing the block to the buckets of the first row nearest the robot base, but always fails when aiming for the second row buckets that are further away, explaining why approximately 50% of attempts fail. This happens because during exploration the block almost never ends up in one of the second row buckets, making it impossible to generate hindsight goals for HER in these cases. Despite the similar results in terms of performance, our method is more sample efficient, as it converges faster than G-HGG, and more significantly, for each policy update our method takes 72.8 seconds on average, while the average time per update when using G-HGG is 396.5 seconds.

The behaviour of the policy learned on this task is also interesting. For the buckets in the first row, the agent is able to learn to reach to the limit of the buckets, which correspond to the full area of reach of the robotic arm, and then

	<i>FetchPush</i>	<i>FetchPickAndPlace</i>	<i>FetchPickObstacle</i>	<i>FetchPickAndThrow</i>
HER	70	180	N/A	N/A
G-HGG	N/A	N/A	175	275
<i>PickPlace@HER</i>	N/A	N/A	250	N/A
QMP-HER	25	150	60	250

Table 6.1: Summary of results in terms of sample efficiency for the different experiments carried out in the Fetch environment. The table shows the **number of policy updates until convergence**, understanding convergence in this scenario to mean when the success rate curve plateaus with a policy fully or partially capable of solving the task.

throws toward the goal location, most of the times being successful. When the goal is to throw the block into a bucket that is in the second row (i.e., most likely using the *FetchPickObstacle* primitive policy), the robot instead persists in trying to reach the buckets of the second row but never opening the gripper to throw toward the bucket (i.e., most likely using the *FetchPickAndPlace* primitive policy).

Balancing exploration and exploitation

One of our earliest experiments consisted of balancing the ratio of exploration and exploitation when selecting the actions during the exploration phase, as over reliance on primitives could lead to convergence at a sub-optimal policy. For more information about the exploration-exploitation trade-off problem see Section 2.3.4.

For the previous experiments, the ϵ value of the ϵ -greedy algorithm was set to 0.3. However, using this value with the QMP-HER resulted in some occasions of bias overestimation that reduced the learning pace, as can be seen in Figure 6.7. Instead, when adjusting this value to be 0.5 (i.e., taking random actions with a probability $\epsilon = 0.5$), the learning accelerates as the exploration becomes more efficient and, thus, prevents overfitting to a sub-optimal policy. For this reason, for all tasks in this chapter, the ϵ value is set to 0.5.

Analysis of primitive behaviour selection during exploration

The key aspect of the method presented in this chapter is the exploration-exploitation decoupling, meaning we have a *behaviour* policy that we use for exploration that

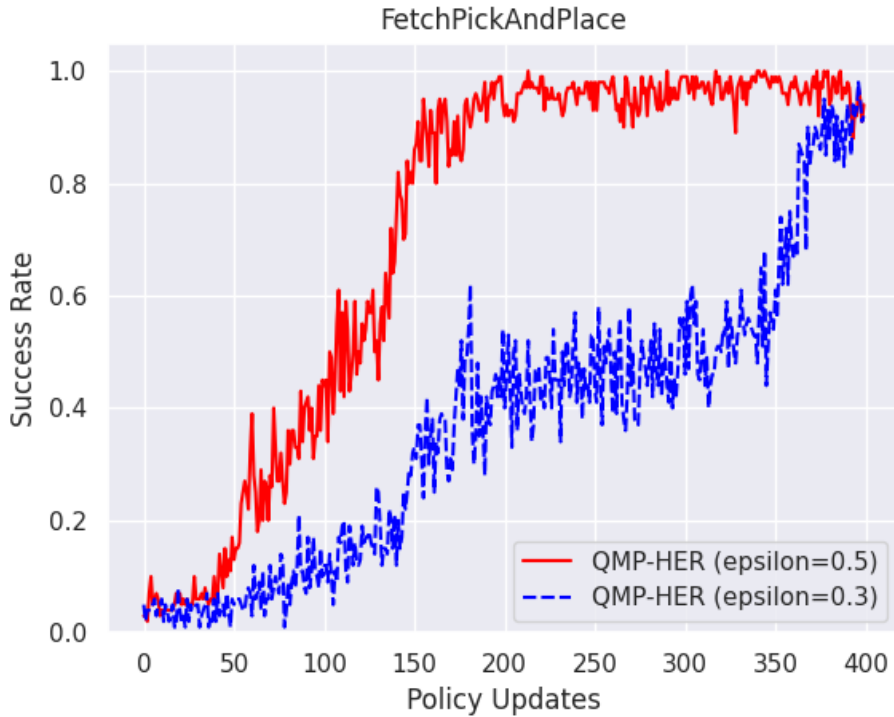


Figure 6.7: Success rate evolution per policy update on the *FetchPickAndPlace* task using the QMP-HER algorithm with different ϵ values for the ϵ -greedy algorithm.

is different from the *target* policy that we update during the learning phase. In particular, we use as *behaviour* policy a QMP that ensembles instances of policies that produce primitive behaviours. In this section, we explore how the usage of these primitive behaviours evolves throughout the learning process, as one of our hypotheses stated in Section 6.4 is that the proposed QMP-HER method, by the end of the learning, should have the same behaviour as original HER as the *target* policy becomes more proficient in solving the task (i.e., it learns a better Q-function for the problem it is trying to solve, and thus, the actions proposed by the *target* policy should more often be the most task-adapted).

To this end, a visualisation mechanism is designed in a confusion matrix format, in which the usage of the different policies is represented as a percentage of actions taken by that policy after a particular number of policy updates. In order to generate this visualisation, we discard those actions that are generated from the ϵ -greedy algorithm, as these actions only provide extra exploration and analysing their usage would not give any insight into the learning of the Q-function. Furthermore, if the

target policy suggests the same candidate action as one of the primitive policies, and that action is taken according to Equation 6.2, this action is only attributed to the *target* policy. If both primitive policies from the QMP suggest the same candidate action, and that action is finally taken, this action is attributed to the primitive policy that is located more to the left on the x -axis of Figures 6.9 and 6.11, by design.

Figure 6.9 shows the usage over time of the *target* policy and the primitives used in the QMP for the *FetchPickObstacle* task, as explained in Section 6.5. In particular, *pick* refers to the instance of the policy trained to solve the *FetchPickAndPlace* task with the objective to lift the block and place it over the obstacle, while *place* refers to the instance of the policy trained to solve the *FetchPickAndPlace* task with the goal to place the block at the goal position.

We choose to see the usage after the first policy update in order to have an approximate idea on what policies are used at the beginning of the exploration, and then explore the usage of the primitive policies after 25 and after 50 policy updates, as this is the period where the algorithm is performing its learning (i.e., the success rate starts to increase in value, as can be seen in Figure 6.5).

The results displayed in Figure 6.9 and 6.10 show that the use of the primitive behaviours happen mainly during the first policy updates, which are the most exploratory interactions of the learning process, and when the Q-function begins to be learned. However, as soon as the success rate starts to increase (i.e., approximately after the 25th policy update), the critic network begins estimating a better Q-function and the actions that the *target* policy proposes are usually more task-adapted. This is clearly the case after the 50th policy update, when the *target* policy is capable of solving the task by itself, as most of the actions taken during the exploration phase come from that policy.

Nonetheless, there are still some occasions where the actions taken are those suggested by the primitive behaviours (see Figure 6.10), more often being the *pick* primitive policy. This can be explained after visually inspecting the behaviour of

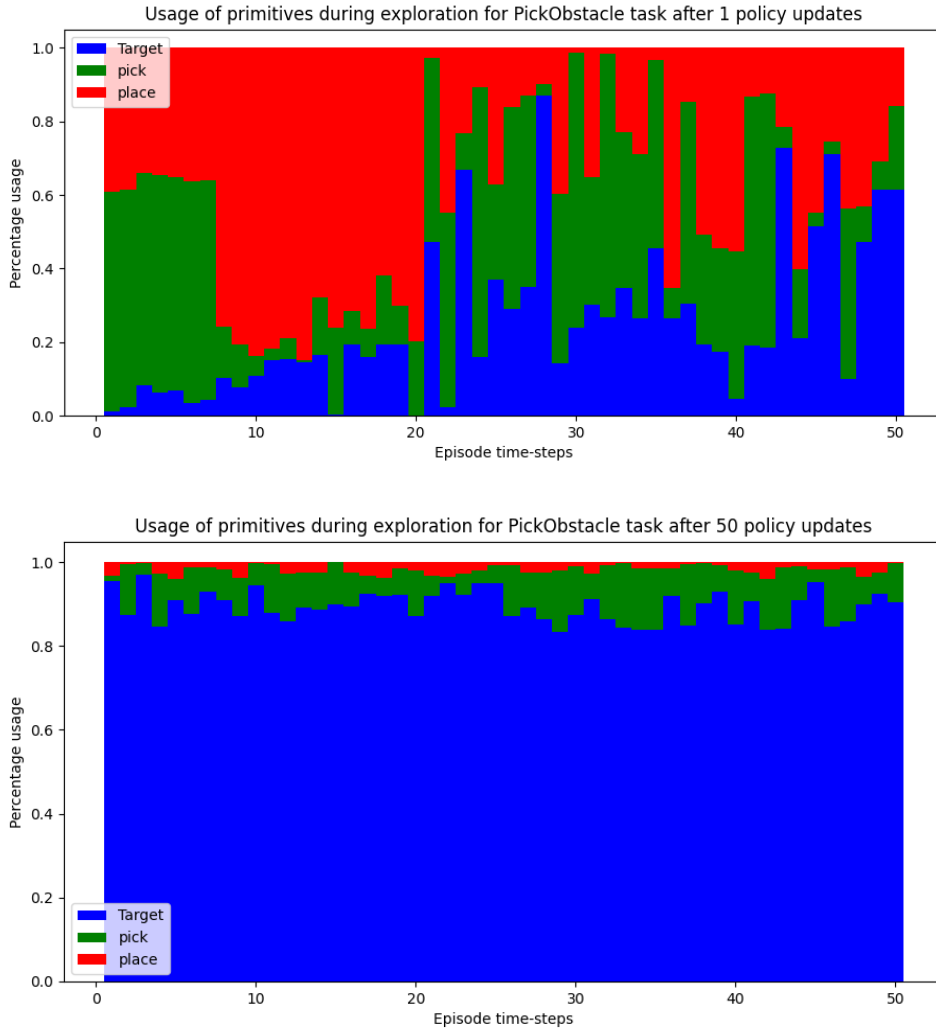


Figure 6.8: Visualisation of the average usage over 25 episodes during exploration of the behaviours of the primitive policies and the target policy after 1 policy update and 50 policy updates on the *FetchPickObstacle* task (simplified to *PickObstacle* in the plot to shorten the title). In this figure, *pick* refers to an instance of the policy trained to solve the *FetchPickAndPlace* task with the objective to lift the block over the obstacle, while *place* refers to an instance of the policy trained to solve the *FetchPickAndPlace* task with the goal to place the block at the goal position.

the *target* policy, as it eventually finds out that the fastest way to complete the task is not placing the block in the goal position but throwing the block so that it falls in the goal position. However, it always lifts the block and goes over the obstacle, meaning that the actions suggested by the *pick* primitive policy are more task-adapted and sometimes more efficient than the *target* policy that keeps being updated.

Figure 6.11 shows the usage over time of the *target* policy and the primitive

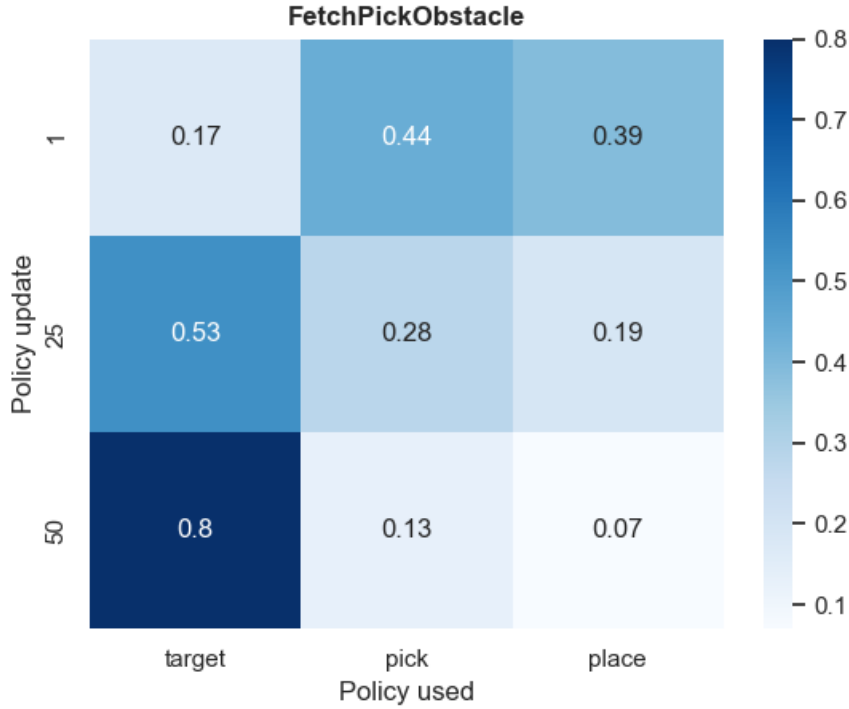


Figure 6.9: Usage of the target policy $\pi(s, g)$ and the primitive behaviour instances *pick* and *place* during the exploration phase after the 1st, the 25th and the 50th policy updates. In this figure, *pick* refers to an instance of the policy trained to solve the *FetchPickAndPlace* task with the objective to lift the block over the obstacle, while *place* refers to an instance of the policy trained to solve the *FetchPickAndPlace* task with the goal to place the block at the goal position.

behaviours used in the QMP for the *FetchPickAndThrow* task, as explained in Section 6.5. In particular, *obstacle* refers to an instance of the policy trained to solve the *FetchPickObstacle* task, setting the objective as the 3D coordinates corresponding to the centre of the target bucket. Similarly, the objectives set for the *pick_goal* are the same, but in this case are executed by an instance of the policy learned to solve the *FetchPickAndPlace* task. Lastly, the *pick_centre* refers to an instance of the policy learned to solve the *FetchPickAndPlace* task with the objective to move the block to a centred elevated location.

In this case, we decided to analyse the behaviour after policy updates that are more spread out in time (i.e., after the 1st, the 50th and the 200th policy updates), because for this task the QMP-HER algorithm takes more time to learn than in the case of the *FetchPickObstacle* task, as shown in Figure 6.6.

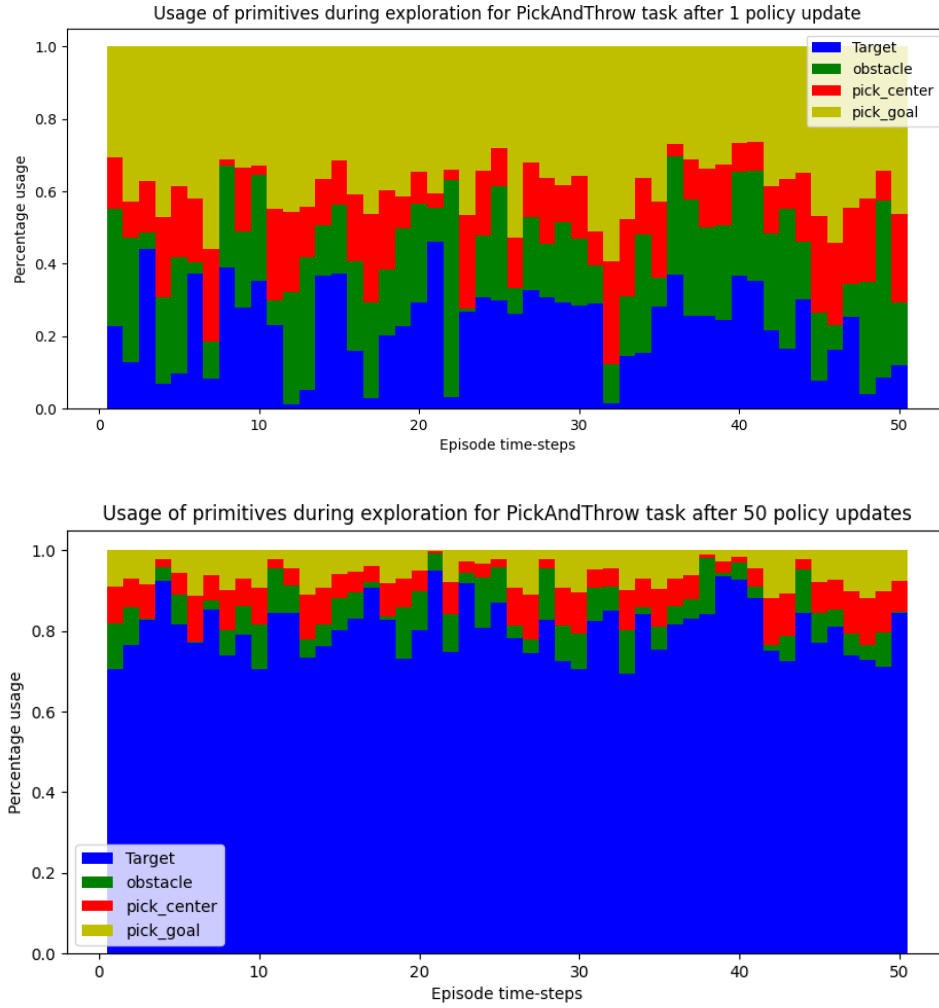


Figure 6.10: Visualisation of the average usage over 25 episodes during exploration of the behaviours of the primitive policies and the target policy after 1 policy update and 50 policy updates on the *FetchPickAndThrow* task (simplified to *PickAndThrow* in the plot to shorten the title). In this figure, *obstacle* refers to an instance of the policy trained to solve the *FetchPickObstacle* task, setting the objective as the 3D coordinates corresponding to the centre of the target bucket, and *pick_center* and *pick_goal* refer to instances of the policy trained to solve the *FetchPickAndPlace* task, with the objectives to move the block to a centred elevated position and towards the the 3D coordinates corresponding to the centre of the target bucket, respectively.

The results displayed in Figure 6.11 show a very similar behaviour of the algorithm as in the previous case: as soon as the *target* policy starts to be good enough to begin having episodes with positive outcomes, the *target* policy also begins specialising for the estimated Q-function that the critic network is learning and thus, the actions the *target* policy proposes are more often chosen as the best by Equation 6.2, or they just simply coincide with the action proposed by one of primitives,

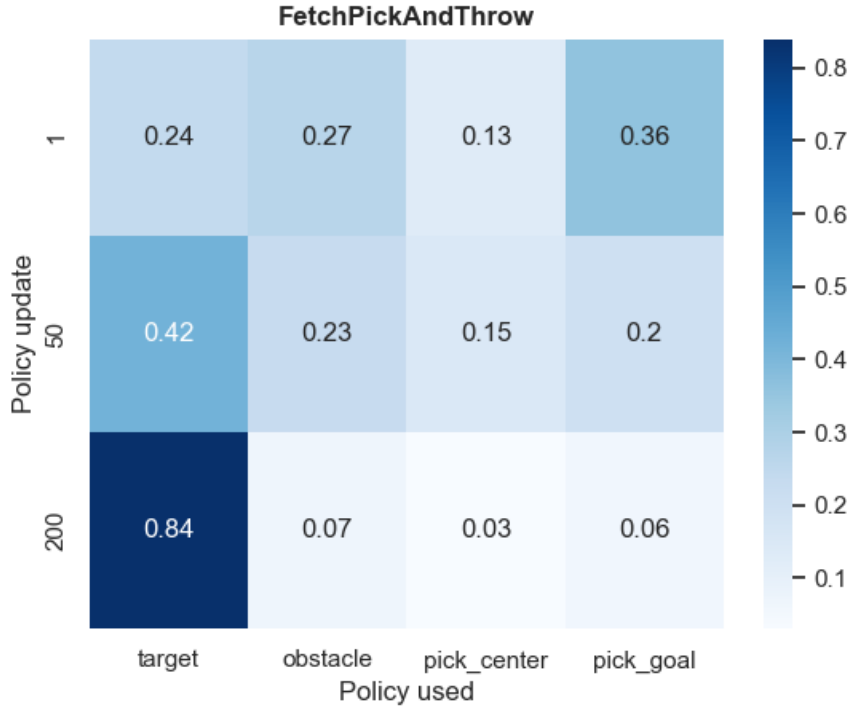


Figure 6.11: Usage of the target policy $\pi(s, g)$ and the primitive behaviour instances *obstacle*, *pick_centre*, and *pick_goal* during the exploration phase after the 1st, the 50th and the 200th policy updates. In this figure, *obstacle* refers to an instance of the policy trained to solve the *FetchPickObstacle* task, setting the objective as the 3D coordinates corresponding to the centre of the target bucket, and *pick_centre* and *pick_goal* refer to instances of the policy trained to solve the *FetchPickAndPlace* task, with the objectives to move the block to a centred elevated position and towards the the 3D coordinates corresponding to the centre of the target bucket, respectively.

meaning that the *target* policy has learned to replicate the primitives behaviour. This way, by the end of the learning process almost all actions during exploration are originated from the *target* policy.

Analysis of sensitivity to primitive behaviours selection

In this subsection we analyse the sensitivity of the QMP-HER method to the selection of primitive behaviours. To do this, we propose various ensembles of primitives $\mathbb{P}$ along with their objectives, $\mathbb{O}$, in order to solve the different tasks of the Fetch environment that have been explored in this chapter.

The first experiments performed involved the use of 7 different instances of the policy learned to solve the *FetchReach* task, setting as goals for these instances the

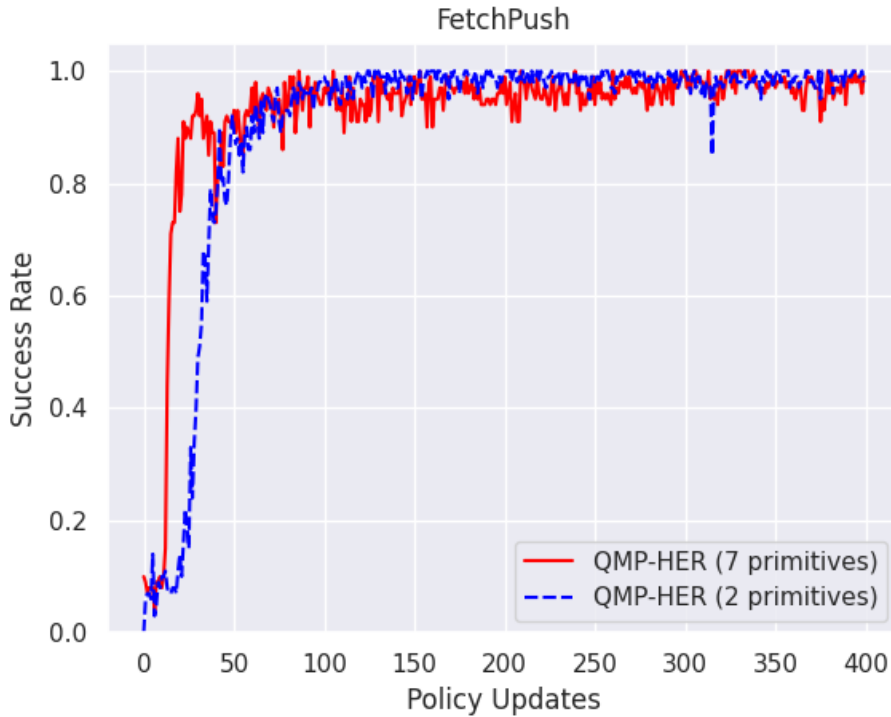


Figure 6.12: Effect of having the 7 primitives described in Section 6.5 as part of the QMP (i.e., using instances of the policy learned to solve the *FetchReach* with slight deviations with respect to the object position, as well as the target goal) to solve the *FetchPush* task (red line), and only using two instances of the *FetchReach* task to reach the object position and the target goal as part of the QMP (blue line).

same goal of the episode, as well as the current position of the block and slight deviations of the current block position with the aim to touch the block from the left, right, front, behind and from the top. The second set of experiment removes the deviated goals, using only the current block position and the episode’s goal. Figures 6.12 and 6.13 show the results of this experiment for the *FetchPush* and *FetchPickAndPlace* tasks, respectively.

While for the *FetchPush* task, the use of more primitives proves beneficial, for the *FetchPickAndPlace* task there is no significant difference in terms of sample efficiency, and even the 2 primitives option is slightly more sample efficient. This is because touching from different angles in a push task during the exploration phase can lead to a more varied collection of interactions with plausible results very quickly. On the other hand, when solving the *FetchPickAndPlace* task, it is more useful always approaching the block the same manner, as most goals are in the air

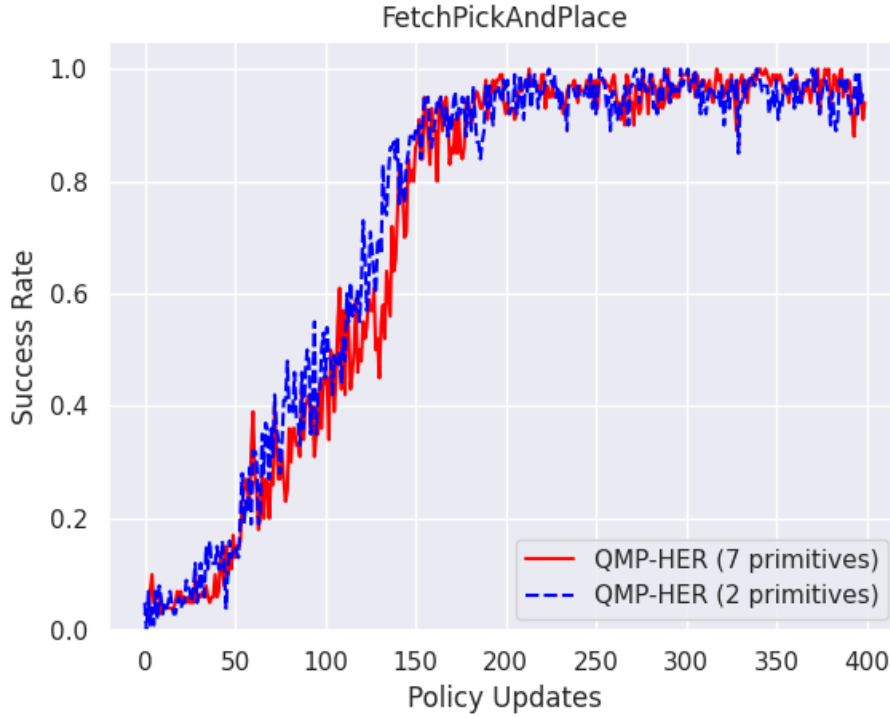


Figure 6.13: Effect of having the 7 primitives described in Section 6.5 as part of the QMP (i.e., using instances of the policy learned to solve the *FetchReach* with slight deviations with respect to the object position, as well as the target goal) to solve the *FetchPickAndPlace* task (red line), and only using two instances of the *FetchReach* task to reach the object position and the target goal as part of the QMP (blue line).

and require grasping the object first.

For the *FetchPickObstacle* task, we compare the sample efficiency and the performance of QMP-HER when using two instances of the policy trained to solve the *FetchPickAndPlace* task, as shown in Figure 6.5, with the sample efficiency and the performance when using those same policy instances, as well as an instance of the policy trained to solve the *FetchPush* with the objective set to be the same as the goal of the episode g . The reasoning behind defining this new objective is to refine the final position of the cube.

The results of this experiment can be seen in Figure 6.14. As can be observed, the combination of primitives did not prove more efficient than simply using two instances of the policy trained to solve the *FetchPickAndPlace* task. This is because during the first iterations, the agent does not know that there is an obstacle in the middle that must be avoided, meaning that at the beginning of the training process

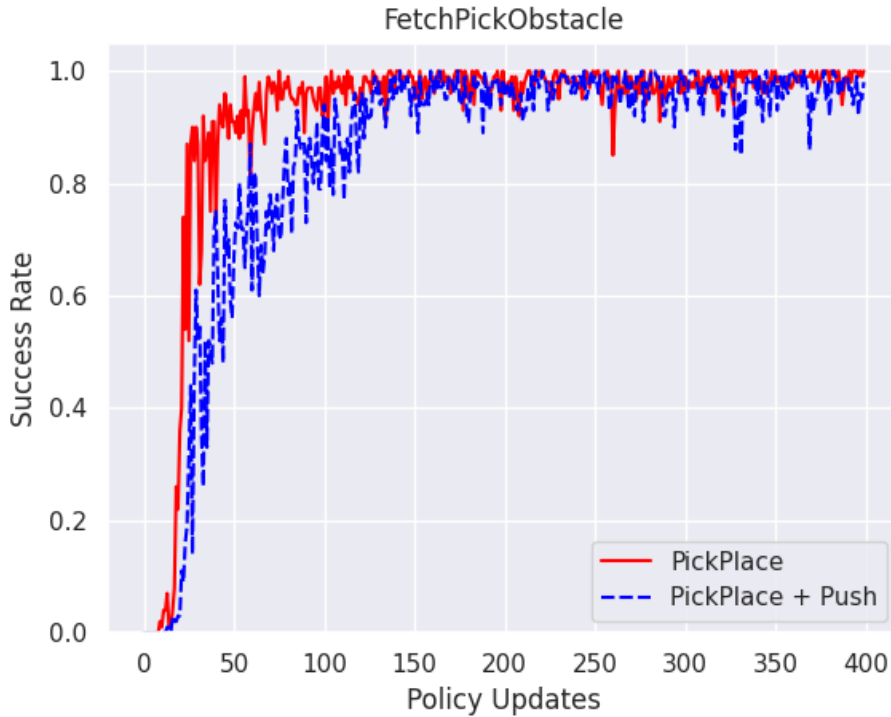


Figure 6.14: Effect of having two instances of the policy trained to solve the *FetchPickAndPlace* task (red line), and a combination of the policy trained to solve the *FetchPickAndPlace* task and the policy trained in the *FetchPush* task (red line) as part of the QMP used for the behaviour policy in order to solve the *FetchPickObstacle* task.

the agent understands that the actions proposed by the policy trained to solve the *FetchPush* task would be more optimal to solve the *FetchPickObstacle* task, and it is only after several failed iterations that it learns that the most optimal action to take at the beginning of the episodes is to use the primitives with a *pick&place* behaviour. Despite this, both methods are more sample and time efficient than the baseline model G-HGG, as adding this extra primitive with the *pushing* behaviour only adds on average 2 more seconds per policy update (i.e., approximately 88 seconds per policy updates, compared to the 117 seconds required by G-HGG).

We perform a similar experiment for the *FetchPickAndThrow* task. In that task, we compare the sample efficiency and the performance of using an instance of the policy learned to solve the *FetchPickAndPlace* task, and an instance of the policy learned to solve the *FetchPickObstacle* task, both combined and independently, as shown in Figure 6.6. In particular, the policy trained on the *FetchPickAndPlace*

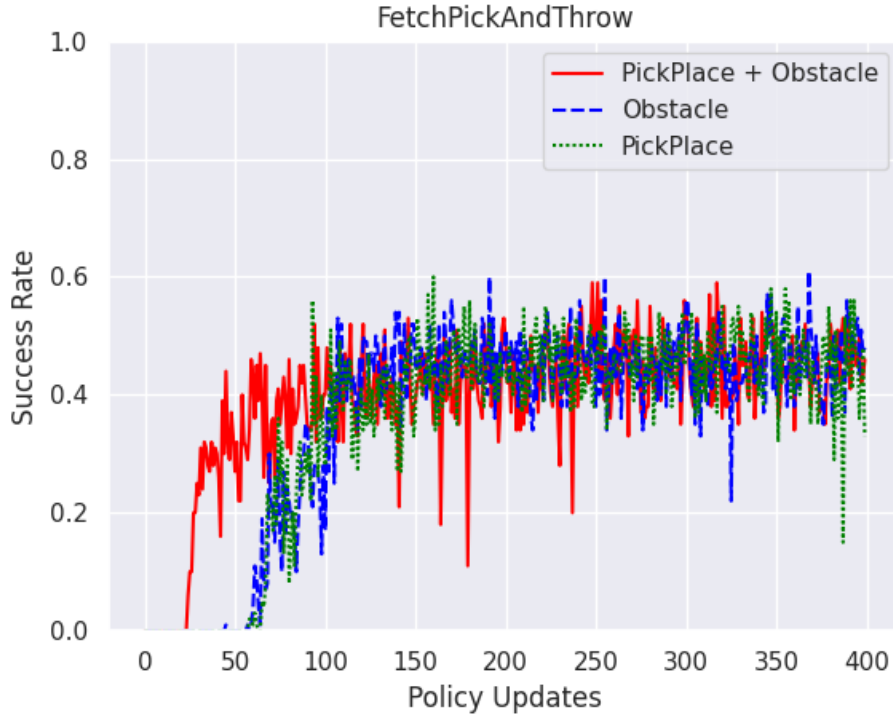


Figure 6.15: Effect of having only an instance of the policy trained to solve the *FetchPickAndPlace* task (green line), only an instance of the policy trained to solve the *FetchPickObstacle* task (blue line) and a combination of both (red line) as part of the QMP used for the behaviour policy in order to solve the *FetchPickAndThrow* task.

task uses the 3D coordinates corresponding to the center of the target bucket as the objective, as well as a centered elevated position, while the policy learned to solve the *FetchPickObstacle* task uses the 3D coordinates corresponding to the center of the target bucket.

Figure 6.15 shows the results of these experiments. As can be appreciated, the use of both types of primitives in the ensemble $\mathbb{P}$ helps the robot start learning earlier than just using one primitive behaviour in the QMP, regardless of the type. It is worth noting that when it comes to the behaviour observed of the final policy, there are no differences that we could determine between them.

The difference in sample efficiency might be because the policy that was learned to solve the *FetchPickObstacle* task can deal better with the buckets in the middle of the first row, as this policy sees the buckets as obstacles, while the policy trained on the *FetchPickAndPlace* task is initially more efficient with dealing with

the buckets on the corners of the first row because it tries to approach the 3D coordinate corresponding to the buckets until they touch the bucket and it falls due to the collision. By using both of these policies, it benefits at the beginning of the environment interactions, while by the end of the learning process it already finds that the most efficient and faster way to solve the task is by throwing the block (if nothing else, for the buckets situated in the first row).

6.6.2 In-hand manipulation tasks

The results obtained by QMP-HER and QMP-HGG for the experiments run on the Shadow Dexterous Hand environment tasks show a small improvement in terms of performance with respect to the baseline models found in Chapter 4, as can be seen in Figures 6.17, and 6.19.

In Figure 6.16, the results for the different experiments run on the *RotateParallel* task are presented. As can be observed, the results in this environment show that our proposed methods had very similar learning pace as their respective baseline models. In other words, the experiments using a QMP-HER had very similar performances to the performance of HER, while the experiments using a QMP-HGG had very similar performances to the performance of HGG. Despite this, for both methods the use of Davenport primitives proves slightly superior than both the baseline model and the alternative set of objectives explored.

This means that in this task, most of the contribution to the learning is originated in the goal generation part of the HGG algorithm, as can be seen in Figure 6.17, where the success rate of the QMP-HGG is almost the same as the original HGG. This is because the goal space in this task is not fully continuous, as the x and y rotation components must be multiples of $\frac{\pi}{2}$ radians, and thus the goal generation from HGG, which ensures the generated goals belong to the goal space, makes the exploration more effective than when using the primitives of the QMP. For this same reason it is more difficult in this task to learn a useful policy using HER, as many of the achieved poses during exploration will not be valid hindsight goals because

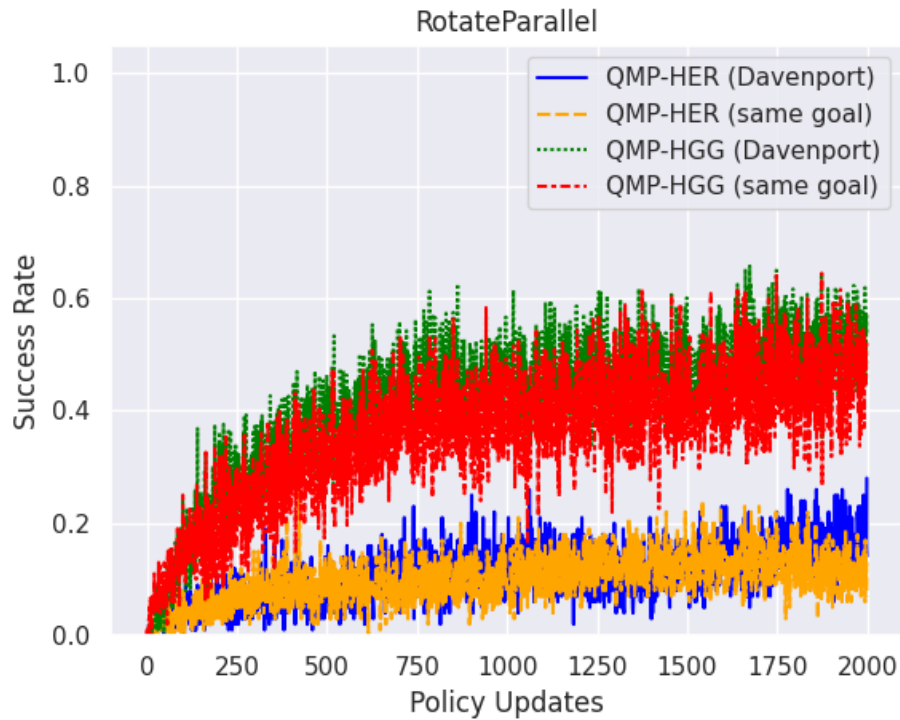


Figure 6.16: Success rate per policy update of the different policies trained in the different experiments explained in Section 6.5 for the *RotateParallel* task of the Shadow Dexterous Hand environment.

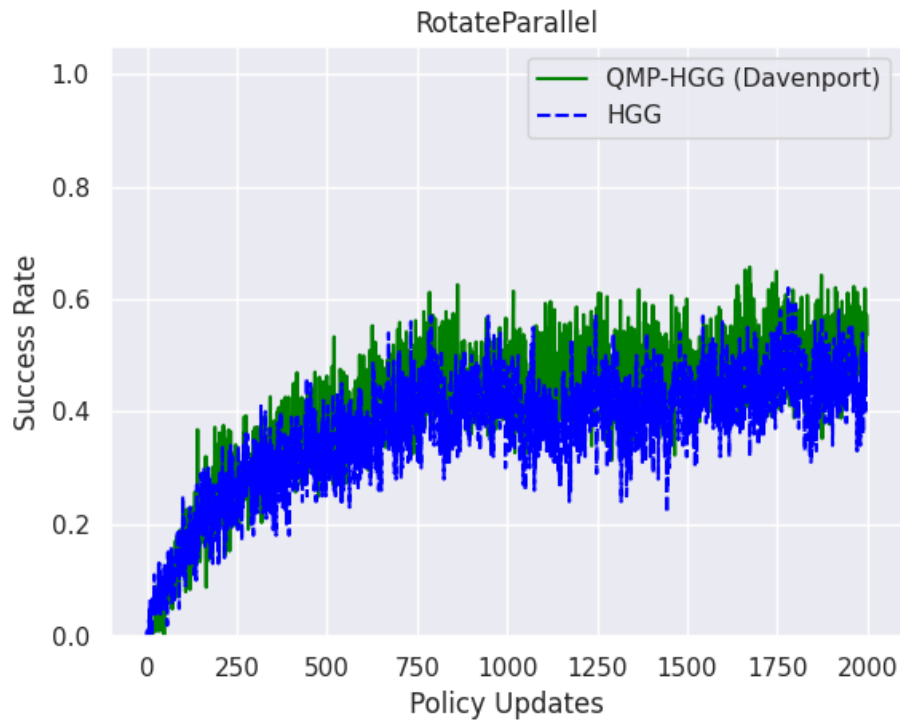


Figure 6.17: Comparison of the best policy found by of proposed QMP-HGG and the baseline model based on HGG for the *RotateParallel* task that was found in Chapter 4.

those poses do not belong to the goal space.

It is noticeable that the QMP-HER with Davenport rotations works in the *RotateXYZ* task, but it is not as successful in the *RotateParallel* task (see Figures 6.16 and 6.18). Again, this is a combination of the nature of the goal space of the *RotateParallel* task and the fact that the Davenport decomposition is never fully performed during exploration, but instead a mix of actions originated from the different primitives and the *target* policy is more likely to happen. Furthermore, even if the full sequenced chain of rotations was performed, this sequential execution does not have a perfect success rate in solving these tasks, meaning that the exploitation that the agent can benefit from by applying a sequential execution of a Davenport rotation decomposition is also limited to the proficiency of the primitive behaviours.

In the *RotateXYZ* task, all experiments had similar results, except the QMP-HER that uses instances of the policies trained to rotate the cube around the axes with the objectives set to be the same as the goal of the episode, as can be observed in Figure 6.18. A similar result is obtained for the QMP-HGG with those objectives, as the learning pace is very similar to the learning pace of the standard HGG. This is because it is not probable that the goal generated by the environment consists of one simple rotation around the axis, meaning that for most of the learning process these primitives are unhelpful.

On the other hand, the QMP-HER with Davenport rotations manages to obtain similar performance as the QMP-HGG. However, when analysing the behaviour of the learned policies, there are no signs of it using a sequence of Davenport chained rotations, not even in the first iterations. This is because at the beginning of the learning process the critic benefits from positive outcomes to understand what the goal of the task is, and the actor, instead of replicating the behaviour, gets refined to try to solve the task faster, as a consequence of the reward design that encourages the robot to solve the task as fast as possible. Another reason why this happens is because, in the *RotateXYZ* task, HER can operate regardless of the achieved cube orientation, as all cube orientations are included in the goal space, meaning that

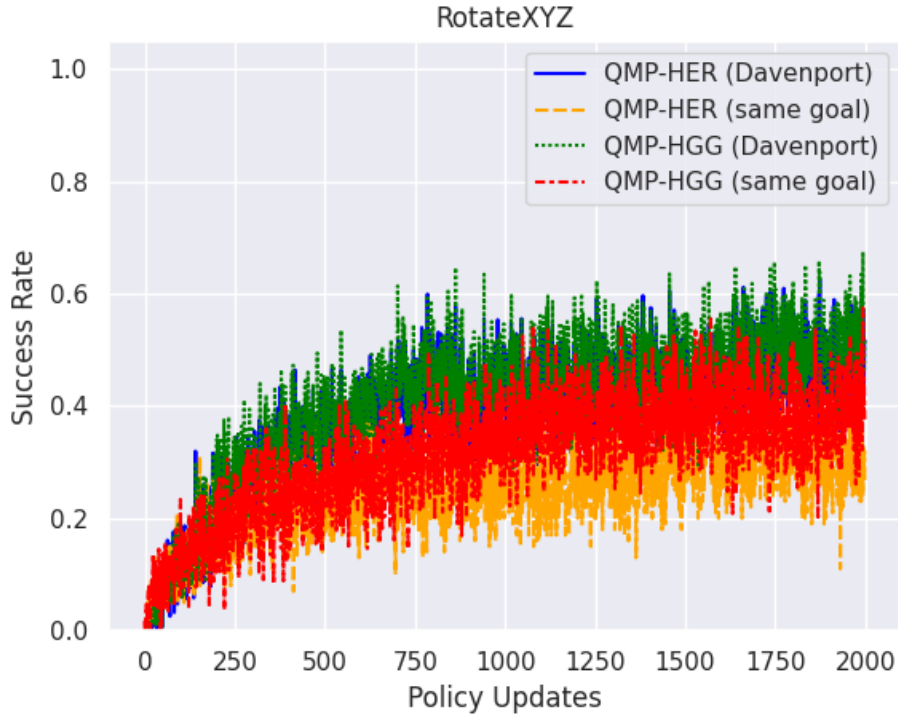


Figure 6.18: Success rate per policy update of the different policies trained for the different experiments explained in Section 6.5 for the *RotateXYZ* task of the Shadow Dexterous Hand environment.

even with failed experiences it can learn positive outcomes, which happens more often than the sequenced Davenport decomposition. Despite this, the benefits of the Davenport occasional exploitation during the exploration phase are evident by the increase of performance.

There is a considerable difference between these methods regarding their computational efficiency, as the QMP-HER (regardless of the objectives) takes approximately 4 days to train, while the QMP-HGG variant takes approximately 7 days.

Despite the performance and the sample efficiency for the *RotateXYZ* task improving when compared to original HGG and HER, it is evident by looking at Figure 6.18 that the success rate stagnates at around 55% and the learning becomes slower from that point. Despite this, as can be seen in Figure 6.19, the QMP-HGG is a more efficient method than the baseline model that was selected in Section 4.4 (i.e., HER) when analysed in terms of performance. However, it is much more time-intensive, as the total training time of this method is approximately 7 days, compared to the

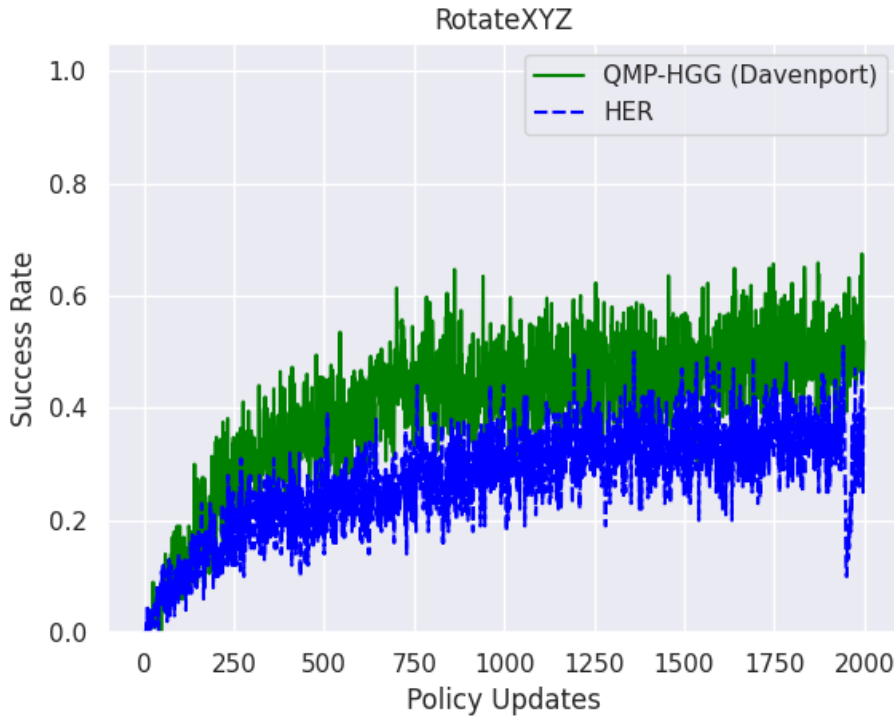


Figure 6.19: Comparison of the best policy found by of proposed QMP-HGG and the baseline model based on HER for the *RotateXYZ* task that was found in Chapter 4.

approximately 3 days that takes training a policy using standard HER.

In both tasks, the set of primitives that contain the Davenport decomposition obtain better performances than the set of primitives that have the objectives set to be the same as the goal for the episode. This suggests that the algorithm benefits from highly exploratory *behaviour* policies, as the behaviours of the primitives are not visibly produced by the learned *target* policy as mentioned earlier. In other words, it is likely that the Davenport rotations are contributing to the learning by adding noise to the action decision-making process rather than by exploiting a trajectory that solves the task. For this reason, it is also possible that DDPG is not the best learning algorithm to solve this task, as it generates deterministic actions instead of learning a probabilistic approach for the action prediction.

Analysis of primitive behaviour selection during exploration

The aim of this section is to analyse the usage of the primitive behaviours for the best models found in the previous section. We use the same strategy used in the analysis

of the Fetch environment result in order to generate the matrix visualisations: for Figures 6.21 and 6.23, we discard from the analysis those actions that are generated from the ϵ -greedy algorithm, if the *target* policy suggests the same candidate action as that of the primitive policies and that action is taken according to Equation 6.2, this action is only attributed to the *target* policy; and if two or more primitive policies from the QMP suggest the same candidate action, and that action is finally taken, this action is attributed to the primitive policy that is located more to the left on the x -axis of Figures 6.21 and 6.23 by design.

Figure 6.21 shows the usage of the *target* policy, as well as the instances of the primitive behaviours that can solve each of the rotations of the z - y - z Davenport decomposition. In particular, z_1 and z_2 are instances of the policy trained to solve the *RotateZ* task with the goal set to be the first and third rotations, respectively, of the Davenport decomposition, while y_1 refers to an instance of the policy trained to solve the *RotateY* task with the goal set to be the second rotation of the Davenport decomposition.

As can be observed, the use of the primitives of the QMP-HGG follows a very similar behaviour as the experiments carried out for the Fetch environment tasks, as the *target* policy eventually becomes the most used policy during exploration because of the actor network becoming specialised to the critic network (i.e. the actions proposed by the *target* policy end up having a higher Q-value because this is the policy that is being updated to solve the task). The policies that reproduce primitive behaviours are mainly used in first iterations of the learning process (and in a completely random way, as can be seen in Figure 6.20), and their behaviour is not later reproduced by the learned *target* policy based on a visual inspection of its behaviour. Despite that, the performance was better when using this set of primitive behaviours. This is because the goal of the primitives is not set to be the same as the goal of the episode, which means that the primitive policies by themselves do not contribute to generating exploitable behaviours, and instead contribute to generating a more exploratory *behaviour* policy. This helps the critic

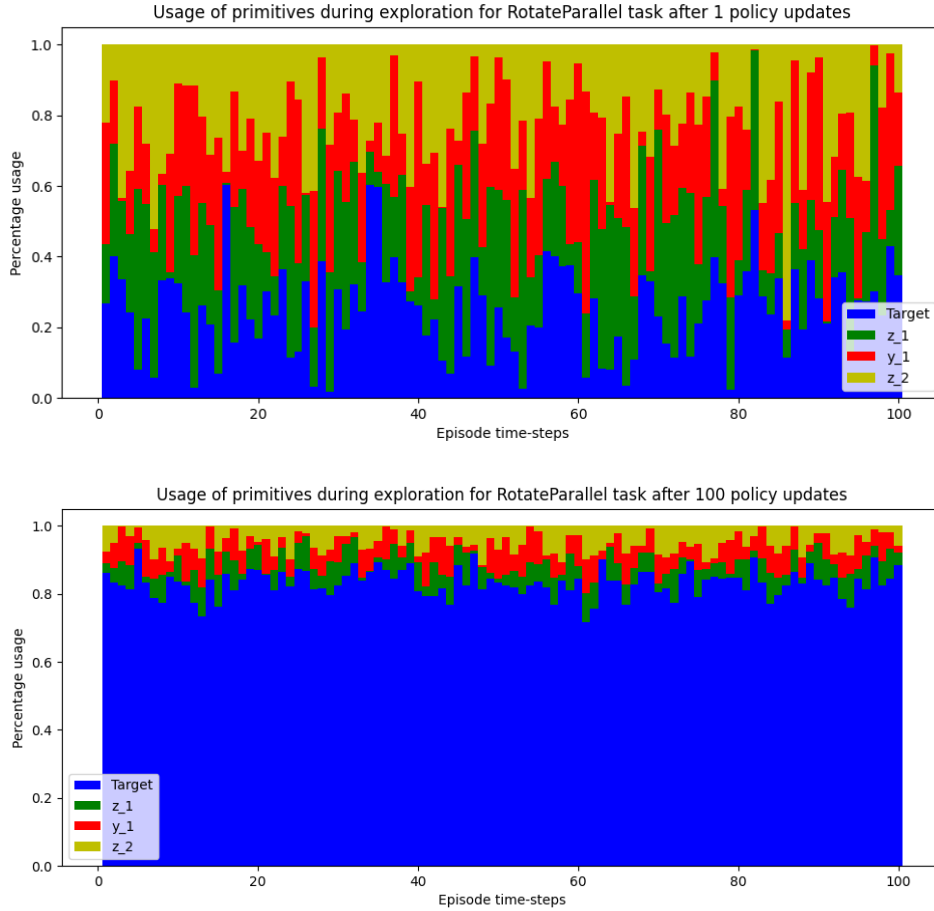


Figure 6.20: Visualisation of the average usage over 25 episodes during exploration of the behaviours of the primitive policies and the target policy after 1 policy update and 100 policy updates on the *RotateParallel* task.

network to become more proficient in estimating the Q-function and, thus, later the actor network (i.e., the *target* policy) also becomes slightly better at solving the task by itself.

This claim is also supported by the use of the primitives in the *RotateXYZ* task, which can be seen in Figures 6.22) and 6.23. Similarly to the policy trained to solve the *RotateParallel* task, the behaviour of the *target* policy when trained on the *RotateXYZ* does not appear to be using a sequence of rotations around the axes based on a visual inspection of its behaviour, and despite that the performance increases with respect to the baseline model. Furthermore, similarly to the *RotateParallel* task, the use of the primitive behaviours happens mainly after the first policy updates, but later on the *target* policy is the most used by a substantial margin.

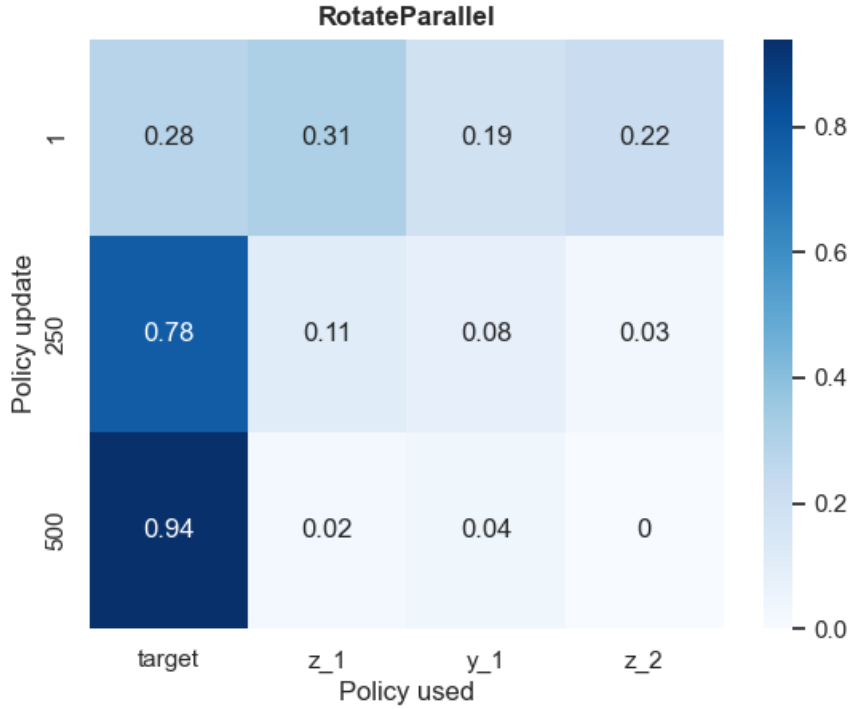


Figure 6.21: Usage of the target policy $\pi(s, g)$ and the primitive behaviour instances z_1 , y_1 , and z_2 during the exploration phase after the 1st, the 250th and the 500th policy updates for the *RotateParallel* task. In this figure, z_1 and z_2 refer to an instance of the policy trained to solve the *RotateZ* task with the goal set to be the first and third rotations, respectively, of the z - y - z Davenport decomposition, while y_1 refers to an instance of the policy trained to solve the *RotateY* task with the goal set to be the second rotation of the z - y - z Davenport decomposition.

6.7 Conclusion

In this chapter, we present a novel algorithm that we call Q-switch Mixture of Primitives Hindsight Experience Replay (QMP-HER). This algorithm improves the sample efficiency and the computation time of agents trained with HER or HGG by making use of primitive skills (i.e., policies) learned by mastering simpler tasks. In particular, it manages to learn using 100 fewer policy updates in the most complex tasks of the Fetch environment (*FetchPickObstacle*, *FetchPickAndThrow*) taking on average only 72.5 seconds compared to the 396.5 seconds it takes the G-HGG. In the Shadow Dexterous Hand, it increases the success rate by 20% while taking only 4 days to train, compared to the 7 days for HGG. Instead of enforcing a curriculum for the behaviour policy using these primitives, the critic network of the target policy

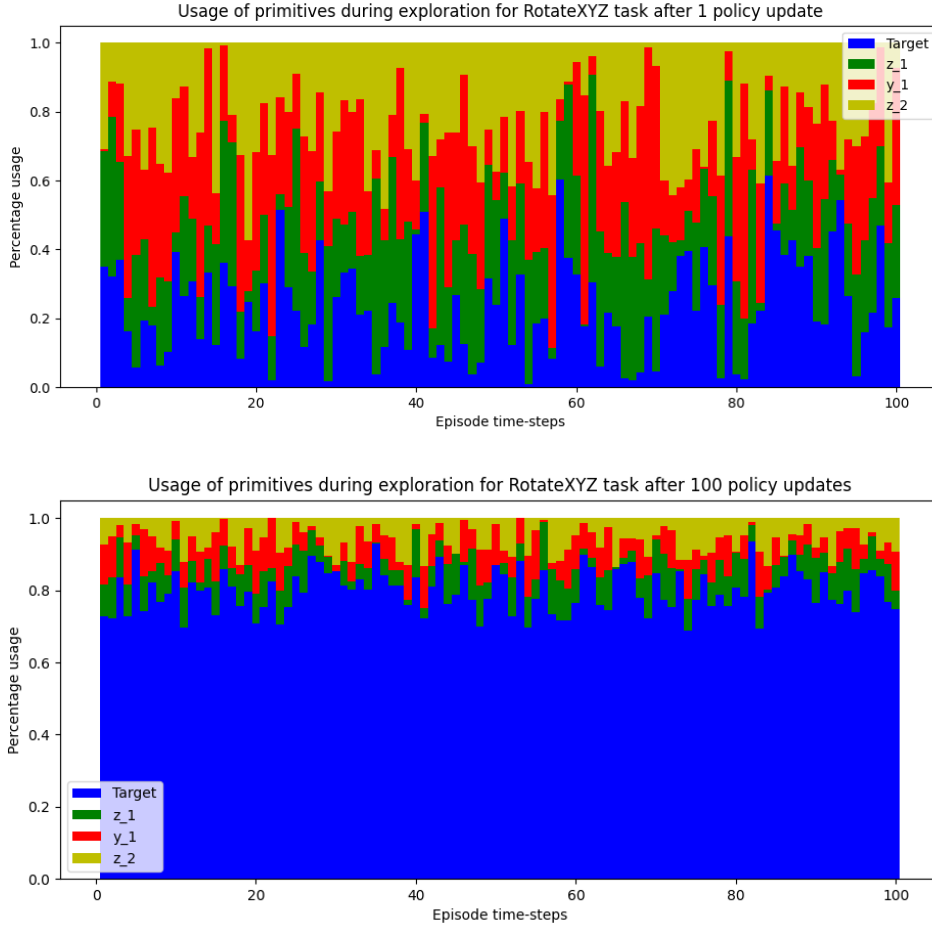


Figure 6.22: Visualisation of the average usage over 25 episodes during exploration of the behaviours of the primitive policies and the target policy after 1 policy update and 100 policy updates on the *RotateXYZ* task.

decides at each time-step during the exploration phase if the actions suggested by the primitive policies might be useful or not to solve the target task. This method can be used together with any off-policy RL algorithm that does not modify the action selection mechanism.

The algorithm has been evaluated in different tasks from the Fetch and the Shadow Dexterous Hand environments. The QMP-HER (or QMP-HGG) algorithm proved to be very sample and time efficient for those tasks that can be fully solved by the primitives (i.e., most of the tasks on the Fetch environment), taking full benefit from the exploitation that comes by introducing this knowledge to the learning process. Despite this, the algorithm is also capable of refining the learned policy instead of fully replicating the behaviours of the primitives, in this way improving

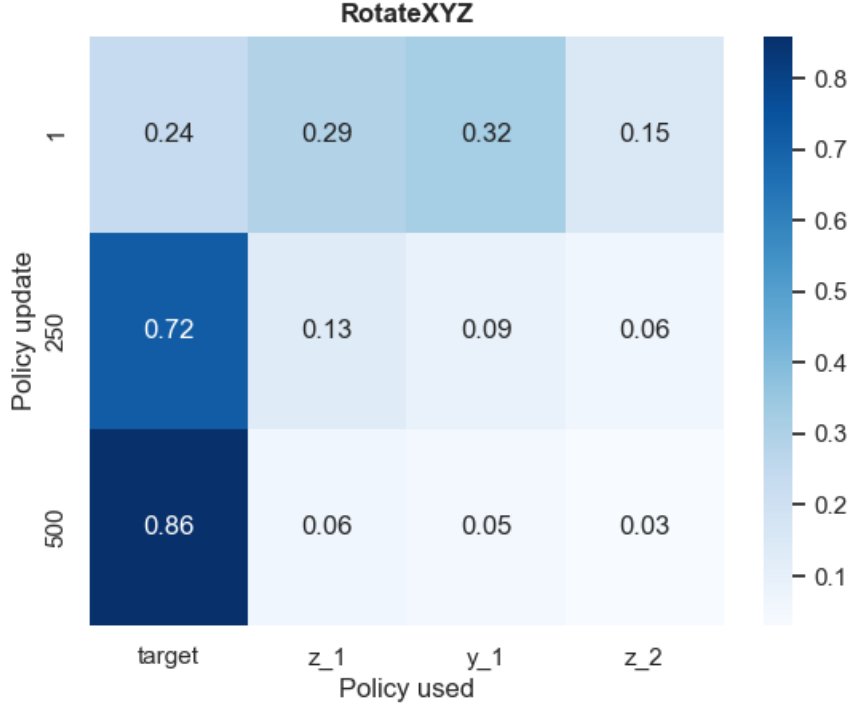


Figure 6.23: Usage of the target policy $\pi(s, g)$ and the primitive behaviours instances z_1 , y_1 and z_2 during the exploration phase after the 1st, the 250th and the 500th policy update for the *RotateParallel* task. z_1 , y_1 and z_2 represent the same as in Figure 6.21.

on the proposals of these primitive behaviours.

The impact of the QMP algorithm, however, is much less significant for those tasks with complex goal spaces, such as the in-hand manipulation tasks. Instead, the hindsight goal generation algorithm selection is the most important factor in these cases, while the primitive behaviours primarily contribute in terms of making the *behaviour* policy more exploratory.

Nevertheless, our algorithm is also competitive in terms of sample efficiency when compared to other existing sample efficient methods, such as the RHER [219], introduced in Section 5.3.1, which manages to solve the *FetchPush* task (i.e. reach 95% of success rate) in approximately 65,000 time-steps and the *FetchPickAndPlace* task in approximately 245,000 time-steps, while our method takes 62,500 time-steps (i.e. 25 policy updates) to reach 95% of success rate when solving the *FetchPush* task and 375,000 time-steps (i.e. 150 policy updates) when solving the *FetchPickAndPlace* task. Regarding the Shadow Dexterous Hand environment tasks, the Hindsight

Planner algorithm, also introduced in Section 5.3.1, manages to reach 50% of success rate in the *RotateXYZ* in approximately 12,500,000 time-steps. In contrast, our method reaches the same success rate in 10,000,000 time-steps (i.e. 2000 policy updates).

Future research arising from this work could include finding better primitives that replicate more specific behaviours, possibly by clustering environment transitions of simple tasks and replicating the behaviour of these clusters using any state-of-the-art behavioural modelling technique, such as decision transformers [250] or trajectory transformers [251].

One of the weaknesses of the algorithm presented in this chapter is the need for manually defined objectives and, therefore, another future direction could explore employing our method in conjunction with other algorithms that propose intermediate goals during the exploration phase, such as the Hindsight Planner [118] or RHER; or even introducing a low-level parameter policy that determines how to instantiate the primitives given an environment state as done in [242].

6.8 Summary

6.8.1 Revisiting the hypothesis

In Chapter 1, the following hypothesis was stated:

- **Hypothesis 3:** *The use of fixed set of basic primitive policies limits the sample efficiency and the performance of robotic manipulation to the expertise of these primitive behaviours. However, we hypothesise that if an algorithm can decide during exploration between when to use one of the existing primitive policies and when to learn a new skill, robots will become more adaptable to dynamic environments and will learn to perform more complex tasks more quickly.*

As has been seen throughout this chapter, this hypothesis is supported by experimental evidence. By automatically deciding when to use certain primitive behaviours during exploration, the QMP-HER algorithm presented in this chapter not

only outperforms state-of-the-art off-policy RL algorithms, but also does so with a very good training time efficiency, adding very little complexity to the main off-policy RL logic.

6.8.2 Summary of chapter contributions

This chapter presented one substantial contribution, which is summarised as follows:

1. **A novel algorithm called Q-Switch Mixture of Primitives Hind-sight Experience Replay (QMP-HER) that improves the sample efficiency in complex manipulation tasks by making use of primitive behaviours trained in simpler environments.** This algorithm can work in conjunction with any off-policy RL algorithm.

Chapter 7

Conclusions

7.1 Chapter overview

This chapter provides answers to the research questions raised in Chapter 1, as well as presenting the main conclusions of the research reported in this thesis. In particular, Section 7.2 answers the research questions and revisits the different hypotheses raised at the beginning of the thesis. Then, Section 7.3 summarises the contributions to the field presented in this thesis, and Section 7.4 presents several potential future research directions. Finally, some closing remarks are provided in Section 7.5 to conclude the thesis. The work described in this thesis was accepted for publication at the 2022 Sixth IEEE International Conference on Robotic Computing (IRC) [252], where it was awarded with the 2nd best PhD workshop paper.

7.2 Answering the research questions

The hypotheses raised in Chapter 1 are discussed in this section in light of the research described in the previous chapters. Each of the associated research questions is addressed to provide more concise descriptions of the various findings of this thesis:

Research Question 1: How important is reward engineering in obtaining sample and time efficiency when training a RL-based algorithm in lifting an object?

- **Hypothesis 1:** *By carefully designing a reward that encourages a particular desired behaviour, standard RL algorithms can learn more efficiently to solve robotic manipulation tasks that involve the lifting of an object, leading to more sample-efficient methods and, thus, to lower training times.*

To answer this question, in Chapter 3 we design a novel intrinsic reward system that encourages an object lifting behaviour, combined with a sparse reward for the xy position of the object, and evaluate this reward scheme in the context of the Real Robot Challenge (RRC). Even though our method achieve a slight improvement in performance over a traditional reward scheme, and did not prove to be more sample efficient overall; it manages to learn faster in the initial interactions with the environment, meaning that a successful lifting behaviour can be achieved earlier when using our reward system compared to the alternative reward systems explored. This was the model we submitted to the Real Robot Challenge (RRC) competition and managed to win Phase 1 of the RRC, beating the rest of solutions proposed by other teams taking part.

Research Question 2: How can we make use of basic primitive skills in a sequential manner to solve more complex tasks? In which scenarios would this approach be most useful?

- **Hypothesis 2:** *A major reason why RL sometimes struggles with sample and time efficiency is because robotic manipulation tasks are too complex to learn from scratch (we refer as learning from scratch to initialising all model parameters randomly and then optimising them using a chosen optimisation algorithm). Instead, we hypothesise that by dividing complex tasks into smaller simpler tasks, that can be learned quickly and then be executed sequentially, the original complex manipulation task can be effectively solved using few interactions with the environment. If this hypothesis is supported, this approach will result in more modular learning, which not only ensures a more sample and time efficiency method, but also a more interpretable algorithm.*

To test this hypothesis, Chapter 5 proposes different solutions to solve obstacle avoidance and in-hand manipulation tasks by sequentially executing learned off-policy RL-based primitive behaviours that are easy and fast to learn. By employing this method, using a limited amount of experience, we achieve better performance in complex manipulation tasks, such as rotating an object with a hand manipulator, than we would obtain otherwise using the same amount of experience and training the same off-policy RL algorithm in a fully end-to-end fashion.

However, the performance obtained using this approach is limited by the expertise of the primitive behaviours, while the performance obtained by the off-policy RL trained in an end-to-end fashion would eventually improve if sufficiently long training times were possible. In other words, the end-to-end fashion can produce more expert policies at the cost of sample and time inefficiency compared to the approach presented in Chapter 5.

Furthermore, the use of primitive behaviours, sequentially executed, is mostly useful in complex environments that have high-dimensional action, observation, and/or goal spaces, as training an off-policy RL algorithm to solve this type of tasks in an end-to-end fashion requires very intensive exploration and, therefore, it takes a very long time.

Research Question 3: How can we reuse primitive skills that are easy to learn in order to accelerate the learning of current state-of-the-art (SOTA) methods in robotic manipulation? Are there strategies that accelerate current SOTA without adding computational complexity?

- **Hypothesis 3:** *The use of fixed set of basic primitive policies limits the sample efficiency and the performance of robotic manipulation to the expertise of these primitive behaviours. However, we hypothesise that if an algorithm can decide during exploration between when to use one of the existing primitive policies and when to learn a new skill, robots will become more adaptable to dynamic environments and will learn to perform more complex tasks more quickly.*

To test this hypothesis, Chapter 6 proposes a novel algorithm, named Q-switch Mixture of Primitives Hindsight Experience Replay (QMP-HER), that makes use of primitive behaviours while exploring the environment only when the use of these primitives is useful to solve the task at hand. By making use of the feedback obtained from the learned action-value function, our algorithm strategically selects the best action amongst all the candidate actions proposed by the primitive policies and the target policy that is being updated to solve the task.

Even though it is almost impossible to improve the sample efficiency of the current state of the art without adding computational complexity (no work in the literature has been found that does so), as has been seen throughout Chapter 6, our proposed method improves the sample efficiency while adding minimal computational complexity, particularly when compared to other model-based works in the existing literature, such as G-HGG.

7.3 Summary of thesis contributions

A number of different contributions to the field have been made in this thesis. These are summarised as follows:

(i) Chapter 3:

- (a) **A novel reward system that combines sparse and dense components in order to encourage a lifting behaviour for a robotic manipulator.** This algorithm can be used with any off-policy RL technique and sets a new state of the art on the ‘*Move Cube on Trajectory*’ task of the TriFinger robot.

- (b) **Developed three different cube manipulation strategies** that can be used with the TriFinger robot, **by making use of force feedback** obtained from tactile sensors allocated on the finger tips of the robot.

(ii) Chapter 5:

- (a) **A novel learning framework for in-hand object 3D rotations based on a sequential execution of primitives learned by any off-policy RL method.** This framework operates using the Davenport chained rotations decomposition, a method widely used in the field of aeronautics.
- (b) **Implementation of two new in-hand robotic manipulation tasks** named *RotateX* and *RotateY* that follow the standards of OpenAI gym environment, meaning it can be used with algorithms implemented to work on that environment.
- (iii) **Chapter 6:**
 - (a) **A novel algorithm called Q-Switch Mixture of Primitives Hind-sight Experience Replay (QMP-HER) that improves the sample efficiency in complex manipulation tasks by making use of primitive behaviours trained in simpler environments.** This algorithm can work in conjunction with any off-policy RL algorithm.

7.4 Recommendations for future research

In this thesis, one main critical aspect has been addressed: the exploration of sample-efficient off-policy RL algorithms tailored for robotic manipulation tasks. However, there are other areas involving RL and robotic manipulation that merit research. In this section, we provide some recommendations for future research.

The most obvious possible direction emerging from this thesis is investigating the integration of multi-task learning and transfer learning techniques in RL for robotic manipulation, which can lead to more efficient and versatile robotic systems capable of performing a wide range of tasks with shared knowledge and experience, continuing the path started in Chapter 6. By enabling robots to leverage previously learned skills and adapt them to new tasks or environments, transfer learning could enhance

the autonomy, flexibility, and task efficiency of robotic manipulators, resulting in more adaptable and capable robotic systems for various applications.

Furthermore, for all chapters except Chapter 3, the observations used come directly from the environment simulator. We must somehow estimate this information in real environments; in Chapter 3 we used a colour segmentation provided by the organisers of the RRC, and force feedback obtained from tactile sensors located on the fingertips of the robot. Therefore, focusing on enhancing the perception capabilities of robotic manipulators through state-of-the-art computer vision, sensor fusion, and tactile sensing technologies could potentially significantly improve their interaction with the surrounding environment, and is a line of research currently being explored by many [253]–[255].

Another possible direction involves addressing the challenges of safety and robustness in RL-based robotic manipulation systems. Ensuring the reliable and predictable behaviour of robots in unstructured environments or when facing unforeseen events is crucial for real-world deployment. By integrating safety constraints and robust optimisation frameworks into RL algorithms, it could be possible to mitigate risks, handle disturbances, and improve the overall safety of autonomous robotic manipulation systems [256]–[259].

Another avenue for future work involves the development of collaborative robotic systems that can seamlessly interact and cooperate with human operators, other robots, or autonomous agents in shared workspaces. The design of intelligent coordination and communication mechanisms based on AI and human-robot interaction principles is important in order to enable robots to work alongside humans in collaborative or assistive tasks, and this topic has become very popular in recent years [260]–[262].

7.5 Closing remarks

The most successful recent robotic manipulator systems are driven by RL algorithms, which are trained through trial and error by interacting with an environment. Be-

cause these algorithms require extensive exploration in order to perform well, particularly when dealing with high-dimensional complex environments, they suffer from sample inefficiency resulting in impractically long training times. The research conducted in this thesis focuses on finding strategies to accelerate the learning process for RL-based algorithms to solve robotic manipulation tasks, and therefore, the approaches explored and developed are of particular interest for those individuals, institutions, or applications where training resources are scarce.

Even though this thesis focuses on sample and time efficiency, it is important to remember the main overarching goal for this type of research: to make robotic manipulation systems intelligent by providing robots with the ability to understand the context in which they operate. Being able to efficiently learn manipulation skills, can move us closer to realising this goal. However, there are still many research lines of inquiry to be investigated before this becomes a reality.

Robots will soon become intelligent enough to have an enormous positive impact on our quality of life if we use their capabilities responsibly. While RL has shown incredible promise as a way to achieve that level of intelligence, there is still work to do. The contributions of this thesis help us take some of the steps along that road to a world where dull, dangerous, and dirty work is no longer the burden of the human worker, and we can reclaim more time to spend in pursuit of health, happiness, and love.

Appendix A

Appendix

A.1 Reward engineering

A.1.1 Grid-search hyperparameter a of Equation 3.3

Table A.1: Scores for different values of the hyperparameter a over 10000 episodes. The best score was obtain setting it to 20.

a	score
10	-6,532 $\pm$ 2,090
20	-6,198 $\pm$ 1,840
30	-6,344 $\pm$ 1,929
40	-7,051 $\pm$ 1,791

A.1.2 Network hyperparameters

Code can be found at https://github.com/RedmondLabUCD/rrc_pos_ori.

- Hidden layer size of the actor: [256, 256, 256]
- Hidden layer size of the critic: [256, 256, 256]
- ϵ -greedy exploration: 0.3
- Learning rate of the actor: 0.001
- Learning rate of the critic: 0.001

- Discount factor: 0.98
- Batch size: 256
- Replay buffer size: 1,000,000
- Clipping range : 5
- Perturbation of the observation space using Gaussian noise: True
- Warm-up time steps: 10000

A.1.3 Extended domain randomisation results

Num	Seed 0		Seed 123		Seed 200	
	Rob1	Sim	Rob6	Sim	Rob5	Sim
1	-12115	-8288	-15639	-9200	-10120	-8379
2	-15167	-9430	-14262	-12781	-16159	-12284
3	-23378	-8290	-8257	-10560	-21487	-9197
4	-21338	-10045	-11242	-9759	-23810	-7111
5	-25971	-7998	-11947	-8762	-10207	-9745
6	-13497	-8878	-18063	-8536	-12042	-9183
7	-17905	-9567	-13341	-10685	-12958	-9670
8	-22253	-9332	-15494	-10022	-15986	-15520
9	-11608	-7669	-12824	-7113	-18867	-10239
10	-11805	-8351	-21261	-15298	-17593	-8557
11	-10500	-8677	-20369	-9053	-16410	-16587
12	-19996	-9367	-23302	-9123	-18469	-11242
13	-20676	-8585	-18059	-8727	-11228	-13206
14	-12666	-7764	-18087	-8337	-10918	-12380
15	-11413	-9877	-13749	-10143	-14727	-23389
<i>Avg</i>	-16686	-8808	-15726	-9873	-15399	-11779

Table A.2: The evaluated scores of DRL agents trained with different seeds on both simulator and real robot. 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.

A.2 Sequential execution of primitive behaviours

A.2.1 Network Hyperparameters

Code for Chapters 5 and 6 can be found at https://github.com/franroldans/custom_gym and <https://github.com/franroldans/qmp-her>

Num	Seed 0				Seed 123				Seed 200			
	Rob1	Rob5	Robo6	Sim	Rob1	Rob5	Rob6	Sim	Rob1	Rob5	Rob6	Sim
1	-9970	-11190	-11512	-5635	-8789	-7328	-10030	-6579	-22824	-12436	-11927	-6965
2	-11412	-9017	-11461	-5029	-8086	-10263	-13828	-6650	-12288	-10107	-19633	-7464
3	-10746	-12084	-10900	-5175	-11328	-9908	-8270	-7180	-10859	-9501	-17123	-8839
4	-7572	-15767	-11904	-5570	-9928	-8228	-9110	-6314	-19834	-11469	-15823	-5422
5	-10883	-10430	-13682	-5430	-13106	-8263	-10662	-5746	-13174	-17859	-9643	-7505
6	-7896	-11427	-10371	-5895	-11644	-10245	-6619	-5932	-18568	-14007	-17594	-7324
7	-7942	-14505	-14028	-4977	-6447	-11307	-12355	-6106	-12754	-10631	-11542	-6473
8	-8461	-11135	-13604	-5698	-9180	-7481	-11375	-6551	-21177	-11721	-14255	-6009
9	-6441	-12871	-9460	-4921	-10841	-6781	-7524	-6560	-12986	-11987	-15043	-7716
10	-11466	-11620	-10030	-5702	-4951	-8415	-10385	-6257	-10326	-12408	-17208	-8291
11	-9513	-5192	-15004	-5259	-9020	-8248	-7751	-6055	-18069	-11129	-19162	-7405
12	-11928	-10046	-11051	-6016	-6250	-9530	-11040	-6722	-19168	-19332	-18368	-6590
13	-14798	-11213	-12311	-5027	-8987	-10205	-11442	-7148	-15011	-15671	-15546	-8015
14	-10501	-12419	-16049	-5531	-8834	-9002	-9852	-6101	-23270	-17366	-19554	-6483
15	-9178	-10589	-11256	-5847	-12243	-8691	-16036	-5732	-13678	-11187	-13829	-8657
Avg	-9914	-11300	-12175	-5448	-9309	-8926	-10017	-6376	-16266	-13121	-15750	-7277

Table A.3: The evaluated scores of DRL agents trained with different seeds on both simulator and real-robot. All DRL agents are trained from scratch without DR. This table is the raw data of Table 3.2 before averaging.

Num	Seed 0				Seed 123				Seed 200			
	Rob1	Rob5	Rob6	Sim	Rob1	Rob5	Rob6	Sim	Rob1	Rob5	Rob6	Sim
1	-8787	-9647	-7396	-5257	-7579	-6335	-6700	-6461	-8000	-10161	-7614	-5413
2	-11681	-6962	-10088	-6809	-6344	-6729	-7252	-5724	-8219	-9995	-8555	-6481
3	-6013	-8187	-6318	-5442	-7376	-6124	-5388	-7291	-13052	-7443	-6707	-6812
4	-10352	-6727	-4912	-5169	-5364	-8417	-7932	-6066	-12318	-8324	-7532	-7491
5	-8789	-7936	-7370	-6081	-9498	-8711	-6013	-6474	-10980	-7155	-8574	-5620
6	-10460	-11045	-7351	-5908	-8529	-7367	-5514	-5788	-11110	-6375	-7533	-8658
7	-8279	-10296	-11356	-6426	-6393	-4906	-8937	-6458	-7776	-6660	-7679	-5728
8	-10261	-9046	-6375	-6043	-4557	-7492	-7335	-5680	-8995	-9554	-6193	-6664
9	-6371	-11219	-6704	-4671	-7729	-6269	-7673	-6544	-9650	-7126	-7821	-6263
10	-10443	-11136	-11567	-6491	-8103	-9095	-9221	-7921	-10734	-9523	-5636	-6289
11	-6236	-7585	-11795	-5751	-5496	-7470	-5685	-5299	-11455	-7360	-6246	-5608
12	-8060	-6528	-10957	-3804	-6306	-9059	-8037	-5429	-8518	-5751	-9680	-4373
13	-9808	-8811	-8557	-5418	-5114	-6558	-8175	-5993	-7719	-9358	-9330	-5051
14	-10962	-11804	-11804	-7607	-6388	-6056	-6038	-6394	-10331	-7711	-8775	-5891
15	-7681	-7249	-10592	-6500	-7062	-6800	-7225	-6278	-10197	-8043	-6610	-5868
Avg	-8946	-8945	-8876	-5825	-6789	-7159	-7142	-6253	-9937	-8036	-7632	-6147

Table A.4: The evaluated scores of DRL agents trained with different seeds on both simulator and real-robot. 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 3.2 before averaging.

- Hidden layer size of the actor: [256, 256, 256]
- Hidden layer size of the critic: [256, 256, 256]
- ϵ -greedy exploration: 0.3
- Learning rate of the actor: 0.001
- Learning rate of the critic: 0.001
- Discount factor: 0.98

- Batch size: 256 for Shadow Dexterous Hand environment tasks, 64 for Fetch environment tasks.
- Replay buffer size: 500,000
- Clipping range : 5
- Warm-up time steps: 10000
- Stop HGG threshold: 0.5 for *FetchPickObstacle* task, 0.9 for *FetchPickAndThrow* and Shadow Dexterous Hand tasks.
- Graph: $n_x = 31$, $n_y = 31$, $n_z = 11$ for *FetchPickObstacle* task, and $n_x = 51$, $n_y = 51$, $n_z = 7$ for *FetchPickAndThrow* task

A.3 Automatic use of primitive behaviours

A.3.1 Network Hyperparameters

Same as previously except:

- Epsilon-greedy: 0.5
- Object deviations for primitive policies goals:
 - PickAndPlace and Push tasks: $[[0.1, 0, 0], [-0.1, 0, 0], [0, 0.1, 0], [0, -0.1, 0], [0, 0, 0.1], [0, 0, 0]]$ left&right, front&behind, and top
 - PickObstacle task: $[0, 0, 0.6]$ to lift the object in a position over the obstacle's height.
 - $[0, 0, 0.6]$ for primitive policy aiming to grasp the object and move it to a centered, elevated location.

Bibliography

- [1] M. T. Mason, *Mechanics of robotic manipulation*. MIT press, 2001.
- [2] M. T. Mason, “Toward robotic manipulation,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 1–28, 2018.
- [3] J. Tegin and J. Wikander, “Tactile sensing in intelligent robotic manipulation—a review,” *Industrial Robot: An International Journal*, vol. 32, no. 1, pp. 64–70, 2005.
- [4] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, “Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review,” *Robotics*, vol. 10, no. 1, p. 22, 2021.
- [5] M. A. Wiering and M. Van Otterlo, “Reinforcement learning,” *Adaptation, learning, and optimization*, vol. 12, no. 3, p. 729, 2012.
- [6] G. Tesauro, “Td-gammon, a self-teaching backgammon program, achieves master-level play,” *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.
- [7] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
- [8] S. G. Khan, G. Herrmann, F. L. Lewis, T. Pipe, and C. Melhuish, “Reinforcement learning and optimal adaptive control: An overview and implementation examples,” *Annual reviews in control*, vol. 36, no. 1, pp. 42–59, 2012.

- [9] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, “Optimal and autonomous control using reinforcement learning: A survey,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2017.
- [10] R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, “Deep reinforcement learning for autonomous driving: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.
- [11] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
- [12] K. Zhu and T. Zhang, “Deep reinforcement learning based mobile robot navigation: A review,” *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.
- [13] P. Kormushev, S. Calinon, and D. G. Caldwell, “Reinforcement learning in robotics: Applications and real-world challenges,” *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.
- [14] N. Akalin and A. Loutfi, “Reinforcement learning approaches in social robotics,” *Sensors*, vol. 21, no. 4, p. 1292, 2021.
- [15] S. Amarjyoti, “Deep reinforcement learning for robotic manipulation-the state of the art,” *arXiv preprint arXiv:1701.08878*, 2017.
- [16] A. Roy, X. Zhang, N. Wolleb, C. P. Quintero, and M. Jägersand, “Tracking benchmark and evaluation for manipulation tasks,” in *2015 IEEE international Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 2448–2453.
- [17] S. Cruciani, B. Sundaralingam, K. Hang, V. Kumar, T. Hermans, and D. Kragic, “Benchmarking in-hand manipulation,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 588–595, 2020.

- [18] J. Cui and J. Trinkle, “Toward next-generation learned robot manipulation,” *Science robotics*, vol. 6, no. 54, eabd9461, 2021.
- [19] R. Padilla, S. L. Netto, and E. A. Da Silva, “A survey on performance metrics for object-detection algorithms,” in *2020 international conference on systems, signals and image processing (IWSSIP)*, IEEE, 2020, pp. 237–242.
- [20] S. Mondié, A. Egorov, and M. A. Gomez, “Lyapunov stability tests for linear time-delay systems,” *Annual Reviews in Control*, vol. 54, pp. 68–80, 2022.
- [21] M. Wüthrich, F. Widmaier, F. Grimminger, J. Akpo, S. Joshi, V. Agrawal, B. Hammoud, M. Khadiv, M. Bogdanovic, V. Berenz, *et al.*, “Trifinger: An open-source robot for learning dexterity,” *arXiv preprint arXiv:2008.03596*, 2020.
- [22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [23] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.
- [24] A. A. Shahid, D. Piga, F. Braghin, and L. Roveda, “Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning,” *Autonomous Robots*, vol. 46, no. 3, pp. 483–498, 2022.
- [25] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, “Dynamic movement primitives in robotics: A tutorial survey,” *The International Journal of Robotics Research*, vol. 42, no. 13, pp. 1133–1184, 2023.
- [26] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, “Diversity is all you need: Learning skills without a reward function,” *arXiv preprint arXiv:1802.06070*, 2018.
- [27] P. Ladosz, L. Weng, M. Kim, and H. Oh, “Exploration in deep reinforcement learning: A survey,” *Information Fusion*, vol. 85, pp. 1–22, 2022.

- [28] E. G. Kaigom and J. Roßmann, “Physics-based simulation for manual robot guidance—an erobotics approach,” *Robotics and Computer-Integrated Manufacturing*, vol. 43, pp. 155–163, 2017.
- [29] M. Gupta and G. S. Sukhatme, “Using manipulation primitives for brick sorting in clutter,” in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 3883–3889.
- [30] A. Roque and S. K. Damodaran, “Explainable ai for security of human-interactive robots,” *International Journal of Human-Computer Interaction*, vol. 38, no. 18-20, pp. 1789–1807, 2022.
- [31] N. Emaminejad and R. Akhavian, “Trustworthy ai and robotics: Implications for the aec industry,” *Automation in Construction*, vol. 139, p. 104 298, 2022.
- [32] A. Hussain, A. Malik, M. U. Halim, and A. M. Ali, “The use of robotics in surgery: A review,” *International journal of clinical practice*, vol. 68, no. 11, pp. 1376–1382, 2014.
- [33] S. Bauer, M. Wüthrich, F. Widmaier, A. Buchholz, S. Stark, A. Goyal, T. Steinbrenner, J. Akpo, S. Joshi, V. Berenz, V. Agrawal, N. Funk, J. Urain De Jesus, J. Peters, J. Watson, C. Chen, K. Srinivasan, J. Zhang, J. Zhang, M. Walter, R. Madan, T. Yoneda, D. Yarats, A. Allshire, E. Gordon, T. Bhattacharjee, S. Srinivasa, A. Garg, T. Maeda, H. Sikchi, J. Wang, Q. Yao, S. Yang, R. McCarthy, F. Sanchez, Q. Wang, D. Bulens, K. McGuinness, N. O’Connor, R. Stephen, and B. Schölkopf, “Real robot challenge: A robotics competition in the cloud,” in *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, D. Kiela, M. Ciccone, and B. Caputo, Eds., ser. Proceedings of Machine Learning Research, vol. 176, PMLR, 2022, pp. 190–204.
- [34] R. C. Luo, M.-C. Ko, Y.-T. Chung, and R. Chatila, “Repulsive reaction vector generator for whole-arm collision avoidance of 7-dof redundant robot

- manipulator,” in *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, IEEE, 2014, pp. 1036–1041.
- [35] R. Savatekar and A. Dum, “Design of control system for articulated robot using leap motion sensor,” *Int. Res. J. Eng. Technol*, vol. 3, pp. 1407–1417, 2016.
- [36] S. KuCuk and Z. Bingul, “The inverse kinematics solutions of industrial robot manipulators,” in *Proceedings of the IEEE International Conference on Mechatronics, 2004. ICM’04.*, IEEE, 2004, pp. 274–279.
- [37] P. Božek, A. Al Akkad M, P. Blištan, and I. Ibrahim N, “Navigation control and stability investigation of a mobile robot based on a hexacopter equipped with an integrated manipulator,” *International Journal of Advanced Robotic Systems*, vol. 14, no. 6, p. 1 729 881 417 738 103, 2017.
- [38] I. Agustian, N. Daratha, R. Faurina, A. Suandi, *et al.*, “Robot manipulator control with inverse kinematics pd-pseudoinverse jacobian and forward kinematics denavit hartenberg,” *arXiv preprint arXiv:2103.10461*, 2021.
- [39] M. Colledanchise and L. Natale, “On the implementation of behavior trees in robotics,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5929–5936, 2021.
- [40] D. C. Dominguez, M. Iannotta, J. A. Stork, E. Schaffernicht, and T. Stoyanov, “A stack-of-tasks approach combined with behavior trees: A new framework for robot control,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12 110–12 117, 2022.
- [41] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, and A. Wkasowski, “Behavior trees in action: A study of robotics applications,” in *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, 2020, pp. 196–209.

- [42] O. Ruiz-Celada, P. Verma, M. Diab, and J. Rosell, “Automating adaptive execution behaviors for robot manipulation,” *IEEE access*, vol. 10, pp. 123 489–123 497, 2022.
- [43] J. Styrud, M. Iovino, M. Norrlöf, M. Björkman, and C. Smith, “Combining planning and learning of behavior trees for robotic assembly,” in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 11 511–11 517.
- [44] X. Cheng, A. Kumar, and D. Pathak, “Legs as manipulator: Pushing quadrupedal agility beyond locomotion,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 5106–5112.
- [45] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.
- [46] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
- [47] R. Bellman, “A markovian decision process,” *Journal of mathematics and mechanics*, pp. 679–684, 1957.
- [48] M. Van Otterlo and M. Wiering, “Reinforcement learning and markov decision processes,” in *Reinforcement learning: State-of-the-art*, Springer, 2012, pp. 3–42.
- [49] Z. Ding, Y. Huang, H. Yuan, and H. Dong, “Introduction to reinforcement learning,” *Deep reinforcement learning: fundamentals, research and applications*, pp. 47–123, 2020.
- [50] R. Bellman, R. E. Kalaba, *et al.*, *Dynamic programming and modern control theory*. Citeseer, 1965, vol. 81.
- [51] A. Boularias, J. Bagnell, and A. Stentz, “Learning to manipulate unknown objects in clutter by reinforcement,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.

- [52] G. Peng, J. Yang, X. Li, and M. O. Khyam, “Deep reinforcement learning with a stage incentive mechanism of dense reward for robotic trajectory planning,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2022.
- [53] C. Daniel, O. Kroemer, M. Viering, J. Metz, and J. Peters, “Active reward learning with a novel acquisition function,” *Autonomous Robots*, vol. 39, pp. 389–405, 2015.
- [54] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight experience replay,” *Advances in neural information processing systems*, vol. 30, 2017.
- [55] S. Singh, R. L. Lewis, and A. G. Barto, “Where do rewards come from,” in *Proceedings of the annual conference of the cognitive science society*, Cognitive Science Society, 2009, pp. 2601–2606.
- [56] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, and N. De Freitas, “Playing hard exploration games by watching youtube,” *Advances in neural information processing systems*, vol. 31, 2018.
- [57] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, “Intrinsically motivated reinforcement learning: An evolutionary perspective,” *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 70–82, 2010.
- [58] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” *Advances in neural information processing systems*, vol. 12, 1999.
- [59] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.
- [60] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, pp. 279–292, 1992.

- [61] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [62] T. Degris, M. White, and R. S. Sutton, “Off-policy actor-critic,” *arXiv preprint arXiv:1205.4839*, 2012.
- [63] M. Hausknecht and P. Stone, “On-policy vs. off-policy updates for deep reinforcement learning,” in *Deep reinforcement learning: frontiers and challenges, IJCAI 2016 Workshop*, AAAI Press New York, NY, USA, 2016.
- [64] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, “Don’t decay the learning rate, increase the batch size,” *arXiv preprint arXiv:1711.00489*, 2017.
- [65] S. Lange, T. Gabel, and M. Riedmiller, “Batch reinforcement learning,” in *Reinforcement learning: State-of-the-art*, Springer, 2012, pp. 45–73.
- [66] S. Ben-David, E. Kushilevitz, and Y. Mansour, “Online learning versus offline learning,” *Machine Learning*, vol. 29, pp. 45–63, 1997.
- [67] P. Auer, “Using upper confidence bounds for online learning,” in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, IEEE, 2000, pp. 270–279.
- [68] J. Rubin, O. Shamir, and N. Tishby, “Trading value and information in mdps,” *Decision making with imperfect decision makers*, pp. 57–74, 2012.
- [69] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen, *et al.*, “A tutorial on thompson sampling,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.
- [70] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction,” in MIT press, 2018, ch. X, Epsilon-Greedy Algorithm.
- [71] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2017, pp. 3389–3396.

- [72] B. Manela, “Deep reinforcement learning for complex manipulation tasks with sparse feedback,” *arXiv preprint arXiv:2001.03877*, 2020.
- [73] H. Zhang, F. Wang, J. Wang, and B. Cui, “Robot grasping method optimization using improved deep deterministic policy gradient algorithm of deep reinforcement learning,” *Review of Scientific Instruments*, vol. 92, no. 2, 2021.
- [74] S. Kumra, S. Joshi, and F. Sahin, “Learning robotic manipulation tasks via task progress based gaussian reward and loss adjusted exploration,” *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 534–541, 2021.
- [75] V. Bulut, “Optimal path planning method based on epsilon-greedy q-learning algorithm,” *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 44, no. 3, p. 106, 2022.
- [76] P. Shukla, H. Kumar, and G. C. Nandi, “Robotic grasp manipulation using evolutionary computing and deep reinforcement learning,” *Intelligent Service Robotics*, vol. 14, pp. 61–77, 2021.
- [77] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [78] I. C. Dolcetta and H. Ishii, “Approximate solutions of the bellman equation of deterministic control theory,” *Applied Mathematics and Optimization*, vol. 11, no. 1, pp. 161–181, 1984.
- [79] C. Gaskett *et al.*, “Q-learning for robot control,” 2002.
- [80] I. Carlucho, M. De Paula, S. A. Villar, and G. G. Acosta, “Incremental q-learning strategy for adaptive pid control of mobile robots,” *Expert Systems with Applications*, vol. 80, pp. 183–199, 2017.
- [81] P. Li, G. Wang, H. Jiang, Y. Jin, Y. Gan, X. Chen, and J. Ji, “A q-learning control method for a soft robotic arm utilizing training data from a rough simulator,” in *2021 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, IEEE, 2021, pp. 839–845.

- [82] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, “Q-learning algorithms: A comprehensive classification and applications,” *IEEE access*, vol. 7, pp. 133 653–133 667, 2019.
- [83] G. Tesauro *et al.*, “Temporal difference learning and td-gammon,” *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.
- [84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [85] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation,” *arXiv preprint arXiv:1806.10293*, 2018.
- [86] A. Camacho, J. Varley, A. Zeng, D. Jain, A. Iscen, and D. Kalashnikov, “Reward machines for vision-based robotic manipulation,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 14 284–14 290.
- [87] S. James and A. J. Davison, “Q-attention: Enabling efficient learning for vision-based robotic manipulation,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1612–1619, 2022.
- [88] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.
- [89] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, “Soft actor-critic algorithms and applications,” *arXiv preprint arXiv:1812.05905*, 2018.
- [90] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, “Natural actor-critic algorithms,” *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

- [91] A. Robbins, M. Ho, and M. Teodorescu, “Model-free dynamic control of robotic joints with integrated elastic ligaments,” *Robotics and Autonomous Systems*, vol. 155, p. 104 150, 2022.
- [92] D. Kawahara, S. Ozeki, and I. Mizuuchi, “A curiosity algorithm for robots based on the free energy principle,” in *2022 IEEE/SICE International Symposium on System Integration (SII)*, IEEE, 2022, pp. 53–59.
- [93] M. Omer, R. Ahmed, B. Rosman, and S. F. Babikir, “Model predictive-actor critic reinforcement learning for dexterous manipulation,” in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, IEEE, 2021, pp. 1–6.
- [94] M. Haklilar and H. Temeltaş, “Guided soft actor critic: A guided deep reinforcement learning approach for partially observable markov decision processes,” *IEEE Access*, vol. 9, pp. 159 672–159 683, 2021.
- [95] J. Björck, X. Chen, C. De Sa, C. P. Gomes, and K. Weinberger, “Low-precision reinforcement learning: Running soft actor-critic in half precision,” in *International Conference on Machine Learning*, PMLR, 2021, pp. 980–991.
- [96] S.-i. Amari, “Backpropagation and stochastic gradient descent method,” *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.
- [97] S. Paul and L. Vig, “Deterministic policy gradient based robotic path planning with continuous action spaces,” in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 725–733.
- [98] C. Do, C. Gordillo, and W. Burgard, “Learning to pour using deep deterministic policy gradients,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 3074–3079.
- [99] S. Nahavandi, R. Alizadehsani, D. Nahavandi, C. P. Lim, K. Kelly, and F. Bello, “Machine learning meets advanced robotic manipulation,” *Information Fusion*, vol. 105, p. 102 221, 2024.

- [100] C. Guo, W. Luk, S. Q. S. Loh, A. Warren, and J. Levine, “Customisable control policy learning for robotics,” in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, IEEE, vol. 2160, 2019, pp. 91–98.
- [101] O. Kilinc and G. Montana, “Reinforcement learning for robotic manipulation using simulated locomotion demonstrations,” *Machine Learning*, pp. 1–22, 2022.
- [102] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International conference on machine learning*, PMLR, 2018, pp. 1587–1596.
- [103] H. Hasselt, “Double q-learning,” *Advances in neural information processing systems*, vol. 23, 2010.
- [104] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.
- [105] T. Joshi, S. Makker, H. Kodamana, and H. Kandath, “Twin actor twin delayed deep deterministic policy gradient (tatd3) learning for batch process control,” *Computers & Chemical Engineering*, vol. 155, p. 107 527, 2021.
- [106] J. Langaa and C. Sloth, “Expert initialized reinforcement learning with application to robotic assembly,” in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2022, pp. 1405–1410.
- [107] R. Fernandez-Fernandez, M. Aggravi, P. R. Giordano, J. G. Victores, and C. Pacchierotti, “Neural style transfer with twin-delayed ddpg for shared control of robotic manipulators,” in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 4073–4079.
- [108] M. Zhan, J. Chen, C. Du, and Y. Duan, “Twin delayed multi-agent deep deterministic policy gradient,” in *2021 IEEE International Conference on Progress in Informatics and Computing (PIC)*, IEEE, 2021, pp. 48–52.

- [109] A. Iriondo, E. Lazkano, A. Ansuategi, A. Rivera, I. Lluvia, and C. Tubio, “Learning positioning policies for mobile manipulation operations with deep reinforcement learning,” *International Journal of Machine Learning and Cybernetics*, pp. 1–21, 2023.
- [110] K. S. Srinivas, R. Kumar, T. Sudheer, M. Somasundaram, and T. Suresh, “Industrial robotic arm design enabled by ai-based deep reinforcement learning using a combination of deep deterministic policy gradient (ddpg) and hindsight experience replay (her).,” *Turkish Online Journal of Qualitative Inquiry*, vol. 12, no. 7, 2021.
- [111] S. B. Remman and A. M. Lekkas, “Robotic lever manipulation using hindsight experience replay and shapley additive explanations,” in *2021 European Control Conference (ECC)*, IEEE, 2021, pp. 586–593.
- [112] D. C. Crowder, J. Abreu, and R. F. Kirsch, “Hindsight experience replay improves reinforcement learning for control of a mimo musculoskeletal model of the human arm,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 29, pp. 1016–1025, 2021.
- [113] Z. Gao, R. Tang, L. Chen, Q. Huang, and J. He, “Continuous shared control in prosthetic hand grasp tasks by deep deterministic policy gradient with hindsight experience replay,” *International Journal of Advanced Robotic Systems*, vol. 17, no. 4, p. 1 729 881 420 936 851, 2020.
- [114] Y. Shao, H. Zhou, S. Zhao, X. Fan, and J. Jiang, “A control method of robotic arm based on improved deep deterministic policy gradient,” in *2023 IEEE International Conference on Mechatronics and Automation (ICMA)*, IEEE, 2023, pp. 473–478.
- [115] Z. Ren, K. Dong, Y. Zhou, Q. Liu, and J. Peng, “Exploration via hindsight goal generation,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

-
- [116] L. Rüschendorf, “The wasserstein distance and approximation theorems,” *Probability Theory and Related Fields*, vol. 70, no. 1, pp. 117–129, 1985.
 - [117] Z. Bing, M. Brucker, F. O. Morin, R. Li, X. Su, K. Huang, and A. Knoll, “Complex robotic manipulation via graph-based hindsight goal generation,” *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 7863–7876, 2021.
 - [118] Y. Lai, W. Wang, Y. Yang, J. Zhu, and M. Kuang, “Hindsight planner,” in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 690–698.
 - [119] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
 - [120] P. Dendorfer, A. Osep, and L. Leal-Taixé, “Goal-gan: Multimodal trajectory prediction based on goal position estimation,” in *Proceedings of the Asian Conference on Computer Vision*, 2020.
 - [121] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.
 - [122] A. Deshpande, S. Sarma, A. Jha, and B. Ravindran, “Improvements on hindsight learning,” *arXiv preprint arXiv:1809.06719*, 2018.
 - [123] X. Cao, H. Wan, Y. Lin, and S. Han, “High-value prioritized experience replay for off-policy reinforcement learning,” in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, 2019, pp. 1510–1514.
 - [124] T. M. Luu and C. D. Yoo, “Hindsight goal ranking on replay buffer for sparse reward environment,” *IEEE Access*, vol. 9, pp. 51 996–52 007, 2021.
 - [125] J. B. Lanier, *Curiosity-driven multi-criteria hindsight experience replay*. University of California, Irvine, 2019.
 - [126] R. Zhao and V. Tresp, “Curiosity-driven experience prioritization via density estimation,” *arXiv preprint arXiv:1902.08039*, 2019.
-

- [127] H. Zeng, P. Zhang, F. Li, C. Lin, and J. Zhou, “Ahegc: Adaptive hindsight experience replay with goal-amended curiosity module for robot control,” *IEEE Transactions on Neural Networks and Learning Systems*, 2023.
- [128] T. Dai, H. Liu, K. Arulkumaran, G. Ren, and A. A. Bharath, “Diversity-based trajectory and goal selection with hindsight experience replay,” in *PRICAI 2021: Trends in Artificial Intelligence: 18th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2021, Hanoi, Vietnam, November 8–12, 2021, Proceedings, Part III 18*, Springer, 2021, pp. 32–45.
- [129] R. Zhao and V. Tresp, “Energy-based hindsight experience prioritization,” in *Conference on Robot Learning*, PMLR, 2018, pp. 113–122.
- [130] M. Fang, T. Zhou, Y. Du, L. Han, and Z. Zhang, “Curriculum-guided hindsight experience replay,” *Advances in neural information processing systems*, vol. 32, 2019.
- [131] Z. Bing, H. Zhou, R. Li, X. Su, F. O. Morin, K. Huang, and A. Knoll, “Solving robotic manipulation with sparse reward reinforcement learning via graph-based diversity and proximity,” *IEEE Transactions on Industrial Electronics*, vol. 70, no. 3, pp. 2759–2769, 2022.
- [132] Y. Yin, Z. Chen, G. Liu, and J. Guo, “A mapless local path planning approach using deep reinforcement learning framework,” *Sensors*, vol. 23, no. 4, p. 2036, 2023.
- [133] N. Vulin, S. Christen, S. Stevšić, and O. Hilliges, “Improved learning of robot manipulation tasks via tactile intrinsic motivation,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2194–2201, 2021.
- [134] Y. Kang, E. Zhao, Y. Zang, L. Li, K. Li, P. Tao, and J. Xing, “Sample efficient reinforcement learning using graph-based memory reconstruction,” *IEEE Transactions on Artificial Intelligence*, 2023.

- [135] M. Al-Gabalawy, “Path planning of robotic arm based on deep reinforcement learning algorithm,” *Advanced Control for Applications: Engineering and Industrial Systems*, vol. 4, no. 1, e79, 2022.
- [136] L. Moro, A. Likmeta, E. Prati, and M. Restelli, “Goal-directed planning via hindsight experience replay,” in *International Conference on Learning Representations*, 2021.
- [137] J. Fong, D. Campolo, C. Acar, and K. P. Tee, “Model-based reinforcement learning with lstm networks for non-prehensile manipulation planning,” in *2021 21st international conference on control, automation and systems (ICCAS)*, IEEE, 2021, pp. 1152–1158.
- [138] Y. Labbé, S. Zagoruyko, I. Kalevatykh, I. Laptev, J. Carpentier, M. Aubry, and J. Sivic, “Monte-carlo tree search for efficient visually guided rearrangement planning,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3715–3722, 2020.
- [139] J. Jacob, T. Bandyopadhyay, J. Williams, P. Borges, and F. Ramos, “Learning to simulate tree-branch dynamics for manipulation,” *IEEE Robotics and Automation Letters*, 2024.
- [140] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, “Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search,” in *2016 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2016, pp. 528–535.
- [141] R. Yang, M. Fang, L. Han, Y. Du, F. Luo, and X. Li, “Mher: Model-based hindsight experience replay,” *arXiv preprint arXiv:2107.00306*, 2021.
- [142] M. Lutter, L. Hasenclever, A. Byravan, G. Dulac-Arnold, P. Trochim, N. Heess, J. Merel, and Y. Tassa, “Learning dynamics models for model predictive agents,” *arXiv preprint arXiv:2109.14311*, 2021.

- [143] R. McCarthy, F. R. Sanchez, Q. Wang, D. C. Bulens, K. McGuinness, N. O'Connor, and S. J. Redmond, "Solving the real robot challenge using deep reinforcement learning," *arXiv preprint arXiv:2109.15233*, 2021.
- [144] Q. Wang, F. R. Sanchez, R. McCarthy, D. C. Bulens, K. McGuinness, N. O'Connor, M. Wüthrich, F. Widmaier, S. Bauer, and S. J. Redmond, "Dexterous robotic manipulation using deep reinforcement learning and knowledge transfer for complex sparse reward-based tasks," *Expert Systems*, vol. 40, no. 6, e13205, 2023.
- [145] R. Nagpal, A. U. Krishnan, and H. Yu, "Reward engineering for object pick and place training," *arXiv preprint arXiv:2001.03792*, 2020.
- [146] R. Zeng, M. Liu, J. Zhang, X. Li, Q. Zhou, and Y. Jiang, "Manipulator control method based on deep reinforcement learning," in *2020 Chinese Control And Decision Conference (CCDC)*, IEEE, 2020, pp. 415–420.
- [147] Z. Wu, W. Lian, V. Unhelkar, M. Tomizuka, and S. Schaal, "Learning dense rewards for contact-rich manipulation tasks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 6214–6221.
- [148] C. Chen, H. Zhang, Y. Pan, and D. Li, "Robot autonomous grasping and assembly skill learning based on deep reinforcement learning," *The International Journal of Advanced Manufacturing Technology*, pp. 1–17, 2024.
- [149] X. Liang, K. Shu, K. Lee, and P. Abbeel, "Reward uncertainty for exploration in preference-based reinforcement learning," *arXiv preprint arXiv:2205.12401*, 2022.
- [150] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Icml*, vol. 99, 1999, pp. 278–287.

- [151] A. Gupta, A. Pacchiano, Y. Zhai, S. Kakade, and S. Levine, “Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 281–15 295, 2022.
- [152] R. Devidze, P. Kamalaruban, and A. Singla, “Exploration-guided reward shaping for reinforcement learning under sparse rewards,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 5829–5842, 2022.
- [153] D. Yang and Y. Tang, “Adaptive inner-reward shaping in sparse reward games,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–8.
- [154] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, *et al.*, “Multi-goal reinforcement learning: Challenging robotics environments and request for research,” *arXiv preprint arXiv:1802.09464*, 2018.
- [155] Z. Deng and J. W. Zhang, “Learning synergies based in-hand manipulation with reward shaping,” *CAAI Transactions on Intelligence Technology*, vol. 5, no. 3, pp. 141–149, 2020.
- [156] Y. Luo, K. Dong, L. Zhao, Z. Sun, C. Zhou, and B. Song, “Balance between efficient and effective learning: Dense2sparse reward shaping for robot manipulation with environment uncertainty,” *arXiv preprint arXiv:2003.02740*, 2020.
- [157] Y. Cui, Z. Xu, L. Zhong, P. Xu, Y. Shen, and Q. Tang, “A task-adaptive deep reinforcement learning framework for dual-arm robot manipulation,” *IEEE Transactions on Automation Science and Engineering*, 2024.
- [158] B. Kim, G. Kwon, C. Park, and N. K. Kwon, “The task decomposition and dedicated reward-system-based reinforcement learning algorithm for pick-and-place,” *Biomimetics*, vol. 8, no. 2, p. 240, 2023.

- [159] G. Konidaris and A. Barto, “Autonomous shaping: Knowledge transfer in reinforcement learning,” in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 489–496.
- [160] A. C. Tenorio-Gonzalez, E. F. Morales, and L. Villasenor-Pineda, “Dynamic reward shaping: Training a robot by voice,” in *Advances in Artificial Intelligence–IBERAMIA 2010: 12th Ibero-American Conference on AI, Bahia Blanca, Argentina, November 1-5, 2010. Proceedings 12*, Springer, 2010, pp. 483–492.
- [161] A. D. Laud, *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.
- [162] H. Nguyen, H. M. La, and M. Deans, “Hindsight experience replay with experience ranking,” in *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, IEEE, 2019, pp. 1–6.
- [163] M. Saeed, M. Nagdi, B. Rosman, and H. H. Ali, “Deep reinforcement learning for robotic hand manipulation,” in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, IEEE, 2021, pp. 1–5.
- [164] R. Zhao, X. Sun, and V. Tresp, “Maximum entropy-regularized multi-goal reinforcement learning,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 7553–7562.
- [165] A. Sehgal, M. Sehgal, and H. M. La, “Aacher: Assorted actor-critic deep reinforcement learning with hindsight experience replay,” *arXiv preprint arXiv:2210.12892*, 2022.
- [166] N. Funk, C. Schaff, R. Madan, T. Yoneda, J. U. De Jesus, J. Watson, E. K. Gordon, F. Widmaier, S. Bauer, S. S. Srinivasa, *et al.*, “Benchmarking structured policies and policy optimization for real-world dexterous object manipulation,” *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 478–485, 2021.

- [167] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “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)*, IEEE, 2017, pp. 23–30.
- [168] R. Madan, H. Sikchi, E. K. Gordon, and T. Bhattacharjee, “Real robot challenge 2021: Cartesian position control with triangle grasp and trajectory interpolation,” *arXiv preprint arXiv:2203.08371*, 2022.
- [169] Q. Yao, J. Wang, and S. Yang, “Real-world dexterous object manipulation based deep reinforcement learning,” *arXiv preprint arXiv:2112.04893*, 2021.
- [170] T. Yoneda, C. Schaff, T. Maeda, and M. Walter, “Grasp and motion planning for dexterous manipulation for the real robot challenge,” *arXiv preprint arXiv:2101.02842*, 2021.
- [171] M. B. Hafez, C. Weber, M. Kerzel, and S. Wermter, “Deep intrinsically motivated continuous actor-critic for efficient robotic visuomotor skill learning,” *Paladyn, Journal of Behavioral Robotics*, vol. 10, no. 1, pp. 14–29, 2019.
- [172] C. Finn, S. Levine, and P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” in *International conference on machine learning*, PMLR, 2016, pp. 49–58.
- [173] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, “Data-efficient deep reinforcement learning for dexterous manipulation,” *arXiv preprint arXiv:1704.03073*, 2017.
- [174] J. Wu, H. Wu, Z. Qiu, J. Wang, and M. Long, “Supported policy optimization for offline reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 278–31 291, 2022.
- [175] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma, “Mopo: Model-based offline policy optimization,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 129–14 142, 2020.

-
- [176] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [177] N. Gürtler, F. Widmaier, C. Sancaktar, S. Blaes, P. Koley, S. Bauer, M. Wüthrich, M. Wulfmeier, M. Riedmiller, A. Allshire, *et al.*, “Real robot challenge 2022: Learning dexterous manipulation from offline data in the real world,” in *NeurIPS 2022 Competition Track*, PMLR, 2023, pp. 133–150.
- [178] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki, “Meta reinforcement learning for sim-to-real domain adaptation,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2725–2731.
- [179] B. Eysenbach, S. Asawa, S. Chaudhari, S. Levine, and R. Salakhutdinov, “Off-dynamics reinforcement learning: Training for transfer with domain classifiers,” *arXiv preprint arXiv:2006.13916*, 2020.
- [180] A. Melnik, L. Lach, M. Plappert, T. Korthals, R. Haschke, and H. Ritter, “Using tactile sensing to improve the sample efficiency and performance of deep deterministic policy gradients for simulated in-hand manipulation tasks,” *Frontiers in Robotics and AI*, p. 57, 2021.
- [181] N. A. Lynnerup, L. Nolling, R. Hasle, and J. Hallam, “A survey on reproducibility by evaluating deep reinforcement learning algorithms on real-world robots,” in *Conference on Robot Learning*, PMLR, 2020, pp. 466–489.
- [182] J. Leitner, A. W. Tow, N. Sünderhauf, J. E. Dean, J. W. Durham, M. Cooper, M. Eich, C. Lehnert, R. Mangels, C. McCool, *et al.*, “The acrv picking benchmark: A robotic shelf picking benchmark to foster reproducible research,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 4705–4712.
- [183] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2012, pp. 5026–5033.

- [184] I. M. Bullock, J. Borràs, and A. M. Dollar, “Assessing assumptions in kinematic hand models: A review,” in *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, IEEE, 2012, pp. 139–146.
- [185] F. R. Sanchez, Q. Wang, D. C. Bulens, K. McGuinness, S. J. Redmond, and N. E. O’Connor, “Hierarchical reinforcement learning for in-hand robotic manipulation using davenport chained rotations,” in *2023 9th International Conference on Automation, Robotics and Applications (ICARA)*, IEEE, 2023, pp. 160–164.
- [186] H. Nguyen and H. La, “Review of deep reinforcement learning for robot manipulation,” in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, IEEE, 2019, pp. 590–595.
- [187] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, “A survey on deep reinforcement learning algorithms for robotic manipulation,” *Sensors*, vol. 23, no. 7, p. 3762, 2023.
- [188] O. Kroemer, S. Niekum, and G. Konidaris, “A review of robot learning for manipulation: Challenges, representations, and algorithms,” *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 1395–1476, 2021.
- [189] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters, “Towards learning hierarchical skills for multi-phase manipulation tasks,” in *2015 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2015, pp. 1503–1510.
- [190] M. J. Ain, E. E. Aksoy, and F. Wörgötter, “Library of actions: Implementing a generic robot execution framework by using manipulation action semantics,” *The International Journal of Robotics Research*, vol. 38, no. 8, pp. 910–934, 2019.
- [191] L. Rozo, M. Guo, A. G. Kupcsik, M. Todescato, P. Schillinger, M. Giftthaler, M. Ochs, M. Spies, N. Waniek, P. Kesper, *et al.*, “Learning and sequencing

- of object-centric manipulation skills for industrial tasks,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 9072–9079.
- [192] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid, “Learning to combine primitive skills: A step towards versatile robotic manipulation,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 4637–4643.
- [193] B. J. Clement and E. H. Durfee, “Top-down search for coordinating the hierarchical plans of multiple agents,” in *Proceedings of the third annual conference on Autonomous Agents*, 1999, pp. 252–259.
- [194] M. Liu, Y. Tan, and V. Padois, “Generalized hierarchical control,” *Autonomous Robots*, vol. 40, pp. 17–31, 2016.
- [195] A. Garland and N. Lesh, “Learning hierarchical task models by demonstration,” *Mitsubishi Electric Research Laboratory (MERL), USA*–(January 2002), pp. 51–79, 2003.
- [196] R. Lioutikov, O. Kroemer, G. Maeda, and J. Peters, “Learning manipulation by sequencing motor primitives with a two-armed robot,” in *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*, Springer, 2016, pp. 1601–1611.
- [197] M. Mayr, K. Chatzilygeroudis, F. Ahmad, L. Nardi, and V. Krueger, “Learning of parameters in behavior trees for movement skills,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 7572–7579.
- [198] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule, “Behavior trees and state machines in robotics applications,” *IEEE Transactions on Software Engineering*, 2023.

- [199] B. Bakker and J. Schmidhuber, “Hierarchical reinforcement learning with subpolicies specializing for learned subgoals,” in *Neural Networks and Computational Intelligence*, 2004, pp. 125–130.
- [200] M. M. Botvinick, “Hierarchical reinforcement learning and decision making,” *Current opinion in neurobiology*, vol. 22, no. 6, pp. 956–962, 2012.
- [201] O. Nachum, S. S. Gu, H. Lee, and S. Levine, “Data-efficient hierarchical reinforcement learning,” *Advances in neural information processing systems*, vol. 31, 2018.
- [202] M. D. Shuster and F. L. Markley, “General formula for extracting the euler angles,” *Journal of guidance, control, and dynamics*, vol. 29, no. 1, pp. 215–221, 2006.
- [203] N. Vuong, H. Pham, and Q.-C. Pham, “Learning sequences of manipulation primitives for robotic assembly,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 4086–4092.
- [204] A. I. Weinberg, A. Shirizly, O. Azulay, and A. Sintov, “Survey of learning approaches for robotic in-hand manipulation,” *arXiv preprint arXiv:2401.07915*, 2024.
- [205] M. Breyer, F. Furrer, T. Novkovic, R. Siegwart, and J. Nieto, “Comparing task simplifications to learn closed-loop object picking using deep reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1549–1556, 2019.
- [206] J. Felip, J. Laaksonen, A. Morales, and V. Kyrki, “Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks,” *Robotics and Autonomous Systems*, vol. 61, no. 3, pp. 283–296, 2013.
- [207] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen, “Automatic grasp planning using shape primitives,” in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, IEEE, vol. 2, 2003, pp. 1824–1829.

- [208] J. D. Morrow and P. K. Khosla, “Manipulation task primitives for composing robot skills,” in *Proceedings of International Conference on Robotics and Automation*, IEEE, vol. 4, 1997, pp. 3354–3359.
- [209] I. Weidauer, D. Kubus, and F. M. Wahl, “A hierarchical extension of manipulation primitives and its integration into a robot control architecture,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 5401–5407.
- [210] A. Skoglund, B. Iliev, B. Kadmiry, and R. Palm, “Programming by demonstration of pick-and-place tasks for industrial manipulators using task primitives,” in *2007 International Symposium on Computational Intelligence in Robotics and Automation*, IEEE, 2007, pp. 368–373.
- [211] L. Marzari, A. Pore, D. Dall’Alba, G. Aragon-Camarasa, A. Farinelli, and P. Fiorini, “Towards hierarchical task decomposition using deep reinforcement learning for pick and place subtasks,” in *2021 20th International Conference on Advanced Robotics (ICAR)*, IEEE, 2021, pp. 640–645.
- [212] T. Li, K. Srinivasan, M. Q.-H. Meng, W. Yuan, and J. Bohg, “Learning hierarchical control for robust in-hand manipulation,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 8855–8862.
- [213] A. Bhatt, A. Sieler, S. Puhlmann, and O. Brock, “Surprisingly robust in-hand manipulation: An empirical study,” *arXiv preprint arXiv:2201.11503*, 2022.
- [214] S. Abondance, C. B. Teeple, and R. J. Wood, “A dexterous soft robotic hand for delicate in-hand manipulation,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5502–5509, 2020.
- [215] N. Gürtler, D. Büchler, and G. Martius, “Hierarchical reinforcement learning with timed subgoals,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 732–21 743, 2021.

- [216] S. Pateria, B. Subagdja, A.-H. Tan, and C. Quek, “End-to-end hierarchical reinforcement learning with integrated subgoal discovery,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7778–7790, 2021.
- [217] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, “Hierarchical reinforcement learning: A comprehensive survey,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–35, 2021.
- [218] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, “Generating adjacency-constrained subgoals in hierarchical reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 579–21 590, 2020.
- [219] Y. Luo, Y. Wang, K. Dong, Q. Zhang, E. Cheng, Z. Sun, and B. Song, “Relay hindsight experience replay: Self-guided continual reinforcement learning for sequential object manipulation tasks with sparse rewards,” *Neurocomputing*, vol. 557, p. 126 620, 2023.
- [220] H. Goldstein, “Classical mechanics,” in 2nd. Reading, Penn., Addison-Wesley, 1980, ch. Appendix B.
- [221] J. Diebel *et al.*, “Representing attitude: Euler angles, unit quaternions, and rotation vectors,” *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.
- [222] E. Gavrilina and V. Chestnov, “Singularity-free attitude control of the unmanned underwater vehicle,” in *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*, IEEE, 2020, pp. 512–519.
- [223] B. Smeresky and A. Rizzo, “Kinematics: On direction cosine matrices,” *Advances in Spacecraft Attitude Control*, p. 19, 2020.
- [224] V. Carrara, “An open source satellite attitude and orbit simulator toolbox for matlab,” in *Proceedings of the 17th International Symposium on Dynamic Problems of Mechanics*, 2015, pp. 1–12.

- [225] D. Kim and H. Leeghim, “A new approach for handling under-actuated system control,” *Transactions of the Japan Society for Aeronautical and Space Sciences*, vol. 58, no. 2, pp. 110–111, 2015.
- [226] C. M. Roithmayr and D. H. Hodges, *Dynamics: Theory and application of kane’s method*, 2016.
- [227] E. G. Hemingway and O. M. O’Reilly, “Perspectives on euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments,” *Multibody system dynamics*, vol. 44, pp. 31–56, 2018.
- [228] J. B. Kuipers, *Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality*. Princeton university press, 1999.
- [229] R. A. Johnson, S. Verrill, and D. H. Moore, “Two-sample rank tests for detecting changes that occur in a small proportion of the treated population,” *Biometrics*, pp. 641–655, 1987.
- [230] T. L. Paine, C. Paduraru, A. Michi, C. Gulcehre, K. Zolna, A. Novikov, Z. Wang, and N. de Freitas, “Hyperparameter selection for offline reinforcement learning,” *arXiv preprint arXiv:2007.09055*, 2020.
- [231] F. R. Sanchez, Q. Wang, D. C. Bulens, K. McGuinness, S. Redmond, and N. O’Connor, “Learning and reusing primitive behaviours to improve hindsight experience replay sample efficiency,” *arXiv preprint arXiv:2310.01827*, 2023.
- [232] E. Ronco and P. Gawthrop, “Modular neural networks: A state of the art,” *Rapport Technique CSC-95026, Center of System and Control, University of Glasgow*. <http://www.mech.gla.ac.uk/control/report.html>, 1995.
- [233] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, “Learning modular neural network policies for multi-task and multi-robot transfer,” in *2017 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2017, pp. 2169–2176.
- [234] K. Chen, “Deep and modular neural networks,” *Springer Handbook of Computational Intelligence*, pp. 473–494, 2015.

- [235] W. Wu and Y. Pan, “Adaptive modular convolutional neural network for image recognition,” *Sensors*, vol. 22, no. 15, p. 5488, 2022.
- [236] H. H. Nguyen, T. N.-D. Tieu, H.-Q. Nguyen-Son, V. Nozick, J. Yamagishi, and I. Echizen, “Modular convolutional neural network for discriminating between computer-generated images and photographic images,” in *Proceedings of the 13th international conference on availability, reliability and security*, 2018, pp. 1–10.
- [237] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko, “Learning to reason: End-to-end module networks for visual question answering,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 804–813.
- [238] Z. Yu, J. Yu, Y. Cui, D. Tao, and Q. Tian, “Deep modular co-attention networks for visual question answering,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 6281–6290.
- [239] G. L. O. Massa, H. Vinuesa, and L. Lanzarini, “Modular creation of neuronal networks for autonomous robot control,” in *2006 IEEE 3rd Latin American Robotics Symposium*, IEEE, 2006, pp. 66–73.
- [240] C. W. Anderson and Z. Hong, “Reinforcement learning with modular neural networks for control,” *Proceedings of NNACIP*, vol. 94, pp. 90–93, 1994.
- [241] L. Kirsch, J. Kunze, and D. Barber, “Modular networks: Learning to decompose neural computation,” *Advances in neural information processing systems*, vol. 31, 2018.
- [242] S. Nasiriany, H. Liu, and Y. Zhu, “Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks,” in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 7477–7484.

- [243] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel, “Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning,” in *International Conference on Machine Learning*, PMLR, 2021, pp. 6131–6141.
- [244] T. Yu, A. Kumar, Y. Chebotar, K. Hausman, S. Levine, and C. Finn, “Conservative data sharing for multi-task offline reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 501–11 516, 2021.
- [245] F. Sasaki and R. Yamashina, “Behavioral cloning from noisy demonstrations,” in *International Conference on Learning Representations*, 2020.
- [246] Y. Wang, C. Yang, Y. Wen, Y. Liu, and Y. Qiao, “Critic-guided decision transformer for offline reinforcement learning,” *arXiv preprint arXiv:2312.13716*, 2023.
- [247] J. Kim, S. Lee, W. Kim, and Y. Sung, “Value-aided conditional supervised learning for offline rl,” *arXiv preprint arXiv:2402.02017*, 2024.
- [248] R. Zhang, Z. Luo, J. Sjölund, T. B. Schön, and P. Mattsson, “Entropy-regularized diffusion policy with q-ensembles for offline reinforcement learning,” *arXiv preprint arXiv:2402.04080*, 2024.
- [249] G. Zhang, A. Jain, I. Hwang, S.-H. Sun, and J. J. Lim, “Efficient multi-task reinforcement learning via selective behavior sharing,” *arXiv preprint arXiv:2302.00671*, 2023.
- [250] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, “Decision transformer: Reinforcement learning via sequence modeling,” *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.
- [251] M. Janner, Q. Li, and S. Levine, “Offline reinforcement learning as one big sequence modeling problem,” *Advances in neural information processing systems*, vol. 34, pp. 1273–1286, 2021.

- [252] F. R. Sanchez, S. Redmond, K. McGuinness, and N. O'Connor, "Towards advanced robotic manipulation," in *2022 Sixth IEEE International Conference on Robotic Computing (IRC)*, IEEE, 2022, pp. 302–305.
- [253] Y. Seo, D. Hafner, H. Liu, F. Liu, S. James, K. Lee, and P. Abbeel, "Masked world models for visual control," in *Conference on Robot Learning*, PMLR, 2023, pp. 1332–1344.
- [254] P. Wu, A. Majumdar, K. Stone, Y. Lin, I. Mordatch, P. Abbeel, and A. Rajeswaran, "Masked trajectory models for prediction, representation, and control," in *International Conference on Machine Learning*, PMLR, 2023, pp. 37 607–37 623.
- [255] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta, "R3m: A universal visual representation for robot manipulation," *arXiv preprint arXiv:2203.12601*, 2022.
- [256] H. Xiong and X. Diao, "Safety robustness of reinforcement learning policies: A view from robust control," *Neurocomputing*, vol. 422, pp. 12–21, 2021.
- [257] G. Ji, J. Yan, J. Du, W. Yan, J. Chen, Y. Lu, J. Rojas, and S. S. Cheng, "Towards safe control of continuum manipulator using shielded multiagent reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7461–7468, 2021.
- [258] A. C. Ak, E. E. Aksoy, and S. Sariel, "Learning failure prevention skills for safe robot manipulation," *IEEE Robotics and Automation Letters*, 2023.
- [259] H.-L. Hsu, Q. Huang, and S. Ha, "Improving safety in deep reinforcement learning using unsupervised action planning," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 5567–5573.
- [260] A. Dahiya, A. M. Aroyo, K. Dautenhahn, and S. L. Smith, "A survey of multi-agent human–robot interaction systems," *Robotics and Autonomous Systems*, vol. 161, p. 104 335, 2023.

- [261] S. W. Abeyruwan, L. Graesser, D. B. D'Ambrosio, A. Singh, A. Shankar, A. Bewley, D. Jain, K. M. Choromanski, and P. R. Sanketi, "I-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops," in *Conference on Robot Learning*, PMLR, 2023, pp. 212–224.
- [262] D. Kim, M. Choi, and J. Um, "Digital twin for autonomous collaborative robot by using synthetic data and reinforcement learning," *Robotics and Computer-Integrated Manufacturing*, vol. 85, p. 102632, 2024.