



Learning from Demonstration using Hierarchical Inverse Reinforcement Learning

Leonor Baptista da Costa Silva Santos

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Gil Gonçalves

Co-supervisor: Liliana Antão

Co-supervisor: João Reis

October 25, 2021

Resumo

Com a evolução conjunta da Indústria e Robótica, os sistemas produtivos tornaram-se cada vez mais complexos, robustos e seguros. No entanto, a sociedade requer que a Indústria se adapte às suas necessidades de uma forma rápida, suave e flexível.

A programação contemporânea incorpora metodologias inteligentes no chão de fábrica. Contudo, estas não são flexíveis ou recicláveis quando expostas a novas configurações de produto ou produção. De igual modo, as metodologias raramente conseguem transferir capacidades a outras tarefas. Tudo isto combinado, consome muitos recursos.

O primeiro objetivo foi criar um cenário de simulação no seio da Indústria, que fosse facilmente transferido para situações reais, ao mesmo tempo que seguisse as técnicas de *Reinforcement Learning*. O segundo objetivo foi o estudo sobre como discretizar uma tarefa. Por fim, o último objetivo envolveu avaliar o impacto da reutilização de modelos pré-treinados em diferentes sub-tarefas.

A metodologia escolhida recorre à *framework robo-gym* para interligar o *OpenAI Gym* com o simulador de física *Gazebo* e assim criar o cenário de *pick and place* modificado. A modificação corresponde à adição da tarefa de ajuste do objeto ao encaixe desejado. O treino do agente envolve demonstrações, tal como definido na metodologia denominada de *Learning from Demonstration*. O algoritmo de treino escolhido foi o *Generative Adversarial Imitation Learning*, que partilha características tanto de *Reinforcement Learning* como de *Inverse Reinforcement Learning*.

A primeira conclusão principal foi a de que discretização de tarefas pode ser realizada através da modulação da função de recompensa. A equação deve considerar um gradiente suave de erro e recompensas positivas associadas à conclusão de cada uma das sub-tarefas. As recompensas positivas devem, por um lado, ser sequencialmente maiores, à medida que a sequência progride. Por outro lado, os incrementos entre as recompensas devem também ser crescentes. Assim, a metodologia de discretização consegue simplificar tarefas ditas complexas e melhora desempenho dos agentes, face às metodologias tradicionais.

Adicionalmente, provámos que o re-treino de modelos pode ser vantajoso quando não são necessárias novas capacidades ou quando o balanço entre adaptação e exploração é positivo. Nesse caso, as curvas de aprendizagem são muito mais estáveis.

Esta proposta reúne um conjunto de regras para flexibilizar e simplificar a programação de uma tarefa industrial, baseada numa função de recompensa. As respetivas análises e conclusões resultaram numa publicação proposta para a conferência *ICAR*. Uma segunda publicação está a ser preparada.

Palavras-Chave— Robótica, Engenharia Robótica, Reinforcement Learning, Inverse Reinforcement Learning, Generative Adversarial Imitation Learning, Demonstrações, Função de Recompensa, Discretização, Hierarquização, Reutilização de Modelos

Abstract

With the joint evolution of Industry and Robotics, manufacturing systems are becoming more complex, resilient, and safer. At the same time, Industry 4.0 answers the requirements to adapt to society demands as fast, seamless, and flexible as possible. Despite bringing intelligence methodologies to the factory floor, the contemporary robotic engineering techniques fail to be flexible or resilient when faced with new configurations of new products or different production parameters. Also, currently used methodologies rarely allow to transfer skills from other tasks. All combined, these factors make the development and updating of robotic systems a cumbersome task that requires extensive resources. To address these limitations, this thesis explores how the use of Learning from Demonstration can contribute to improve robotic engineering.

The first goal was to create a manufacturing simulation scenario that could easily transfer to real situations while reacting to Reinforcement Learning Techniques. The second objective was to study how to discretise a complex task. The last goal assessed the impact of reusing pre-trained models in different tasks.

The methodology used the robo-gym framework, that connects the OpenAI Gym with the Gazebo physics engine, to create a modified pick and place task, where an object had to be fitted in a goal pose. The training involved expert demonstrations as part of the Learning from Demonstration scope. The algorithm employed was the Generative Adversarial Imitation Learning, which shares both Reinforcement Learning and Inverse Reinforcement Learning characteristics.

The first key finding was that task discretisation could be achieved with reward function modelling. It can be done with a default smooth gradient error and positive rewards associated with sub-tasks completion. The positive rewards must be sequentially higher as well as the increments between them. This discretisation approach simplifies the complexity associated with the tasks and boosts performance compared to the sequential modelling approach.

Secondly, we have proved that retraining models can be sometimes advantageous even when new skills are not required, or when the trade-off between adaptation and exploration is positive. In that case, the learning curve is more stable. This proposal gathers guidelines to flexibilise and simplify engineering associated with a manufacturing task based on a reward function.

The work developed in this thesis resulted in a paper already submitted to ICAR and in a second paper under preparation.

Keywords— Robotics, Robotics Engineering, Reinforcement Learning, Inverse Reinforcement Learning, Generative Adversarial Imitation Learning, Expert Demonstrations, Reward Function, Discretization, Hierarchisation, Model Reutilization

Acknowledgements

Ao Professor Doutor Gil Gonçalves, orientador, por proporcionar-me as ferramentas e indicações para percorrer este caminho.

À Liliana Antão, co-orientadora, pelas horas infindáveis de discussão e revisão, pela confiança de que o barco iria chegar a bom porto, pelo ânimo e pela irmandade na Robótica. Ao Professor Doutor João Reis, pelas ideias fora da caixa e pelos vários dois cêntimos que se calhar já davam para comprar mais um livro.

Aos membros do DIGI2 Lab, por me acolherem no mundo das questões pertinentes.

I would like to express my great gratitude to Matteo Lucchi, for the help outside borders, the detailed guidelines and kind clues when I faced the first obstacles in simulation.

À minha mãe e ao meu pai, em *ex aequo*, por serem os pilares, o porto seguro, a paciência e o exemplo.

Aos amigos de todas as horas, pelo amparo e pelos ouvidos, mesmo quando as minhas piadas não são entendidas. Em especial aos A, B, B, F, M, P.

Aos colegas desta batalha, pela multiplicação da motivação e resiliência.

Aos burpees por me ensinarem perseverança no meio do sufoco.

E à banda sonora desta viagem: AC/DC, Alter Bridge, The Amazons, The Antlers, Arctic Monkeys, Avenged Sevenfold, Beartooth, Billie Eilish, The Black Keys, blink-182, Bullet For My Valentine, Childish Gambino, Cigarettes After Sex, Dire Straits, Disturbed, Dorothy, DragonForce, Dream Theater, Eric Clapton, Explosions In The Sky, Falling In Reverse, Feng Suave, FINNEAS, Five Finger Death Punch, FKJ, Foals, Foo Fighters, Glass Animals, GNR, Green Day, Guns N' Roses, Interpol, Iron Maiden, Jimi Hendrix, John Frusciante, Kasabian, Kiss, The Kooks, Led Zeppelin, Linda Martini, Linkin Park, A Lot Like Birds, Matt Bellamy, Metallica, Muse, My Chemical Romance, The National, Nightwish, Nirvana, Nothing But Thieves, Oasis, The Offspring, Ornatos Violeta, Our Last Night, Pantera, Papa Roach, Pearl Jam, Periphery, Pink Floyd, Placebo, Polyphia, Post Nebbia, Queens of the Stone Age, Radiohead, Rage Against The Machine, Ramin Djawadi, Read Hot Chili Peppers, Rex Orange County, Rise Against, Royal Blood, System Of A Down, Sleeping With Sirens, The Smashing Pumpkins, Snow Patrol, Steve Vai, The Strokes, Sum 41, Tame Impala, Tash Sultana, Tokio Myers, Trivium, U2, Unknown Mortal Orchestra, Van Halen, The Who, Xutos e Pontapés...

Leonor

*“Dragged by the force of some inner tide
At a higher altitude with flag unfurled
We reached the dizzy heights of that dreamed of world
Encumbered forever by desire any ambition
There’s a hunger still unsatisfied
Our weary eyes still stray to the horizon
Though down this road we’ve been so many times”*

Pink Floyd. High Hopes. 1994

Contents

1	Robots and Industry: challenges and opportunities	1
1.1	Context	1
1.2	Problem Definition	2
1.3	Motivation	3
1.4	Objectives	4
1.5	Keywords and Research Questions	4
1.6	Document Overview	5
2	Background	7
2.1	Reinforcement Learning	7
2.2	Learning from Demonstration	14
2.3	Inverse Reinforcement Learning	21
2.4	Generative Adversarial Networks	26
2.5	Hierarchical approaches	30
2.6	Chapter Synopsis	32
3	Related Work	35
3.1	Reinforcement Learning Approaches	35
3.2	Inverse Reinforcement Learning Approaches	42
3.3	Hierarchical Inverse Reinforcement Learning Approaches	46
3.4	Generative Adversarial Imitation Learning Approaches	51
3.5	Chapter Synopsis	53
4	Robotic Systems Simulators	55
4.1	Robotics Simulators Comparison	55
4.2	Chapter Synopsis	60
5	Learning from Demonstration using HIRL	61
5.1	Case Study	61
5.2	Proposed Solution	63
5.3	Case Study’s Low-level Simulation	64
5.4	Simulation Assessment	71
5.5	Chapter Synopsis	74
6	Environment Description and Training Setup	75
6.1	Differences to the Fetch Environment	75
6.2	Observation Space	77
6.3	Action Space	79
6.4	Environment State on Reset	80
6.5	Training Algorithm Setup	80
6.6	Chapter Synopsis	82

7	Reach Sub-Task	83
7.1	Reward Function	84
7.2	Building Expert Trajectories	85
7.3	Experiments and Results	87
7.4	Chapter Synopsis	88
8	Pick Sub-Task	89
8.1	Reward Function	90
8.2	Building Expert Trajectories	91
8.3	Experiments and Results	93
8.4	Chapter Synopsis	125
9	Move Sub-Task	127
9.1	Reward Function	128
9.2	Building Expert Trajectories	129
9.3	Experiments and Results	131
9.4	Chapter Synopsis	138
10	Fit Sub-Task	141
10.1	Reward Function	142
10.2	Building Expert Trajectories	144
10.3	Experiments and Results	147
10.4	Chapter Synopsis	151
11	Conclusions and Future Work	153
A	Training specifications	157
B	Pick Results - Reward Function 1	159
C	Pick Results - Reward Function 2	171
D	Pick Results- Reward Function 3	179
E	Pick Results- Reward Function 4	183
F	Pick Results- Reward Function 5	187
G	Pick Results- Reward Function 6	193
H	Pick Results- Reward Function 7	201
I	Move Results- Reward Function 1	205
J	Move Results- Reward Function 2	207
K	Move Results- Reward Function 3	209
L	Move Results- Reward Function 4	211
M	Move Results- Reward Function 5	213
N	Move Results- Reward Function 6	215
O	Move Results- Reward Function 7	217
P	Move Results- Reward Function 8	219

Q Move Results- Reward Function 9	221
R Fit Results- Reward Function 1	223
S Fit Results- Reward Function 2	225
T Fit Results- Reward Function 3	227
U Fit Results- Reward Function 4	229
References	231

List of Figures

1.1	Hierarchical relation between operations, tasks and products	3
2.1	Reinforcement Learning as Machine Learning sub-field	7
2.2	Example on Tic Tac Toe Game	8
2.3	Agent-Environment Loop	9
2.4	Transition Graph Example	10
2.5	Backup diagrams	12
2.6	Control policy derivation and execution	15
2.7	Policy derivation methods	16
2.8	Policy derivation categories	17
2.9	Correspondence Issue	18
2.10	Generalised Policy Interaction	21
2.11	IRL as modern IOC	23
2.12	IRL Techniques	24
2.13	Generative adversarial networks training	26
3.1	Hybrid architecture for a robotic locomotion system	36
3.2	Adaptive Hierarchical Memory Structure for Autonomous Robots: Proposed System	37
3.3	System overview of the automatic robot skills learning system	40
3.4	Skills Reproduction Module in automatic robot skills learning system	40
3.5	Feature Construction for Inverse Reinforcement Learning-Performance Results	42
3.6	Feature Construction for Inverse Reinforcement Learning-Generalisation Results	43
3.7	Robot Navigation in Crowds- Results	44
3.8	Long-Horizon Tasks with Delayed Rewards- Augmentation Results	47
3.9	Long-Horizon Tasks with Delayed Rewards- Parallel Parking Demonstrations	47
3.10	Long-Horizon Tasks with Delayed Rewards- Convergence Results	47
3.11	Long-Horizon Tasks with Delayed Rewards- Robustness Results	48
3.12	Long-Horizon Tasks with Delayed Rewards- Difficulty Tests	48
3.13	Probabilistic Prediction of Interactive Driving Behaviour- Modelling	49
3.14	GAIL Results	51
3.15	HGAIL results for the Reach and Grasp Environments	52
3.16	Comparative results between HER, GAIL and goalGail	52
4.1	OpenAI Gym environment: Fetch Pick and Place	56
4.2	V-REP Simulator	57
4.3	Unity simulator	58
4.4	Gazebo simulator	58
4.5	The robo-gym architecture	59
5.1	Case study description.	62
5.2	Proposed solution for maximising model reuse in different task specifications	63
5.3	UR5 Robotic Manipulator model	65
5.4	Robotiq 2 finger gripper models	66
5.5	Client-Server Interaction	67

5.6	Object Manipulation	69
5.7	Results on the reaching task with one controllable joint	73
6.1	Definition of the angle ϕ	76
6.2	Gripper real pose on observation and action spaces	77
6.3	Fit Measure where the object (orange) overlaps half the goal (blue), $f = 0.5$	79
7.1	Sub-task 1 - Reaching the object	83
7.2	Sub-tasks on reaching the object	85
7.3	Required number of steps for expert demonstrations on the reaching task	86
7.4	Reach task, object trajectory vs gripper trajectory on expert demonstration 7	86
7.5	Success rate curve for reach agent, with ten expert demos	87
7.6	Reach - Evolution of the cumulative reward per episode.	88
7.7	Reach- Train Accuracy and loss for Reward proposal 1	88
8.1	Sub-task 2 - Picking the object	89
8.2	Sub-tasks on picking the object	91
8.3	Required number of steps for expert demonstrations on the pick task	92
8.4	Pick task, object trajectory vs gripper trajectory on expert demonstration 8	92
8.5	Pick- Success rate curve for pick agent trained anew, with ten expert demos and reward model 1	98
8.6	Pick - Success rate curve for pick agent trained anew, with ten expert demos and reward model 2	102
8.7	Pick- Success rate curve for pick agent trained anew, with ten expert demos and reward model 3	105
8.8	Pick- Success rate curve for pick agent trained anew, with ten expert demos and reward model 4	110
8.9	Pick- Success rate curve for pick agent trained anew, with ten expert demos and reward model 5	112
8.10	Pick- Success rate curve for pick agent trained anew, with ten expert demos and reward model 6	115
8.11	Penalisation profile for early gripper closing. The area withing grey dotted line on the figure a) is zoomed in on figure b)	118
8.12	Pick- Success rate curve for pick agent trained anew, with ten expert demos and reward model 7	119
8.13	Pick- Success rate curve for pick agent trained anew, with ten expert demos on rewards 6 and 7	121
8.14	Pick- Success rate curve, with ten expert demos on all reward models	122
8.15	Two sub-goals. The success rate of the proposed sub-goal-based reward methodology (red) vs a traditional reward method (blue). The light colours represent the reaching sub-goal and the darker colours the picking.	124
8.16	Hierarchical results on the pick sub-task	125
9.1	Sub-task 3 - Moving the object. The red shape represents the goal pose and the blue one is the real object.	127
9.2	Sub-tasks on moving the object	129
9.3	Required number of steps for expert demonstrations on the moving task	130
9.4	Move task, object trajectory vs gripper trajectory on expert demonstration 1	130
9.5	Move- Success rate curve, with ten expert demos on all reward models	133
9.6	Three sub-goals. The success rate of the proposed methodology (red) vs the traditional method (blue).	133
9.7	Move- Traditional agent, Episode 991- Trajectory plot	134
9.8	Hierarchical results on the move sub-task, knowing how to reach	135
9.9	Hierarchical results on the move sub-task, knowing how to pick	136
9.10	Move retrain- Train Accuracy and loss	137

9.11	Required number of steps for expert demonstrations on the reaching task, retraining the move agent from the picking models	137
10.1	Sub-task 4 - Fitting the object. The red sub represents the goal pose and the blue one is the real object.	141
10.2	Intersection area to the fit measurement	142
10.3	Sub-task 4- Fitting the object	144
10.4	Fit task, object trajectory vs gripper trajectory on expert demonstration 3	145
10.5	Fit task, reward evolution on expert demonstration 3	146
10.6	Fit task, reward evolution on expert demonstration 3	146
10.7	Fit- Success rate curve, with ten expert demos on all reward models	148
10.8	Hierarchical results on the fit sub-task, knowing how to pick	149
10.9	Hierarchical results on the fit sub-task, knowing how to move	150
B.1	Pick- Reward proposal 1 Example 1 Episode 167	160
B.2	Pick- Reward proposal 1 Example 2 Episode 314	161
B.3	Pick- Reward proposal 1 Example 3 Episode 4	162
B.4	Pick- Reward proposal 1 Example 4 Episode 149	163
B.5	Pick- Reward proposal 1 Example 5 Episode 292	164
B.6	Pick- Reward proposal 1 Example 6 Episode 506	165
B.7	Pick- Reward proposal 1 Example 7 Episode 382	166
B.8	Pick- Reward proposal 1 Example 8 Episode 370	167
B.9	Pick- Reward proposal 1 Example 9 Episode 64	168
B.10	Pick- Training behaviour for Reward proposal 1	169
B.11	Pick- GAIL training behaviour for Reward proposal 1	170
C.1	Pick- Reward proposal 2 Example 1 Episode 4	172
C.2	Pick- Reward proposal 2 Example 2 Episode 9	173
C.3	Pick- Reward proposal 2 Example 3 Episode 165	174
C.4	Pick- Reward proposal 2 Example 4 Episode 154	175
C.5	Pick- Training behaviour for Reward proposal 2	176
C.6	Pick- GAIL training behaviour for Reward proposal 2	177
D.1	Pick- Reward proposal 3 Example 1 Episode 344	180
D.2	Pick- Training behaviour for Reward proposal 3	181
D.3	Pick- GAIL training behaviour for Reward proposal 3	182
E.1	Pick- Reward proposal 4 Example 1 Episode 259	184
E.2	Pick- Training behaviour for Reward proposal 4	185
E.3	Pick- GAIL training behaviour for Reward proposal 4	186
F.1	Pick- Reward proposal 5 Example 1 Episode 80	188
F.2	Pick- Reward proposal 5 Example 2 Episode 609	189
F.3	Pick- Training behaviour for Reward proposal 5	190
F.4	Pick- GAIL training behaviour for Reward proposal 5	191
G.1	Pick- Reward proposal 6 Example 1 Episode 8	194
G.2	Pick- Reward proposal 6 Example 2 Episode 843	195
G.3	Pick- Reward proposal 6 Example 3 Episode 786	196
G.4	Pick- Reward proposal 6 Example 4 Episode 1000	197
G.5	Pick- Training behaviour for Reward proposal 6	198
G.6	Pick- GAIL training behaviour for Reward proposal 6	199
H.1	Pick- Reward proposal 7 Example 1 Episode 751	202
H.2	Pick- Training behaviour for Reward proposal 7	203
H.3	Pick- GAIL training behaviour for Reward proposal 7	204

I.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 1	205
I.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 1	205
I.3	Move- GAIL training behaviour for Reward proposal 1	206
J.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 2	207
J.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 2	207
J.3	Move- GAIL training behaviour for Reward proposal 2	208
K.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 3	209
K.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 3	209
K.3	Move- GAIL training behaviour for Reward proposal 3	210
L.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 4	211
L.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 4	211
L.3	Move- GAIL training behaviour for Reward proposal 4	212
M.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 5	213
M.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 5	213
M.3	Move- GAIL training behaviour for Reward proposal 5	214
N.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 6	215
N.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 6	215
N.3	Move- GAIL training behaviour for Reward proposal 6	216
O.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 7	217
O.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 7	217
O.3	Move- GAIL training behaviour for Reward proposal 7	218
P.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 8	219
P.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 8	219
P.3	Move- GAIL training behaviour for Reward proposal 8	220
Q.1	Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 9	221
Q.2	Move- Evolution of the cumulative reward per episode, for Reward proposal 9	221
Q.3	Move- GAIL training behaviour for Reward proposal 9	222
R.1	Fit- Success rate curve for fit agent trained anew, with ten expert demos and reward model 1	223
R.2	Fit- Evolution of the cumulative reward per episode, for Reward proposal 1	223
R.3	Fit- GAIL training behaviour for Reward proposal 1	224
S.1	Fit- Success rate curve for fit agent trained anew, with ten expert demos and reward model 2	225
S.2	Fit- Evolution of the cumulative reward per episode, for Reward proposal 2	225
S.3	Fit- GAIL training behaviour for Reward proposal 2	226
T.1	Fit- Success rate curve for fit agent trained anew, with ten expert demos and reward model 3	227
T.2	Fit- Evolution of the cumulative reward per episode, for Reward proposal 3	227
T.3	Fit- GAIL training behaviour for Reward proposal 3	228

U.1	Fit- Success rate curve for fit agent trained anew, with ten expert demos and reward model 4	229
U.2	Fit- Evolution of the cumulative reward per episode, for Reward proposal 4	229
U.3	Fit- GAIL training behaviour for Reward proposal 4	230

List of Tables

4.1	Comparison between simulators	60
5.1	PID gains for the effort joint trajectory controller	65
5.2	Validation test: specifications	72
5.3	Reach task on the single joint, HER+SAC parameters	72
6.1	Observation's space description for the modified pick-and-place case study	77
6.2	Observation's space description on the picking task	79
6.3	Observation and Action spaces summed up	82
8.1	Proposed parameters of Eq. 8.1 for Reward Proposal 1	94
8.2	Pick Reward 1 - Observed situations and respective cumulative reward range.	97
8.3	Pick - Case distribution for Reward 1, between episodes 0 and 99, inclusively.	98
8.4	Pick - Case distribution for Reward 1, between episodes 100 and 299, inclusively.	99
8.5	Pick - Case distribution for Reward 1, between episodes 300 and 599, inclusively.	100
8.6	Pick - Case distribution for Reward 1, between episodes 600 and 808, inclusively.	100
8.7	Proposed parameters of Eq. 8.1 for Reward Proposal 2	101
8.8	Pick Reward 2 - Observed situations and respective cumulative reward range.	101
8.9	Pick - Case distribution for Reward 2, between episodes 0 and 67, inclusively.	103
8.10	Pick - Case distribution for Reward 2, between episodes 68 and 173, inclusively.	103
8.11	Proposed parameters of Eq. 8.1 for Reward Proposal 3	104
8.12	Pick Reward 3 - Observed situations and respective cumulative reward range	104
8.13	Pick - Case distribution for Reward 3, between episodes 0 and 199, inclusively.	106
8.14	Pick - Case distribution for Reward 3, between episodes 200 and 299, inclusively.	106
8.15	Pick - Case distribution for Reward 3, between episodes 300 and 399, inclusively.	107
8.16	Pick - Case distribution for Reward 3, between episodes 400 and 756, inclusively.	108
8.17	Proposed parameters of Eq. 8.1 for Reward Proposal 4	108
8.18	Pick Reward 4 - Observed situations and respective cumulative reward range	109
8.19	Pick - Case distribution for Reward 4, between episodes 0 and 365, inclusively.	109
8.20	Proposed parameters of Eq. 8.1 for Reward Proposal 5	111
8.21	Pick Reward 5 - Observed situations and respective cumulative reward range	112
8.22	Pick - Case distribution for Reward 5, between episodes 0 and 399, inclusively.	113
8.23	Pick - Case distribution for Reward 5, between episodes 400 and 801, inclusively.	113
8.24	Proposed parameters of Eq. 8.1 for Reward Proposal 6	114
8.25	Pick Reward 6 - Observed situations and respective cumulative reward range	114
8.26	Pick - Case distribution for Reward 6, between episodes 0 and 449, inclusively.	115
8.27	Pick - Case distribution for Reward 6, between episodes 450 and 1000, inclusively.	116
8.28	Pick - Case distribution for Reward 6, between episodes 1000 and 1540, inclusively.	117
8.29	Proposed parameters of Eq. 8.1 for Reward Proposal 7	117
8.30	Pick Reward 7 - Observed situations and respective cumulative reward range	119
8.31	Pick - Case distribution for Reward 7, between episodes 0 and 399, inclusively.	120
8.32	Pick - Case distribution for Reward 7, between episodes 400 and 699, inclusively.	120
8.33	Pick - Case distribution for Reward 7, between episodes 700 and 810, inclusively.	121

8.34	Pick Reward Functions - Success rate comparison	123
8.35	Observed situations summary.	125
9.1	Parameters' combinations tested for the moving task.	131
10.1	Parameters' combinations tested for the fitting task.	147
10.2	Sub-goal reward increments on the four sub-tasks	149
A.1	Machine's Specifications	157
A.2	Training Parameters	158

Abbreviations and Symbols

ACNN	Associative Chaotic Neural Network
AI	Artificial Intelligence
AL	Apprenticeship Learning
BC	Behavioral cloning
CBN-IRL	Constraint-based Bayesian Nonparametric Inverse Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
EDDS	Discrete Event Dynamic System
FEM	Feature Expectation Matching
FIRL	Feature Construction for Inverse Reinforcement Learning
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Networks
GASIL	Generative Adversarial Self-Imitation Learning
GPU	Graphics Processing Unit
GTAL	Game-theoretic Apprenticeship Learning
HER	Hindsight Experience Replay
HGAIL	Hindsight Generative Adversarial Imitation Learning
HIRL	Hierarchical Inverse Reinforcement Learning
HRL	Hierarchical Reinforcement Learning
HMM	Hidden Markov Model
IL	Imitation Learning
IOC	Inverse Optimal Control
IRL	Inverse Reinforcement Learning
MDP	Markov Decision Process
ML	Machine Learning
LfD	Learning from Demonstration
LfF	Learning from Failure
LTM	Long-term Memory
PID	Proportional Integral Derivative
PPO	Proximal Policy Optimization
ROS	Robot Operating System

RL	Reinforcement Learning
SAC	Soft Actor Critic
STM	Short-term Memory
TRPO	Trust Region Policy Optimization

Chapter 1

Robots and Industry: challenges and opportunities

Industry and robots are deeply intertwined [1]. While Industry's evolution demanded robot's new capabilities, developments in robotics allowed for better performance when it comes to productivity and safety in Industry [2, 3]. More than ever, with Industry 4.0, there are new and diverse demands, which require new tools, such as Artificial Intelligence (AI) or Cloud Robotics [4]. Therefore, there is a two-way development process: societal changes request innovation, which causes society to evolve. Thus, robots must improve to keep up with the challenges, but the good news is that there are more resources and opportunities than ever [5].

1.1 Context

The way robots evolved in terms of roles, behaviour, and programming methods is strongly related to how Industry has changed these past years [6]. By only understanding the Manufacturing Industry's challenging needs, one can improve robots' performance, productivity and safety.

At first, in the mid-twentieth century, industrial robots were introduced to perform repetitive tasks, usually involving heavy loads, substituting humans in intense and harsh labour, starting at the so-called Industry 1.0 [7]. This resulted in a general increase in productivity and quality as a consequence of the repeatability robots allowed. Even though the usage of the robot induced increased efficiency in mass production, they had some limitations [8]. They performed well-modelled tasks, were placed and used in a fixed position, did not react to any adversity or perturbation, and repeated the same sequence of actions regardless of the surroundings or human presence, which might endanger human lives. Many safety measures had to be implemented, such as building safety fences and full emergency stops when an operator is nearby [9].

In recent years, Industry demanded more resilience, greater flexibility, and adaptability towards product customisation to react to consumer habits and quick changes in societal trends - the desired mass customisation [10]. The products need to be constantly updated, reinvented and must meet people's needs and tastes. With the traditional fully automated solutions, these production

requirements could not be met, and robots started to be seen as an extension of human capabilities. So, the paradigm in robotics changed. Robotics' systems are now easier programmed and take into account information regarding the surroundings. On the one hand, new programming languages and development techniques were merged into the systems. On the other hand, improvements in sensitisation help with decision-making, environment awareness, information integration and systems' collaboration [1]. These are the challenges faced by contemporary manufacture, Industry 4.0, which brings physical and software together, creating the cyber-physical systems where data flows are crucial to performance [11].

1.2 Problem Definition

Current industrial context demands fast and robust **product customisation**. This variability, as well as the environmental uncertainties, are not compatible with the traditional programming methods. The common process demands explicitly programming the robot to perform each manufacturing task from scratch with traditional approaches. Every time a new task emerges, or a change in the process's flow (the sequence of operations) or its parameters (like objects' positions) occurs, there is a need for laborious and cumbersome changes in the robot's program.

In recent years, more recent and intelligent solutions have emerged within the paradigm of Machine Learning (ML) methods. These newer solutions allow for some flexibility, but mostly related to changes in object's positions and other similar parameters, generalising the base operations to other positions by training the created model in the specific process environment. However, the concern with model **transversality and robustness** has not been thoroughly explored regarding the intelligent and flexible solutions available. Firstly, the initial design of the solution with the applied methods is complex. Secondly, if the manufacturing process suffers a modification in the sequence of operations for the robot, the model usually needs to be retrained, where its convergence is typically slow. Even though some or most of the robot's operations were already programmed (cases when the operations are the same and only their order change), these are not reused, and the full behaviour is retrained, involving more time-consuming steps.

Furthermore, all the limitations explained above usually lead to high costs associated with change-overs and product adaptations. They also require specialised programming resources for reprogramming purposes. Given this, there is a clear need to **merge** intelligent approaches with methods that facilitate customisation and flexibility, either in similar tasks or in others prone to **discretisation** into similar operations.

Overall, manufacturing systems **suffer from complexity and lack of transversality**. The complexity is preponderant when the tasks require long sequences of actions. The lack of adaptability is clear-cut when similar tasks must be reprogrammed entirely, not using the potential of available models.

1.3 Motivation

The majority of manufacturing products can be understood, from the assembly point of view, as a combination of **discrete hierarchical tasks** instead of a massive sequence of individual actions [12]. As in Fig. 1.1, a product like a car engine can be assembled in a set of small tasks, each including smaller operations. Both operations and tasks, resembling the human perspective, might be transversely applied to other products or tasks, respectively. For example, moving a manipulator between two points is an operation employed in stacking and attaching tasks. Likewise, those tasks can be required for different assembly products: many mechanical parts need screwing.

The intricate sequence of movements and actions required for the product to be ready comes with significant uncertainties like the ever-coming variants of a product family or the parts' positioning. These main assembly tasks' characteristics require ease in reconfiguration [13]. The need for a robot's hierarchical understanding promotes a shift from a complex sequence of movements to a set of layers of simpler reusable operations. This change in paradigm is helpful in tackle easier problems and enable the production of derived/family products, promoting flexibility in the productive system and speeding up the programming required to change the product assigned. It prevents the operator from explicitly hard-coding the robot's movements or using a single-product oriented coding [14]. Ideally, the hierarchical approach allows the robot to reuse blocks of knowledge at different times.



Figure 1.1: Hierarchical relation between operations, tasks and products

AI researchers have been working on applying ML methods like Reinforcement Learning (RL) to manufacturing tasks, where **Learning from Demonstration (LfD)** or Imitation Learning (IL) have proved to be promising fields. This field accelerates the learning process, which is particularly important given the challenges imposed by Industry 4.0, as mentioned above [15]. The rising interest in IL follows the programming demands and resembles the way humans teach each other: it is a natural and easier way of teaching. The demonstration also avoids the need for hard-coding most of the possible situations the robot must be prepared to be robust [16].

Hierarchical Reinforcement Learning (HRL) is a new research area that results from several efforts to address RL's limitations in more extensive or complex tasks. By combining LfD methods with a hierarchical approach, one could join the generalisation capabilities, flexibility, and speed of each method, to better align with industry demands while propelling relevant advancements in applying AI methods in industrial robotics.

1.4 Objectives

The objectives set for this thesis were shaped by **discretisation**, **hierarchisation**, and **adaptability**. Given the identified problem and motivation, we have defined the following objectives:

- Identify the common intelligent methodologies which are appropriate to robotic tasks, where the desired behaviour is known;
- Compile key concepts regarding those approaches, identifying relevant examples and similar solutions;
- Compare the performance between traditional and hierarchical **LfD**;
- Define an appropriate case study to test the hierarchisation and discretisation hypothesis, under the **LfD** scope;
- Compare available simulation tools to implement and validate the case study;
- Organise a set of guidelines to divide a task into smaller sub-goals;
- Generate a dataset of expert demonstration for the task;
- Develop a system that applies **LfD** and hierarchical techniques to make the robot:
 - learning the correct behaviour, based on the recorded demonstrations.
 - adapt to task's variability (different sub-tasks or product specifications);
 - adapt to environment uncertainties (sensor errors or irreproducibility);
 - optimise the task learned, solving it better than the demonstrator (time or precision).

In the end, the robot should generalise the given knowledge and autonomously reproduce the task demonstrated. Ideally, simulation implementation will not be the final goal and should be as flexible as possible for future integration into real environments.

1.5 Keywords and Research Questions

With the **current demands** for flexible and robust programming techniques regarding robots in manufacturing, and the given the **objectives** defined above, the main **research keywords** were **robot demonstration**, **imitation learning**, **robot programming** and **hierarchical inverse reinforcement learning**. The research questions derived from the keywords above, which are the main focus of this project are:

- Is it possible to **program** a robot, within a manufacturing environment, to perform its tasks using **demonstrations** and Inverse Reinforcement Learning (**IRL**)?
- By using a robotic **simulator**, is it possible to provide detailed demonstrations of the optimal behaviour? If yes, can it be done in a simple and flexible manner?

- How can one make an intelligent agent understand **task discretisation**?
- Is it possible to make the **agent reuse** previously learnt concepts?

1.6 Document Overview

The document is divided into three main parts. The first presents an **analysis** of the available concepts and projects related to the scope of this thesis, including Chapters 2 to 4. The second part includes the **methodology**, namely the proposed solution, the low and high-level implementations, in Chapters 5 and 6. Then, Chapters 7 to 10 correspond to **tests, validation and conclusions**.

Regarding **context analysis**, Chapter 2 introduces necessary concepts to understand the approaches, such as LfD, RL and hierarchical frameworks. Following the literature review scheme, Chapter 3 focus on state-of-the-art proposals. It also presents examples of algorithm chosen for this thesis and refers to evaluation metrics commonly used in these situations. Still, in the analysis phase, Chapter 4 studies and compares some of the available simulators in robotics.

Following the introductory phase, Chapter 5 defines the case study chosen and proposed solution. This chapter also explains all the modifications required to address the problem in terms of low level simulation, and a validation test to ensure all works correctly. Completing the low-level implementations, Chapter 6 describes the high-level simulator and all constituting elements.

The **tests performed** and corresponding results are presented and analysed on Chapters 7, 8, 9 and 10. The information is divided into sub-tasks. Each of them includes a description of the specific experiment, relevant considerations and demonstrative episodes. An analysis of reward function variants is also presented, with outcomes in terms of learning metrics or observed situations. These chapters end with considerations regarding the sub-goal versus the traditional approach and how reusing previous models can impact training performance.

Despite all chapters including a synopsis that sums up the important considerations, the document ends with Chapter 11. This last part gathers the final reflections on the problem defined in Chapter 5 and proposes some avenues for future work.

Chapter 2

Background

Now, a quick review of the main concepts related to this thesis' scope is presented: learn a task by demonstration, the [RL](#) methodology and [IRL](#). [LfD](#) is a valuable concept for transforming automation systems into intelligent systems, which resorts to many different tools like supervised learning or [RL](#). By chronological order, [RL](#) was the first to be formally defined. So, we start by presenting the concepts necessary to the solution. Then, we continue to a more general scope and explain how robotics can profit from demonstrations, introduce the challenges faced, and give examples of some methods used. The [IRL](#), under the hood of [LfD](#), also appears to solve some of [RL](#)'s difficulties.

After, we give examples of hierarchical frameworks and how intelligence methods could be improved with them. Finally, the chapter explains how Generative Adversarial Networks ([GAN](#))s work as they are used in [LfD](#) algorithms, like the one we've chosen to use. The main aspects here enumerated aim to provide a solid understanding of the problem tackled.

2.1 Reinforcement Learning

According to *Sutton and Barto* [17], [RL](#) is an [ML](#) field that explores how to map states/situations to actions while maximising the overall reward. It is then a closed-loop problem because the chosen action directly influences the following environment's states. [RL](#) differs from supervised learning in the sense that there are no correct recipes for every possible interaction. Nor is [RL](#) within the unsupervised learning field because it finds the best reward instead of hidden data associations. It is then the third field in the [ML](#) world (Fig. 2.1).

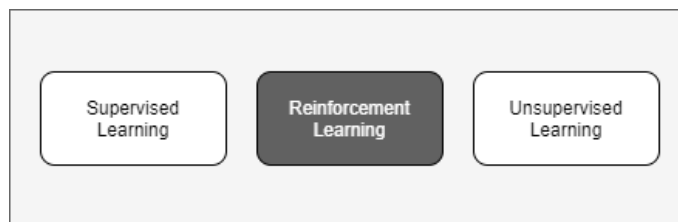


Figure 2.1: Reinforcement Learning as Machine Learning sub-field

The **RL** learning process is all about finding the balance between **exploration** and **exploitation**. The system should replicate actions that led to higher rewards (exploitation, greedy actions) and discover new actions that will potentially achieve better rewards (exploration, nongreedy actions). Moreover, **RL** considers that many uncertainties will arise. The agent must deal with them when trying to fulfil the defined goal (by successive cause and effect) while constantly interacting with the environment.

2.1.1 Problem Elements

RL problems usually include an agent, an environment, a policy, the reward signal, and a value function. The **agent** is the actor that makes decisions, and the **environment** is what surrounds the agent and reacts to its actions. The **policy** allows the choice of actions for a given environment's state, ranges from simple associations to complex functions. The policy is only updated after many experiments based on the outcome and regardless of the game evolution. The entire behaviour is valued if the goal was reached or penalised the other way around.

The **reward signal** is the feedback given by the environment about the action taken. It provides the agent with the perception of whether or not the action contributed to the goal. The final objective is to maximise the cumulative rewards by combining sequences of actions. On the other hand, the **value function** gives insights into the possible rewards the system can accumulate from the state onward. At last, the **model** can also be part of the **RL** elements. It is the knowledge that allows guessing the environment's reaction.

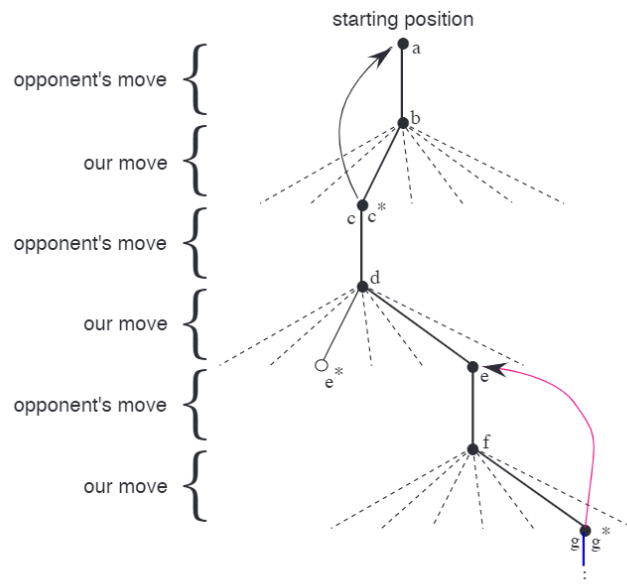


Figure 2.2: Sequence of agent's moves and opponent's move, where e was an exploratory move in Tic Tac Toe. Source: [17]

Given the example of a tic-tac-toe game of Fig. 2.2, each move made (solid lines) is chosen from a pool of possible actions (dashed lines). When exploiting the acquired knowledge, the agent

chooses the action with a higher value (c^*). If an exploratory approach is taken, a different action is chosen (e), and it causes a backup (curved line).

Agent-Environment Interaction

The agent interacts with the environment in a closed-loop, as according to Fig. 2.3, a discrete system with time instants $t = 0, 1, 2, \dots \in \mathcal{N}$. The agent reads the current state $S_t \in \mathcal{S}$ ($\mathcal{S}$ is the set of possible states) and received reward $R_t \in \mathcal{R}$, and selects the action $A_t \in \mathcal{A}(S_t)$ (set of actions available at the current state). Later, at $t + 1$, the environment reacts and outputs the next state S_{t+1} and the numerical reward R_{t+1} . The rewards give information on what is the ultimate agent goal and not how to achieve it.

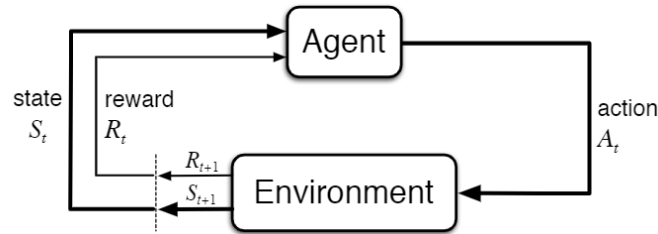


Figure 2.3: Closed-loop agent-environment and relation to actions, rewards and states. Source: [17]

The policy at the instant t (π_t) maps the environment's state into an given action ($\pi_t(a|s)$), which gives the probability of the next action to be $A_t = a$ for a given state $S_t = s$. Reinforcement learning algorithms define how to change the policy (π_t) to maximise the cumulative rewards.

The abstraction specified with Fig. 2.3 stresses the RL nature: the problem can be approached with the closed-loop model abstraction, regardless of its details, such as sensors, control or specific objectives.

2.1.2 Markov Property and Markov Decision Process

RL problems are usually framed under the Markov Property and as a Markov Decision Process (MDP). The Markov Property models a signal in which the next state ($t + 1$) only depends on the current one and the action taken at the moment (t). So, according to *Sutton and Barto* [17], and under a discrete approach for simplicity purposes, the probability distribution for:

- the current state $S_t = s \in \mathcal{S}$
- taking the action $A_t = a \in \mathcal{A}(S_t)$
- reaching the next state $S_{t+1} = s' \in \mathcal{S}$
- with the respective reward $R_{t+1} = r \in \mathcal{R}$

is given interactively by Equation 2.1, the Markov Property:

$$\begin{aligned} p(s', r|s, a) &= Pr(R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_t, S_t, A_t) \\ &= Pr(R_{t+1} = r, S_{t+1} = s' | S_t = s, A_t = a) \end{aligned} \quad (2.1)$$

The Markov Property is one of the **RL** main pillars, which simplifies the approach but requiring the states to be informative. If this property is satisfied, the task is an **MDP**. The **MDP** can be defined by its state and action sets and the environment's step dynamics (Eq 2.1).

From the probability in Eq.2.1, it is possible to compute:

- the **expected rewards** for a given state-action pair:

$$r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a) \quad (2.2)$$

- the probability of a **state transition**:

$$p(s'|s, a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a) \quad (2.3)$$

- the **expected reward** for the sequence $s - a - s'$ (triple):

$$r(s, a, s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r p(s', r|s, a)}{p(s'|s, a)} \quad (2.4)$$

Additionally, the **MDP** is summarised into a *transition graph*, in which the states, actions and transition probabilities are depicted, as exemplified in Fig 2.4.

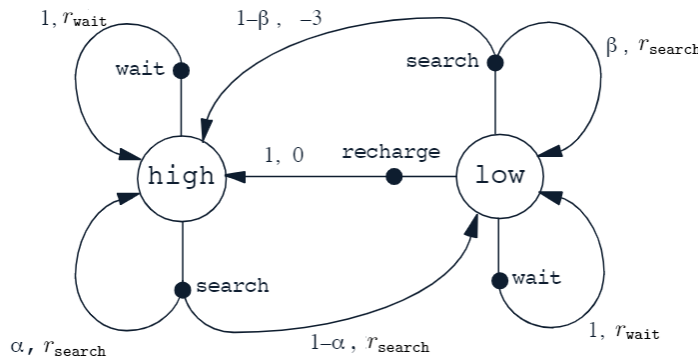


Figure 2.4: Example of a transition graph for an **MDP** problem. The big white circles represent states; the small black circles represent the actions, and the arrows are the transitions' triples with a given probability. Source [17]

An **MDP** can be defined through the rewards and the model. The algorithms that solve **RL** problems can be either model-based or model-free.

2.1.3 Value Functions

While reward functions are a response from the environment, the agent estimates how good a given state is through value functions. The evaluation considers the expected future rewards or the expected return. So, the *value* of a state s for a policy π comprises the returns R_t . This quantity is called the *state-value function for policy π* and is computed according to:

$$v_{\pi}(s) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \quad (2.5)$$

Where γ is the *discount rate*, $0 \leq \gamma \leq 1$, which determines the weight of future rewards. The closer to the current instant t , the more important is the reward. At the instant $t+k$, the reward is weighted by γ^{k-1} .

The expected return by taking the action a at the state s and under the policy π is called *action-value function*:

$$q_{\pi} = (s, a) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \quad (2.6)$$

Both state-value function (Eq. 2.5) and action-value function (Eq. 2.6) are estimated through experience as the number of times the state appears approaches infinity. These methods of averaging random samples are called *Monte Carlo methods*.

Another solution, alternatively to Monte Carlo, is to keep a single record of v_{π} and q_{π} as parameterised functions that satisfy recursive relationships. The *Bellman Equation* (Eq. 2.7) for v_{π} stresses the relationship between the state value and the values of successive states. Overall, the value of a state (s) is the sum of the successive discounted values of the following states.

$$\begin{aligned} v_{\pi}(s) &= \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi}(s')] \end{aligned} \quad (2.7)$$

This translates the backup operations yielded from the Bellman Equation (Eq. 2.7): the current state and rewards give information about the previous sequence of states. The behaviour is depicted in Fig. 2.5.

The solution of the Bellman Equation (Eq. 2.7) is unique and equal to v_{π} .

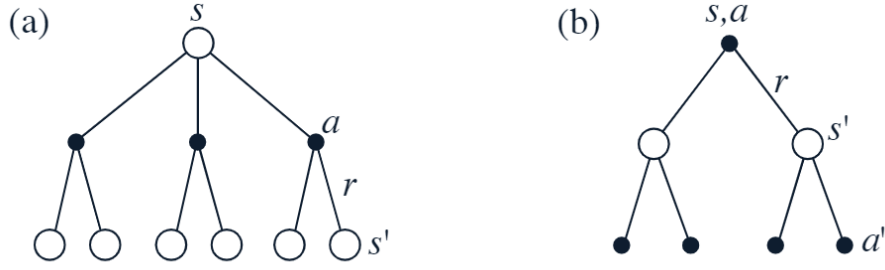


Figure 2.5: Backup diagrams for a) v_π and b) q_π . Source: [17]

2.1.4 Optimal Value Functions

The *optimal policy*, π_* , is the best policy for all states s . The corresponding *optimal state-value function* is:

$$v_*(s) = \max_\pi v_\pi(s) \quad (2.8)$$

Similarly, the *optimal action-value function*, q_* :

$$q_*(s, a) = \max_\pi q_\pi(s, a) \quad (2.9)$$

The Eq. 2.8 can be re-written according to the self-consistency condition by Bellman equation, as the *Bellman optimality equation*:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E} \left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a \right] \\ &= \max_{a \in \mathbf{A}(s)} \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_*(s') \right] \end{aligned} \quad (2.10)$$

The optimal policy will be the one that only assigns non-zero probabilities to the actions that reach the maximum value in the Bellman optimality equation. In other words, if the policy is *greedy* regarding the optimal evaluation function v_* (Eq. 2.10), it is the optimal policy.

However, solving the Bellman optimality equation (Eq. 2.10) is not the go-to approach most of the time. It is an exhaustive search that relies on three principles:

1. the environment's dynamics are thoroughly known;
2. there are enough computational resources to explore all solutions
3. the Markov property is respected.

For the situations where one or more principles do not apply (commonly), the **RL** consists of computing approximate solutions. These alternative approaches might be useful to focus on the frequently encountered states.

2.1.5 Summary

The **RL** concepts to keep hereafter are:

- **RL** is a closed-loop problem because the actions taken based on the current state will influence the next state;
- Among supervised learning and unsupervised learning, the **RL** is a different paradigm within the **ML** field;
- **RL** alternates between exploration and exploitation to find the best sequence of actions;
- The **agent** and the **environment** continuously interact with each other;
- The choices the agent makes are called the **actions**;
- The **state** translates how is the environment and influences the agent's choices, and the **rewards** are the actions' evaluation;
- The **policy** is the set of rules that the agent follow to select the actions;
- The **return function** is the prediction of future rewards the agent aims to maximise;
- The **discount factor** (γ) weights the next rewards over the distant ones;
- If the environment's current state gives enough information about the previous states (informative states) and allows the agent to predict the future, the system lies under the **Markov Property**, and it is called an **MDP**;
- **Value functions** map a state or action-state pair into the corresponding expected return;
- If a given value function associates the maximum return for a state/state-action pair, it is called the **optimal value function**.
- The **Bellman optimality equation** (Eq. 2.10) is the condition that a policy must satisfy to be the optimal policy;
- In some cases, it is impossible to calculate all possible situations accurately, so approximate methods are used instead.

2.2 Learning from Demonstration

According to *Nehaniv et al.* [18], **LfD** or **IL** expresses the interaction between two agents resulting in observational learning, just as animals do. Even though the imitation process can occur in terms of action-, program- or effect-level, the output can be significantly different and impact the task's success. Regarding action-level imitation, the teacher's movements are thoroughly copied by the imitator. The program-level imitation is based on a sequence of steps, usually involving calling hierarchical routines. However, the rule (rather than the exception) is that the teacher and the apprentice have different embodiment. Therefore, the exact copy of the movements is quite hard to do and can sometimes become impossible.

Consequently, the mapping between the two agents can also be unfeasible. The exact translation differs significantly when feasible, with different environmental consequences (for example, translating human arm motion into an industrial robotic manipulator). So, the effect-level imitation consists of a "functional imitation" of the effects caused in the environment. In other words, a successful imitation reaches the same purpose: both hummingbirds and helicopters fly. Moreover, jumping into an abstraction layer, one imitation is considered successful if the action goal is identified and correctly attained, without further effects on the surroundings, which leads to the next important factor: perception.

Different environment perceptions lead to distinct success evaluations and uncertainties on desired goals. By formally defining the sequences of goals, the imitation's assessment and quantitative comparison are possible by computing error metrics regarding states achieved and sub-goals completed. The generalisation comes with the correspondence model between the actors and the minimisation of unsuccessful evaluated attempts. The **LfD** approach generally lies within the Supervised Learning field because the learning agent has access to a labelled training dataset: the desired behaviour is known from the beginning. However, other approaches are using **RL**, where the agent needs to explore the environment by trial and error. To define the solution scope and unify the diverse developments in **LfD**, the dominant questions, according to *Billard et al.* [19], are: what to imitate, how to imitate, when to imitate and who to imitate.

2.2.1 Learning from Demonstration in Robotics

LfD in robotics and its vast spectrum is the focus of *Argall et al.* [20]. At the root of the problem, robots are expected to learn the correspondence between the world's state and the next action to take. This mapping decision is based on a **policy** and can be developed by a field in **ML**: robotic **LfD**. That policy is learnt by the robotic systems based on the teachers' examples, which demonstrate the desired robot behaviour, i.e., the sequence of state-action pairs. The survey explains the advantages of an **LfD** approach: no need for a thorough understanding of the environment's dynamics, simpler approach to building policies, communication is more natural for humans, and the learning process is more directed to the shown behaviour.

Framing an LfD problem

Although LfD spreads across a wide variety of problems, they all have common aspects. Let's call the set of environment's states S and the actions A . The actions in A can represent both motion movements (low-level) or logical decisions (high-level). The states' sequence is triggered by taking some action, whose interaction is associated with some probability, in which the transition (T) is:

$$T(s'|s, a) : S * A * S \rightarrow [0, 1] \quad (2.11)$$

Another assumption is that not all state is fully observable, and the subspace that the observer can access is called Z . For observability purposes, sensors should be part of the robotic system and provided the appropriate perception. Likewise, there is a possible mapping between the observable and non-observable states:

$$M : S \rightarrow Z \quad (2.12)$$

It is then possible to define the policy (π) that chooses the action based on the observable state at that time:

$$\pi : Z \rightarrow A \quad (2.13)$$

The demonstration (D) triggers the policy derivation. Later, the system interacts with the world, as clarified in Fig. 2.6. This is an example where LfD approaches the issue in an RL style.

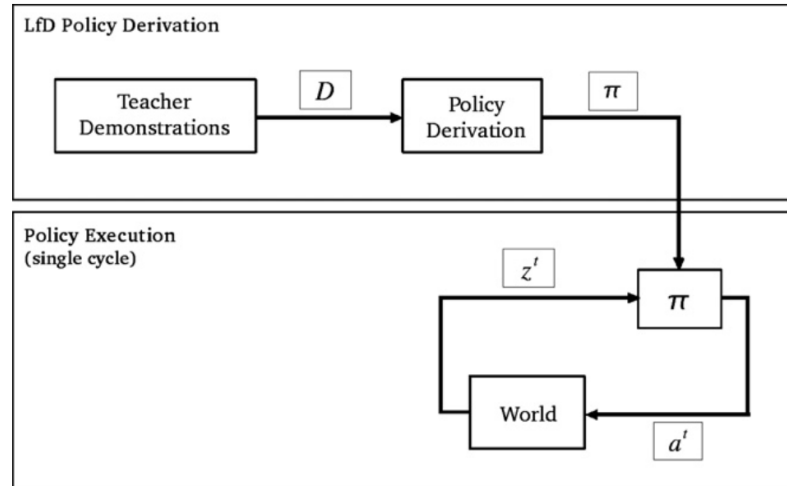


Figure 2.6: Control policy derivation and Execution, source:[20]. In LfD, the teacher demonstrates the desired behaviour (D) so that the agent can derive the policy (π). This is a step prior to execution, which occurs only once. The policy execution is then a continuous loop, where both the outside world and the agent interact. On the one hand, the agent chooses the next action (a') based on the policy (π) and the observable state (z'). The environment reacts to every action (a), displayed by the new state, z' .

Designing an LfD System

Following the *Argall et al.*[20] survey information, the design of a LfD System deals with the choice of **demonstration** (i), **state-space** (ii) and **policy** (iii).

Considering problem (i), the demonstration design comprises the **demonstration's source** (i.a) and the **demonstration strategy** (i.b).

- For the demonstrator problem (i.a), there are two agents: the one that controls the demonstration and the other that performs the demonstration *per se*. For instance, the same human agent controlling the demonstration can execute it employing its own body (as when he sorts a set of materials) or teleoperate a robotic arm (for welding purposes);
- Regarding the strategy employed in the learning process (i.b), there are two main groups: **batch learning** and **interactive learning**. These two approaches differ on the training moment: the first learn the policy after all data has been collected, and the latter does so every interaction.

The state-space issue (ii) distinguishes between a **discrete representation + a set of boolean features** or a **continuous representation + state representation**.

When computing the policy (iii), not only it has to be designed (iii.a) but also should be built on afterwards (iii.b). In terms of policy derivation (iii.a), there have been used **mapping functions** (the policy tries to map directly the observations into actions), **system models** (world dynamics, $T(s'|s, a)$, and rewards $R(s)$ are considered) or **plans** (which learns the pre- and post-conditions associated to an action, $L(preC, postC|a)$, creating a recipe based on the user intentions). Fig. 2.7 sums up these three methods.

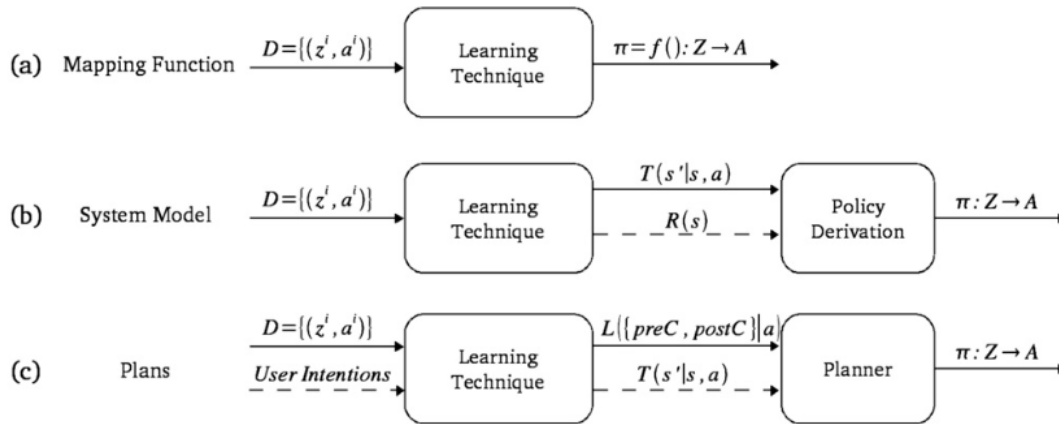


Figure 2.7: Policy derivation methods: mapping function, system model and plan. Source: [20]

Afterwards, there is a need to choose the learning technique, which deeply relies on the problem's nature. For the mapping function issues, **classification** or **regression** can be chosen. When employing systems models, the choice falls upon **engineered rewards** or **learned rewards**. Fig.2.8 synthesises these options.

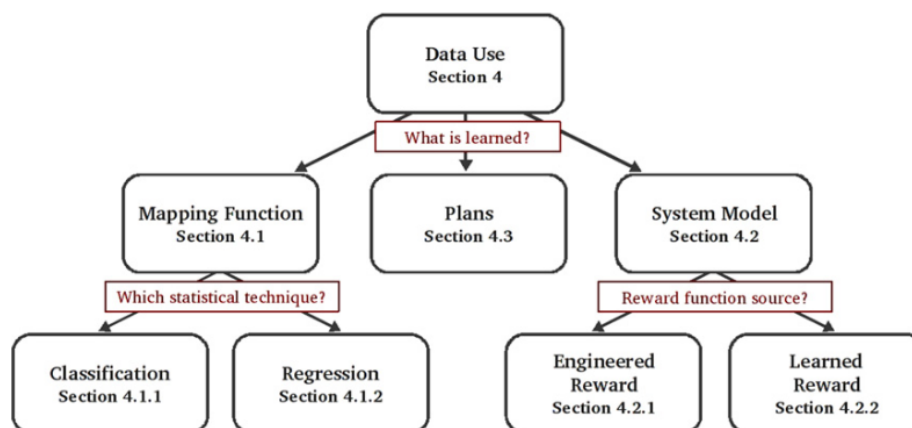


Figure 2.8: Policy derivation and correspondent approaches for each one. Source: [20]

The **mapping function method**, which calculates a function that approximates the system state to the action demonstrated, tries to find the teacher’s policy and generalise it towards other situations. The method’s outputs are the robot’s actions, and the inputs are the robot’s perceptible states. The classification approach is used for discrete outputs, whereas the regression calculates continuous actions. The classification derivation pairs up similar states (inputs) and controls basic motion, action primitives, or complex behaviours. For basic motion (low-level control), there have been used Gaussian Mixture Models [21], Decision trees [22], Bayesian networks classifier [23] and k-Nearest Neighbours classifier [24].

The motion primitives approach is used in a sequence of state mappings, and the proposed approaches include the use of the K-Nearest Neighbours Classifier and Hidden Markov Model (HMM). High-level behaviours can also be controlled with mapping methods and similar approaches, like HMM and Support Vector Machines.

The regression approach is the continuous dual of the classification in policy derivation. It is commonly applied to low-level control problems rather than to high-level ones. The techniques used include Lazy Learning, Locally Weighted Regression, Receptive Field Weighted Regression, Locally Weighted Projection Regression, Neural Networks, Gaussian Mixture Regression and Sparse Online Gaussian Processes.

A different approach to determine the policy is **System Models**, as previously explained in Fig. 2.7. It considers the state transition between states $T(s'|s, a)$ and the policy is derived as $\pi : Z \rightarrow A$. The problem framing is usually under the RL methodology, which determines the transition function according to the robotic system’s demonstration and additional exploration. Another important piece to the RL solution is the reward function, either manually defined or learnt through the demonstration. The source of the reward function is the factor that differs between **engineered reward functions** and **learned reward functions**.

In the **engineered reward** functions, the reward function is manually defined by the user. It sometimes helps the robot to find the rewards, avoiding long periods of initial random exploration. However, the robot’s performance is heavily dependent on the rewards and the demonstrations

provided. In some cases, robot agents exchange information and can increase the learning speed. These are the situations where **LfD** specialises in **RL**.

If instead of manually defining the reward function, the robot can **learn it**, it can be advantageous when the system is too complex to model. These issues are generally addressed using **IRL**. Normally, the greater the similarities between demonstration and proposed behaviour, the greater is the assigned reward. Furthermore, the robot can receive some guidance, post demonstration, to improve its performance. Hence, if the system correctly learns the reward function, the correct behaviour is also discovered. The implementations can derive only the reward function - $R(s)$ - or also the transition function - $T(s'|s, a)$.

Despite the diversity in policy derivation methods, there are some obstacles to the robot's performance. The demonstration datasets might reflect the inaccuracies or limitations inherent to the human teacher, like movement speed or sub-optimal behaviours. Other limitations include non demonstrated states or poor quality data. This has room for improvement, and many researchers have been focused on solving this issue, like incorporating other corrective sources.

Sources of Demonstration

The first obstacle when creating demonstrations is directly connected to the correspondence problem. It translates into the difficulties of mapping the teacher execution to the recorded demonstrations (a record). Still, it is also present in the embodiment process when the demonstration is mapped to the learner. These challenges are more prominent when the teacher is human, and a real robot is a learner. Fig. 2.9 clarifies the sequence of transformations since execution to the learner.

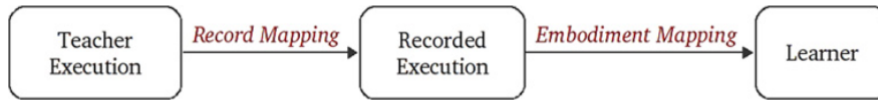


Figure 2.9: Correspondence Issue: record and embodiment mapping. Source: [20]

It is pointed out by *Argall et al.*[20] that in some cases, the embodiment problem might not occur like when the demonstration is executed directly in the robot. The author calls these cases as **demonstrations**. When all mappings are present, it is called **imitation**.

In a demonstration, the robot senses the states by its sensors or infers from the data. The demonstration subdivides into **teleoperation** or **shadowing**. Teleoperation means the robot is directly controlled by the human agent (ex. joystick, kinesthetics, speech), whether in shadowing the robot senses and tries to emulate the human agent executing the task. Although there is some simplification in the embodiment mapping, the technique has several limitations: teleoperation can only be used in robots that support the feature or systems with bearable low-level control. Some teleoperation examples are [25] and [26], and shadowing are [27] and [28].

The imitation approach can be recorded by adding **sensors to the teacher's body**, which might require specialised sensors and imply overhead, or by **externally record the execution**, which can be more general but less accurate. The examples include [29], [30] and [31].

2.2.2 Challenges and Novelties

In simple words, *Chernova et al.* [16] explains that **LfD**, applied to Robotics, aims to simplify robots' programming. Without explicitly hard-coding all possible (even unlikely) situations, the knowledge is transferred from the teacher to the robotic system. **LfD** solves the low-level control (i.e. motion control) required to tackle a particular task but has still room for improvement:

- *Grollman et al.* [32] describe the importance of unsuccessful demonstrations not to be used as an example but to explore different successful approaches from the beginning, called Learning from Failure (**LfF**). It allows the system to deal with corrective issues or new situations otherwise perfect demonstrations do not contribute to;
- *Dong et al.* [33] point out the need to recognise real-time human movements after learning (offline) the provided demonstrations and extracting the associated relevant features, even those not explicitly pointed out. This Probabilistic Flow Tubes approach allows the robot to continue tasks if need be;
- *Mohammad et al.* [34] explore the possibility of an imitator to decide whether or not human behaviour should be learnt and find its boundaries of interest for building a demonstration;

Moreover, *Chernova et al.*[16] point out that **LfD** (still within the low-level field) can be used to correct mechanisms rather than simply add new demonstrations. It is the case of [35], which proposes a framework to correct online policies according to the human teacher's execution experience and then overcome the difficulties of designing the best-suited policy. The editorial also speaks of the high-level control provided by **LfD**. *Jakel et al.*[36] explain the necessity of demonstrations when programming a dexterous manipulator as the demonstration based programming with contact constraints. This situation embraces the correspondence problem, and the trajectory choice is solved due to the power of human demonstrations.

All these projects stress **the need for flexible ways of programming systems with some degree of complexity** that otherwise could not respond to the robustness required. In fact, **LfD** brings the power of demonstration to robotics' systems and shows them the desired behaviour in a more human-friendly way. However, one of the main concerns shared by all the projects is related to the **demonstration quality** and the way the system will **react when faced with new situations and many uncertainties**. When should the system consider or discard the demonstration? What is the impact of incorrect demonstrations? How to prepare the system for unexpected situations and undesired environment's behaviours? How to choose the best policy regarding the demonstrations' undemonstrated state? All these questions arise and may not be the best approach to simplify the production of a product family.

2.2.3 Summary

The key ideas to keep in mind after reading the [LfD](#) section are:

- Inspired from the natural interactions between animals, the [LfD](#) techniques consist on **observational learning** between two agents;
- Due to **different embodiment**, apprentices might not be able to reproduce the teacher's actions thoroughly;
- At the **effect-level**, a successful imitation means the agent was able to **identify and reproduce** the actions, without any further consequences on the environment;
- A **policy** is what maps environment's state to agent's actions;
- After the teacher demonstration, the agent derives the policy and then executes it, continuously **interacting** with the surroundings;
- The system design includes decisions regarding the **demonstration nature, state-space and policy derivation**;
- The system models approach takes into account the transition between states and environments' rewards. The **reward function** can be either manually defined or engineered through [IRL](#) methods;
- Demonstrations can be recorded from several points of view, and the corresponding **mapping** can translate into additional challenges;
- The greater concerns in the research among [LfD](#) include the programming flexibilisation, the ability to cope with complexity, how the agent should deal with failure and the elimination of poor demonstrations.

2.3 Inverse Reinforcement Learning

Russell [37] explains that before learning some task with **RL**, the first thing to do is to determine the reward function that the agent is trying to optimise, given:

1. the measurements of an agent's behaviour along the time, and for diverse circumstances;
2. the measurements of the agent's sensors;
3. the physical environment's model.

Russell [37] points out that **IRL** can be an effective way of learning from demonstrations (**LfD**). Especially for complex tasks, **IRL** might be desirable over manually building a reward function with intuitive knowledge. Moreover, the reward function is a more compact, robust and simpler way of learning the teacher's behaviour than direct mapping.

2.3.1 Reward Functions within IRL

Abbeel and Ng [38] view the **IRL** approach as a way to overcome the difficulty of explicitly writing the reward function from an expert's demonstration. Under the **MDP** framing, many solutions arise to find the optimal or sub-optimal policy. However, all of them consider the reward function is available and easier defined than the value function. Nonetheless, for example, there are many considerations to be aware of in autonomous driving, which does not guarantee that the agent will drive well. The authors consider that the constant *generalised policy iteration* (Fig. 2.10) is undesirable in real applications, i.e., value (v) and policy (π) functions interact until desired performance and consistency is achieved. Having to perfect the policy and reward functions manually makes the **RL** algorithms and optimal control methods unappealing for Industry.

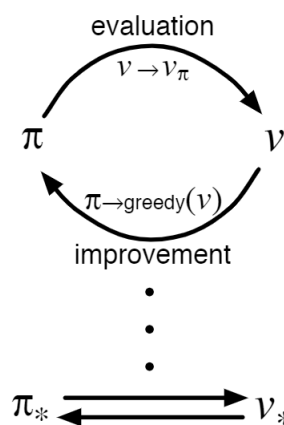


Figure 2.10: Generalised Policy Iteration: policy function and value function constantly improve each other until the consistency and desired performance has been reached. Source: [17]

For dealing with the apprenticeship learning challenges, solutions like supervised learning might not be robust enough to generalise according to the uncertainties that come along. Moreover,

Abbeel and Ng [38] state that reward functions are the pillars to RL approaches and is the "most succinct, robust and transferable definition of the task". Like that, the IRL methodologies search to optimise the reward function by creating a linear combination of the most important features, even if the expert's reward function is not known *à priori*.

For Apprenticeship Learning (AL) definition purposes, Abbeel and Ng [38] specify a MDP problem without a reward function. The MDP is in the form of a (S, A, T, γ, D) tuple where :

- S is the set of possible states;
- A is the set of actions;
- T is the set of state transitions probabilities, $p(s'|s, a)$;
- γ is the discount factor in the interval $[0, 1]$;
- D is the initial-state distribution

Assuming the later definitions, there is some features' vector $\phi : S \rightarrow [0, 1]^k$ and a true reward function $R^*(s) = w^* \cdot \phi(s)$, $w^* \in \mathbb{R}^k$, where $\|w^*\|_1 \leq 1$, for attaining rewards between 0 and 1. The real reward vector is unknown and w^* specifies then relative weights between data.

The policy (π) that maps states to probability distributions over actions is calculated as:

$$\begin{aligned} E[V^\pi(s_0)] &= E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi\right] \\ &= E\left[\sum_{t=0}^{\infty} \gamma^t w \cdot \phi(s_t) | \pi\right] \\ &= w \cdot E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi\right] \end{aligned} \quad (2.14)$$

If defining the expected discounted feature value vector, $\mu(\pi)$, as the *feature expectations* (Eq. 2.15):

$$\mu(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi\right] \in \mathbb{R}^k \quad (2.15)$$

Then the policy is the result of:

$$E[V^\pi(s_0)] = w \cdot \mu(\pi) \quad (2.16)$$

As the reward function (R) is a linear combination of the features π , the feature expectations $\mu(\pi)$ (Eq. 2.15) directly relates to the expected sum of the delayed rewards (Eq. 2.14). Thus, Eq. 2.16 stresses that the feature expectations will translate the expected sum of discounted rewards.

2.3.2 IRL and Inverse Optimal Control

Ab Azar *et al.*[39] compiled the major differences between Inverse Optimal Control (IOC) and IRL methods. IOC is a classic approach to system control that employs stability control laws, Lyapunov Function, Proportional Integral Derivative (PID) gains. The IRL is a modern IOC approach (Fig. 2.11) that uses ML contributions to determine a reward function from expert demonstrations. Regarding system behaviour, IOC is a classic way of system control in which stability is guaranteed, and the system is expected to behave optimally. The IRL methodology overcomes the lack of information on system dynamics, making it more robust, especially for uncertainly behaved environments. However, selecting an IOC IRL algorithm relies on the system and environment model, demonstration nature, control law or policy, cost/reward function and system behaviour.

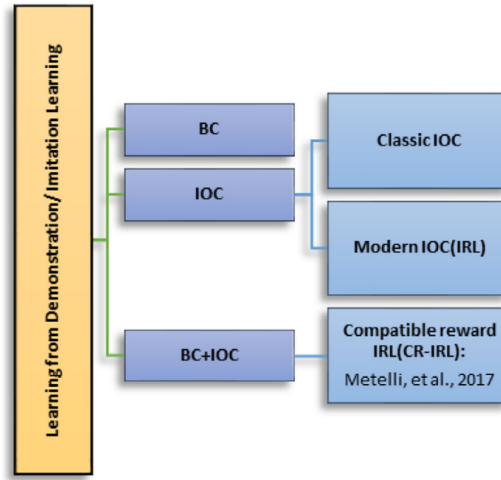


Figure 2.11: IRL as modern IOC. Source: [39]

Similarly to *Abbeel and Ng* [38], *Ab Azar et al.*[39] formulate the IRL methodology. Given:

- set of states S ;
- set of actions A ;
- a policy $\pi \in \Pi : S \rightarrow A$;
- a MDP framework;
- initial state $s(0) = s_0$;
- final state $s(t_f) = s_{t_f}$;
- the next state given by $s_{t+1} = M(S, A, T, \gamma) R$;
- the expected cumulative reward function for the expert policy:

$$V(s_t, a_t) = \max_{\pi} E\left[\sum_t \gamma^t R(s_t, a_t) | \pi\right]; \quad (2.17)$$

- the demonstrated trajectory $D = \{(s_1, a_1), \dots, (s_n, a_n)\}$;
- a weighting vector $w \in \mathbb{R}^n$;
- features and constraints vector $\phi : (S, A) \rightarrow \mathbb{R}^n$;
- reward function $R(s, a) = w^T \phi(s, a)$;

The objective is to find the reward function $R(s, a)$ that matches the expert's demonstration, given the expected cumulative reward functions:

$$R = IRL(S, A) \quad (2.18)$$

2.3.3 IRL Techniques

As depicted in Fig. 2.11, **IRL** is the modern approach to **IOC**, which tries to find the optimal policy given an expert's demonstration. The approaches to recover the reward function group as Direct, Indirect or Direct-Indirect, as in Fig. 2.12.

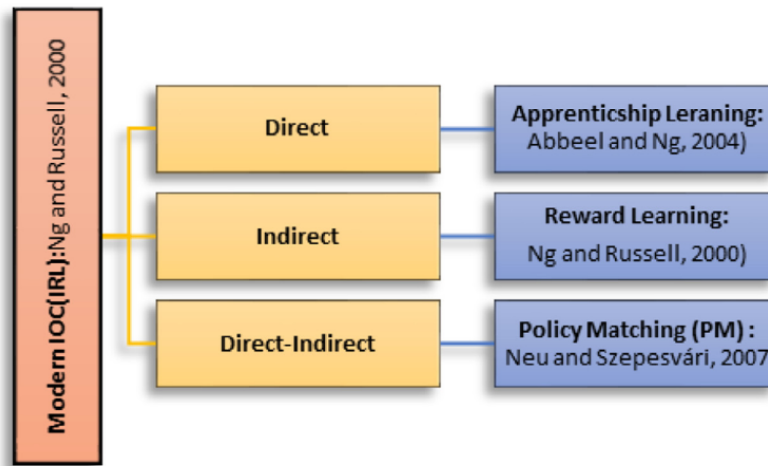


Figure 2.12: **IRL** Techniques. Source: [39]

Direct techniques try to find the optimal policy by minimising the deviation error between the expert's policy and the chosen framework of supervised learning. **Indirect** methods obtain the policy by finding a reward function. The **direct-indirect** method combines a loss function that tries to minimise the agent's deviation to expert's policy with the policy obtained from the **MDP**.

For **AL** (Direct **IRL**), finding the closest policy $\pi \in \Pi$ to the expert policy π^E is an optimisation problem. If the expert's policy is considered to outperform all the policies from Π , the following inequality for delayed rewards applies:

$$E \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | \pi^E \right] \geq E \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | \pi \right] \quad (2.19)$$

Following the Eq. 2.16, the inequality transforms into:

$$w^T \mu(\pi^E) \geq w^T \mu(\pi) \forall \pi \quad (2.20)$$

Then the IRL problem translates into the minimisation problem in which the expert's policy μ_E outperforms the chosen policy, where ϵ max (max-margin problem) is that margin:

$$\|\mu(\pi) - \mu_E\| \leq \epsilon \quad (2.21)$$

The existing algorithms within the AL scope include *Max-Margin Planning*, *Feature Expectation Matching (FEM)*, *Parameterized policy class*, *Feature Construction in IRL* and *Generative Adversarial IRL*. This last method has been used in recent studies, as it is a model-free IL algorithm that shows great performance in high-dimensional problems. Its base elements, GANs, are overviewed in the next section (Section 2.4).

2.3.4 Summary

Regarding the IRL methodology:

- IRL methodology takes into account the measurements of the agent's behaviour, measurements of the agent's sensors and the physical environment model;
- building the reward function through IRL might be a process more compact, robust and simple than deriving it manually;
- the generalised policy interaction is undesirable in real applications;
- IRL searches to optimise the reward function from the expert's demonstration;
- the features expectations for a given policy ($\mu(\pi)$) directly relate to the reward function (R);
- IRL is a modern approach to IOC;
- IRL includes direct (AL), indirect and direct-indirect implementations;
- when designing an IRL solution, it must be taken into account the following challenges: ill-posedness, non-convexity, data availability, complexity and dimensionality, feature selection and the ability to generalise.

2.4 Generative Adversarial Networks

GAN are an example of generative models [40], that are able to reproduce the data distribution of a provided dataset. This is the pillar to **AL** algorithms like Generative Adversarial Imitation Learning (**GAIL**), that we employed in this thesis. When proposing the **GAN** methodology, *Goodfellow et al.* [41] created a procedure to train two models interchangeably. The **generator** model G identifies the data distribution and tries to replicate the behaviour. At the same time, the **discriminator** model D aims to distinguish whether a sample was generated by G or belongs to training data (real vs fake). The common objective of each model is to potentiate the occurrence of mistakes by the other model, thus being called *adversarial nets*. According to Eq. 2.22, the interaction resembles a "minmax two-player game" [41]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log (1 - D(G(z)))] \quad (2.22)$$

, where x is the data sample, p_{data} represents the data distribution, $p_z(z)$ are the input noise variables, $G(z)$ is the generator's result given the input noise variables, $D(x)$ is the discriminator's probability that x belongs to the data rather than the generator's distribution p_g

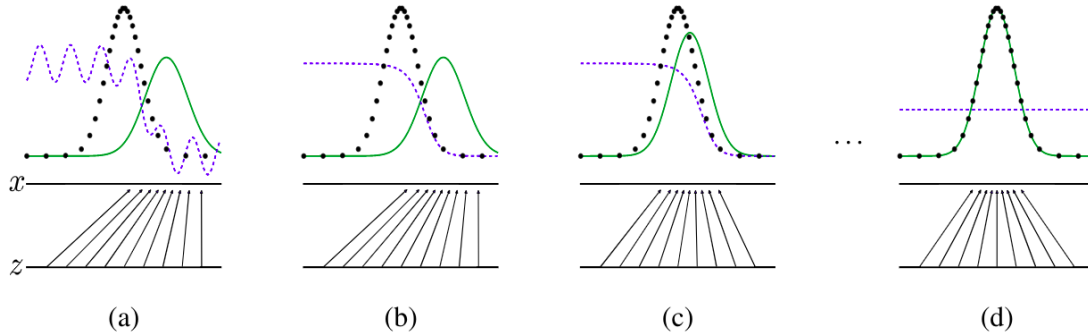


Figure 2.13: "Generative adversarial nets are trained by simultaneously updating the discriminative distribution (D , blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x . The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of a low density of p_g . (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$. (c) After an update to G , the gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{data}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$." Source: [41]

As shown in Fig. 2.13, the discriminator (D , blue, dashed line) is optimised alternately with the generator (G , green, solid line). The black dotted line represents the real data distribution (p_x),

which the generator tries to mimic. Optimising only the discriminator would be computationally expensive but would also overfit on finite datasets. The aim is to keep the discriminator close to the optimal solution as the generator changes. Thus, for each generator (G) optimisation step, the discriminator (D) is optimised k times. The global optimality is achieved when the generator's data distribution matches exactly the data provided, $p_g = p_{data}$.

2.4.1 Loss functions

As the training progresses, it is important to analyse the accuracy and loss functions of either discriminator and generator. They give clues on when the training has stabilised or if the systems are unstable or diverging. Moreover, the higher the network loss, the more frequently it is committing mistakes identifying data. First, the discriminator loss is analysed, and then the generator loss is looked over. As an adversarial network works antagonistically, it is expected that a decrease in one model's loss with result in an increase in the other. However, the change does not occur in the same proportion, only in terms of direction.

Discriminator Loss

Cross-entropy computes the difference between two probability distributions. Thus, it can be used to optimise models like logistic regressions or neural networks. This is the case with GAN [40]: the cost function for the discriminator is the cross-entropy cost. This loss is minimised throughout the training when the real/fake data classifier uses a sigmoid output. In this case, there are two mini-batches of data: the one with the real data samples, which are labelled with 1 and the one composed of generated data, labelled with 0 for all samples. Then, the lower the cross-entropy, the better is the discriminator's performance. The expected behaviour is for the discriminator loss to converge and oscillate around 0.5.

Generator Loss

Following the discriminator's logic and Eq. 2.22, the generator's loss could be computed as the minimisation of $\log(1 - D(G(z)))$. However, the discriminator successfully rejects fake (generated) samples with high confidence, which results in a poor gradient with no necessary information to update the network's weights. As a consequence, the loss is computed as maximising the discriminator's fake samples prediction. The generator then forces the discriminator to mistake the samples [40].

Generator Value Loss

Generator value loss is similar to the simple generator loss: generator value loss will follow every time the generator loss increases. The difference is that the generator value loss corresponds to the temporal difference error and focus on the difference between timesteps' prediction and outcome.

2.4.2 Accuracy

Regarding how close the discriminator is to classify the samples into fake or real data, the model's accuracy plays an important role. The discriminator is evaluated regarding the samples correctly classified as fake data (discriminator accuracy on generator, Eq. 2.23) or correctly classified as real data (discriminator accuracy on expert, Eq. 2.24).

In the end, the discriminator's accuracy is the mean value of each accuracy pair, resulting in the ratio of correctly classified samples over the number of samples evaluated.

$$acc_{expert} = \frac{\sum D(x) == 0}{nr_generated} \quad (2.23)$$

$$acc_{expert} = \frac{\sum D(x) == 1}{nr_real} \quad (2.24)$$

2.4.3 Generative Adversarial Imitation Learning

Among the several applications of GANs, they can be used in the field of IRL, where one recent and powerful algorithm employing adversarial learning stands out: GAIL, proposed by *Ho and Ermon* [42]. Instead of learning a cost function that explains the expert behaviour (such as Maximum causal entropy IRL) and then mapping it into a policy, the proposed GAIL algorithm aims to **directly learn a policy matching the expert's behaviour**. Recalling the model's classification, GAIL is classified as an AL technique. It is model-free and seeks to scale into high-dimensional problems.

Mathematically, the **common IRL approaches** fit a cost function to the expert behaviour, transforming this first stage into an optimisation problem:

$$maximise_{c \in \mathbb{C}} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi}[c(s, a)] \right) - \mathbb{E}_{\pi_E}[c(s, a)] \quad (2.25)$$

where, Π is the set of stochastic policies, π_E is the expert policy, s belongs to the state space $\mathbb{S}$, a belongs to the action space $\mathbb{A}$, c is a cost function from the a family of functions $\mathbb{C}$. The $\mathbb{E}_{\pi}[c(s, a)] = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)]$ is the discounted expectation with respect to the generated trajectory and $H(\pi) = \mathbb{E}[-\log \pi(a|s)]$ is the discounted causal entropy for the policy π . The expert policy is then found by an RL procedure which chooses a cost function $c \in \mathbb{C}$, using a high-entropy methodology. This cost function should assign low costs to the expert policy and high costs to the other policies, following the rationale:

$$RL(c) = \arg \min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi}[c(s, a)] \quad (2.26)$$

The authors focused on developing an algorithm that could express complex expert behaviour without minutely hand-crafting the considered features. At the same time, the IRL method must not overfit when dealing with small datasets. According to *Ho and Ermon* [42], AL may fail to imitate when the set of cost functions ($\mathbb{C}$) does not include the cost function that translates

the expert behaviour, also failing to recover the corresponding policy. However, [AL](#) can scale to higher-dimensional problems, employing policy function approximation.

The proposed [GAIL](#) aims to tackle efficiency in large environments and dispenses to careful tuning the algorithm. It is drawn a connection between [IL](#) and [GAN](#) in which [GAIL](#) searches for a saddle point in Eq.2.27, in model-free imitation problems for large environments.

$$\mathbb{E}_{\pi}[\log(D(s, a))] + \mathbb{E}_{\pi_E}[1 - \log(D(s, a))] - \lambda H(\pi) \quad (2.27)$$

The algorithm starts by finding a function approximation for both a parameterised policy with weights θ (π_{θ}) and a discriminator network with weights w (D_w). Then, [GAIL](#) alternates between the Adam gradient on w for D and a Trust Region Policy Optimization ([TRPO](#)) step on θ for π . The latter step intends to limit the policy change due to noise in the policy gradient. The first step feeds the learning signal to the policy, approaching the agent's behaviour to the expert's.

2.4.4 Summary

The important concepts to retain regarding [GANs](#) are:

- [GANs](#) can mimic the dataset behaviour by reproducing the data distribution of the expert. Therefore, they can generate new data samples that aim to be mistaken for real data. This is achieved by training two models as opponents: a **discriminator** and a **generator**. The first aims to classify data as real or fake, and it is updated k times for each generator update. The generator samples from input noise variables, transforms them to the data space and generates a sample. The generator aims to fool the discriminator, and each one plays an important role when optimising the other's performance.
- There are two key measures to analyse how the training is progressing: the **loss function** and the **accuracy**. The first is a measure of how often each model is committing mistakes, and it is divided into **discriminator loss** and **generator loss**. When one of the losses increases, the other decreases but not in the same magnitude. When both losses stabilise, the model is believed to have found an optimal solution. The **discriminator's accuracy** can be evaluated over all samples, fake samples or real samples. It gives insights into how the discriminator misclassifies data samples and if it is failing more often on the fake or real samples. This is a useful tool when debugging and trying to improve the models.
- One of the applications of [GANs](#) is the [GAIL](#) algorithm. This method is part of the [IRL](#) scope and uses the two adversarial nets to find a well-suited policy. Instead of fitting a cost function to the expert's behaviour, the methodology involves training a generator net until the discriminator cannot distinguish properly between real data or fake data. When using [GAIL](#), there is an optimisation step where models like the [TRPO](#) or Deep Deterministic Policy Gradient ([DDPG](#)) are used. Those models are then loaded into the scenario, after completing the training.

2.5 Hierarchical approaches

According to Hengst [43], "HRL decomposes an RL problem into a hierarchy of sub-problems or sub-tasks such that higher-level parent-tasks invoke lower-level child tasks as if they were primitive actions". Despite being an RL problem with all its characteristics, HRL joins the perspective of layers and discretises how information should be perceived. As stated by Hengst[43], this methodology might not find the optimal solution RL would, but shows clear advantages like reduced computational complexity, compact representation or reusability.

2.5.1 Hierarchical Framework Example

Le et al. [44] use a **hierarchical guidance framework** combined with an expert, to speed up learning and minimise expert related costs. They proposed a framework that can leverage "the hierarchical structure of the underlying problem to integrate different modes of expert interaction"[44], such as in IL or RL.

Another concern is for the high-level expert to trigger only low-level learning when necessary. The formalism there defined includes a two-level hierarchy: the **high-level (HI)** and the **low-level (LO)**. The HI corresponds to choosing the sub-task to be executed next, i.e. the *meta-controller*. The LO consists of executing the actions for the given sub-task, i.e. *sub-policies*. For the elevator example, the sub-tasks the agent must choose from include "going to the elevator", "taking the elevator down" and "walking out of the elevator". These sub-tasks include LO actions such as walking, turning or waiting.

Each sub-task/sub-goal within the HI level are represented as $g \in \mathbf{G}$. The primitive actions belonging to the LO level are denoted as $a \in \mathbf{A}$. The learning agent begins to choose a sub-goal g , executes the corresponding a actions until the sub-task is complete and moves on to the next sub-goal until the task is complete. The HI level has H_{HI} (HI horizon) sub-goals maximum, and the LO level has H_{LO} which means the maximum number of primitive actions to accomplish the sub-task or choose the next one. Thus, the maximum number of primitive actions for a given trajectory is $H_{FULL} = H_{HI}H_{LO}$.

The final goal is to learn simultaneously:

- the observed state space $\mathbf{S}$:
- the meta-controller's HI-level policy $\mu : \mathbf{S} \rightarrow \mathbf{G}$;
- the sub-goal policies $\pi_g : \mathbf{S} \rightarrow \mathbf{A}$, for each subgoal $g \in \mathbf{G}$.

In the end, it is desired to obtain the maximum reward combining both the meta-controller and the sub-policies. Moreover, there is a termination function $\beta_g : \mathbf{S} \rightarrow \{True, False\}$ to end the execution of the policy π_g .

Algorithm 1: Hierarchical Guidance [44]

```

1 for  $h_{HI} = 1 \dots H_{HI}$  do
2   observe state  $s$  and choose sub-goal  $g \leftarrow \mu(s)$  for  $h_{LO} = 1 \dots H_{LO}$  do
3     observe state  $s$ 
4     if  $\beta_g(s)$  then
5       break
6     end
7     choose action  $a \leftarrow \pi_g(s)$ 
8   end
9 end

```

For each sub-policy π_g , the LO-controller determines a *LO-level trajectory* comprising both states and actions $\tau = (s_1, a_1, \dots, s_H, a_H, s_{H+1})$, where $H \leq H_{LO}$. The choice of a the sequence of sub-policies (g_i) constitute the *HI-level trajectory* $\tau_{HI} = (s_1, g_1, s_2, g_2, \dots)$. Combining the HI-level trajectory with the respective LO-level trajectories (τ), it is obtained a *hierarchical trajectory* $\sigma = (s_1, g_1, \tau_1, s_2, g_2, \tau_2, \dots)$. At last, the *full trajectory*, takes the hierarchical trajectory (σ) and substitutes each τ_i for the corresponding set of actions and states.

2.5.2 Summary

The hierarchical formalism, according to *Le et al.* [44]:

- **speeds up learning** and reduces the **cost of expert feedback** in **LfD** approaches;
- creates levels of hierarchies such as the the **HI-level** and **LO-level** controllers;
- higher levels are responsible for **choosing the sub-tasks' sequence** to be executed;
- lower levels control the **specific primitive actions** required to complete a sub-task;

2.6 Chapter Synopsis

LfD algorithms arise from **observational learning** employed by many animals when teaching a diverse range of abilities, humans included. Thus, **LfD** approaches became a trend when it comes to programming agents to perform **complex tasks**. However, the **different embodiment** between the agent and the teacher (for example, between a human and a robotic manipulator) might lead to difficulties in mapping the actions and movements from one to another. Consequently, it is common to choose an **effect-level imitation** rather than thoroughly copying the movements. Thereby, the effect-level imitation requires the agent/learner to understand the desired **demonstration's outcome** before reproducing those behaviours.

The **policy** (π) is the transform function that enables the agent to map the **environment's state** (s , which might only be partially observable) into the **available actions** (a). This is a continuous process in which the policy has its roots in the teacher's demonstration. However, the way the policy is determined deeply depends on the **problem's characteristics** such as the demonstration source and state-space. With these considerations, the policy can be derived by means of a **mapping function**, a **system model** or **plans**. Regarding system models, the translation of the learning technique into the policy derivation depends on a **reward function** given by the environment according to the **agent's performance**. The reward function is a key factor to a successful learning process that is either **manually defined** or **engineered** through other methodologies.

Within **LfD**, one can employ methods from the **RL field** which aims to **maximise overall rewards** when performing a given task. To do so, the algorithms take advantage of both **exploration** and **exploitation**. The former consists of trying different approaches, expecting to discover new action's sequences. The latter reproduces the best actions recorded until then. The **reward signal** is given to the agent by the environment, allowing to evaluate the agent's performance towards the desired goal. Therefore, the **interaction is continuous**: the agent performs a given action a based on the environments' state. In its turn, the environment responds to the action by sending the reward r and moving to the next state.

Addressing the quality of a demonstration, *Grollman et al.* [32], *Dong et al.* [33], and *Mohammad et al.*[34] try to overcome the suboptimal behaviour of human demonstrations: by learning with unsuccessful demonstrations, recognising new sources of demonstration or deciding whether or not to incorporate the information. Many questions are still to be answered, but the **LfD** algorithms are very promising methodologies.

One of the **RL** pillars is the Markov Property and the **MDP**. The **Markov Property** says that if the states are informative, the probability of moving to the state s_{t+1} with the reward r only depends on the current action (a_t) at the current state (s_t). If the system fulfils the rule, it is classified as a **MDP**, with the respective expected rewards, state transition probabilities and state-action-state transition probabilities. Moreover, the **MDP** is commonly identified as a transition graph with the states, actions and transition probabilities depicted. The rewards are only accessed after the decisions are taken and give information regarding a short-time horizon. So, a value function estimates how good a given state is and the best strategy to adopt after that. Both stochastic and

approximate models are considered to find the best policy, and, once again, the choice relies on the challenge's nature.

When considering the measurements of the agent's behaviour along the time, the sensors' measurements and the environment's model (model-based or model-free), there are approaches to determine the corresponding reward function. This methodology is called **IRL** and can overcome the undesired and time consuming **generalised policy iteration**. The main goal is to search and optimise the reward function, which is not known *à priori*, given the states, actions and being supported by a set of features and constraints (ϕ). Algorithms like **GAIL**, within the **IRL**, scope show great performance in high-dimensional problems and use **GANs**.

Then, given the rise of **GANs** as recent and powerful **adversarial methods**, it was explained how two models, namely the discriminator and the generator, can work **against** each other for a **greater good**. After receiving a dataset with demonstrative behaviours, the generator must learn the normal data distribution and reproduce those to fool the discriminator. The more accurate the discriminator, the further the generator will improve. This balance, sometimes fragile, will teach the model how to create expert-like samples. Their behaviour can be analysed through **function losses** or **accuracy metrics**. Among the common applications of **GAN**, like data reproduction in images, one of the usages is related to the **IRL** field. Specifically, the **GAIL** methodology employs them to overcome the problem of fitting a cost function to the expert behaviour. The **GANs** are crucial here to create new samples (generator) that will be rated by the discriminator. At the end, when the training has stabilised, there can be extracted a model, like **TRPO** or **DDPG**.

At last, it was presented a **hierarchical approach** to **speed up learning** in **LfD** methodologies. The main objective is to divide the problem in **higher and lower decision layers**. The higher levels should be responsible for crafting a **strategic sub-task sequence**, and the lower layers determine the **sequence of action primitives** required to complete a sub-task. The advantage comes when the layers work synchronously and turn the complex problem into smaller and simpler tasks.

Chapter 3

Related Work

The current chapter presents a review regarding the state of the art solutions and the common challenges faced in the field. This chapter approaches four categories of solutions. First, the main aspects of the [RL](#) approaches are presented, and how combining hierarchical structures help with performance. Then, there are examples on [IRL](#), which exploit demonstrations and the importance of selecting correct features. The chapter also includes the state-of-the-art solutions regarding the Hierarchical Inverse Reinforcement Learning ([HIRL](#)), including many of the advantages mentioned before. In the end, there is a performance comparison of a generative algorithm in common [RL](#) environments and its performance on pick and place tasks.

3.1 Reinforcement Learning Approaches

[RL](#) approaches are one of the main fields in [ML](#) and simplify the control policy determination through exploration techniques. This makes it easier to program complex robotic systems and allows the system to explore new promising behaviours. It brings interesting aspects to programming, especially in complex environments, as in the case of the manufacturing systems, where the tasks are too complex to being programmed in a completely manual way.

[HRL](#) has proven useful to the Robotics field, and many concepts are being integrated, namely motion primitives. Motion primitives allow the system to converge much faster among the [LfD](#) approaches available.

Hierarchical Reinforcement Learning Hybrid Architecture (2000)

Huber [\[45\]](#) proposes an architecture that aims to incorporate changes using online learning techniques while controlling complex robotic systems. On the one hand, the proposal seeks to reduce the number of exploratory actions needed to find the optimal control policy by using temporal abstractions and closed-loop strategies. On the other hand, it aims to adapt the solution to more complex tasks by expanding the abstraction towards successively higher abstraction layers.

The hierarchical control architecture deals with the system as a Discrete Event Dynamic System (DEDS). The system is built in an abstract and discrete state-space, where policies are continuous abstract actions. Every time a more complex strategy is deployed, other closed-loop actions are introduced. A 4-legged robot was used to validate the approach of the locomotion scenario.

As summarised in Fig. 3.1, the proposed architecture comprises a closed-loop control policies block, the DEDS supervisor block and the Reinforcement Learning Component. The **closed-loop control policies block** provides faster learning by modelling a set of abstract actions. The actions need to be reactive and able to deal with any perturbations that might come up, which reduces the need to acquire compensatory action knowledge. The robustness and stability are guaranteed due to the abstract actions' physical space partitions.

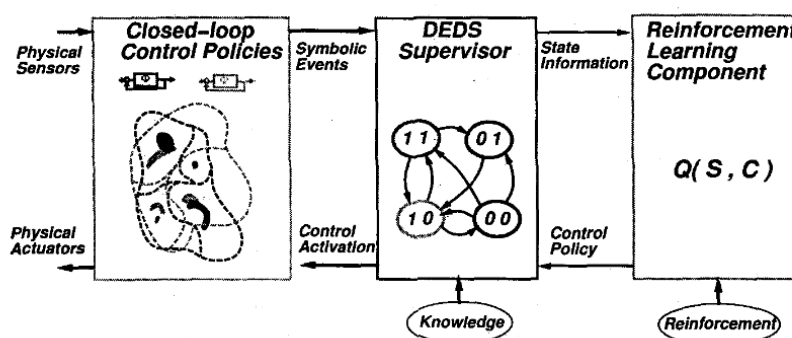


Figure 3.1: Hybrid control architecture proposed for a robotic locomotion system. Source: [45]

The **DEDS Supervisor** is responsible for reducing the state-space and, consequently, the dimensions of the information fed to the **RL** block. Thus, the learning time of the control policy drops significantly. This aspect can be used further to increase the flexibility and transversality of intelligent robotic solutions. For instance, many abstract actions can be reused to build a similar or customised product when producing a product's family. At last, the **RL Component** uses an **MDP** approach and the *Q-Learning* algorithm to compute the control policy. This is the block that will determine which action to perform and inform that to the adjacent block.

The experiments performed with the four-legged robot proved that a robust strategy for the turning gait manoeuvre is **learnt rapidly** (in about 600 learning steps). For a more complex rotation manoeuvre, it was compared the results that **reused** previously learnt policies versus their absence. The outcome was a **faster learning process** and best performance overall (higher rewards). At the same time, it allowed overcoming hidden states due to the abstraction layer employed- **increased robustness**.

Hence, in this thesis, we intend to exploit the advantages of an abstract approach to the actions available and reduce the state-space for improved learning rates. These are essential features when it comes to allowing **flexibility** for the robot in case of change-overs within a given manufacturing process. However, the main problems with this approach include:

- The abstraction is designed to deal with vertical hierarchical layers, i.e. every time a new

layer is added, it corresponds to a higher level in control. This creates limitations in terms of new skill's integration (new tasks of the same layer to be included);

- Q-learning (RL algorithm) is not very fast and might be outperformed in terms of convergence speed. Additionally, all the problems associated with RL also apply. So, employing only the Q-learning algorithm makes this solution less flexible;
- Regarding environment validation, the hybrid control architecture was not tested when different tasks were required. This solution seems to use only the policies provided initially and does not specify what happens when skills are required later in the process.

Adaptive Hierarchical Memory Structure for Autonomous Robots (2008)

Obayashi *et al.* [46] trained an agent to learn, memorise and associate recollections. It is used an actor-critic RL method in the learning phase. The memory system is accomplished with the adaptive hierarchical structure. At last, the association process is implemented with an Associative Chaotic Neural Network (ACNN). The system proposed, depicted in Fig. 3.2, takes advantage of hierarchisation and divides the problem into two blocks: memory and learning.

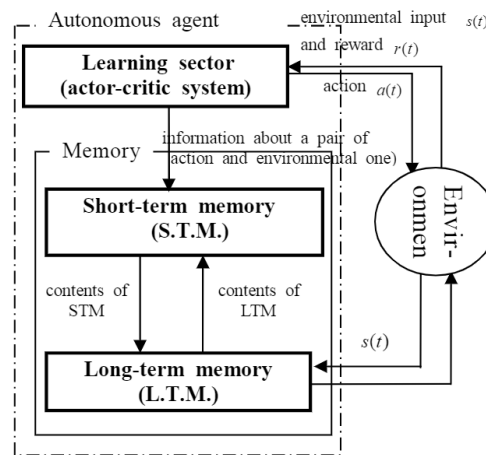


Figure 3.2: Proposed System [46]

The **learning block** incorporates the actor-critic methodology to choose the best sequence of actions based on the environment's state and rewards. The **memory block** comprises both Short-term Memory (STM) and Long-term Memory (LTM). The **short-term memory** remembers the learning paths collected in the learning phase and discards all the other unused data. In parallel, the **long-term memory** saves the useful experiences in the STM.

The results and effectiveness were tested in mazes. The conclusion derived from the trials was that by executing smaller demonstrations, the agent could generalise using both STM and LTM. In the case of the larger scale maze, the demonstrations are not enough, and the algorithm fails to exit the maze. These results express:

- how the hierarchical approaches can simplify complex problems;

- how the [RL](#) approaches might not respond to problems that require great exploration and many demonstrations.

In conclusion, the adaptive hierarchical memory structure is an example of **hierarchical structures**' usage. In this case, hierarchy is applied differently from the previous example and relates to the knowledge gathered (divides the data between long and short memory). As explained by the authors, the memory division simplifies the problem, **deconstructing its complexity**. However, it does **not** seem to **scale properly** to larger mazes. This shows a great **disadvantage** in high-dimensional problems and therefore would most likely fail to perform well in problems where the complexity often changes. For situations like setup updates in industrial manufacturing, the systems need to be flexible enough and scale properly to more complex (or not) tasks. Moreover, this approach requires the model to use big datasets and might need to be shown the same demonstrations again. The fact that the [STM](#) will be erased might also be a drawback when trying to reuse older models. Given that, memory hierarchisation is not the best solution for the problem ahead.

Although [HRL](#) performs better than simple [RL](#) overall, it **shares many limitations** with common [RL](#). One of those limitations is related to exploration. Both [RL](#) and [HRL](#) take long to converge and require many episodes to test different behaviours. So, that is why [IRL](#) shows a performance boost. When demonstrating expert behaviours to the agent, it should explore similar behaviours and, in theory, **converge faster** as the agent will less likely test nonsense actions. Examples of [IRL](#) approaches will be given next section.

Overcoming Exploration in Reinforcement Learning with Demonstrations (2018)

Given the exploration problem in robotics that require executing several steps efficiently over a long horizon, *Nair et al.* [47] proposed joining [RL](#) with demonstrations to guide their agent's exploration. Their method is built on top of [DDPG](#) and Hindsight Experience Replay ([HER](#)) and was tested in a block-stacking problem (stack 2 to 6 blocks on top of each other in a target location). The blocks have random initial positions and a fixed goal location. The authors used 100 human demonstrations. The demonstrations were obtained using teleoperation in virtual reality, which can be difficult for non-expert operators, like those typically existing in manufacturing shop-floors. The combination of [DDPG](#) with demonstrations was achieved by four different strategies:

- A second **demonstration buffer** (besides the one in the [DDPG](#) algorithm) that stores demonstration data in the format (s_t, a_t, r_t, s_{t+1}) . Each minibatch's examples drawn from the demonstration batch are used as an off-policy replay data for both actor and critic updates.
- A new [RL](#) loss function is only used on the demonstration examples for training the actor - the **behaviour cloning loss**. This loss is the standard in [IL](#) and helped improve the learning process but prevented the learned policy from improving beyond the demonstration.
- To prevent the sub-optimal demonstrations problem stated above, a **Q-Filter** was used. The [IL](#) loss is only applied to states where the critic $Q(s, a)$ establishes that the actor's action is worse than the demonstration.

- **Resets to demonstrations states** were utilised to overcome the problem of sparse rewards in very long horizon tasks, where some training episodes were reset using states and goals from demonstration episodes. The authors use the final state reached in the demonstration as the goal, as in HER. This allows to expose the agent to higher reward states but is only applicable during training.

This work was validated on the simulated robotics' environment OpenAI's Fetch, where the agent receives all object's positions as its observations. The action space was composed of two variables: the end-effector 3D position and the desired distance between the gripper's fingers. The following tasks were also tested using a fully sparse reward and compared with Behavioral cloning (BC) and HER: Fetch Push (block placed on top of a table is moved to a target position also on the table (gripper blocked)); Fetch Slide (block placed on a table is moved to a given target position, outside the robot's reach); Fetch Pick-and-place (block placed on top of a table must be moved to a target location in the air).

The proposed method showed significantly better results, being faster at learning the tasks when compared to HER and performing better than BC. When considering the number of timesteps for convergence, a speedup of four times was achieved in pushing and two times in sliding, compared to HER. When it comes to the pick-and-place task, the HER baseline cannot solve it at all, while RL+demonstrations achieves almost 100% after 1M episodes.

Regarding the block stacking task, two different rewards were tested, a fully sparse (only gets a reward if all blocks are in one place) and a "step" reward (reward when the robot moves a block into its target location). Using only HER, the robot fails to perform correctly in any number of blocks, achieving 0% of success rate with both sparse and step rewards. By utilising BC and BC+HER, the agent is only able to achieve 65% with a sparse reward and two blocks. For 3, 4, 5, and 6 blocks, the robot cannot achieve any success, with both sparse and dense rewards. With the proposed method, the authors are able to achieve a 99% success rate with two blocks, 89% with three, and 1% with 4, all with sparse rewards. With step rewards, the success rates range from 91% to 4% for 4, 5, and 6 blocks, respectively.

Although this method shows promising results, the RL + demonstrations approach takes about one million timesteps to achieve 100% of success rate in a simple pick and place, 150 million for 88% success rate in stacking three blocks, and 300 million for a success rate of 54% in stacking four blocks. Besides this aspect, for the algorithm to be successful, 100 demonstrations are needed at least.

An Automatic Robot Skills Learning System from Robot's Real-World Demonstrations (2019)

Li *et al.* [48] tried to simplify robot control by using expert demonstrations and avoiding explicit programming policies by hand. The studied tasks include object reaching and block stacking, which are some of the common robotic tasks. In the real world, two 6-DOF UR5 Robotic arms with Robotiq85 grippers and two Asus Xtion PRO cameras were used.

The first problem pointed out was associated to the embodiment problem between an human and a robotic manipulator. The second problem was the inefficiency and safety issues associated to exploration in real robotic systems. So, the authors proposed a framework with four main modules, which are depicted in Fig. 3.3. Each module is responsible for:

- **Data collection:** Record demonstrations from the robots in the real world, using RGB cameras, that would keep track of object and gripper positioning;
- **Data Processing:** Pre-process demonstrations, extracting features, sampling and transforming data to the simulation scenario;
- **Training:** Train a policy to learn how to perform the skills, using **DDPG+HER+** demonstrations;
- **Skill reproduction:** Transfers the policy actions to the real robotic arm and translates the camera images to the simulation format, according to Fig. 3.4.

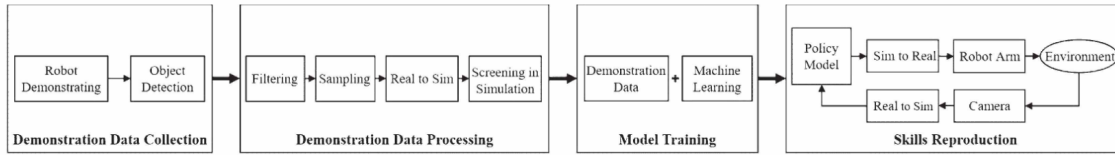


Figure 3.3: System overview of the framework proposed by *Li et al.*[48]. The system was designed for automatic robot skills learning and includes four modules: demonstration data collection, demonstration data processing, model training and skills display. Source: [48]

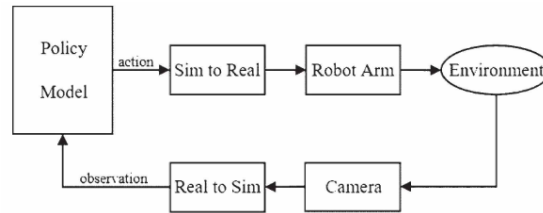


Figure 3.4: Skills Reproduction Module in the learning system. Source: [48]

Mujoco and OpenAI gym were used as the simulator, which offered integration with **RL** implementations and allowed bench-marking with other projects. In terms of algorithms, they compared simple **DDPG+HER**, **DDPG+HER** with simulated demonstrations and **DDPG+HER** with real-world simulations. This last approach corresponds to the proposed methodology. The results show the simple **DDPG+HER** fail to solve the task, while the approaches with demonstrations can do so. So, it is clear that demonstrations play a key role in tackling complex and uncertain tasks, such as in real scenarios. The results show 100% of test success rate and 50% of train success rate, and it is able to perform better than the expert, acquiring new undemonstrated skills.

Despite the advantages evidenced by the demonstrations in real-world robotic tasks, we can identify two main problems. The first has to do with the **framework architecture**. The framework

was designed to work in the specific situation where the robots and camera were defined beforehand. So, different scenario specifications require reconfiguration, programming and adaptation in terms of integration and modelling. The demonstrations are also very complicated to acquire, requiring filtering that can reduce data significance and diversity if too strict. The second problem is related to the **robotic task** *per se*. When the task needs to be updated, the intelligent modules need to adapt too. All demonstrations, processing and training need to be performed from scratch. On the other hand, the observation space only considers the gripper and object positioning, which does not extrapolate to other manufacturing tasks that require more moving the manipulator. The wrist orientations are also fixed, which can deal with simple cubic objects, which is a disadvantage.

3.2 Inverse Reinforcement Learning Approaches

IRL solutions resolve the policy search problem inherent to RL. This proved to perform well when the policy functions are too complex and hard to design manually. Other issues are related to the undemonstrated data that the RL systems cannot overcome by not expanding the understanding behind the demonstration's observations. Here, the main focus shifts to **feature choice** and **engineering** when designing an accurate and responsive system. The reason behind this is the necessity to choose the best representation that shows the **impact** of the expert actions while minimising dimensionality. In short, it is crucial to choose the best **representativity** for the smallest complexity, thus focusing on feature selection and engineering.

Feature Construction for Inverse Reinforcement Learning (2010)

Levine *et al.* [49] present an algorithm for Feature Construction for Inverse Reinforcement Learning (FIRL). It can select and design reward features from a set of component features. The objective also includes the determination of the optimal policy according to the demonstrated examples. Another interesting feature considered in the proposed solution is the optimisation step that tries to find out where the contemplated features are insufficient to solve the problem.

Later, in the experiments' phase, the FIRL approach is compared towards three other IRL algorithms: LPAL, MMP and Abbeel&Ng, in a grid world. The FIRL proves to be slower than the others but scales better for higher dimension problems. Even not being provided with the perfect features, the FIRL (black line) proves to be comparable to all the other algorithms with perfect or primitive features. Both for deterministic and non-deterministic models, the algorithm reaches low levels of wrong predictions. The increase of accuracy and the corresponding decrease in mispredictions is demonstrated in Fig. 3.5.

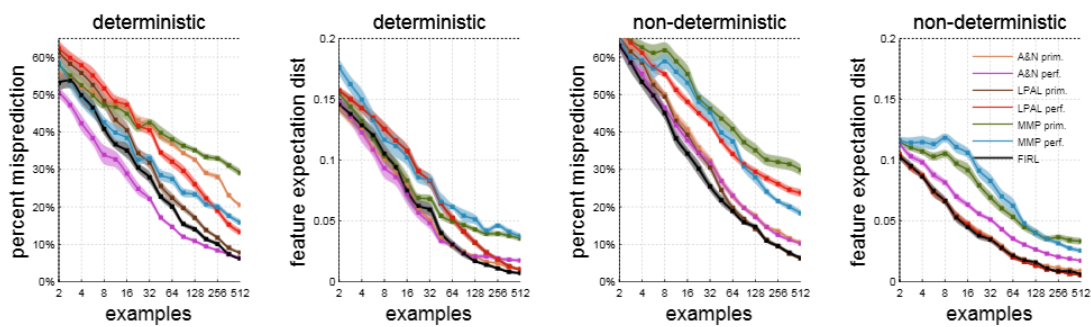


Figure 3.5: Accuracy comparison for FIRL, LPAL, MMP and Abbeel&Ng. All algorithms except FIRL used perfect or primitive features provided. Mispredictions generally decrease when increasing the number of examples given. Source [49]

The experiments related to transference between environments proved that FIRL could generalise the learnt rewards to different environments. It can also build features from primitive components. According to Fig. 3.6, the general accuracy decreases in an environment with distracting

colours range between 2 and 20. However, the **FIRL** approach is more robust by being able to interpret the rewards.

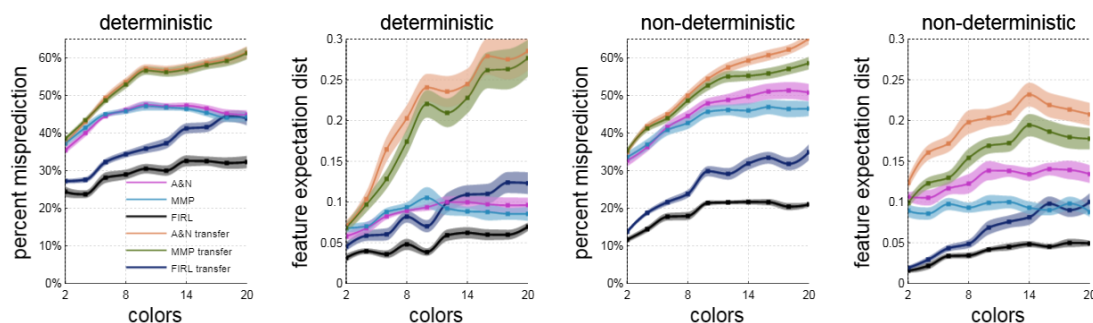


Figure 3.6: Accuracy comparison of **FIRL** and Abbeel&Ng, for environments other than a training one. The **FIRL** approaches are less affected by colour distractions. Source [49]

The **FIRL** algorithms provided great insights on the design of reward features. The algorithm proves to find precedence relationships and can transfer between environments. This makes the approach robust to interference and valuable to understand the agent’s motivations from observed demonstrations. However, the algorithm has some issues related to convergence guarantee and fitting step (need to penalise excessively deep trees). The proposed solution is interesting from the flexibility to task changes problem’s point of view: it allows one to determine ideal policies by perfecting the features considered and overcoming the **RL** dependence on reward functions. Nonetheless, it lacks abstraction layers so that it can be able to deliver different products.

Robot Navigation in Crowds (2014)

Vasquez *et al.* [50] employ two **IRL** approaches to solve a robot navigation problem in crowds, where obstacles constantly move. The **IRL** methods compared were **Max-Margin IRL**, **Maximum Entropy IRL** and a manually weight’s tune. Two different metrics were defined: the objective evaluation, which includes path length and smoothness, and the subjective metrics that consider intrusions. The metrics were useful to compare **IRL** methods, the generalisation ability, and different sets of features. The demonstration through teleoperation brought important insights as humans naturally predict the obstacles movements, proving the value of **LfD** programming.

It was concluded, according to Fig. 3.7, that both **IRL** algorithms perform similarly, and both accomplished better results than the manually tuning approach (the smaller the bars, the better the performance). The different performance measures for different feature sets prove the **IRL** dependence on the data significance. It suggests further improvement of feature sets has a more direct impact than choosing different **IRL** algorithms.

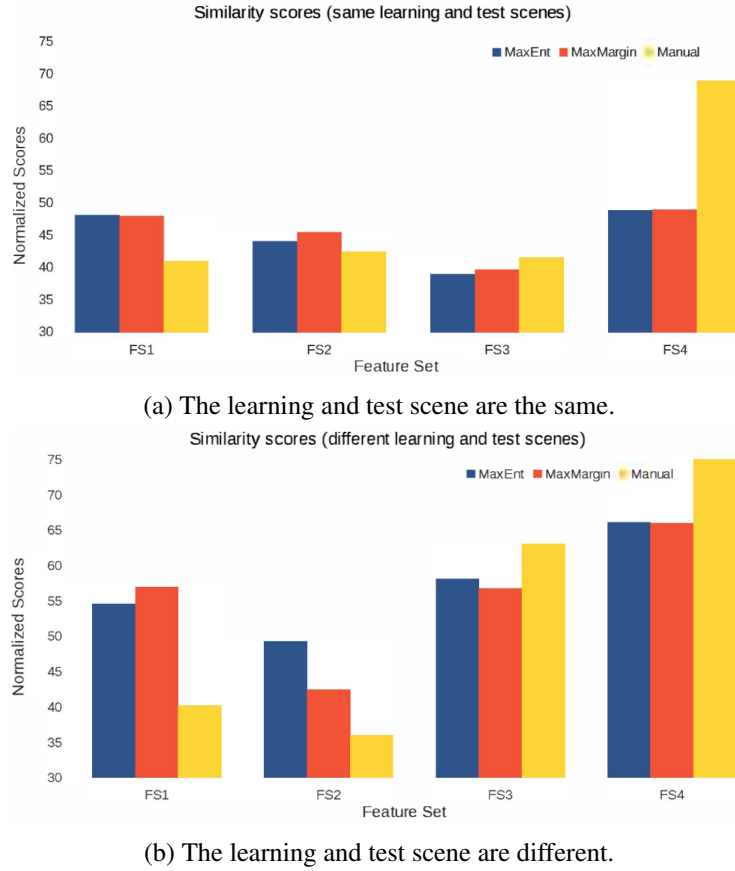


Figure 3.7: Normalised Scores per feature set and learning algorithms; Better results represented by smaller similarity scores. In the right chart, learning and test scenes are the same. In the left chart, learning and test scenes are different. Adapted from [50].

However, the framework considers the environment is dynamic by re-sampling and re-planning every ten timesteps. Either way, the methodology does not deal with different objectives along the way, only analyses differences in the environment. Regarding generalisation, the approaches do not maintain the performance within acceptable limits when changing environments, making it unsuitable for variations within a task, like in our context of mass customisation, where manufacturing a product family brings a wide variety in robot's processes.

Inferring Task Goals and Constraints using Bayesian Nonparametric Inverse Reinforcement Learning (2020)

When faced with complex manipulation tasks, *Park et al.*[51] observed that IRL failed to recover the reward functions. That problem was common when there were explicit environment constraints, like forces or axis alignments, which are hard to transmit to the robot. Those constraints were frequently restrained to a part of the demonstrations. Thus, the authors proposed the Constraint-based Bayesian Nonparametric Inverse Reinforcement Learning (CBN-IRL) algorithm. CBN-IRL can deconstruct a complex demonstration into sub-tasks, each of them associated

with a sub-goal and a local constraint. Each sub-goal g will be assigned with the specific sub-goal reward R_g , giving the agent sequential goals.

However, the challenge is finding the best-suited demonstration partitioning while associating them with the rewards and constraints for maximum likelihood. The **CBN-IRL** algorithm was tested in **2D navigation tasks** and **5D tool manipulation tasks**. The **first** experiment showed that **CBN-IRL** did infer the task of following a wall, when dividing the trajectory into two. The **CBN-IRL** could successfully generalise when being showed a different sinusoidal trajectory, while the other algorithms did not. Regardless of the number of sub-goals, the **CBN-IRL** did produce good results, while the other methods were more impacted by that, producing trajectories less similar to those demonstrated.

The **manipulation tasks** were more complex and simulate realistic manufacturing scenarios. Those tasks included clamp-passing tasks, where the **CBN-IRL** agent performed consistently well. That agent showed the best generalisation capabilities. However, the number of sub-goals seem to have more impact in this situation: the higher the number of sub-goals, the more difficult it is to generalise to different clamp configurations.

Despite the robustness and advantages of sub-goals in complex chores, the **CBN-IRL** does not offer ease in reconfiguration. Tasks with different objectives but common skills are not reused and do not simplify robot's reprogramming. This fault is a considerable disadvantage in systems where **flexibility** and **customisation** is required very often.

3.3 Hierarchical Inverse Reinforcement Learning Approaches

Similarly to [IRL](#) approaches, [HIRL](#) tries to find the optimal reward functions. However, [IRL](#) algorithms sometimes fail to fully determine the function when delayed reward problems are considered or when many reward functions yield the same behaviour. Hence, adding the hierarchical aspect can improve the algorithms further when the complex task comprises a sequence of simpler sub-tasks, as in the case of assembly chores.

Long-Horizon Tasks with Delayed Rewards (2016)

Krishnan et al. [52] address a complex problem by transforming it into a sequence of smaller tasks with shorter horizons, considering only a few demonstrations. Later, the learnt sub-tasks are used to define local rewards in the global structure. The proposed solution reaches efficiency in smaller datasets by identifying changes in local linearity instead of identifying all systems.

The changes in local linearity correspond to changes in motion, for example, when objects contact or different force/torque is required. These physical aspects are often easier to describe and these states are called *transition states*. Another improvement, before sub-task identification, is the definition of rewards not at the end-points' segments but by augmenting the state-space. These new variables can memorise the previous transition states and be implemented employing features in [IRL](#).

The framework uses [HIRL](#), and the sub-tasks are modelled using Linear Gaussian systems. The two main goals include learning the locally linear sub-tasks from the demonstrations' features and create local rewards based on the learnt sub-tasks and the global task. The algorithm was evaluated towards different benchmarks and followed the proposed solution, including:

1. the generation of task-specific demonstrations;
2. learning the reward function with the [HIRL](#) approach;
3. comparison between the [HIRL](#) solution and a Q-Learning [RL](#) agent, regarding convergence.

In a **GridWorld test**, the agent has to sequentially collect the rewards 0 and 1 represented in the green square in Fig. 3.8. To do so, the agent must go through the narrow passage twice, which causes the [RL](#) agent to struggle because it does not know the previous states. The [HIRL](#) agent, due to space augmentation, solves the problem much faster and achieves the higher rewards (blue line in Fig. 3.8.d). For the **Parallel parking problem**, five demonstrations were given to the agent, which are represented in Fig. 3.9.

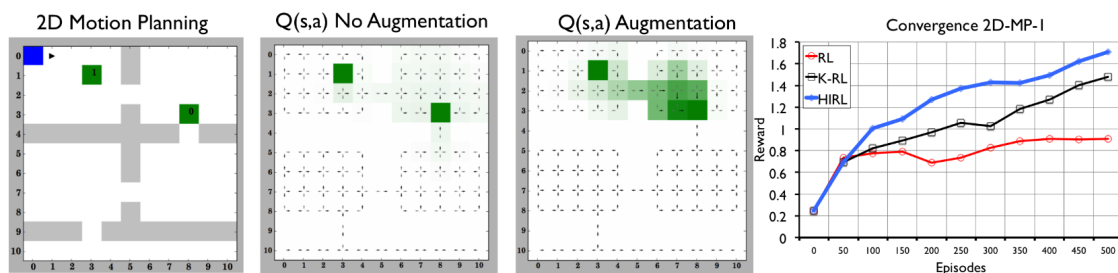


Figure 3.8: **Augmentation Results:** For the test where the agent must collect the green squares sequentially (0-1) and go through a narrow passage to do so, the **HIRL** agent achieves greater success by using the space augmentation, i.e. by keeping track of past states. Source: [52]

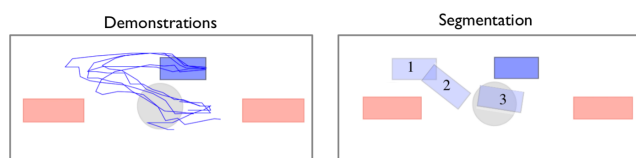


Figure 3.9: **Parallel Parking Demonstrations.** Source: [52]

Compared to **RL** and **IRL** approaches in a parallel parking task with noisy dynamics, the **HIRL** approach converges faster. The convergence gap is especially visible when only some variables are available to the agent, which is where the velocity is not observable. Fig. 3.10 demonstrates these conclusions and proves the advantages of the segmentation/hierarchy in complex tasks and noisy environments.

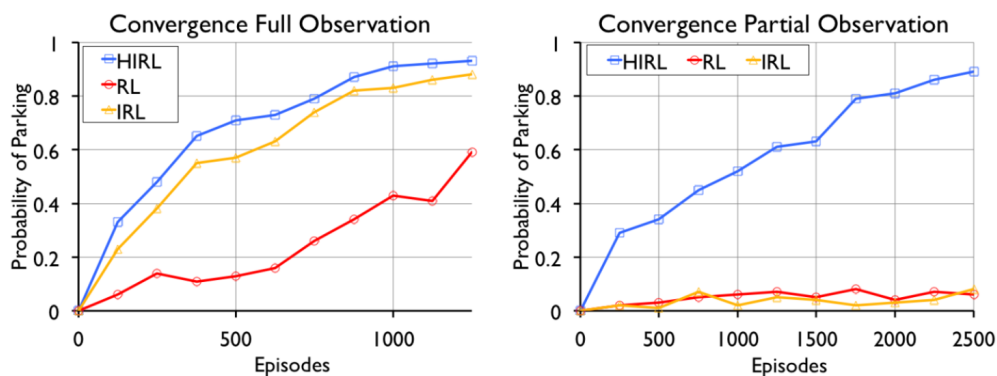


Figure 3.10: **Convergence Results:** The **HIRL** agent, represented by the blue line, converges faster than **RL** and **IRL** agents, when learning the parallel parking task. The gap is higher when only a part of the state space is observable due to the hierarchical aspect that solves sub-tasks sequentially. Source: [52]

The **HIRL** algorithm was also compared in the parallel parking problem. It aimed to evaluate the response dependence to the number of demonstrations and the robustness to system noise. According to Fig. 3.11[A], the **HIRL** agent can park in most cases, surpassing the other agents. The fewer the number of demonstrations, the more explicit the **HIRL** agent's performance. In

Fig. 3.11[B], the **HIRL** agent is the most robust agent to system noise, even with only five demonstrations acquired.

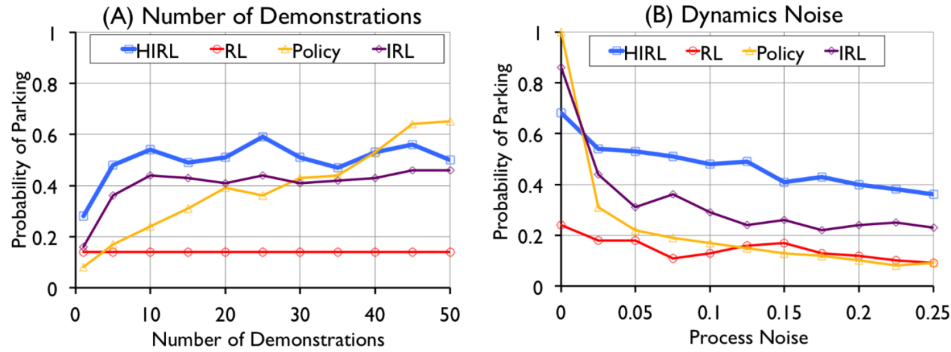


Figure 3.11: **Robustness Results:** The **HIRL** agent, represented by the blue line, achieves great results for a lower number of demonstrations (a) when compared to other **RL** and **IRL** agents. Even though the accuracy in parallel parking is reduced when the system's dynamics includes noise, the **HIRL** algorithm gets the best performance overall, with only five demonstrations (b). Source: [52]

A test was created with decreasing space between obstacles to assess the response to more challenging tasks. Once again, the **HIRL** approach reached the best performances. The biggest gap between that agent and the **RL** agent happens between 0.5 and 1.2, according to Fig. 3.12.

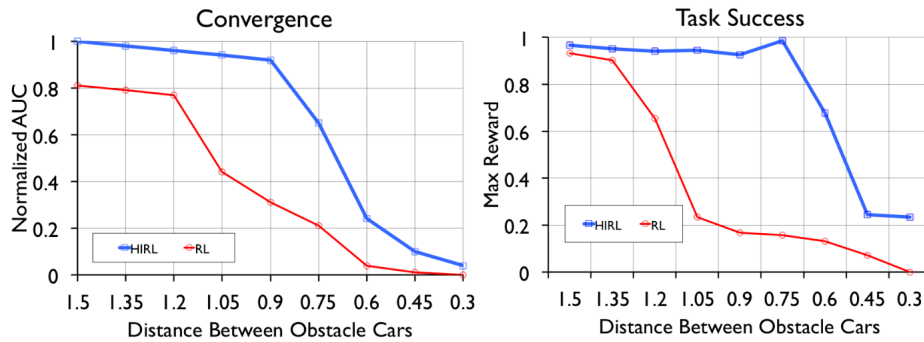


Figure 3.12: **Difficulty Tests:** The experiment evaluates the response of two agents when the space between obstacles decreases. The **HIRL** agent, represented by the blue line, achieves the max reward for a great range in distances, only decreasing when the obstacles are very close. Thus, the **HIRL** agent achieves better rewards for all the demonstrated distances Source: [52].

The demonstrated results prove **HIRL** to be a good approach when choosing the appropriate sub-task abstractions. The framework solves two different problems:

- learning the sequence of sub-tasks;
- extracting additional features from the learnt tasks.

Summing up, the **HIRL** agent converged much faster than the other agents and achieved the maximum rewards overall. When considering these results, the **HIRL** approach seems appropriate to solve the hierarchical problem of manufacturing customised products, and the robustness appears to suit well task change-overs.

Probabilistic Prediction of Interactive Driving Behaviour (2018)

Sun et al. [53] use a probabilistic **HIRL** approach to autonomous vehicles trajectory generation. The choice of a probabilistic approach is linked to the need to solve the uncertainties of human behaviour. The need for the interactive characteristic addresses the dependence on historical data and future plans. The authors employed the **HIRL** framework to incorporate discrete and continuous driving decisions, such as humans driving. The probabilistic approach should consider the agent's historical trajectory and those of the surrounding vehicles.

The generation of trajectories by humans when driving is naturally a hierarchical and probabilistic process that combines both discrete and continuous driving decisions. The discrete decisions learn complex patterns similar to a game, such as yielding or passing. The continuous decisions influence the speed, acceleration and smoothness. Fig. 3.13 depicts the case when a predicted vehicle tries to change lane, and the trajectory is computed through a mixture of both distributions (discrete and continuous).

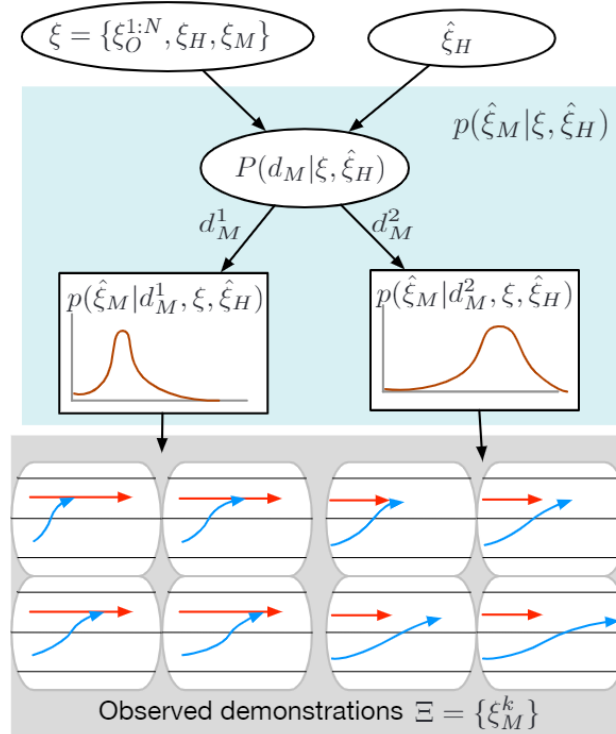


Figure 3.13: Model for the probabilistic and hierarchical trajectory-generation Process when the predicted vehicle (in blue) tries to change lane to the one where the agent's vehicle (host, in red) is currently on. Source: [53].

The hierarchical probabilistic models proved their worth for interactive prediction and learning the different layers in the observed demonstrations. Once again, the hierarchisation of tasks allows for convergence and simplifies complex missions as the driving example or an assembly task.

The experiments used a road-map and four different vehicles: one (red) merging into the lane, one (blue) maintaining the course in that lane and two (black) other vehicles ahead and behind the blue one. The results show the proposed method achieves better overall performance compared to [HMM](#) and Neural Networks. The predictions are accurate in terms of conservatism and non-defensiveness, matching the ground truth.

At last, this work proposes a prediction relying on future plans of the host vehicle instead of directly making predictions on historical information. This is possible by including the [HURL](#) method that captures the many layers of the human trajectory generation. Accordingly, the method shows to be promising in other problems involving task hierarchisation, regarding the different natures of the sub-tasks as continuity and distribution. Nonetheless, in a manufacturing environment, the sequence of tasks are well known. The challenge is not choosing the next behaviour according to the predicted circumstances. Instead, the objective is to create a hierarchy within the model so that different tasks can be replaced interchangeably, with as little adjustments as possible.

3.4 Generative Adversarial Imitation Learning Approaches

Ho and Ermon [42] evaluated the **GAIL** algorithm by applying it to classic control environments and high dimensional tasks. The analysis consisted on the relative performance achieved and the behaviour for variable number of demonstrations. The **GAIL** performance was compared to the expert behaviour (performance of 1.0), a random agent (performance of 0.0), **BC**, Feature Expectation Matching (**FEM**) and Game-theoretic Apprenticeship Learning (**GTAL**) algorithms. Another comparison factor was the influence of the number of expert demonstrations required to achieve the desired behaviour. Analysing Fig. 3.14, **GAIL** with the **TRPO** produces the policies with the best performance over all environments, regardless of the demonstrations provided. For the higher-dimensional environments, **GAIL**'s performance is accentuated when compared to **BC**, **FEM** or **GTAL**. Regarding the number of expert demonstrations, **GAIL** requires smaller data batches to produce good policies.

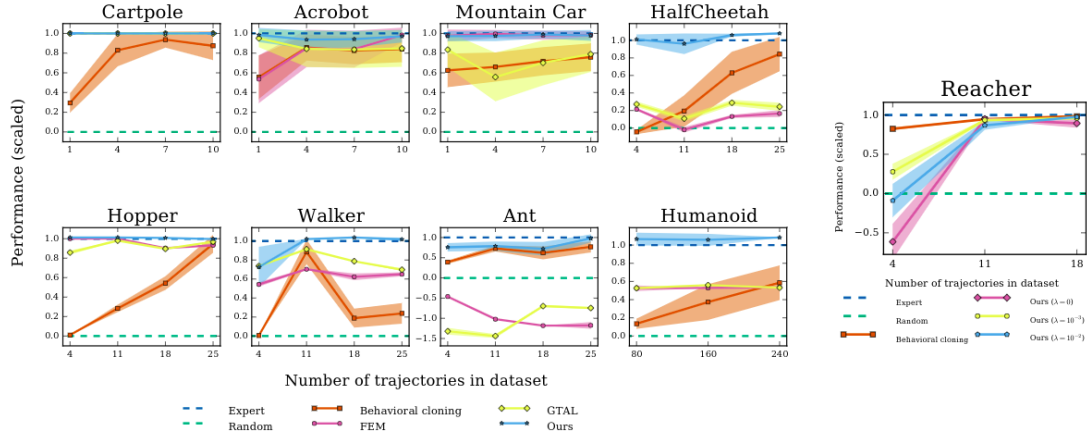


Figure 3.14: **GAIL** comparative results: performance of learned policies. The y-axis is negative cost, scaled so that the expert achieves one and a random policy achieves 0; Causal entropy regularisation λ on Reacher. Source: [42]

Similarly, *Liu et al.* [54] applied the **GAIL** algorithm to two Robotics environments (OpenAI's Gym Fetch): the reach and the pick and place tasks. The evaluation metrics used were the distance error and the success rate throughout an episode. The distance error consists on the difference between the desired position and the gripper position at the end of each episode. The success rate was evaluated as the number of times the agent reached the target withing a given margin, for the maximum number of iteration steps. Analysing Fig. 3.15, **GAIL** (blue line) performs better than the proposed Hindsight Generative Adversarial Imitation Learning (**HGAIL**) [54], Generative Adversarial Self-Imitation Learning (**GASIL**) and Proximal Policy Optimization (**PPO**), both in terms of higher and faster success rate and lower distance errors.

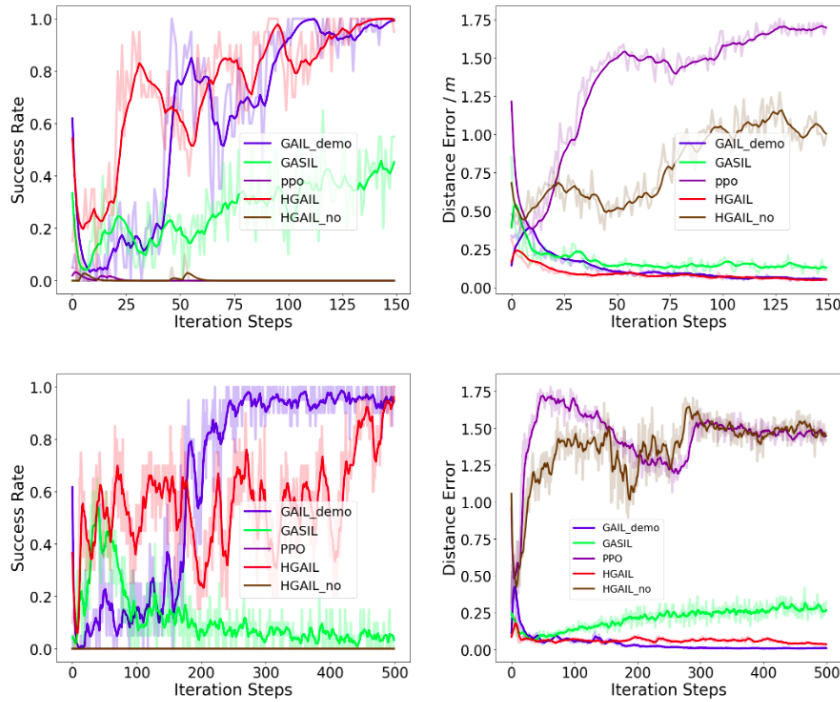


Figure 3.15: Learning curves for **HGAIL**, **GAIL**, **PPO** and **GASIL**. The first row depicts the results on the reaching task, and the second row the task of object grasping. The success rate translates the percentage of successful gripper positioning over the episode horizon. The second column shows the evolution of the distance error, the difference between the desired and the real gripper positioning. Source: [54]

Ding et al. [55] studied how **GAIL** and **BC** would behave on the four rooms environment, a pointmass block pusher, a simple robotic pick and place and a two-cube robotic stack. The goal-conditioned tasks which include bottlenecks, like those mentioned, normally take long before converging to a desired behaviour. The metrics used to compare **GAIL**, **BC** and the proposed algorithm comprising experience replay (*goalGail*) was the success rate over training, such as depicted in Fig.3.16. One disadvantage emphasised by the following results is that **GAIL** is not able to outperform the demonstrator, which requires the demonstrations to be carefully acquired.

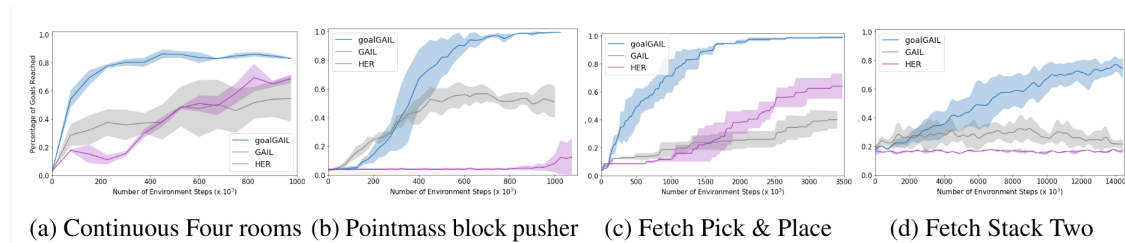


Figure 3.16: Comparative results between **HER**, **GAIL** and *goalGail* on a) Four rooms task, b) point-mass block pusher c). Source: [55]

3.5 Chapter Synopsis

The way robots are programmed has been evolving for the past years. The solutions are using many **ML** aspects when trying to respond to the desired system's **flexibility**.

The **LfD** approaches include **RL** and **IRL** architectures. The **RL** techniques take advantage of exploration and exploitation to learn the tasks. As justified in Section 3.1, **RL** is commonly used in robotics systems but might fail to deliver the optimal solution. These situations often require additional demonstrations and demand many resources to explore new possibilities. However, when the **RL** is combined with task hierarchisation, the system achieves better performance.

The following methodology addressed was **IRL**, along Section 3.2. Those approaches can design reward features without manual intervention when the correct and necessary features are observed. They can identify some motivations behind the human's demonstration when designing the policy function. Some issues stated as drawback the agent's inaccuracy when the complexity increases.

Combining both **IRL** advantages and Hierarchical approaches, Section 3.3 presents the advantages of both hierarchical approaches and **IRL** methodologies. The hierarchical structures' abstraction layer allows the agent to deconstruct the complex task into simpler sub-tasks, resulting in increased convergence and better performance overall. These aspects, when joined together, appear to be very promising in completing the assembly of a family of products because:

- the assembly nature involves sequences of actions and operations to achieve the desired product, which matches the way hierarchy works;
- the product diversification within a family can constitute a great challenge in terms of complexity, to which a policy function must respond. The **IRL** methodology seems to be well suited to identify the key aspects in policy generation.

GAIL is an **AL** algorithm that is classified as **IRL**. It has been used in recent problems because it shows high convergence and good performance, especially in high-dimensional case studies. This approach differs from other **IRL** approaches: instead of fitting a cost function to the expert behaviour, the **GAIL** methodology is model-free. It alternately improves the discriminator weights (D_w) and policy weights (π_θ). So, using the **GAN**'s paradigm, both policy and discriminator work against each other to develop the best model possible. Overall, the **GAIL** shows better performance in difficult tasks like reaching and grasping two objects. In the latter, the performance boost is clearer, as picking and stacking two cubes is considered very complex. Solutions involving **GAIL** have proven to be extremely powerful in the field of **LfD**. The evaluation metrics used when analysing the agent's performance are related to the success rate and the final error.

Chapter 4

Robotic Systems Simulators

Testing theories, ideas, and designs before implementing the solutions in "real-life systems" is crucial. Thus, this concern is also essential in research fields, according to *Plasencia et al.* [56]. In robotics, this issue is fundamental to avoid tremendous expenses in specialised machinery. State-of-the-art solutions require simulators equipped with sophisticated features and tools. On the one hand, simulators need to be flexible and accurate enough to mimic the environments. On the other hand, computational costs, complexity and timeliness are essential constraints. Simulators advantages include lower development costs (only intellectual costs implied), safer/controlled environments and flexibility on choosing which and how many sensors to apply [57].

Simulators differ significantly in performance, licences' type, physical complexity, programming language and available models. However, the primary purpose of testing and validating a framework in an adequate simulator is to easily transfer the solution to the real system, applying as few modifications as necessary. For short, a *"robotic simulator builds the bridge between 'artificial creatures' theory and robotics"* [56].

This chapter compares the candidate robotics simulators available at the moment, focusing on simplicity, integration and flexibility aspects. The considered simulators were: OpenAI Gym, V-Rep, Gazebo, Unity and robo-gym.

4.1 Robotics Simulators Comparison

According to *Plasencia et al.* [56], the robotics simulators used in learning and/or research environments are of massive importance in exploring, testing, debugging, and validating approaches. Concisely, the characteristics that were pointed out as distinctive factors regarding the development of new robotics solutions were:

- soft learning-curve;
- user-friendly interface;
- hierarchical organisation of models;

- possible integration with contemporary scientific advances;
- available developers' community.

Many available simulators are employed in robotics problems, but only a few are ready to implement machine learning techniques. Among the simulators referred in the literature, this comparison focused on the characteristics of the Gym, V-Rep, Gazebo, Unity and robo-gym.

4.1.1 OpenAI Gym

Gym [58] is a *"toolkit for developing and comparing reinforcement learning algorithms"* [59] which was created by OpenAI. Their main objectives were to develop the RL field while putting up a universal benchmark and standardising environments that ease publications' implementation in real systems. This necessity comes with the promising results of RL and deep learning methodologies when applied to difficult and complex problems.

The framework provides environments dedicated to testing and validating algorithms among the machine learning research community, such as the one depicted in Fig.4.1. The tool is designed so that the researchers focus more on the algorithm implementation and applicability rather than creating a specific environment or tuning physical environmental aspects [56]. The environment's versioning ensures reproducibility and meaning to previous tasks and scenarios. Moreover, Gym is an open-source platform (<https://github.com/openai/gym>) that has been adopted worldwide, resulting in a supportive community that flattens the learning curve.

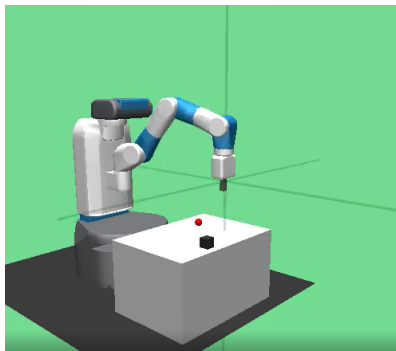


Figure 4.1: Gym Environment: Fetch Pick and Place. Source: <https://gym.openai.com/envs/FetchPickAndPlace-v1/>

According to *Brockman et al.* [58], the following design decisions are the framework's root:

- focuses on implementing environment's interface, rather than different agent's interfaces;
- considers both the sample complexity (time to learning) and the final performance;
- stimulates community to share ideas rather than competing for results;
- cares for environment's versioning, guaranteeing reproducibility;
- monitors the environments and performance by default.

4.1.2 V-REP

V-REP (Fig. 4.2), now CoppeliaSim (<https://www.coppeliarobotics.com>), is a robotic simulator that supports C/C++, Python, Java, Lua, Matlab or Octave languages [56]. The simulator is not open-source, but it is available with an educational licence. The V-REP simulator provides various sensors and robot models (like classic robotic arms and mobile robots). Thus, it is highly customisable and comes up with different algorithms like collision detection and inverse kinematics. Programming techniques can also be customised as add-ons, plug-ins or Robot Operating System (ROS) nodes. V-Rep has no dedicated node within ROS, unlike Gazebo 4.1.4, but it is possible to overcome this drawback if need be. Models are easily imported or modified without the need to dig into XML files, making the V-Rep a user-friendly simulator. The simulations are easily controlled via ROS Services.

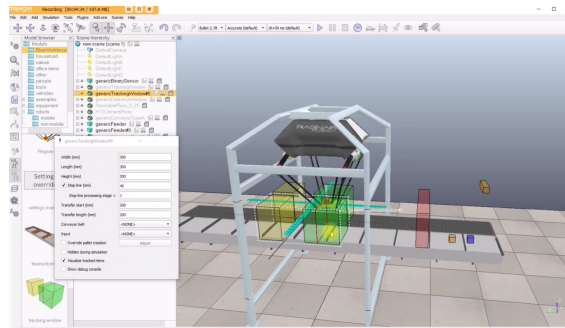


Figure 4.2: V-REP Simulator. Source: <https://youtu.be/MX3VXvZFk0U>

According to *De Melo et al.* [57], other advantages include support for multi-robot applications, factory automation simulations, fast prototyping, and remote monitoring. Thus, the platform is highly versatile and dynamic.

4.1.3 Unity

De Melo et al. [57] points out that **Unity** (depicted in Fig. 4.3) is a simulator that has been used in game development, both 2D and 3D. It also allows augmented and virtual reality, support for collision mapping, space ambient occlusion and others. These characteristics make it possible to be used for robotics simulations, even not being its principal function.

Many dependencies are required to integrate the Unity simulator to ROS, making it complex to apply in industrial scenarios. Moreover, it is not open source and requires licensing. Regarding simulation realism, Unity is one of the most recommended simulators available, according to *De Melo et al.* [57]. However, it is possible to import other pre-existing robotic environments through a URDF file. The interface is easily tackled and allows assembling other objects easily.

By being used mainly in gaming environments, Unity has a small community focused on robotics environments. This aspect results in less support for development difficulties and fewer resources to reuse. Thus, the development time increases due to successive trial and error trials.

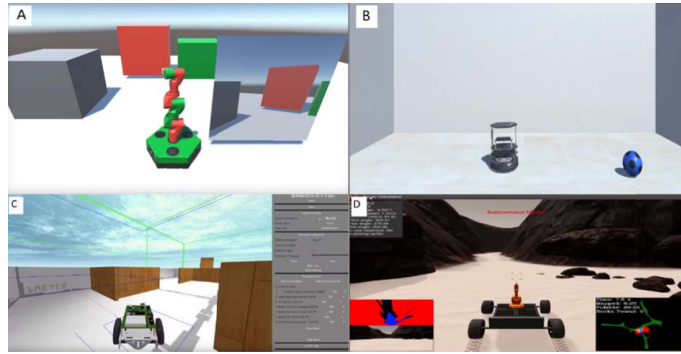


Figure 4.3: Example applications on Unity simulator. Source: *De Melo et al.* [57]

4.1.4 Gazebo

Gazebo (<http://gazebo-sim.org/tutorials>), depicted in Fig.4.4, is an open-source platform that integrates with **ROS**, which is a dedicated Robotics operative system that requires learning and experience, which means the learning curve is somehow complex. Thus, it is mainly programmed using C++, having some support for Python.

Regarding environment modelling, Gazebo offers an editor with simple shapes [60]. It is possible to create new environments or reuse others from the community but requires familiarity with SDF, an XML based format. Also, the edition of pre-existent models might not be possible, which makes the tool less user-friendly, and more limited in terms of freedom to scene editing. Thus, *Nogueira* [60] classifies Gazebo as a more challenging tool for implementing "cognitive theories" than V-Rep.

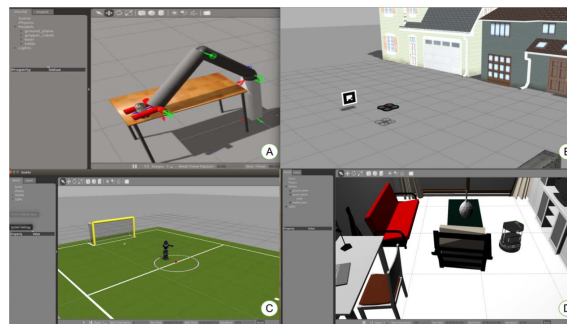


Figure 4.4: Example applications on Gazebo simulator. Source: *De Melo et al.* [57]

4.1.5 Robo-Gym

Unlike the ones above, Robo-Gym is not a simulator itself, but works as a framework proposed by *Lucchi et al.* [61] to reduce the gap between task simulation and real robots. They aim to provide researchers with a tool that allows a smooth transfer between simulated environment and real-world robotics, which is called **robo-gym** (<https://github.com/jr-robotics/>

[robo-gym](#)). The robo-gym framework uses OpenAI Gym that has been increasingly incorporated into state-of-the-art [RL](#) problems. Additionally, robo-gym provides an interface to industrial hardware, independent of any license or additional difficulties regarding the systems' integration.

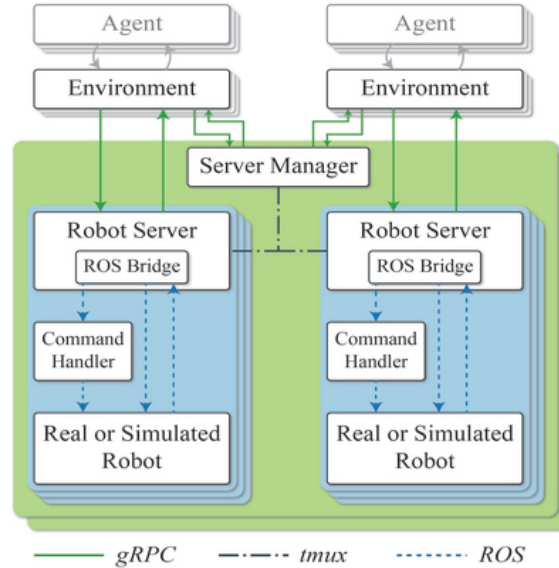


Figure 4.5: The robo-gym architecture. Source: [61]

As depicted in Fig.4.5, the framework's architecture includes two main parts: the server-side and the environment-side. The **server side** is represented in green and corresponds to the real/simulated robot (based in Gazebo and [ROS](#)) alongside the required communication mediums. The **environment side** is the interface to OpenAI Gym (agent + environment).

Concerning community support, all the help to the OpenAI Gym framework also applies to the robo-gym's environment side. The models provided are of industrial robotic manipulators (Universal Robots UR5 and UR10), which would be a good starting point to the problem's industrial context, and the manipulation of the given environment can be achieved similarly to OpenAI Gym. Regarding software license, this is open-source software that makes it easier to develop research projects. Compared to the previous tools, the main advantage is the transferability between simulation scenarios and real-world systems, with the ability to handle distributed hardware and software. Thus, the analysis of different algorithms and robotic problem framing is more straightforward, bringing pragmatic approaches to state-of-the-art technologies.

4.2 Chapter Synopsis

Simulators arose with the necessity to test, support and validate the robotics developments before implementation. The simulators previously referred to in this chapter are summed up in Tab. 4.1. The table includes information on how easy it is to adapt to the simulator, how intuitive the interface is, if it is an open-source framework, and if there is access to support among the established community. Other specific information is displayed, like the availability of robotics environments, programming language, and support to distributed hardware.

Simulator	Learning-curve	User-friendly Interface	Open-source	Community Support	Available Robotics Environments	Program. Language	Distributed Hardware Support
Gym	flat	yes	yes	yes	yes	Python	no
V-Rep	medium	yes	no	yes	partially	C/C++, Python, Java, Lua, Matlab or Octave	no
Gazebo	medium	poor	yes	yes	yes	C++ (Python)	no
Unity	medium	yes	no	no	poor	C#	no
robo-gym	flat	yes	yes	yes	yes	Python	yes

Table 4.1: Comparison between simulators

After considering the problem's nature and the advantages and disadvantages of each tool, it was decided to use **robo-gym** as the task's simulator. The main reasons were the support provided, employing Python language, which eases implementations for [RL](#) approaches; it provides distributed hardware support and mainly allowing the case study transference to real-world much more effortless. This framework combines the advantages of Gazebo in terms of the physical engine, in which there is a vast community to support share models and help with questions or problems. By employing Gazebo, which was designed to work with [ROS](#), robo-gym makes the transference between simulated to real environments almost instantaneously. The required modifications are the connection with the robotics' agents and the connection to external sensors. At the same time, robo-gym shares the advantages of OpenAI Gym without the MuJoCo limitations. With the integration with the Gym framework, it is possible to benchmark the results and use the [RL](#) algorithms available.

Chapter 5

Learning from Demonstration using HIRL

Until now, we have identified the problem with robotic systems flexibility, reusability and robustness to environment's variability or uncertainty. Given the state-of-the-art methodologies studied, the ones under the scope of LfD have shown good performance in complex tasks. And, in the previous chapter, we have compared and contrasted software that could be appropriate to robotic simulation. So, with the required tools, we are now able to define the case study and propose a flexible approach to the task's context. In this chapter, we present all the implementations carried out in the low-level simulation as well as the validation steps.

5.1 Case Study

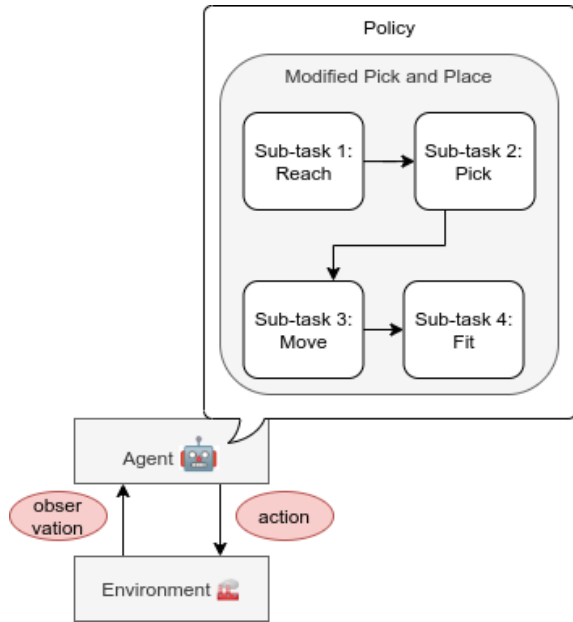
The scenario chosen for this thesis was a **modified pick and place task**. The traditional one is common in manufacturing and can be **divided** into three main operations: reaching the object, picking it, and moving it to another place. This division is the key in this study. We could have chosen any other task that supported discretisation, but the pick and place is the **basis** of any operation the robotic manipulator performs. It requires the manipulator to move itself and objects around. But why call it "modified pick and place"? We do so because we consider a fourth sub-task after the traditional pick and place, the **fitting**, which involves working on the product rather than moving it. The options can include screwing, soldering, inserting, but we opted for fitting in an object, like when building a puzzle (the final product).

The fitting operation involves placing the object in a 3D location with a given orientation. In the scope of product customisation, the sub-task of fitting the object tries to **simplify** how a product is built. On the one hand, this task can be interpreted as an abstract operation that allows reconfiguration according to the product or situation. On the other hand, the fitting itself makes it easier to represent object placement: building a product is modelled like building a puzzle where part placement is similar to fitting the pieces in. Also, this approach works regardless of the

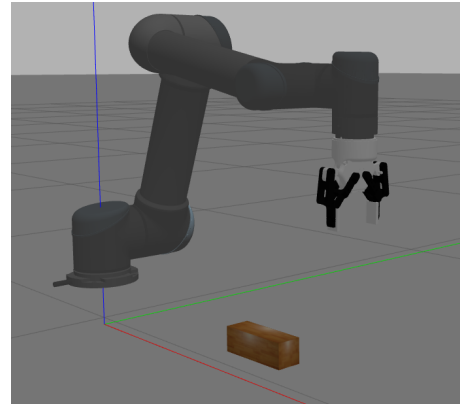
object's shape and only requires its spatial representation. The feature of working with different objects, provides the robotic agent with **enhanced flexibility and adaptability**.

Fig. 5.1a shows the integration of the agent, the environment and the modified pick and place task. According to the **RL** paradigm, the agent chooses its action based on the environment's observed state and the policy developed. That policy should complete the task that involves, sequentially, four sub-tasks. Those are reaching, picking, moving and fitting.

Given the modified pick and place task, we have defined three components: an industrial **robotic manipulator**, a **gripper** and an **object**. The manipulator gives support to the other two and moves them around the environment. The gripper is essential to grasp the parts and tools employed in the manufacturing task. The object simulates any product's part or tool as it is abstract enough in its shape.



(a) Case study- The modified pick and place task.



(b) Simulated environment on reset, includes the UR5, the Robotiq 85 and the object

Figure 5.1: Case study description.

The Fig. 5.1b displays the simulated environment on reset, where the manipulator's base frame is set at $X = (0, 0, 0.1)$ m to simulate the attachment base in a workspace. The manipulator was placed near the origin to eliminate coordinates offsets, impacting **RL** algorithms' performance. When transferring the coordinates to another base frame, applying the corresponding homogeneous transformation is necessary. The manipulator takes an "elbow up" configuration, which is more suited to pick and place tasks and minimises collisions.

5.2 Proposed Solution

The solution developed includes two parts. The first was creating the simulation scenario, and the second is the framework for model flexibilisation. The **simulation scenario** was created to be compatible with real-world tasks, integrates all software parts, allows to add more objects and deals with control and supervision necessary. The **framework** proposes a form of task discretisation, where the modified pick and place is modelled as four sequential sub-tasks. According to Fig. 5.2, the solution allows model re-utilisation for different configurations, either product variation or other tasks with the same base operations. Given the related work in the area, we have opted for using **GAIL** as the intelligence algorithm, because it is within the **LfD** scope. It shows performance advantages on complex tasks and uses reward functions. This middle ground between **RL** and **IRL** is what makes **GAIL** interesting. Like the **RL** methodologies, the **GAIL** requires the definition of a reward function, which makes it possible to adapt the environment's feedback to each sub-goal. On the flip side, the **IRL** nature is what helps reducing the exploration the agent needs to perform. Between models, the framework **reuses** both generator and discriminator networks.

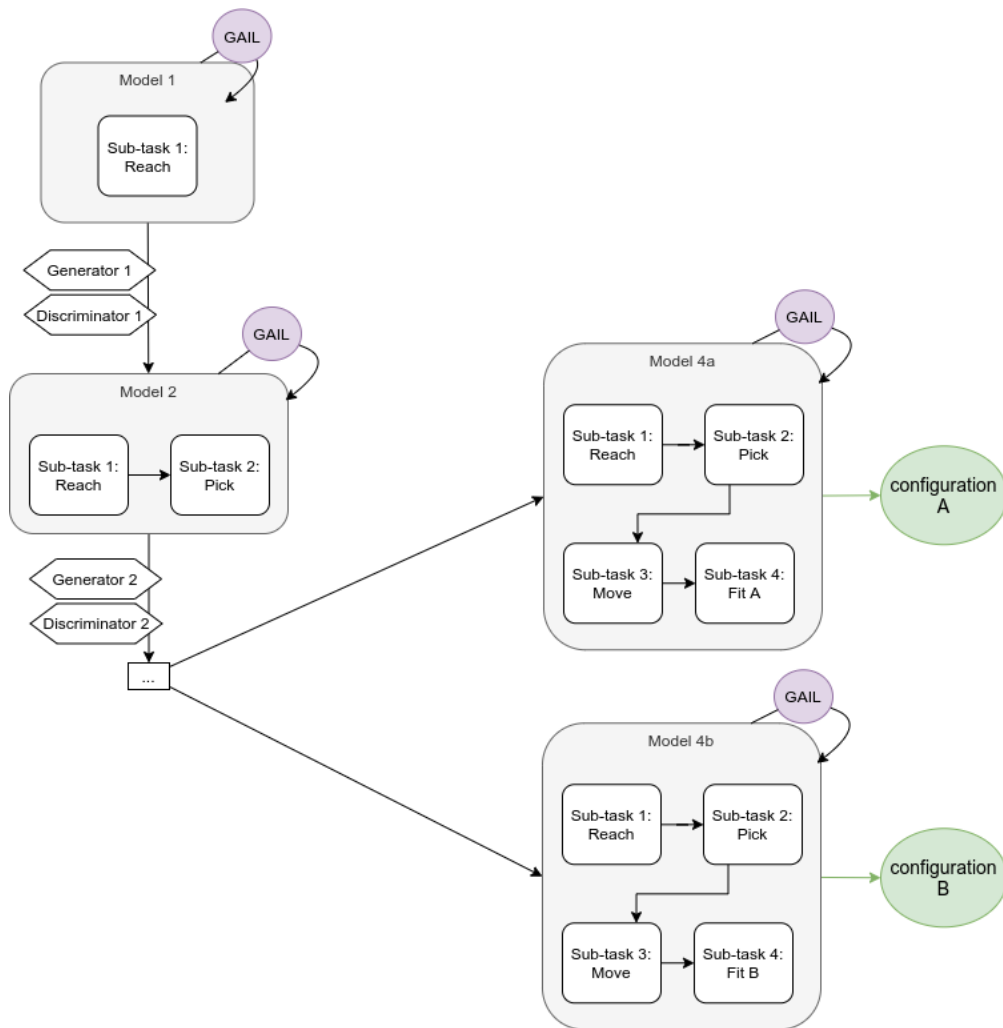


Figure 5.2: Proposed solution for maximising model reuse in different task specifications

5.3 Case Study's Low-level Simulation

When adapting the robo-gym simulation scenario, it was essential to modify its **low-level components**. Those modules comprise a server manager and several robot servers. Referring to Fig.4.5, the **server manager** is a central tool that bridges each environment (in an OpenAI Gym style) and each simulated/real robot (in ROS). It also defines the protocol of the messages (using gRPC) exchanged every control cycle, reporting any communication errors or invalid data. The protocol definition is critical when dealing with such diversity throughout the data flow: ROS and Gazebo versioning, real vs simulated robot, Python vs C++ programming language.

Then, for each robot (coded as a **robot server**), the server manager has to instantiate a compatible **ROS bridge**. This communication interface works as a driver and translates the information between the RL agent and the ROS topics. The main operation includes setting the robot's state (and any other object in the respective ROS world) and collecting the state to input the environment. Due to its nature, the bridge has to be flexible enough to contact both simulated and real robots. To meet the case study's requirements in the scope of this thesis and to respect project scalability and further integration, a **new ROS Bridge was developed**, in Python, based on the original one. On the Robot server-side, the cycle rate is of 20Hz. Regarding ROS, the corresponding Joint State Controller [62] publishes at a rate of 125Hz (period of 8ms), and the arm joints are monitored at a 10Hz rate.

The **environment in ROS and Gazebo was also updated** to meet the simulation scenario necessities. Namely, there was the need to incorporate a robotic arm, a gripper and objects simulating product parts, shaped as a parallelepiped. These modifications were developed on the simulator's lower level at the corresponding ROS bridge. The physical model simulation and low-level control fell back on ROS and Gazebo tools which try to meet pragmatism and realism, bringing new advantages to the pre-existing environments. Thus, this chapter describes the simulator's low level and the modifications proposed.

5.3.1 ROS Model and Control

After presenting the available simulators in the Chapter 4 and opting for the robo-gym framework, it was required to adapt the available scenarios. Although robo-gym simplifies the information transfer between the agent and the hardware, it does not provide all the tools to bring up the challenges faced by an assembly task. Nonetheless, the robo-gym's advantages are valuable enough to justify the modifications to be enumerated next. Moreover, it will solve a research gap related to the simulation scenarios for robotics purposes in the RL field. With the Universal Robot's UR5 [63] environment as the starting point, there was the need to incorporate a gripper that could handle several parts and move them across the environment.

For the robotic arm plus the gripper, the modifications were carried on and tested on **ROS Melodic** and **Gazebo 9**. These specifications were to meet the robo-gym requirements at the time of this thesis.

5.3.1.1 UR5 Manipulator

The assembly task mainly requires an industrial robotic manipulator to support the gripper and move around the parts. The UR5 (Fig. 5.3) is a robotic arm from Universal Robots for which the Gazebo ROS model is available at the ROS industrial Git repository, alongside UR3 and UR10 [64]. By having the UR5 model integrated with the robo-gym, which corresponds to the real robot available in our laboratory, this robot was chosen to be part of the simulation scenario.

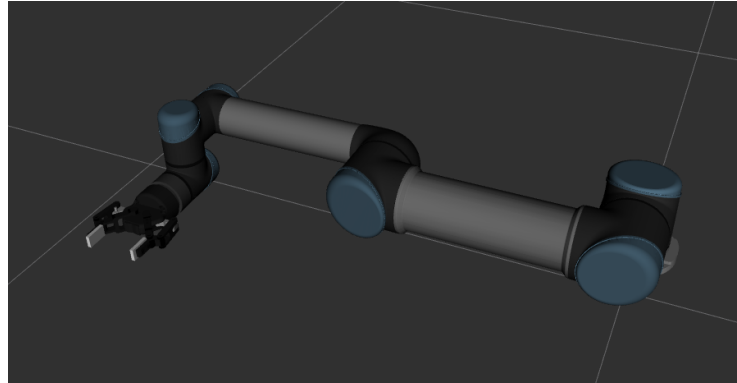


Figure 5.3: UR5 Robotic Manipulator model with gripper attached

In the scope of this thesis, there was the need to manipulate objects resembling parts and tools. Thus, a gripper was attached to the already provided UR5 model to move objects around the workspace. The integration was accomplished by the manipulation of *urdf xacro* files and by using the **MimicJointPlugin** to make the simulated joints work in Gazebo.

UR5 control

The manipulator joints' control was accomplished by publishing commands as a Joint Trajectory Point [65]. The published message associates the joint name to its corresponding position at each control cycle in radians. Moreover, the joint controller is an effort controller [62], which requires a PID module. The PID gains used for the joint (in ROS bridge) are enumerated in Tab.5.1, which were tuned by trial and error.

Joint	P	I	D
shoulder pan joint	10000	200	1
shoulder lift joint	10000	200	1
elbow joint	2000	20	1
wrist 1 joint	500	1	1
wrist 2 joint	500	1	1
wrist 3 joint	10	0.1	1

Table 5.1: PID gains for the effort joint trajectory controller

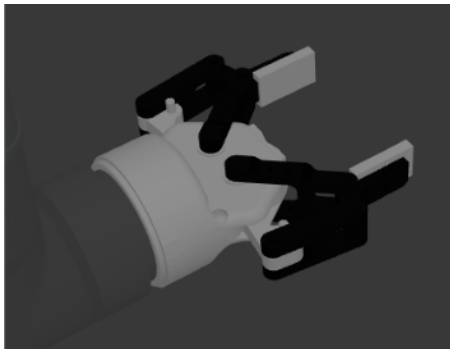
Regarding the **PID** adjustments, the joints required higher **PID** gains because a slight angle variation will result in a more significant shift in the end-effector positioning. The described control relies on the implementations performed for the original UR5 **ROS** bridge, with improvements in **PID** gains and data handling.

5.3.1.2 Robotiq Gripper

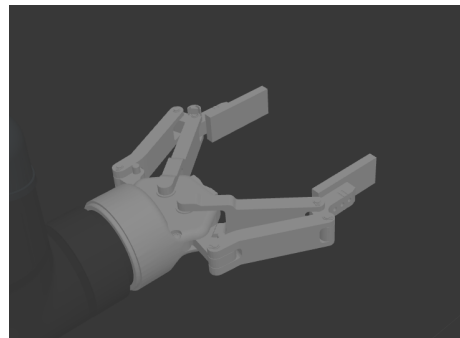
Among robotics developments, the Robotiq grippers are commonly chosen [66, 67, 68] and the models from the original ros-industrial repository [69] have been continuously updated, given the substantial number of repository forks and closed issues. The Robotiq gripper models are also included in **RL** implementations [70] and integrated with OpenAI Gym environments [71].

The grippers used in this scenario are Robotiq 85 (Fig 5.4a) and Robotiq 140 (Fig. 5.4b). The choice for the two-finger grippers is related to:

- the available two-finger gripper at our laboratory (3D printed similar to Robotiq’s structure);
- Robotiq grippers are widely used among other simulations.



(a) Robotiq 85 Gazebo model



(b) Robotiq 140 Gazebo model

Figure 5.4: Robotiq 2 finger gripper models

The additional modifications applied to the models provided in the ros-robotiq repository [69] include joint limits corrections in the urdf files and Gazebo support for mimicking gripper joints. These modifications aimed to solve the broken joints in the Gazebo simulation and make it easier to interact with objects that would fly away or slip out of the gripper before. These changes were proposed to the main repository as a Pull Request (<https://github.com/ros-industrial/robotiq/pull/186>) and solved common issues reported among users.

The gripper’s joints had to simulate the physical behaviour and interact with objects to allow model validation and real-world transferability. The desired behaviour was achieved by employing the **MimicJointPlugin** [72]. For object interaction and object grasping, the plugin used was the **Gazebo grasp fix plugin** [73] which simplifies the material properties tuning for both the gripper and the objects. The plugin attaches the object to the palm/hand gripper when the gripper finger links apply opposing forces. This plugin helps reduce object slippage and collisions that cause

the object to fly away. Those situations are reported by many users as the main problem when the gripper interacts with the object, being a source of unstable behaviours.

Gripper action control

The grippers available at the research laboratory do not include any force/torque sensors. Therefore, the control cannot rely on result signals to evaluate grasping. The methodology incorporated consists of an Action Client that sends goals in a predefined format to the gripper joints.

For the action control [74], there is a client and a server applications that communicate via *ROS Action Protocol*, according to Fig. 5.5. To perform object grasping, the chosen controller was a *GripperActionController* [75] and the client application uses a *SimpleActionClient* [76]. During the interaction between the client (gripper controller) and server controller (ROS control at the joints), there are exchanged control messages in both directions, according to provided tutorials:

- from client to server (as request): goal message and cancel messages
- from server to client (as a response): feedback, result and state messages

The **goal message** sent by the client application is a *GripperCommandGoal* [77], which comprises the desired position goal in millimetres between the gripper's fingers. The message's format is `float64 position, float64 max_effort`. The data is then cast to `float32` for consistency with OpenAI Gym.

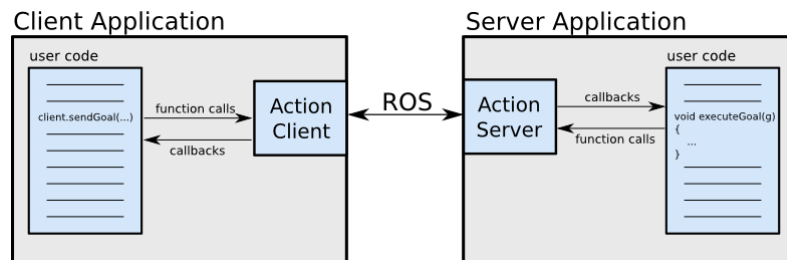


Figure 5.5: Client-Server Interaction. Source [74]

In this scope, the max effort applied to the joint is not limited by the client (an optional argument), but the Gazebo models define that as 1000 N . This constraint means that the gripper joints will apply the effort needed to reach the goal. So, the command to be sent is the fingers position.

Gripper module

The functionalities regarding the gripper control were implemented in the form of a specific module to promote scalability and flexibility. This separate implementation allows for easier gripper's integration or removal according to the bridge or the environment. For the gripper incorporation, it is only required to instantiate a dedicated object (at the bridge initialisation) and call the open/close methods (every control cycle, whenever the joints are updated). Thus, this gripper module, which is independent of the gripper model used, deals with:

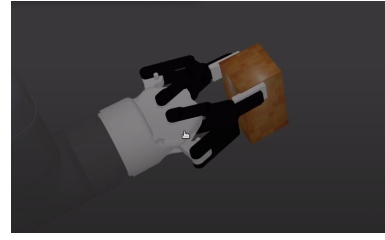
- the creation of the goal message specifying the desired gripper gap size, where the finger's positions come from the environment's actions;
- the conversion between the relative gap size (0-1 or 0%-100%) to the corresponding distances in meters for each gripper specifications;
- the routines regarding the [ROS](#) action client, either being the creation, waiting for service availability or sending the commands;
- providing simple methods to open or close the gripper, or more precisely to request a given gap (in percentage or absolute distance);
- returns the gripper pose for control and comparison at the environment level, including Cartesian coordinates concerning the base frame and the corresponding Euler angles;
- initialising the gripper as fully open, for safety purposes.

5.3.1.3 Objects and Manipulation Issues

We selected parallelepipedic objects for simulation purposes, mainly due to their simplicity and because they can simulate a product's parts and tools. The object is an abstraction of any object in an industrial environment, regardless of the situation or task. This object allows for grasping, insertion and stacking operations, which are simple but versatile enough to produce a reliable and accurate simulation. Differently from a cube, this object's shape introduces the orientation factor, in which the object's pose influences actions. The object's model representations are displayed in Fig. 5.6a, whose dimensions are: $0.05m * 0.05m * 0.15m$. After, the parallelepiped object can be substituted by any tool or part with little work involved in real environments.

When the gripper is asked to close its fingers around an object, the server application will not reach the fully closed position. As depicted in Fig. 5.6b, this happens due to the object dimensions preventing the fingers from closing. Consequently, the server application will not return with the **result message** meaning the goal (of no gap) was achieved. It is not desirable for the gripper to close a given distance according to the object's width, as there might be issues with the grasping or the effort applied. Even if this was possible in simulation, the 3D printed gripper could not sense the gap size, and it would not be a trustworthy problem modelling. Following the same rationale, **feedback messages** would not be realistic, even if they were available, which is not the case.

Instead of inducing whether the object is grasped or not through the effort analysis, the solution found was similar to the one in OpenAI's Gym robotic environments. It consists of comparing the difference between the gripper and the object and the finger's encoders. Moreover, it was required to grasp the object at most $2cms$ from its centre until the grasp plugin was activated. This threshold was empirically chosen and allowed the plugin to be triggered successfully, even if it slides initially. In the end, the $2cms$ guarantee that the object remains grasped, regardless of the inaccuracies in the physics parameters.

(a) Object, dimensions $0.05 * 0.05 * 0.15$ (m)

(b) Object Grasping: closing gripper

Figure 5.6: Object Manipulation

Regarding object stabilisation, the task required adjusting physical properties on both gripper's fingers and objects. The modified properties were applied to the SDF Gazebo model, specifically on the collision elements of the respective link. For the gripper fingers, the modifications took place in the urdf file where the gripper was attached to the arm and where the other Gazebo properties were specified. Following the specific tutorial [78], it was necessary to adjust the friction and contact properties by testing various combinations. The parameters hereby started by following the suggestions among the Gazebo community. However, tests like grasping and moving led us to other adjustments.

- **Adjustments for both gripper fingers and objects:**

- *Contact parameters:*

- * the k_p (dynamically 'stiffness'- equivalent coefficient for contact joints) had to be high enough to avoid object deformation, simulating rigid bodies. Results in $k_p = 10000$ (dimensionless).
 - * the k_d (dynamically 'damping'- equivalent coefficient for contact joints) had to be low enough to avoid slippage between the object and the fingers. Results in $k_d = 0.01$ (dimensionless).

- *Bounce parameters:*

- * to minimise object bounce and undesired behaviour, the restitution coefficient was set to 0 and the threshold to $1e+06$, high enough to ignore bounce whenever collisions involve small forces.

- **Adjustments only for objects:**

- *Contact parameters:*

- * the max velocity was set to $0.1m/s$, meaning that when a collision with the cube occurred, the object's velocity has a maximum of $0.1m/s$. However, according to Gazebo documentation the actual max collision speed will be the minimum between the limit speed of the intervening objects. This definition allows to achieve object's observations that are not far from the workspace when an undesired collision occurs. The observation will allow to penalise the action to avoid collisions.

– *Friction parameters:*

- * the torsional coefficient was set to 10, so that object rotations were minimised. The probability of success grasping should be increased and not influenced by small contacts between gripper and object by reducing the coefficient.

• **Adjustments only for gripper fingers:**

– *Friction parameters :*

- * both μ and μ_2 , which represent the coefficients of friction along both directions, were set to 1 (dimensionless) to avoid noisy movements when starting to grasp.

Object module

The solution proposed to deal with objects followed the same principles that shaped the gripper module's creation. Thus, the object model was flexible enough to deal with any entity, regardless of the type. The implementation is divided between objects' instantiation (at the world file) and the objects' control for the remaining operations.

First, the objects are instantiated in the corresponding world file, with the corresponding type in the name tag. When transposing the approach to the real-world solution, the objects would be recognised through a vision system and associated with their width, height and depth measurements. So, the dedicated objects' controller is responsible for:

- randomising the cube's pose (position and roll) within the manipulator's workspace, at every environment reset. In the real world, this task simulates the initial positioning of the product's component, carried out by a human operator or the surrounding automation system;
- getting the cube's id, dimensions, position and orientation at every control cycle. Again, this can be achieved, in the real world, by a vision system;
- converting the cube's information to an allowed format (like an array) to be sent over the server manager towards the environment.

5.3.2 Integration with OpenAI Gym

After preparing the simulator's low level, it is required to build the environment side: a gym environment that implements the base functions following OpenAI Gym's formatting. The base methods include environment reset, step and reward. The **reset** is responsible for preparing the environment. The **step** method is when the agent performs an action and receives the next observation. The environment provides a **reward** at every step, which helps the agent understand how valuable its action was. The Gym environment is the interface to the training algorithms and follows the specific framework to allow benchmarking and easily employing several algorithms. When adapting the environment to this simulation scenario, the goal is to craft the methods based

on the information gathered by the low-level robot server, act according to the agent and evaluate the behaviour through the reward computation.

5.3.2.1 State Representation

The first contribution to the robo-gym environment side was the state definition, which has been reported as a source of many issues by the JR Robotics team. The state arriving from the server-side was shaped as an ordered list, comprising information on the joint angles, velocities and collisions. The access to the data was performed by hard coding the indexes at which the data was stored. While the joints' order at a ROS level was specified alphabetically, that representation is not natural for humans at the environment level. Thus, dealing with the standard order made sense: from the base frame to the end-effector. The joints' state was displayed as normalised values to accomplish better results with RL algorithms.

To make data handling more straightforward, a dedicated class (`env_state`) was created:

- allows to save and access the information in a dictionary, avoiding issues with the data sorting;
- deals with the creation of the server message, to be passed via server manager and which had to respect the specified formatting;
- reads and decodes the list received from the server manager with the information regarding the surroundings' state;
- creates the observation message to be sent to the agent.

5.4 Simulation Assessment

After the scenario construction, the following assessment was performed at a high level, to **test synchronisation and validate the simulation scenario**. The task defined to check low-level functionalities and convergence was similar to the Fetch-Reach environment provided by OpenAI Gym. This intermediate step was crucial to evaluate whether the simulator could work as a reliable tool in an assembly task, avoiding synchronisation problems related to the ROS topic's exchange. It was also the first test to validate the model and evaluate how the joint space would impact the agent. The joint space was the representation used at the current time in the robo-gym.

For short, the assessment task's objective was to control the arm joints in order to position the end-effector in a pre-defined target, which was randomised at every reset.

Bearing in mind the objective of simulator validation, the reaching task was restricted to a **single controllable joint** (shoulder pan joint). The observation space comprised joint **angles**, **velocities** and the end-effector **pose**, the latter estimated using Gazebo services which can easily be substituted by a vision system in the real world. The target position was placed at the end-effector's circumference of reachable points. Thus, the simplified environment (Goal Env) would avoid collisions, and the dimensionality was reduced from six joints to one. Now, the action

space corresponds to control the shoulder pan joint between -1 and 1. The observation space included information on the shoulder joint positioning and speed, alongside the end-effector pose and distance to the goal position. The validation test components are summed in Tab. 5.2.

Task objective	control the manipulator to reach a given pose with the end-effector
Observation space	1 joint value ($joint \in [-1; 1]$); 1 joint velocity ($speed \in [-inf; +inf]$); gripper pose in cartesian coordinates (x, y, z); distance between the gripper and the desired pose, in cartesian coordinates (x, y, z)
Action space	shoulder pan joint value ($joint \in [-1; 1]$)
Reward	-1*(euclidean distance between end-effector and randomised goal); +100 if the goal was reached
Max Number of Steps	500
Environment type	gym.GoalEnv
Algorithm used	Soft Actor Critic (SAC) + HER

Table 5.2: Validation test: specifications

The training used HER combined with SAC to reproduce the results published with the OpenAI Fetch environments. Algorithms that require several simultaneous workers, like DDPG, are not compatible due to the ROS limitations of a single simulation per machine. Thus, all the decisions regarding algorithms must respect that limitation. The off-policy model chosen was SAC because it was reported to converge to success with only one worker. The model was initialised with the arguments enumerated in Tab. 5.3.

Parameter	Description	Argument
policy	Policy model to use	MlpPolicy, implements actor critic, using Multilayer Perceptrons (2 layers of 64)
model_class	Off policy model which will be used with HER	SAC
n_sampled_goal	Number of sampled goals for replay (offline sampling)	4
goal_selection_strategy	Strategy for sampling goals for replay	future
online_sampling	Sample HER transitions online	True
total_timesteps	number of episodes	1e5

Table 5.3: Training parameters on the reaching task while controlling the shoulder pan joint. The model uses HER+SAC.

5.4.1 Results

According to Tab. 5.3, the agent trained on the reach environment for $1e5$ episodes, for the validation test. The callback function computed evaluation metrics every 500 episodes. Based on that information, three curves were plotted. Those were the **success rate evolution** on Fig. 5.7a, the **number of steps** required to complete the episodes on Fig. 5.7b and the **cumulative reward**, on Fig. 5.7c. Although all curves show high variances and seem not to have stabilised, the results are meaningful. In the final episodes, the success curve shows the agent achieving a success rate of around 80% (Fig. 5.7a). The mean episode length also decreases overall, which means the agent is successfully solving the task, and thus, the episodes take fewer steps. Likewise, shorter episodes result in higher episodic rewards, just as depicted in Fig. 5.7c.

These results prove that all the pieces in the simulator work together, synchronised. However, there are two things worth mentioning: the **episode duration** and the **joint's representation**.

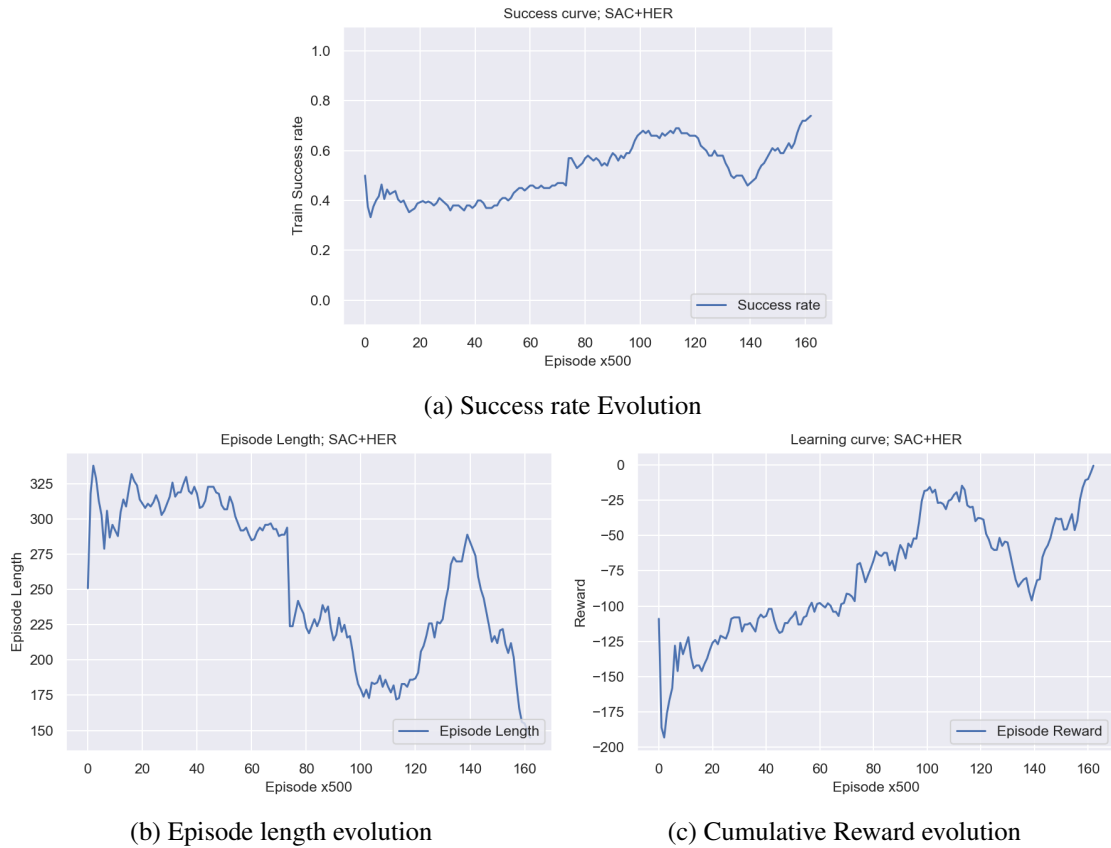


Figure 5.7: Results on the reaching task with one controllable joint

Regarding episode duration, each episode takes, on average, one minute and a half. Thus, training the agent is time-consuming, being essential to reduce the number of interactions required until convergence. Regarding the joint's representation, the agent was trained in joint space, which is translated into seven joint variables. However, that representation is high dimensional and depends on the manipulator used. Also, it is not a natural representation in space and does not

directly match the object’s localisation. For those reasons, we found it **necessary to create a kinematics model** that would easily convert between joint space (required for ROS control) and Cartesian space (only three variables). The need for that module came as a consequence of this validation test, and it is used during the next experiments. The kinematics module is responsible for calculating the homogeneous transformations between joints/links and computes forward kinematics. For the inverse kinematics, the model retrieves the joint’s combination closest to the actual states, which does not correspond to a collision state. Thus, the collision problem is simplified without restricting too much the joints on possible values (action space). The module’s advantage also resides in its flexibility. It supports any manipulator, being only necessary to fill in the joint’s offsets, the Denavit–Hartenberg parameters, and the type of joints.

5.5 Chapter Synopsis

The simulation scenario created aimed to provide a realistic environment to train and evaluate algorithms in the RL scope while making it easier to transfer the knowledge to a real-world agent. The modifications were performed both at the ROS level and at the robo server level.

In ROS and Gazebo, the modifications were related to the models and simulation parameters. First, it was required to integrate a Robotiq gripper model with a UR5 manipulator. Likewise, objects were added to the scenario to simulate tools and parts within an industrial cell. The integration and control were performed using ROS control (such as topics and services) and Gazebo. For scalability and flexibility, **on the server side**, the developments include the construction of the server-side of robo-gym in Python. There was the need to develop three modules: the **gripper module**, the **object module** and the **kinematics model**. All three help the user by adding an abstraction layer that deals with the implementation details and allow reconfiguration.

The reaching test allowed us to validate basic functioning and synchronisation of the simulation scenario. It took advantage of the published results of SAC+HER in the Fetch environment from OpenAI’s Gym. As those results were reproduced with this environment, the validation test was completed with success. It was also possible to identify the **most significant difficulties**: the training time constraints and the limitation of one worker per rollout.

In terms of **contributions**, these low-level developments include the pull request in the Robotiq repository, the three control modules, the simulation environment for object manipulation and the data structuring. The contributions related to the simulation scenario are available at the [GitHub repository](#) and were proposed to the corresponding repository owned by the Institute for Robotics and Mechatronics at the JOANNEUM RESEARCH in Austria.

Chapter 6

Environment Description and Training Setup

Following the simulator validation, the next step is to evaluate whether the agent can execute the proposed case study, which differs from the typical pick and place task. The environment includes a UR5 manipulator with a gripper attached to it and one object put down, according to the description in Chapter 5.

This chapter presents the differences between the Fetch environment (already available with OpenAI Gym) and the proposed environment (which uses Gazebo physical engine instead of MuJoCo). After, there is a description of the observation and action spaces. At last, a description of the algorithm chosen and the setup used for training are given.

6.1 Differences to the Fetch Environment

The Fetch environment, specifically the *FetchPickAndPlace-v1* [79], proposed by OpenAI, was the starting point for building our observation and action spaces. Regarding the *FetchPickAndPlace-v1*'s **observation space** and the conclusions in Chapter 5, the manipulator representation was changed from joint space to Cartesian space. The translation between spaces is achieved with the **kinematics module** implemented after the validation test (Section 5.5). That modification reduces from seven variables to three coordinates. The central point between the fingers represents the manipulator pose to simplification, similar to the Open AI approach. However, some observation variables displayed in the Fetch environment are not directly available in Gazebo. Among those variables, the object's absolute and relative velocities are not realistic to implement in real scenarios. Those speeds might be hard to capture and are associated with cumulative uncertainties from jitter, sensors and the vision system. Moreover, if the solution is to be implemented in the real world as straightforward as possible, our goal was to simplify integration procedures, not adding new challenges to the vision systems. On the other hand, they increment the spaces' complexity, which impacts training performance profoundly.

Regarding the **action space**, the gripper control cannot deal with partially open or closed poses, as explained in Section 5.3.1.2. The gripper orientation is also fixed, pointing down (opposing to z axis) to ease the pick and place. However, as the object is not cuboid, and the object placing requires a particular orientation. For that reason, the agent must control the gripper roll angle (ϕ). Then, the rotation matrix ($R_{gripper}$) to be used by the kinematics module (Section 5.4.1) is given by Eq. 6.1. The ϕ represents the angle between the end-effector's x axis and the gripper's normal vector, according to Fig. 6.1. The angle ϕ is measured in the direct direction, namely counterclockwise.

$$R_{gripper} = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ \sin(\phi) & -\cos(\phi) & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad (6.1)$$

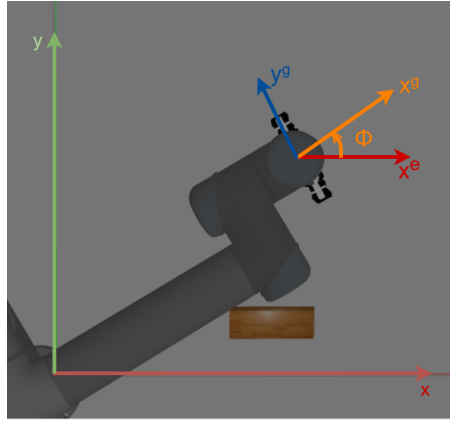


Figure 6.1: Definition of the angle ϕ , representing the gripper orientation. The red axis represents the x -axis in the end-effector frame. ϕ is the angle between the x^e axis and the gripper's normal axis, x^g . The fingers open and close, following the y^g direction.

In terms of the environment's class, it is not coded as a `GoalEnv` like the Fetch environments because there is no need to display the `achieved_goal` and `desired_goal` in a *dictionary*, alongside the observation. That would be the case if the algorithm invoked the `HER` methodology. However, it can be done as future work, calling the respective methods to retrieve the data and then encapsulating them in a dictionary with the actual observation. Thus, the class used for the environment is the simple `gym.Env` [80], which is compatible with the training algorithm employed.

6.2 Observation Space

The observation space is defined as a OpenAI Gym Box Space [81], a continuous space of $n = 11$ variables each of them belonging to a clear interval, whose upper and lower bounds are known. Each observation space component is explained in Tab. 6.1:

Notation	Description	Formatting	Max Values	Min Values
$X_{gripper}$	real gripper pose (m)	(x_g, y_g, z_g)	(1.35, 1.35, 0.40)	$(-0.1, -1.35, 0)$
Δ_d	object to target (m)	$(\Delta x, \Delta y, \Delta z)$	(2.70, 2.70, 2.70)	$(-1.35, -2.70, -2.70)$
X_{object}	object pose (m)	(x_o, y_o, z_o)	(1.35, 2.70, $+\infty$)	$(0, -1.35, 0)$
c	gripper state	float	1	0
g	grasping	float	1	0
f	fit	float	1	0

Table 6.1: Observation's space description for the modified pick-and-place case study

The **real gripper pose** ($X_{gripper}$) is represented as Cartesian coordinates in the base frame. Gripper pose's limits are related to UR5's arm length (approximately 0.85m) plus a tolerance (0.5 m) that considers the gripper dimensions. Regarding max height, 40cm was considered a good trade-off between a small enough space to reduce the agent's exploration and being big enough to perform tasks. The minimum values discard the workspace for negative x values. This lower limit allows to reduce once again the complexity associated with large spaces, reproduces the industrial environment where the workspace would be restricted to a workbench, and tries to mitigate the disadvantages of using a single continuous box space. Simply put, the gripper workspace is spherical, but the Gym space has to be cuboid. Thus, the cuboid space should fit inside the workspace and avoid singularities or unreachable points like the robot's base links. The graphical representation of the observation space and the robot's workspace is depicted in Fig. 6.2.

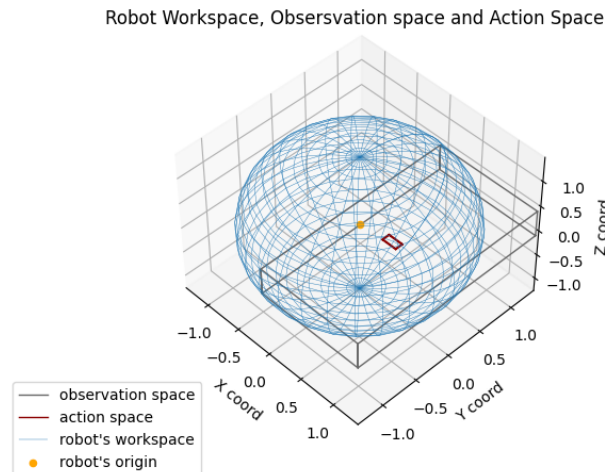


Figure 6.2: Gripper real pose on Observation and Action spaces, compared to robot's workspace

The **object to target** component (Δd) is the distance vector between the real object position

and the defined goal pose (Eq. 6.2). It can be twice the gripper range, at most, for the case where both object and goal are at the robot's workspace edge with the maximum possible separation.

$$\Delta_d = (\Delta x, \Delta y, \Delta z) = X_{goal} - X_{object} \quad [\text{m}] \quad (6.2)$$

For consistency purposes, the **object pose** (X_{object}) is represented as Cartesian coordinates, in relation to the world frame. One alternative representation would be the object pose in the gripper frame (computed as the difference of those poses). Still, that option would introduce space discontinuities when changing from pick to the following stages, i.e. that modelling option would demand changing the variables again to the cube pose or the difference between the goal and the object. On the other hand, it would not correctly represent the task's objective: moving the cube instead of the gripper. Furthermore, the model could not identify and overcome errors related to deviations in grasping configuration. For example, picking the object slightly further and placing the gripper at the same goal does not result in the same cube to goal distance, not the same reward and would lack reproducibility. This design choice is another difference to the Fetch environment.

The **gripper state** and the **grasping** variables give information on whether the gripper is open/closed and if the gripper is grasping an object, respectively. The gripper is classified as open if the gap between fingers is 75% – 100% of the maximum distance (for the modelling to work out with any gripper model). It is classified as closed otherwise. The fingers slide according to the y^g in Fig. 6.1, opening or closing. The gripper must be closed at most 2cms from the object centre to classify an object as successfully grasped. On the one hand, the 2cm threshold was chosen to guarantee that it remains grasped even if the object slides initially. More details on that were provided in Section 5.3.1.3. On the other hand, that assures the object remains parallel to the floor when grasped, resulting in easier fit and place. Although both variables are boolean, that is not compatible with the other continuous variables in the box space. Thus, they were coded as float variables with limit values being 0 and 1. However, those variables do not take intermediate values because the controller is as general as possible and independent of any vision system, implying that partially open states are not supported (Section 5.3.1.2). The gripper and grasping states represent, respectively:

- Gripper state c : 0 means open gripper and 1 closed gripper.
- Grasping state g : 0 means the object is not grasped and 1 otherwise.

The last variable f is denominated **fit**. In this situation, the agent must learn how to place the object with the correct orientation. This variable explains the overlap between the desired (goal) area and the current (object) area, projected on the xy plane. Suppose the information on the height is important, as stacking operations, the fit measure can correspond to the volume. The overlap is given as a percentage, where 0 means no overlap and 1 corresponds to fully coincident areas. The example of Fig. 6.3 represents the situation where the object overlaps the goal by 50%. The fit representation is particularly useful when one product is made of several parts. This variable can be adapted to other tasks the robot might be required to perform. For example, in tasks like screwing or insertion, this information can translate the percentage of completion.

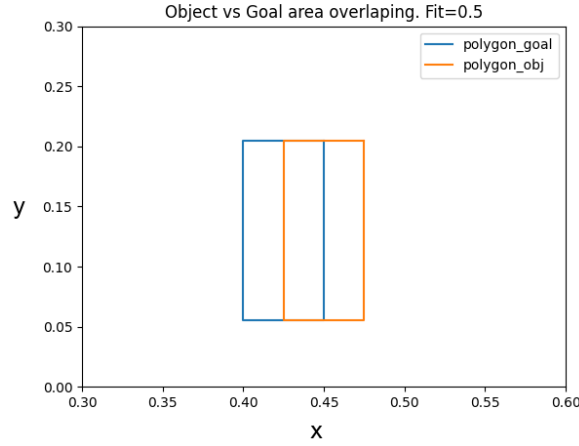


Figure 6.3: Fit Measure where the object (orange) overlaps half the goal (blue), $f = 0.5$

6.3 Action Space

The action space comprises the desired gripper pose, the gripper state (open or closed) and the wrist orientation. The first three variables can be grouped as the **desired gripper pose**. When controlling $Y_{gripper}$, the agent will directly cause the real gripper to move ($X_{gripper}$). However, the pose variation will not be instantaneous: it is influenced by the joint's max speed ($\pi rad/s$) and the [PID](#) (Tab. 5.1). In practice, the [PID](#) controller means that the gripper pose will be less sensitive to minor variations. Additionally, if the order sent is constantly oscillating between two points, the gripper will remain around a central position.

The following action variable is the **desired gripper state**: it is the action to open or close (y_c) the gripper and works as the gripper state c in the observation space. Likewise, that boolean variable is mapped into a continuous interval as a consequence of the Box space. Suppose further developments allow dictionary spaces to work with the algorithm implementations. In that case, choosing the appropriate variable types to match the model would avoid unnecessary exploration regarding possible values between 0 and 1.

The last action variable is related to the **wrist orientation**, perpendicular to the floor. While the object is not grasped, the orientation is automatically computed to be perpendicular to the object. After moving the object, the robot must choose the correct orientation based on the information given by the fit in the observation (f). In this case, the object is symmetrical, and thus a variation of π is enough to cover all overlapping situations. The detailed information on the action space is provided in Tab. 6.2. The spatial representation is plotted in red in Fig. 6.2.

Notation	Description	Formatting	Max Values	Min Values
$Y_{gripper}$	desired gripper pose (m)	(x_d, y_d, z_d)	(0.5, 0.15, 0.05)	(0.3, 0.05, 0.03)
y_c	desired gripper state	float	1	0
ϕ	wrist orientation (rad)	float	π	0

Table 6.2: Observation's space description on the picking task

For the reaching task, the gripper state is fixed as open. Controlling the gripper does not impact the success of this task and speeds up convergence. The action space gripper state at this task is, for all steps, given by Eq. 6.3:

$$y_c(t) = 0, 0 \leq t \leq M \quad (6.3)$$

6.4 Environment State on Reset

Beyond observations and actions, the Gym framework defines the environment reset. It corresponds when the environment goes back to an initial state and the agent tries to solve it again. The reset happens after any termination step, which include:

- successfully completing the episode and achieving success;
- reaching the maximum number of allowed steps during an episode;
- any other situations considered dangerous and properly identified, like collisions.

On reset, the manipulator and the object take their initial poses. The manipulator starts in an "elbow up" configuration to favour the pick and place task while avoiding collisions with the floor, the object or itself. The gripper starts at $X_{gripper}(t_0) = (0.63, 0.10, 0.34)$, which is a point not too close or too far away from the object. This point is outside the action space to show the agent the impact of the trajectory on the task's success and separate the "resting" from the "working" poses.

The object starts on the floor, with the biggest base facing down. The object's central point is at $X_{object}(t_0) = (0.40, 0.10, 0.025)$. It starts parallel to the base x-axis. However, any orientation is compatible as the gripper automatically assumes a perpendicular orientation for grasping. After picking the object, the agent controls the orientation ($\phi(t)$). The goal pose is at $G = (0.45, 0.13, 0.025)$ with the orientation of $\theta = \frac{\pi}{4}$.

6.5 Training Algorithm Setup

The [GAIL](#) [42] algorithm (classified as [IRL](#)) trained the agent on different rewards and sub-tasks. This methodology aims to overcome the programming burden of tasks prone to uncertainty, like non-reproducibility in object's poses. It also aims to develop more flexible algorithms that could ease the programming required to change between tasks or different products. The modelling procedures here gathered can be viewed as a set of guidelines to solve problems made of sub-tasks and tricks to speed up convergence and model recycling.

As explained in Chapter 2, this [IRL](#) algorithm combines [RL](#) techniques with an expert, shortening exploration by directing the agent towards the desired actions following the demonstrations provided. The directed exploration is accomplished by having the agent acting similarly to the experts during the first episodes. This algorithm allows the agent to learn the **expert's behaviour** by training two neural networks as rivals. Despite [GAIL](#) being part of the [IRL](#) world, it shares the requirement of a crafted reward function with [RL](#) methods. The [GAIL](#) implementation used

was provided by the *HumanCompatibleAI* repository [82]. The specific training parameters can be found in Tab. A.2 (Ap. A), which were based on the results and specifications presented by Ding *et al.* [55]. The training ran remotely and required two computers. The specifications of both machines are summed up in Tab. A.1 (Ap. A). The robot simulator comprising Gazebo and ROS ran on a laptop, in which the low-level simulator was developed.

The IRL algorithm executing on the desktop used the NVIDIA GeForce RTX 2080 Ti Graphics Processing Unit (GPU), which helped reduce the training duration. The desktop had enough memory to handle the nets and the train. Nonetheless, as the robo-gym simulator was being refactored, the simulator version on the desktop was different. Consequently, the simulations were not as smooth and reliable as in the laptop. The laptop itself had not enough memory to handle the training. The solution was to connect both machines to a local network, run the server manager on the laptop (alongside Gazebo and ROS), and perform the GAIL procedures on the desktop. The disadvantage was the network dependency, which would cause the training to fail if one of the machines had connectivity issues.

6.5.1 Expert Demonstrations

Part of the IRL procedures involves recording expert demonstrations. When using GAIL from the corresponding repository, the typical approach includes pre-training a policy with a simpler algorithm such as PPO and then feeding some generated trajectories to the IRL algorithm [83]. Our approach **does not involve pre-training an agent**. Instead, the demonstrations are recorded manually, *i.e.* the scripted procedures decide the next gripper pose traditionally. The only requirement for the demonstration to be compatible with the implementation is to be stored as a `TrajectoryWithRew` [84]. By manually recording the desired trajectory in robotics, we avoid spending time training another model while reusing the expertise of a human operator. This data type includes 4 `ndarrays` for an entire expert episode lasting N steps:

- the **observation** array with $N + 1$ elements: the reset plus N trajectory observations;
- the **action** array with N elements encoding the gripper’s pose and state for each step;
- the **reward** array with N elements evaluating the agent’s performance;
- the **info** array with N elements for additional debug

Among the total demonstrations, ten were chosen, which is close to the number of demos for the Half-Cheetah environment, reported by Ho and Ermon [42]. That environment is comparable as the maximum number of steps is 500 and both observation and action spaces are continuous with dimensions 17 and 6, respectively. The best results in [42] were achieved with seven expert demos. In this case, ten demos were used as the problem includes interaction with a sensitive external object. The sets of ten demos were recorded for each test, to use the correct sequence of tasks and corresponding rewards. Each set is analysed when the tests are presented.

6.6 Chapter Synopsis

The differences between the proposed environment and OpenAI’s Fetch relate to the integration with Gazebo and the real-world implementation. The **observation space** includes fewer variables for simplicity and used the Cartesian representation instead of the joint space, which was the previous implementation provided by robo-gym. The **action space** does not allow a partially open gripper configuration. Although both observation and action spaces were designed for the pick and place task, we consider those universal enough to be applied in different robotics tasks with minor adjustments. Furthermore, the described variables are more connected with the robotic tasks developed in real-world scenarios as that information is simpler to acquire. Table 6.3 sums up the variables used during the task.

	Notation	Description
Observation space	$X_{gripper}$	real gripper pose (m)
	Δ_d	object to target (m)
	X_{object}	object pose (m)
	c	gripper state (0-1)
	g	grasping (0-1)
	f	fit (0-1)
Action space	$Y_{gripper}$	desired gripper pose (m)
	y_c	desired gripper state (0-1)
	ϕ	wrist orientation (rad)

Table 6.3: Observation and Action spaces summed up

The algorithm used to train the agent on the task was the [GAIL](#), a methodology from the [IRL](#) family. Not only does the agent become more robust to uncertainties and flexible to task’s variations. It also takes advantage of the demonstrated behaviours and tries to optimise those. The **expert trajectories** were recorded manually, which means those can be achieved without training a previous model. Instead, there can be human interaction, which makes it faster to acquire successful behaviour.

During training, the episodes were recorded, and we kept track of the observations, actions and rewards for each step. Those logs were analysed afterwards for performance analysis and comparison.

Chapter 7

Reach Sub-Task

The solution was proposed in Chapter 5 alongside the low-level implementations regarding the simulation scenario. So, after defining the observation/action spaces and the algorithm specification (Chapter 6), we have all the pieces to start testing. According to the case study in Section 5.1, the first identified sub-task of the modified pick and place is the reaching.

The **reaching** task consists of positioning the gripper near the object at a specific distance and orientation. It is the first sub-task and, thus, the simplest of all. After starting from the reset position (described in Section 6.4), the agent should **guide the gripper down towards the object** until the gripper's fingers are placed around it. The final pose is depicted in Fig. 7.1. The action space was **restricted** at this stage, meaning the gripper was set as open, and thus the agent did not control its state ($Y_c = 0$).

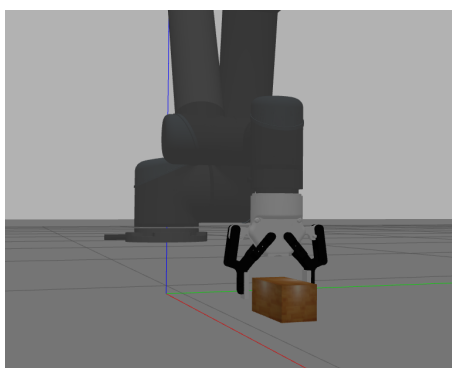


Figure 7.1: Sub-task 1 - Reaching the object

This chapter starts by explaining the proposed reward function and the expert trajectories' recording. Then, we present and analysed the results, which consider the success rate, the discriminator's accuracy and the net losses.

7.1 Reward Function

The reward function for the reach model is **dense**, which means the agent receives **feedback** on the performance at each step. This reward choice tries to **speed up convergence**, as the training duration is the critical resource. The reward function will receive further developments as new sub-tasks are introduced and have two components. The **reward for reaching the object** R_r is assigned when the agent positions the gripper at most 2cms from the object's centre. The E_{g-o} is the **default reward** that reflects the gripper error. According to Eq. 7.1, this value is the difference between the gripper's central point and the object's centre of mass. The E_{g-o} value is always negative, implying that while the gripper is away from the object, the task is far from completion. This component is smooth throughout the steps compared to the additional rewards and penalties, making it easier for the agent to find a gradient.

$$E_{g-o}(t) = -|X_{gripper}(t) - X_{object}(t)| \quad (7.1)$$

The complete reward equation is defined for each step (t) according to Eq. 7.2. There, t_0 is the initial step, M is the maximum number of steps for a given episode, S_o is part of the observation space which corresponds to the object pose (Tab. 6.1, Eq. 7.3), γ_r indicates whether the gripper reached the object successfully at that step t (Eq. 7.4). The reward γ_r can only take the value 1 at most once per episode and makes it the last step.

$$r(t) = \begin{cases} R_r * \gamma_r(t), & \gamma_r(t) = 1 \cap X_{object}(t) \in S_o \\ E_{g-o}(t) + P_c, & otherwise \end{cases} \quad (7.2)$$

$$(0, -1.35, 0) \leq X_{object}(t) \leq (1.35, 2.70, \infty) \quad (7.3)$$

$$\gamma_r(t) = \begin{cases} 1, & |X_{gripper} - X_{object}| \leq 0.02 \\ 0, & otherwise \end{cases}, \forall \sum_{\tau=t_0}^M \gamma_r(\tau) \leq 1 \quad (7.4)$$

In this sub-task, the maximum number of steps an agent explores the environment is set to $M = 500$. That means the episode will stop in two situations: the agent successfully solved the episode or the M was exceeded. The cumulative rewards depend on the episode length, the occurrence of success, and the gripper error.

7.2 Building Expert Trajectories

According to Section 6.5.1, the reaching test required recording ten demonstrations. Each expert episode included $N + 1$ observations, N actions and N rewards. Next, we explain each demonstration's characteristics and the rules behind each action and analyse the statistics regarding the number of steps (N).

The reach expert demonstrator includes two sub-tasks, namely **approaching** and then **positioning** the gripper at the object, as depicted in Fig. 7.2. The approaching phase is critical to ensure the object is not pushed away by the gripper following a lateral trajectory and to maximise grasping success due to the object's parallelepipedic shape. By dividing the sub-task into two, we are able to introduce diversity into the demonstration set, as explained next.

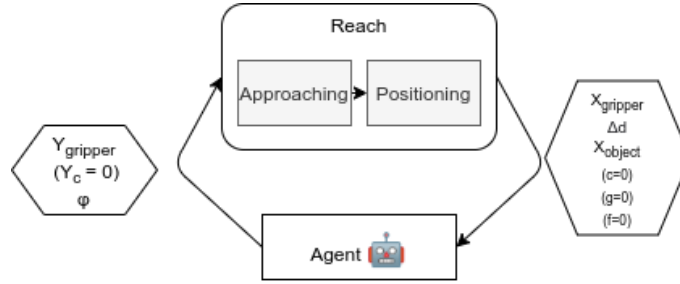


Figure 7.2: Reaching the object: approaching and positioning sub-tasks. The action space is represented in the left diamond box and the observation space in the rightmost one. During this task, the gripper is open to simplify all training procedures and speed up convergence

The **approaching phase** is about positioning the centre of the gripper right over the object's top face ($2 * z_o$) and near the centre of mass, with some noise added to bring variance to the demonstrations. Being $X_{object} = (x_o, y_o, z_o)$ the object pose on the world frame, the approaching pose is computed according to Eq. 7.5, where x_r and y_r are random samples from a continuous uniform distribution in the interval $[-1, 1[$ and z_r is a random sample from a continuous uniform distribution in the interval $[0, 1[$. The agent is not required to replicate the approaching phase as there is no extra reward modelled for that case. However, the agent must choose a trajectory from above, similar to the demonstrated, given the reasons previously mentioned.

$$Y_{gripper} = X_{object} + (0, 0, z_o) + (x_r * 0.05, y_r * 0.025, -z_r * 0.01) \quad [\text{m}] \quad (7.5)$$

After completing the approach sub-task, it follows the **positioning**. The reaching point is the same for all demos: the object's centre of mass. This aims to teach the agent to correlate the action with the object pose (X_{object}) displayed in the observation space. It also stresses the need for precise positioning to the object slip the least. The positioning is considered complete when the gripper is less than 2cm from the object's centre.

Twenty different **trajectories** were recorded for different approach positions. It is important to mention that even with little noise in the approaching pose, there were situations where the agent could not complete the task. Those were discarded, proving the need for robust solutions that

could overcome issues like noise in the encoders or object shifting. The episodes lasted between 40 and 54 steps, with a mean value of 48 steps, according to Fig. 7.3.

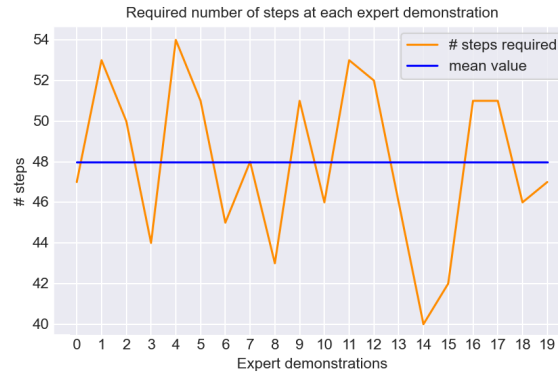


Figure 7.3: Required number of steps for expert demonstrations on the reaching task

Demonstration 7 exemplifies one of the expert trajectories, where the number of steps equals to the mean value. That episode is depicted in Fig. 7.4, where the gripper trajectory is plotted in blue. The two phases are clear: the first segment for the approaching steps and a smaller part for the positioning.

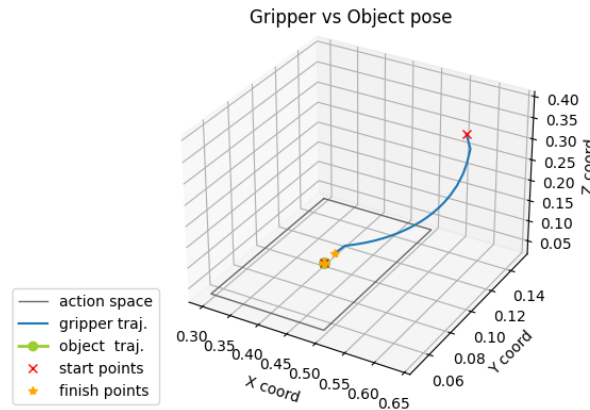


Figure 7.4: Expert demonstration 7. The blue curve represents the gripper trajectory, starting on the red cross and finishing on the yellow star. That gripper's trajectory includes 2 phases: the approaching phase (Eq. 7.5) and the positioning, defined by the smaller bit of the curve. The object does not move because its respective start and finish points are the same. The grey line represents the action space's limits which contain the valid gripper points ($Y_{gripper}$).

7.3 Experiments and Results

The agent easily learned the first sub-task of reaching the object. The several values tried for the reward R_r behaved similarly, and thus the choice was conditioned by the subsequent sub-tasks. That justification will be detailed in Chapter 8. For this experiment, the reward for reaching was $R_r = 5$, and the corresponding equation is in Eq. 7.6.

$$r(t) = \begin{cases} 5 * \gamma_r(t), & \gamma_r(t) = 1 \cap X_{object}(t) \in S_o \\ E_{g-o}(t) + P_c, & otherwise \end{cases} \quad (7.6)$$

According to Fig. 7.5, the agent accomplished the sub-task with a success rate of over 80%. The green points represent every episode the object exits the observation space. When the green points are more concentrated, as around episode 500, the success decreases as a consequence. Those episodes are associated with lower cumulative rewards, on average, because the gripper error is much greater. Those cumulative rewards are plotted in Fig. 7.6. The successful episodes correspond to overall rewards around 0, and the learning curve is smooth over episode 500, suggesting model convergence.

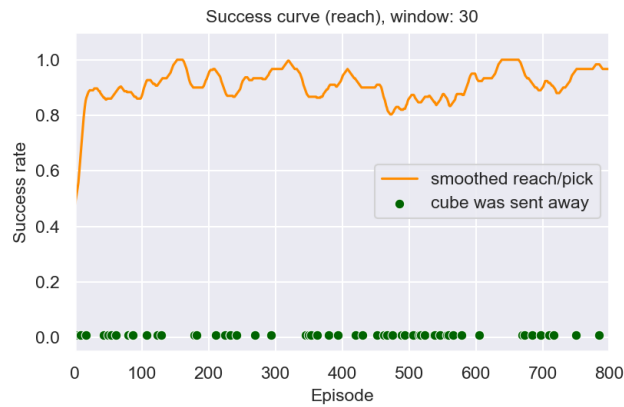
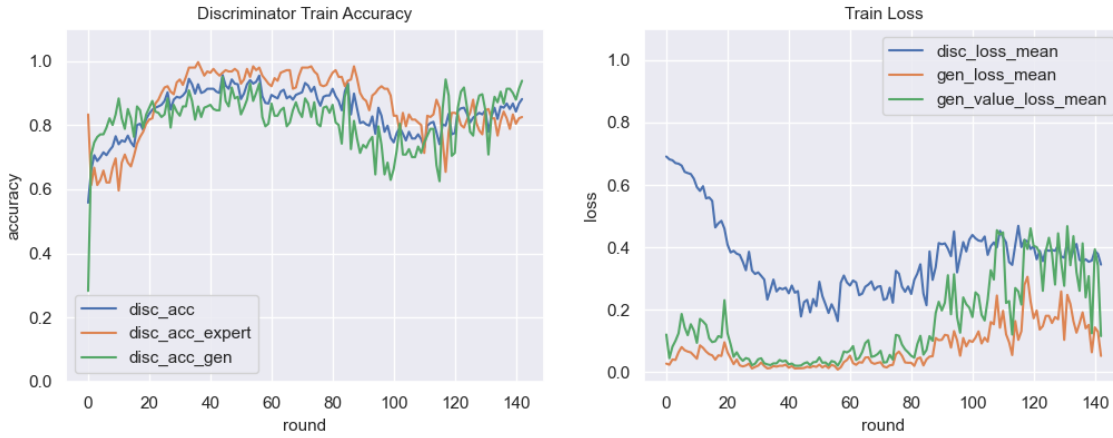


Figure 7.5: Success rate curve for reach agent, with ten expert demos. The model uses the distance (negative) between the gripper and the object as a dense reward. Additionally, there's a bonus of +5 when the gripper reaches the cube. Whenever the object exits the observation space, the episode ends with the corresponding reward as the negative distance between gripper and object.

Fig. 7.7 shows the behaviour of the networks, both the discriminator and the generator. In those graphs, each round corresponds to one generator training plus four discriminator rounds. Regarding accuracy (Fig. 7.7 left), the discriminator starts by misclassifying the expert demonstrations (false negatives). After round 20, the accuracy increases and remains over 0.8. The losses (Fig. 7.7 right) also decrease after round 20. This shows the model has become stable. The discriminator loss remains over 0.2, but that is the desirable behaviour, Section 2.4.1 explains.



Figure 7.6: Reach - Evolution of the cumulative reward per episode.



(a) Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 1

(b) Discriminator loss, Generator loss and Generator value loss, for Reward proposal 1

Figure 7.7: Train Accuracy and loss, for Reward proposal 1.

7.4 Chapter Synopsis

This chapter presented the first sub-task, which will be the base for the developments with the following three sub-tasks (pick, move and fit). The reaching training aimed to teach the agent how to approach the object with the gripper correctly. The fingers were stuck open, which eases the task and forces the agent to perform the trajectory from above.

The reward function presented tried to create a gradient given by the gripper error, E_{g-o} . The reward for reaching is assigned if the gripper error is less than 2cms . This requirement maximises the success of grasping by minimising the consequences of object sliding or difficulties in achieving the next sub-goals.

The set of demonstrations differ on the approaching pose, which was randomised in every episode. Despite the quick convergence, the model proves to have achieved stable behaviours. That was proven by the learning curves, the discriminator's accuracy and the network's losses.

Chapter 8

Pick Sub-Task

After learning the trajectory towards the object and its positioning between the gripper's fingers, the next step is to understand grasping. Concerning the previous reaching procedures, the picking task requires understanding more variables in the observation space (c, g) and the action space (y_c). Also, reaching and picking should be performed sequentially, making it plausible to divide the task into two sub-goals. Despite being the second task, it is considered the most difficult because the interaction between the gripper and the object is critical and has a tremendous impact on the assignment's success. This will be evident in the following reward function study.

First, the agent must **learn the mapping** between the continuous values of the action space variable y_c (Tab. 6.2) and the boolean representation of **open/closed** fingers. Secondly, the agent must understand that closing the fingers **too early** might make it fail the episode, either by preventing the gripper from reaching the object or projecting it out of the observation/action space. At last, the agent must grasp the object at most 2cms from its centre ($|E_{g-o}| \leq 0.02$) to keep balance when moving, avoid slippage and improve object placing. The **observation** and the **action spaces** are the same for this task, whose definition can be found in Sections 6.2 and 6.3. At the end of this task, the agent is expected to grasp the object according to Fig. 8.1.

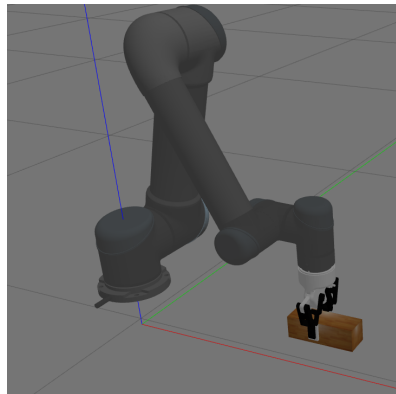


Figure 8.1: Sub-task 2 - Picking the object

We start by presenting how the reward function has been upgraded and which parameters are tested in the scope of the reward function. Then, it is explained how the expert trajectories were

recorded, regarding how they differ from the reach demonstrations. Next, the results are presented, divided into two categories: the first approach trains the model from scratch, and the second reuses the pre-trained models regarding the reaching sub-tasks. After, we present the advantages of using a sub-goal methodology, when compared to the traditional reward modelling.

8.1 Reward Function

The reward function for the reach+pick model derives from the reaching one (Eq. 7.2). Besides the reward base (the gripper error E_{g-o}) and the reward for reaching (R_r), we added four new components, whose impact we studied in Section 8.3. Those new reward components are:

- R_p , the reward for **picking** the object by closing the fingers and grasping the object at most $2cms$ from its centre. If at a given episode the agent performs the reach and the pick tasks simultaneously, the reward assigned is $R_r + R_p$;
- P_o , penalty if the object **exits the observation space** due to a collision with the gripper. This component aims to discourage trajectories that would cause dangerous situations like objects being projected into the air or exiting the industrial workstation;
- P_m is assigned when the object **moves** over $2cms$ from the reset position, and the episode ends at that step. This penalty tries to avoid situations where that would create undemonstrated situations;
- P_c is a penalty related to sending $Y_c = 1$ too earlier. It tries to demonstrate that **closing the fingers before reaching the object** will act against optimisation and might even cause undesirable and undemonstrated situations such as object sliding or collisions.

The complete reward equation is defined for each step (t) according to Eq. 8.1:

$$r(t) = \begin{cases} P_o, & X_{object}(t) \notin S_o \\ P_m, & |X_{object}(t_0) - X_{object}(t)| > 0.02 \cap X_{object}(t) \in S_o \\ R_r * \gamma_r(t) + R_p * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ E_{g-o}(t) + P_c, & otherwise \end{cases} \quad (8.1)$$

Where: t_0 is the initial step, $M = 500$ is the maximum number of steps for a given episode, S_o is part of the observation space corresponding to the object pose (Tab. 6.1, Eq. 7.3). The γ_r indicates whether the gripper reached the object successfully at step t (Eq. 7.4), γ_p takes the value 1 when the grasping has been successful (Eq. 8.2). To avoid repetitive reaching and grasping, both γ_r and γ_p can only take the value 1 at most once per episode. Without this constraint, the agent might

repeat those actions to accumulate reward, without completing the episode, compromising the task's optimisation and completion.

$$\gamma_p(t) = \begin{cases} 1, & |X_{gripper} - X_{object}| \leq 0.02 \cap g = 1 \cap \sum_{\tau=t_0}^t \gamma_r(\tau) = 1 \\ 0, & otherwise \end{cases}, \forall \sum_{\tau=t_0}^M \gamma_p(\tau) \leq 1 \quad (8.2)$$

The mean cumulative rewards depend on the function gains, penalties and rewards. However, the main events are associated with more significant variations between timesteps to stress the actions' impact.

8.2 Building Expert Trajectories

For the expert demonstration on the pick task, the datatype is also `TrajectoryWithRew`, which allows compatibility between models and the algorithm. The demonstration includes the approaching and positioning tasks, like previously. However, we added the gripper's state control (Y_c) and the object grasping at the end. Although the demonstration performed the reaching and picking sequentially, it is accepted if the agent performs those simultaneously.

The agent can now control the gripper's closing/opening and pose, as evidenced in Fig. 8.2 at the left. The environment's information represents the gripper pose ($X_{gripper}$), the distance to the target (Δd), the object pose (X_{object}), the gripper state (c) and the grasping state (g). The fit variable f is not fixed but will not be accounted for in the reward at this stage.

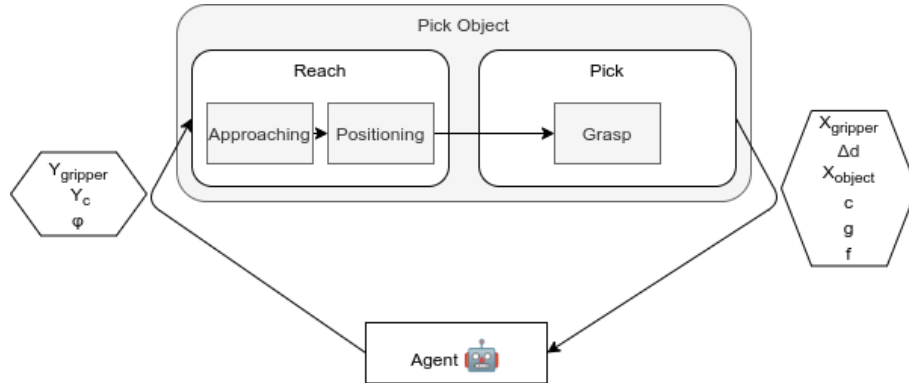


Figure 8.2: Picking the object - reaching and picking sub-tasks. The action space is represented in the left diamond box and the observation space in the rightmost one. The agent controls the gripper's closing/opening, differing from the previous study.

Similarly to the previous demos, the approaching pose has some random noise and follows Eq. 7.5. The situations where the object moves from the reset pose are not demonstrated. However, it can happen as the object is pushed or carried by the gripper. There was the need to record new sets of demonstrations every time the reward gains changed. So, for each combination of parameters, there were recorded ten different expert demonstrations. All of those demonstrations

differ on the approaching point, like for the reaching task. Specifically for the reward functions 1 to 4 (Sections 8.3.1.1 to 8.3.1.4), the number of steps per expert episode range from 44 to 59. According to Fig. 8.3, the episodes lasted on average 49 steps, aligned with the reaching task (Fig. 7.3).

The difference between the episodes' length at each demonstration in Fig. 8.3 shows the impact of the different approaching poses and the joint's PID. The trajectory number 8 is depicted in Fig. 8.4, for exemplification. The gripper finishes the trajectory closer to the object's centre of mass, as both final points overlap.

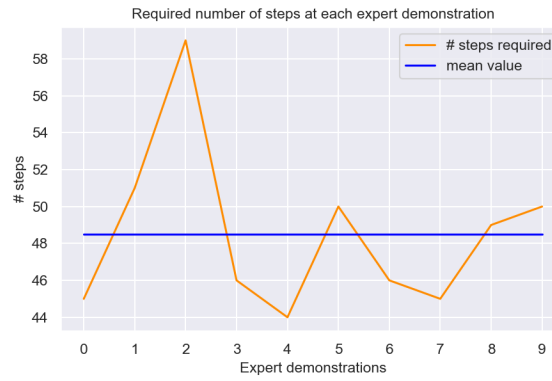


Figure 8.3: Required number of steps for expert demonstrations on the pick task

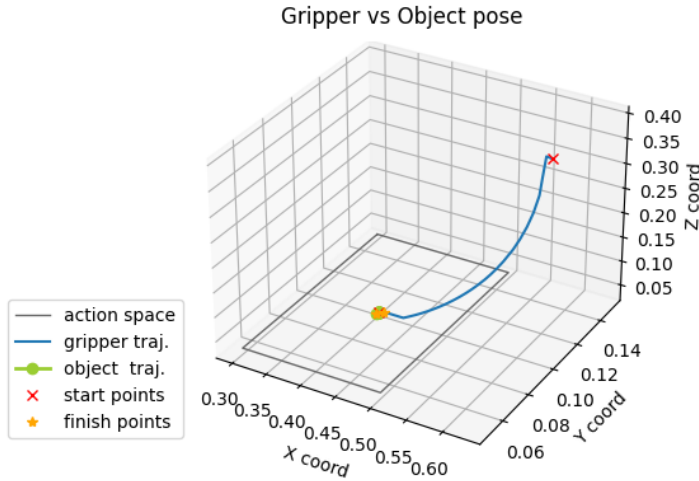


Figure 8.4: Expert demonstration 8 on the pick task. The blue curve represents the gripper trajectory, starting on the red cross and finishing on the yellow star. That gripper's trajectory includes 2 phases: the approaching phase, according to Eq. 7.5, and the positioning. After that, the gripper closes, and the trajectory is complete. The object does not move because the green points coincide for the entire episode and its respective start and finish points are the same. The grey line defines the action space's limits, where the valid gripper points ($Y_{gripper}$) belong.

8.3 Experiments and Results

Following the generic reward function definition, we proceed with the experiments. For short, the tests tried to compare the two main proposals with the traditional [RL/IRL](#) solutions. The first comparison tries to understand the performance of sequential modelling versus the **task discretization**. The second analysis studies the **impact of reusing** the generator (PPO model) and the discriminator models.

For the first analysis regarding task **discretization**, two types of tests were conducted. The **first batch of tests** trained the **agent from scratch** with seven different combinations of the reward function parameters (Eq. 8.1). A specific group of demonstrations were required every time the reward function changed and the corresponding analysis is carried on in Section 8.3.1. All of the results were compared and led up to the best parameters combination choice. The second test involved training an agent without dividing into reaching and picking sub-tasks. That test used the parameters chosen previously and adds information to the impact of discretization.

For the **second analysis** on the impact of model re-usage aims to understand how **flexible** can this approach be, when changing production configurations. So, the training performance is compared to the situation where the agent knows *a priori* how to perform the previous sub-task. This will validate the approached model hierarchisation.

8.3.1 Training from Scratch

In this section, the agent must learn to pick the object **without any previous knowledge** of approaching or reaching the object. There are compared variants of the reward function defined at Eq. 8.1, looking for the best-suited combination of parameters. The corresponding seven tests are described in the following sections, which led up to the parameter choice based on the best agent's behaviour. For comparison purposes, there were used **10 expert demos** for each reward function.

8.3.1.1 Reward Function 1 - First Approach

Recalling Eq. 8.1, the first reward proposal stresses how reaching and picking the object is different from randomly moving the gripper, by assigning high rewards to those achievements. If the object exits the observation space, the episode ends with P_o as the final reward. No additional penalties are used at this stage, then P_m and P_c do not apply here. The equation parameters chosen for the first tests are given in Tab. 8.1. The reader can find the figures depicting the episodes in Appendix B.

Both R_r and R_p had to be great enough to contrast with the negative rewards (which correspond to normal gripper movements) to make it evident for the agent what were the goals. Thus, arbitrary large rewards were chosen: $R_r = 100$ and $R_p = 150$. The penalty P_o had to be lower than

Parameter	Value
k_{pick}	1
R_r	+100
R_p	+150
P_o	-10
P_m	(NA)
P_c	(NA)

Table 8.1: Proposed parameters of Eq. 8.1 for Reward Proposal 1

the exploratory moves to avoid dangerous behaviours, namely the objects exiting the observation space. By substituting the parameters on Tab. 8.1, Eq. 8.1 transforms into Eq. 8.3.

$$r(t) = \begin{cases} -10, & X_{object}(t) \notin S_o \\ 100 * \gamma_r(t) + 150 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ -|X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (8.3)$$

The training analysis included examining both reward and trajectories plots to identify modelling failures. The **reward plots** (such as the one depicted in Fig. B.1a) refer to a single episode and identify the reward taken at each step. The majority of those rewards (blue lines) are directly mapped into the gripper error (E_{g-o}) if no other event occurs. The green line identifies the reward threshold corresponding to a $2cms$ error, above which the reach/pick is considered successful. The graph also displays information on the gripper state (orange line), being either open (low) or closed (high). The **trajectory plots** (example in Fig. B.1b) show the gripper trajectory (blue line) versus the object trajectory (green line). This refers to the observations recorded for an entire episode. For clarity, the grey line delimits the action space. The start and endpoints of both trajectories are represented respectively by a red cross and a yellow star. Five relevant situations were identified, which were grouped by the gripper-object interaction:

Case A - Purely exploratory episodes

When no reaching or picking was performed, it results in a cumulative reward within **interval** $] -70; -15[$ (Tab. 8.2). Fig. B.1 depicts that situation, in which the gripper is positioned at one side of the object, with the fingers closed, **pushing the object to the side**. This trajectory can be seen in Fig. B.1b, as the object line (green) moves in the x-axis direction, similarly to the gripper trajectory (blue) after approaching the object. The second evidence that supports this is related to the reward curve, which is proportional to the gripper error (Eq. 7.1). The gripper is closed when the reward curve gets closer to the reach threshold (steps 20 to 50). So, if the gripper is closed, the gripper's central point will be near the object's central point, but the reaching is not performed. In this situation, the predominant aspect that prevented a successful episode was **closing the fingers too early**, at the approaching phase. Even though Fig. B.1a shows that the gripper state (y_c) was constantly alternating, the gripper's fingers remained closed for most of the episode. This is the

consequence of the **PID** controller and the mapping between the continuous and boolean variables. When replaying the episode, that situation is confirmed;

Case B - Object projected outside action space

This category includes episodes where the object exits the action space, when the gripper approaches it. Even after adjusting the collision parameters, some actions result in longer slides or object projection, both undesirable and dangerous in real situations. Suppose the object just slides out of the action space and remains in the observation space. In this case, the episode may accumulate lower rewards than case A, as a consequence of a greater distance between the gripper and the object (E_{g-o}) for the remaining steps, with no chance to undo the situation. This modelling choice aims to teach the agent the undesirable behaviours when approaching the object, whose episodes have cumulative rewards **between -408 and -33** (Tab. 8.2). For example, we refer to two situations where the object moves to different points outside the action space. In **episode 134** (Fig. B.3), the agent took a trajectory that caused a collision, and the object moves far from the action space. That collision happened at step 218, as suggested by the reward curve falling abruptly (Fig. B.2a). The object flew parallel to the yy axis (Fig. B.2b), suggesting a collision from behind (opposite to the direction of y). **Episode 4** (Fig. B.2) also depicts an episode in which the object exits the action space. However, the object just slides and stops right after the action space's limits, at $X_{object} = (0.582; 0.151; 0.024)$. This situation is less dangerous than episode 134, and similar to Case A. Therefore, it is acceptable that episodes like episode 4 have cumulative rewards similar to those in Case A. Comparatively, episodes like 134 should be penalised. So, **the further** the object lands and the **earlier** in the episode, the **lower** the cumulative reward;

Case C - Object projected outside observation space

If besides exiting the action space, the **object exits observation space**, the step receives a different penalty, and the episode ends. This penalty is set as -10 (Tab. 8.1), which is much lower than the rewards for gripper error E_{g-o} ($[-1, 0]$). The cumulative reward lies in **interval** $]-67; -14[$ and all of them lasted less than 500 steps. **Episode 149**, on Fig. B.4, demonstrates when the object is projected out of the observation space, flying to points with $x_o > 1.35$. Like the previous example, the gripper performs a wrong approach that should be avoided mainly for safety purposes. The difference between this case and episode 134 (Fig. B.3) might be the direction and force applied during the collision. However, when trying to replicate the results, it was verified that the physics engine had a **stochastic** component affecting reproducibility. It is believed some of that randomness has to do with noise in the joint's encoders, which also happen in the real world. This is also derived from network delays and deviations in the cycle time (network latency). Episodes like 149 introduce cumulative errors in the encoders by **stalling the engine**. The solution found to minimise uncertainty and suppress error accumulation was to reset the simulation every 20 episodes. Like the expert demonstration, the gripper approaches the object at step 25 (Fig. B.4a) and tries

to pick it by closing the fingers. However, the gripper moved the object before the collision, creating a new undemonstrated situation to decode. Thus, this episode was also crucial in terms of **environment exploration** and shows **generalisation capability**;

Case D - Sub-goal 1 completed

The agent **only reaches** the object and cannot complete the picking sub-task. In these situations, the cumulative reward lies within **interval**] - 36; 85[(Tab. 8.2), which results in a higher ranking, on average, when compared to previous situations. **Episode 292**, represented in Fig. B.5, shows that even if the gripper reaches the desired pose initially, the second sub-goal is not accomplished. Instead, the agent moves the object during the early training stages when grasping it at one end. Episodes like episode 506 (Fig. B.6) show that the agent reduces the gripper error (E_{g-o}), places the gripper near the accepted points, but **hesitates to close the fingers**, not completing the goal. Other cases combine a successful reach with the projection of the object, such as the situation depicted in **episode 382** (Fig. B.7). This specific episode lasted 496 steps and registered the cumulative reward of 41.80532, which will induce the agent to reproduce the behaviour, even being penalised. One evidence suggesting the need to change the values in Tab. 8.1 is this episode receiving a high cumulative reward;

Case E - Both sub-goals completed

Then, there are successful episodes where both **reaching and picking are performed**. The agent can complete those tasks non-sequentially, *i.e.* the agent can receive the reaching and picking rewards consecutively or with some steps in between. Keeping the reaching reward is essential to give information on where the agent can grasp the object. It is also crucial to promote continuity when retraining agents. One of the concerns was not to assign several reaching rewards, to prevent the agent from repeating the reaching instead of picking. Fig. B.8 depicts the agent reaching, moving away from the position and then grasping the object. The agent receives the reaching reward of $R_r = 100$ at step 248 and the picking reward at step 351, according to Fig. B.8a. Fig. B.8b suggests the agent was exploring approaching points in the direction of the x-axis when the object moved 1cm to the right. Completing the episode with the object in a new pose suggests the agent generalised, even after starting at the same point. When the agent chooses a direct trajectory, similar to the expert demonstration, the result is similar to **episode 64**, depicted in Fig. B.9. In this case, the agent performs the reaching and the picking sub-tasks simultaneously, which is close to the optimised way to do so, proving the agent was able to improve the behaviour. The cumulative reward for successful episodes belongs to the interval **interval**]220; 250[, where the highest rewards correspond to situations with fewer steps, short trajectories, and where reaching and picking are performed simultaneously.

Cases and Rewards Analysis

All previous cases are summed up in Tab. 8.2 with their corresponding reward ranges. **Case B** includes all episodes where the object exits the action space but not the observation space, and neither reaching or picking is performed. **Case C** represents every episode where the object exits the observation space, being implicit that it also escapes the action space. When the object is reached but not picked, the episode is classified as **case D** (resulting or not in the object exiting the spaces). **Case E** is when the agent successfully reaches and picks the object. Lastly, if none of the latter applies and the agent only performs exploration, it is classified as **case A**.

Case	Description	Cumulative Reward Interval
A	no reaching or picking	] - 70; -15[
B	object out of action space (no reaching)	] - 408; -33[
C	object out of observation space (no reaching)	] - 67; -14[
D	reaching (might exit spaces)	] - 36; 85[
E	success: reaching + picking	]200; 248[
Expert demonstrations		]243; 245[

Table 8.2: Pick Reward 1 - Observed situations and respective cumulative reward range.

Some successful situations (classified as E) achieve higher rewards than expert demonstrations, such as episode 64 (Fig. B.9). Also, the situations where the **object exits the observation space** (case C) are perceived as **less dangerous than the ones exiting the action space** (case B) because the cumulative rewards are higher. Nonetheless, **case A is comparable to C in terms of rewards** but in practice, the agent must avoid the latter. Relatively to episode 496 (Fig. B.7), the **reach reward (R_r) hides the fact that the object was projected** out of the observation space and might encourage the agent to reproduce it.

When looking at the **training** as a whole, the first metric analysed was the success rate and its evolution during the 808 episodes. The corresponding calculations used the linear convolution operator (*) to perform the moving window operations. The first step creates an array labelling each episode as 1 if successful and 0 otherwise (Eq. 8.4). Then, the success rate (SR, Eq. 8.5) is the convolution between the data array and the average operator (*avg_op*, array with 30 elements 1). The success rate was smoothed over a ten elements window (*smooth_op*), according to Eq. 8.6.

$$data[i] = \begin{cases} 1, & \text{success} \\ 0, & \text{otherwise} \end{cases} \quad (8.4)$$

$$SR = \frac{data * avg_op}{30} \quad (8.5)$$

$$SR_{smooth} = \frac{SR * smooth_op}{10} \quad (8.6)$$

The **success rate** evolution (SR_{smooth}) for the reward proposal 1 is depicted in Fig. 8.5. At first sight, there are four identifiable phases:

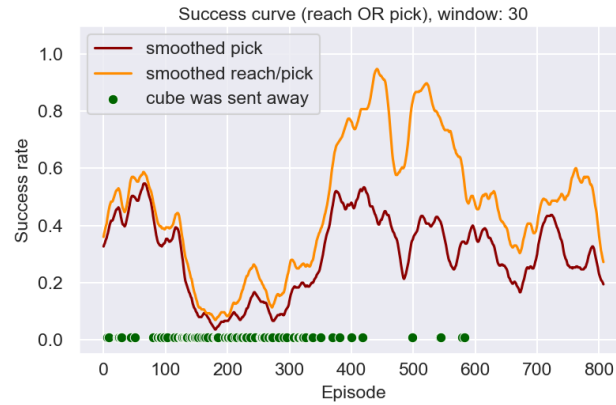


Figure 8.5: Success rate curve for pick agent trained anew, with ten expert demos and **reward function 1**. The function uses the distance (negative) between the gripper and the object as a dense reward, bonus of +100 when the gripper reaches the object for the first time, and +150 when the gripper picks the object, penalty of -10 every time the object exits the observation space.

- i From **steps 0 to 100**, the agent replicates the expert behaviour, with a success rate of 50%. However, analysing its reward plots, it is clear that the agent's exploration is focused on the y_c variable, as it constantly oscillates between open and closed fingers (Fig. B.1a and B.9a). Until episode 100, there have been 45 successful episodes. The agent took on average 60 steps to complete them (between 24 and 150 steps), according to Fig. B.10b. These first 100 episodes are classified in Tab.8.3. It is visible that when the agent performs reaching but not picking (case D), it does so due to the object leaving the action space. In 43% of the episodes, the object exits the action or/and the observation space, which must be avoided. However, the distribution of the cases is a consequence of the rewards discussed in Tab. 8.2. These situations correspond to episodes with cumulative reward around -70 in the graph of Fig. B.10a. Although the agent achieves success, the oscillation in the reward plot of amplitude 300 (Fig. B.10a) suggests the policy is still far from convergence. The first 20 rounds in Fig. B.11 support this conclusion: the high accuracy in the expert data (100%) concerning the generated data, which is increasing (Fig. B.11a) and the decrease in the discriminator loss (from 0.6 to 0.2, in Fig. B.11b). The success rate was expected to begin like depicted and then decrease, similar to the situations in Fig. 3.15;

Case		Nr of occurrences	Percentage (0-100%)
D	A	6	6%
	B	28	28%
	C	15	15%
	total	6	6%
	out of act space	5	5%
	out of obs space	0	0%
E		45	45%

Table 8.3: Pick - Case distribution for Reward 1, between episodes 0 and 99, inclusively.

- ii **Between steps 100 and 300** there is an abrupt decrease in the success rate, and the agent fails to reach or pick the object (Fig. 8.5). The learning curve (Fig. B.10a) reflects that lack of success, whose average cumulative reward ranges between -100 and 0. The justification is the object exiting both action (35.5%) and observation spaces (25%), according to both Tab. 8.4 and the frequent green points in Fig. 8.5. An increase is registered in the purely exploratory episodes (6% to 20%), where the gripper grasps the object at the wrong pose and moves it around, like the episode 167 (Fig. B.1). The occurrence of case A suggests that the agent tries to solve the undemonstrated situations where the goal point changes, trying to generalise. There are 28 successful episodes (case E), represented in Fig. B.10b from occurrence 45 to 73. There is also an evident increase in the steps required in an episode to achieve success. Thus, the **agent is focused on exploring new situations and interacts more with the environment**;

Case		Nr of occurrences	Percentage (0-100%)
D	A	40	20%
	B	71	35.5%
	C	50	25%
	total	11	5.5%
	out of act space	5	2.5%
	out of obs space	3	1.5%
E		28	14%

Table 8.4: Pick - Case distribution for Reward 1, between episodes 100 and 299, inclusively.

- iii Subsequently, **from step 300 to 600** the agent can complete the reach between 70-90% of the time but succeeds in picking 20-50% of the times, according to Fig. 8.5. On average, the agent reaches the object (case D) 28.7% of the time, and the success is not compromised by the object leaving the workspace (only 4.7% of the cases). This gap was due to object movements after small contact, and those episodes were similar to episode 292 (Fig. B.5). It is also clear that the agent learnt to avoid case D (object exiting observation space) and took more episodes exploring (case A occurred 17.7%). For cases A and D, the exploration occurs at the left side of the object. That predominance shows a bias even with the object starting at the centre of the observation space ($X_{object} = (0.4; 0.1; 0.25)$) and the gripper $y_g(0) \approx 0.1$. Consequently, the learning curve plot (Fig. B.10a) shows an increase in the cumulative reward at this stage, but the policy has not yet converged as there are oscillations of the amplitude of nearly 200. Having a look at the number of steps required to reach and pick the object, the curve in Fig. B.10b between the successful episodes 73 and 181 shows that the agent took longer to complete the episodes (> 100 steps), with some cases lasting over 400 steps. This behaviour favoured **exploration over optimisation**. The discriminator loss (Fig. B.11b) between rounds 60 and 120 appears to swing around 0.3, but the generator loss and value loss increase. Thus, the **policy has not converged yet**;

Case		Nr of occurrences	Percentage (0-100%)
D	A	53	17.7%
	B	37	12.3%
	C	16	5.3%
	total	86	28.7%
	out of act space	11	3.7%
	out of obs space	3	1.0%
E		108	36%

Table 8.5: Pick - Case distribution for Reward 1, between episodes 300 and 599, inclusively.

- iv At last, the **final episodes** show that the discrepancy between cases D and E decreased and the situations where the object exits the spaces are residual (case B - 0.96% and case C - 0%), according to Fig. 8.5 and Tab. 8.6. Although the loss curve (Fig. B.11b) for rounds 120 to 140 is smoother than previously, the agent's behaviour is unsatisfactory. The success rate is lower than 50%, which will not scale well when the remaining tasks are added. Even though the learning curve is positive in this interval (Fig. B.10a), it still shows big oscillations and the number of steps required to complete the episode range between 50 and 450 steps.

Case		Nr of occurrences	Percentage (0-100%)
D	A	108	51.92%
	B	2	0.96%
	C	0	0%
	total	34	16.35%
	out of act space	0	0%
	out of obs space	0	0%
E		64	30.67%

Table 8.6: Pick - Case distribution for Reward 1, between episodes 600 and 808, inclusively.

The agent replicates the expert behaviour during the first part of the training. After, it explores rather than exploiting the behaviour demonstrated. Then, the agent tries to converge towards a suitable behaviour, where the learning curve is more stable, the success rate increases and the model losses decrease. However, the model's behaviour derived from Reward function 1 needs **improvement on the success rate, occurrence of collisions, and discrepancy between cases D and E**. Next, we study new combinations of reward function parameter and their impact. The pick sub-task has proven to be unstable, as collisions produce non-observable states and prevent the agent from undoing the situation. Consequently, those collisions undermine picking and make the success rate decrease in the surrounding episodes.

8.3.1.2 Reward Function 2 - Removing penalty P_o and early episode finish

This reward function proposal aims to study if by finishing the episode earlier and penalising that step with $P_o = -10$ it prevented the agent from sending the gripper near the object, which usually resulted in a much lower step reward. This hypothesis was based on the sequence of cases: when dramatic collisions, such as those in C, occurred, it was registered a decrease in episodes classified as D and E. This change was motivated by the low success rate between episodes 100 and 300 (Fig. 8.5), when only 14% of those episodes were successful (Tab. 8.4). This new reward function (Eq. 8.7) differs from the previous Eq. 8.3 by **not considering the penalty P_o and by not finishing the episode earlier** every time the object exits the observation space (case C). The updated parameters are defined in Tab. 8.7. The figures corresponding to this training are grouped in the Appendix C.

$$r(t) = \begin{cases} 100 * \gamma_r(t) + 150 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ -|X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (8.7)$$

Parameter	Value
k_{pick}	1
R_r	+100
R_p	+150
P_o	(NA)
P_m	(NA)
P_c	(NA)

Table 8.7: Proposed parameters of Eq. 8.1 for Reward Proposal 2

During training with this reward function, all the situations described in Tab. 8.2 occur. However, as the approach to case C changed, the reward range changed too. The new intervals are represented in Tab. 8.8. The main difference is case C, as the interval is substantially lower, going from $] -67; -14[$ to $] -11.9E9; -202[$. Here, the episode does not end earlier, and the agent accumulates rewards proportional to the gripper error E_{g-o} . The models display higher losses because there are episodes where the cumulative reward is lower than -100 and observations out of the possible range. **The following reward functions should take that into account as an improvement.** Similarly, episodes classified as case D also receive lower cumulative rewards, as the object can exit the observation space after reaching, and those rewards accumulate for the entire episode. As the expert demonstrations were reused, the expert reward range is maintained.

Case	Description	Cumulative Reward Interval
A	no reaching or picking	$] -62; -20[$
B	object out of action space (no reaching)	$] -2800; -45[$
C	object out of observation space (no reaching)	$] -11904812695; -202[$
D	reaching (might exit spaces)	$] -3619; 52[$
E	success: reaching + picking	$]239; 248[$
Expert demonstrations		$]243; 245[$

Table 8.8: Pick Reward 2 - Observed situations and respective cumulative reward range.

After analysing the reward ranges, the next step consists of analysing the training, taking a similar approach to reward proposal 1 (Subsection 8.3.1.1). The new success rate is plotted in Fig. 8.6, using the same success window and smoothing operator in Eqs. 8.5 and 8.6, to enable comparisons. At first sight, there are two training phases:

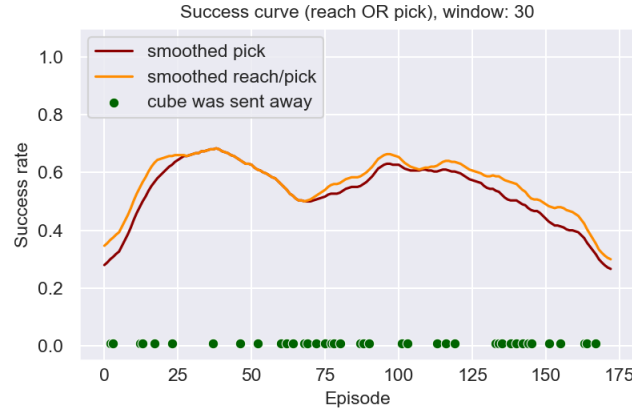


Figure 8.6: Success rate curve for the pick agent trained anew, with ten expert demos and **reward function 2**. The function uses the gripper error as a dense reward, a bonus of +100 when the gripper reaches the object for the first time, a bonus of +150 when the gripper picks the object, no action or penalisation is carried out when the object exits the observation space.

- i The **first 68 steps** show that most times the agent performs reaching, it can also complete picking, according to Fig. 8.6. Specifically, there are only two episodes where reaching is performed without picking, being both cases due to the object exiting the action space, as reported in Tab. 8.9. Case C happens 17.65% of times in the first part of the training, which is close to the corresponding first part in the previous section. In episode 4, the reward curve (Fig. C.1a) falls abruptly when the object jumps out of observation space and stays there for the remaining steps. However, as the cumulative rewards span over a wide range of values, the learning curve (Fig. C.5a) oscillates too much. Also, the generator loss (Fig. C.6b) is incredibly high for rounds 0 to 13. Another reason for those high losses is part of the observation being out of the defined limits. Although this behaviour is **unstable** and **must be improved**, it shows some **advantages on the frequency of cases E**. The success rate is over 50% for this reward function (Fig. 8.6), which was below 50% for the corresponding phase in the previous proposal (Fig. 8.5). This part of the training resulted in 41 successful episodes that lasted around 50 steps (Fig. C.5b), and the reward range (22-150) is comparable to the corresponding situation in the previous example (Fig. B.10b). An episode worth mentioning is episode 9 (Fig. C.2), where the object was attached to the gripper most of the episode, as the reward curve (Fig. C.2a) remains approximately constant for steps over 150. Episode 9 is a purely exploratory episode (case A). Still, it is evident that even though the approach and the positioning were almost perfect, the agent failed by the conjugation of three factors: changing the gripper state at a high rate, the effect of the **PID** and the Gazebo grasp plugin. All those factors made the gripper open for only a fraction of

seconds. That was interpreted by the plugin as noise, keeping **the object attached to the gripper**;

Case		Nr of occurrences	Percentage (0-100%)
D	A	2	2.94%
	B	11	16.18%
	C	12	17.65%
	total	2	2.94%
	out of act space	2	2.94%
	out of obs space	0	0%
	E	41	60.29%

Table 8.9: Pick - Case distribution for Reward 2, between episodes 0 and 67, inclusively.

- ii The remaining **steps (68 to 173)** show a smaller decrease in the success rate (Fig. 8.6), in comparison to the corresponding period in Fig. 8.5. This time, the success rate does not fall under 20%. Instead, there are 52.38% of episodes achieving success, according to Tab. 8.10. However, after reaching the object (case D), there are two episodes where the object ends up out of the observation space. One of these situations is depicted in Fig. C.3. There, the green line (Fig. C.3b) shows the object trajectory spreading out of the workspace. Analysing Fig. C.3a into detail, it can be concluded that after reaching the object, the gripper moves past the goal pose and can be inferred that it collides with the object from behind ($y > 0.3$), probably with the fingers closed. This reward function also seems to optimise the successful episodes: the required number of steps to success remain lower than 100, on average, according to Fig. C.5b. Another episode worth mentioning is episode 154 (Fig. C.4), in which the collision was so dramatic that the object reappeared at the world origin point. The object final point in Fig. C.4b appears in $X_{object} = (0; 0; 0)$ and the green lines depict the trajectory. In terms of the reward plot (Fig. C.4a), it is also clear that the reward is residual between steps 90 and 140, which justifies the low cumulative reward of -2815.30 . Despite the instability observable in the learning curve plot (Fig. C.5a), the discriminator accuracy (Fig. C.6a) is smooth, and the agent seems to be learning.

Case		Nr of occurrences	Percentage (0-100%)
D	A	2	1.90%
	B	18	17.14%
	C	26	24.76%
	total	4	3.81%
	out of act space	2	1.90%
	out of obs space	2	1.90%
	E	55	52.38%

Table 8.10: Pick - Case distribution for Reward 2, between episodes 68 and 173, inclusively.

Although episodes with X_{object} outside the observation space's limits cause **undesired model losses** (generator loss > 0.5), this proposal shows that having penalties (P_o) much lower than

the step error **prevent the agent from replicating situations where it approaches the object**. The penalty P_o made the agent see the object as a critical element to avoid the steps following a collision. However, it is important to improve the model's loss by **discarding observations** outside the allowed boundaries and **reducing the discrepancy** between episodic rewards.

8.3.1.3 Reward Function 3 - Penalising agent by finishing the episode

This reward proposal tries to combine the advantages offered by **removing penalty P_o** while reducing the model's loss by **eliminating the high range of rewards and the unexpected observations**. If the reward range is smaller, the agent should better distinguish between the rest of the cases.

The new parameters for the reward function (Eq. 8.1) are identified in Tab. 8.11, which results in Eq. 8.8. In practice, the reward function uses the same rewards as Eq. 8.7, but ends cases B earlier, with penalty $P_o = E_{g-o}$ instead of $P_o = -10$.

Parameter	Value
k_{pick}	1
R_r	+100
R_p	+150
P_o	E_{g-o}
P_m	(NA)
P_c	(NA)

Table 8.11: Proposed parameters of Eq. 8.1 for Reward Proposal 3

$$r(t) = \begin{cases} -|X_{gripper}(t) - X_{object}(t)|, & X_{object}(t) \notin S_o \\ 100 * \gamma_r(t) + 150 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ -|X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (8.8)$$

The situations observed are the same as previously. There are situations where the agent purely explores the workspace (case A). There are episodes where the object exits only the action space (case B) or the observation space (case C) without even being reached. In some cases, the agent correctly positions the gripper without picking the object (case D) or completes both sub-tasks (case E). The cumulative reward ranges for the recorded episodes are enumerated in Tab. 8.12.

Case	Description	Cumulative Reward Interval
A	no reaching or picking	] - 75; -20[
B	object out of action space (no reaching)	] - 455; -26[
C	object out of observation space (no reaching)	] - 571; -2.88[
D	reaching (might exit spaces)	] - 30; 90[
E	success: reaching + picking	] 224; 248[
expert demonstration		] 243; 245[

Table 8.12: Pick Reward 3 - Observed situations and respective cumulative reward range

Comparing the reward ranges to the previous examples (Tab. 8.2 and 8.8):

- **case A** remains in the same range, as nothing related to exploration was changed;
- **case B** is in the same range as the reward proposal 1 (Tab. 8.2) and smaller than those in proposal 2 because the episodes do terminate earlier. The episodes from proposal 2 where the object reenters the workspace (such as episode 154 depicted in Fig. C.4) were not registered, mitigating the lower bound of -2800 in Tab. 8.8. This fact suggests that even in a dataset four times bigger, the collisions recorded were not as forceful, showing improvements;
- **case C** includes episodes in a range between proposals 1 and 2. On the one hand, the interval is broader than in proposal 1 regarding lower and higher bounds. On the other hand, the lower bound is higher than in Tab. 8.8, the consequence of finishing the episode earlier. This tries to instigate exploration around the object pose;
- for the same reasons, the reward range for **case D** is narrower than for the reward proposal 2. Similarly to proposal 1, the lower bound is now considerably higher than the -36 in Tab. 8.2. Thus, the coherence in the results shows that the agent seems to explore more with this reward combination;
- for the **successful episodes**, the reward range is approximately the same.

After analysing the training in terms of episode reward magnitude, the following study is related to the classification of those episodes. The **success rate** evolution (SR_{smooth}) for the reward proposal 3 is plotted in the Fig. 8.7. The analysis focuses on four periods:

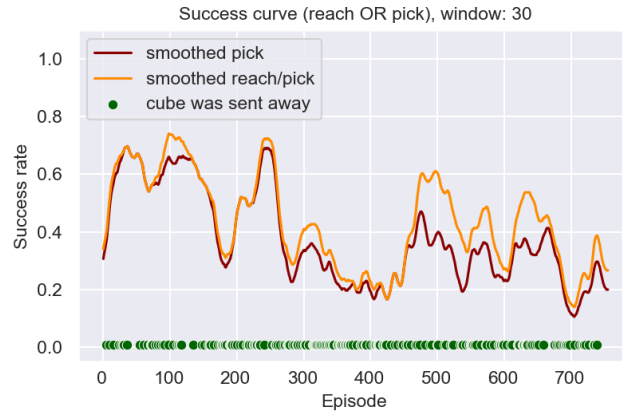


Figure 8.7: Success rate curve for pick agent trained anew, with ten expert demos and **reward function 3**. The function uses the gripper error as a dense reward, a bonus of $+100$ when the gripper reaches the object for the first time, a bonus of $+150$ when the gripper successfully picks the object. When the object exits the observation space, the episode ends with the corresponding reward

- i From **steps 0 to 200**, it is observed that the agent has copied the expert behaviour during the first episodes. Consequently, there is a significant percentage of successful episodes (56.6%) and few exploratory episodes (case A happens 2% of the time), according to Tab. 8.13. Then, the success rate increased from 73 occurrences to 113, compared to episodes 0 to 300 for reward 1. Regarding performing reaching but not picking, the percentage is now residual (under 4%), and all of those were due to the object exiting the agent's workspace. The learning curve, depicted in Fig. D.2a, shows that the averaged episodic reward is positive. So, there are more successful episodes (case E) than episodes involving collisions (such as cases B and C). However, the curve oscillations show that the model has not achieved convergence. In terms of successful episodes (Fig. D.2b), most of the 113 episodes lasted less than 100 steps, and the average was around 50 steps. The discriminator loss starts by decreasing as expected (Fig. D.3b), as the generator starts to learn the behaviour. The generator loss, depicted in the same figure, appears to be much smoother and lower than the observed in Fig. C.6b, for proposal 2.

Case		Nr of occurrences	Percentage (0-100%)
D	A	4	2%
	B	32	16%
	C	46	23%
	total	5	2.5%
	out of act space	3	1.5%
	out of obs space	2	1%
E		113	56.5%

Table 8.13: Pick - Case distribution for Reward 3, between episodes 0 and 199, inclusively.

- ii Between **steps 200 to 300**, the success rate decreases to 47%, and the object exits the observation space 32% of the time, according to Tab. 8.14. That permutation of 10% suggests that the gripper tested different approaches to the object, which ended in collision. Despite the decrease in the success rate, this shows an improvement compared to proposal 1. The percentage of cases A, B and D remained near those registered in Tab. 8.13.

Case		Nr of occurrences	Percentage (0-100%)
D	A	1	1%
	B	17	17%
	C	32	32%
	total	3	3%
	out of act space	2	2%
	out of obs space	1	1%
E		47	47%

Table 8.14: Pick - Case distribution for Reward 3, between episodes 200 and 299, inclusively.

iii Between **steps 300 to 400**, the agent shows the lowest success rate, fluctuating between 20% and 40%, according to Fig. 8.7. The learning curve (Fig. D.2a) shows a persistent decrease in the reward, reaching an average of -100 near episode 400. Despite the decrease, the agent did not accomplish the reach sub-goal as well. Instead, the agent ended up colliding with the object more often, as case C represents 46% in this interval, according to Tab. 8.15. These results show that the agent focused on approaching the object with different trajectories. In some cases, it tries to reach the object from one side, but it often sends the correct trajectory with the fingers closed, such as the situation depicted in Fig. D.1. This situation is not heavily penalised, as the episodic reward is around -9.38. Instead, the episode is terminated, preventing unexpected observations and not assigning step rewards too low. The cumulative reward encourages the agent to repeat the behaviour (as it happens in the following episodes). When comparing this situation to the results found with reward proposal 1, additional penalties like P_o prevent the agent from replicating those last steps. In this episode, the last steps were dedicated to approaching instead of picking the object, which undermined the success rate. These episodes attempt to penalise the agent by lasting less than 500 episodes and feeding lower rewards of the same magnitude as before.

Case		Nr of occurrences	Percentage (0-100%)
D	A	7	7%
	B	16	16%
	C	46	46%
	total	5	5%
	out of act space	1	1%
	out of obs space	2	2%
E		26	26%

Table 8.15: Pick - Case distribution for Reward 3, between episodes 300 and 399, inclusively.

iv Finally, the discriminator accuracy (Fig. D.3a) on both expert and generated data decreases, despite the loss function (Fig. D.3a) appearing to stabilise. Between **steps 400 to 756**, the learning curve (Fig. D.2a) ranges between -100 and 150, on average. The learning curve confirms the results in Tab. 8.16, where there is a significant increase in episodes classified as B (from 16% to 28.4%) and a decrease in episodes classified as C. Then, the agent reduces the most dangerous collisions (cases C), which means handling those episodes is enough to teach the agent to avoid them. The trade-off is related to the number of steps required to complete an episode, which increases, from episode 186 onward, according to Fig. D.2b. Comparing the success curve for this reward function (Fig. 8.7) and reward 1 (Fig. 8.5), it is visible that the success rate oscillates between the same values. Still, there was an improvement related to case D. The gap between partially completed episodes (case D) and successful episodes (case E) is smaller as episode D decreased from 16.35% to 8.7%. In 21 out of 31 episodes, the agent could not pick the object as it exited the observation or the action space.

Case		Nr of occurrences	Percentage (0-100%)
D	A	25	7.0%
	B	101	28.4%
	C	96	27.0%
	total	31	8.7%
	out of act space	13	3.6%
	out of obs space	8	2.2%
E		103	28.9%

Table 8.16: Pick - Case distribution for Reward 3, between episodes 400 and 756, inclusively.

After this analysis, it can be concluded that episodes where the object exit the observation space are **unstable factors**. On the one hand, they need to happen so that the agent **understands how to interact** with the object. On the other hand, these episodes must **receive a penalty** not to be encouraged. Compared to reward 1, this reward proposal **improves the success rate** as the exploration around the object is **encouraged**. The generator model is much more **stable**, also making it a better reward. There are **no observations out** of the pre-defined conditions, and there are **no over penalising** rewards. We conclude that providing rewards outside the action space and stopping the episode is sufficient to **punish** the agent while **maintaining continuity** in the assigned rewards.

8.3.1.4 Reward Function 4 - Penalty for moving the object

As both the object pose and gripper pose on reset were always the same, no expert demonstration would exemplify how the agent would behave **when the object had moved**. This next reward proposal aims to study if that **compromised success rate by requiring more exploration**. Then, this proposal is similar to the previous parameters, enumerated in Tab. 8.17.

Parameter	Value
k_{pick}	1
R_r	+100
R_p	+150
P_o	$E_{g-o}(t)$
P_m	$E_{g-o}(t)$
P_c	(NA)

Table 8.17: Proposed parameters of Eq. 8.1 for Reward Proposal 4

The difference is: if **the object moves over 2cms** from the reset pose ($|X_{object}(t) - X_{object}(t_0)| > 0.02$), the episode **finishes** with the final reward P_m . The penalty P_m was applied for gripper errors above 2cms to be consistent with the threshold defined for reaching/picking. This aims to

minimise undemonstrated situations, and the respective equation is Eq. 8.9.

$$r(t) = \begin{cases} |X_{gripper}(t) - X_{object}(t)|, & |X_{object}(t_0) - X_{object}(t)| > 0.02 \cap X_{object}(t) \in S_o \\ 100 * \gamma_r(t) + 150 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (8.9)$$

It is observed that for this reward proposal, there is a new situation. Every episode that finishes before 500 steps due to the object moving more than $2cms$ from the reset pose is classified as **case F**. These episodes are penalised by finishing earlier and receiving the penalty of P_m . As seen before in reward proposal 3, finishing the episode earlier with a greater but consistent reward is enough to teach the agent the wrong behaviour. According to Tab. 8.18 and 8.19, there were no episodes of **type A**, which is understandable as exploration usually results in the object being moved around. Every episode that would be **classified as B** ended up finishing earlier and being classified as F too. Despite every **episode C** being inherent of case F, there is a clear difference between those. If the object moves more than $2cms$ between cycles and exits the observation space, the episode was more critic than others. Thus, for analysis purposes, these situations are inspected separately. However, the rewards are calculated the same way and accordingly to Eq. 8.9.

Case	Description	Cumulative Reward Interval
A	no reaching or picking	(NA)
B	object out of action space (no reaching)	(NA)
C	object out of observation space (no reaching)	] - 12; 0[
D	reaching (might exit spaces)	]93; 94[
E	success: reaching + picking	]241; 245[
F	object moved over $2cms$	] - 11; -4[
Expert demonstrations		]243; 245[

Table 8.18: Pick Reward 4 - Observed situations and respective cumulative reward range

Case	Nr of occurrences	Percentage (0-100%)
A	0	0%
B	0	0%
C	20	5.5%
total	2	0.5%
D	0	0%
out of act space	0	0%
out of obs space	0	0%
object moved over $2cms$	2	0.5%
E	30	8.2%
F	313	85.8%

Table 8.19: Pick - Case distribution for Reward 4, between episodes 0 and 365, inclusively.

The reward range for **episodes classified as C** is aligned with those episodes observed earlier, in which the leading cause is the gripper being closed at the object approach. According to the data in Tab. 8.19, those situations only occur 5.5% of the time and thus have little impact on the training. There are only two episodes **classified as D** where the agent reaches the object but fails to pick it, as it is penalised by moving the object. Episode 259 on Fig. E.1 is one of those situations. According to both trajectory (Fig. E.1b) and reward plots (Fig. E.1a), the agent approaches the object but moves it before and after reaching it properly. Thus, the partial success was due to the reward function modelling, and the plots suggest the agent would achieve success in the following steps. However, the majority of the training had episodes of **type F**, which happened 85.8% of the time. The corresponding cumulative rewards ranged between -11 and -4 , closer to case C's rewards. However, this reward range does not reflect the difference in the behaviours in which the object moves, but the agent can still achieve success and does not cause dangerous situations.

The success rate evolution for the reward proposal 4 is depicted in Fig. 8.8. The first observation is that the **success rate is considerably lower** than the previous trains, never rising above 30%. The overall success rate remains at 8.2%, according to Tab. 8.19. The second remark is that the reach curve follows the picking curve evolution. There are only two episodes when resetting the episode prevented the agent from achieving success (Fig. E.1). Finally, the blue points in Fig. 8.8 show how often the episodes **ended earlier due to the penalty P_m** .

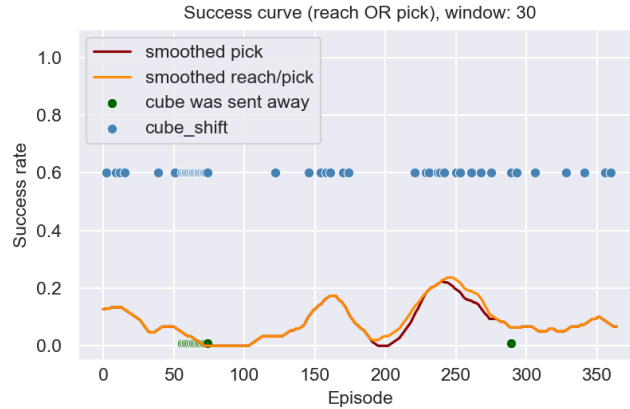


Figure 8.8: Success rate curve for pick agent trained anew, with ten expert demos and **reward function 4**. The function uses the gripper error as a dense reward, a bonus of $+100$ when the gripper reaches the object for the first time, a bonus of $+150$ when the gripper successfully picks the object. When the object moves more than 2cms , even if remaining in the observation space, the episode ends with the corresponding reward.

Fig. E.2a shows the rewards being around 0 and rising only whenever the successful episodes happened. At first sight, this reward curve seems an improvement as there are **fewer oscillations**, and the episodic reward ranges only between -10 and 250 . However, this only happens because episodes classified as A, B or C **were mitigated**. On the other hand, the lower the range, the better the agent understands between successful episodes, **promoting optimisation**. Namely, with this lower range, the required number of steps (Fig. E.2b) to complete the episodes remained significantly **smaller**, of around 60 steps on average. In terms of accuracy and loss, there is little

to say because the training was brief. Nonetheless, the accuracy (Fig. E.3a) is rising smoothly towards 1, but the discriminator loss (Fig. E.3b) is taking **longer** to decrease, suggesting that the training is **inefficient**.

This proposal's motivation is to study whether **convergence** is compromised by the object **moving away from the reset pose**. That particular situation has **not** been demonstrated and then requires the agent to **explore it before solve it**. Another factor that might delay convergence is the **lack of reproducibility** that comes from the Gazebo simulator. Namely, when succeeding at an episode, the agent will try to replicate the behaviour that might not have the same output due to **randomness**, for example, on the joint encoders. Considering the previous analysis, it is safe to conclude that finishing the episode earlier when the object moves is a **penalty that undermines exploration**. The direct consequence is the **low success rate** in Fig. 8.8.

8.3.1.5 Reward Function 5 - Adjusting rewards R_r and R_p

As induced from the previous training, reducing the reward range might help the agent with optimisation. Specifically, the little **differences** between successful episodes are more **noticeable** because the difference between all cases is also smaller. Next, we tested that hypothesis by **reducing** both R_r and R_p . The new parameters are described in Tab. 8.20, which correspond to Eq. 8.10. This time, there is **no additional penalty** every time the agent moves the object. However, the episode finishes if there is any observation out of the predefined limits.

Parameter	Value
k_{pick}	1
R_r	+10
R_p	+15
P_o	$E_{g-o}(t)$
P_m	(NA)
P_c	(NA)

Table 8.20: Proposed parameters of Eq. 8.1 for Reward Proposal 5

$$r(t) = \begin{cases} 10 * \gamma_r(t) + 15 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (8.10)$$

It was necessary to record **new** expert trajectories that would match the new reward equation and followed the same rules as before. This time, the ten expert episodes lasted between 51 and 67 steps, requiring an average of 58 steps. In terms of reward range, the expert trajectories received episodic rewards between 11.97 and 13.75, resulting in an average of 12.91. As expected, their range of values reflected the impact of **non-optimal** trajectories, and the gripper error E_{g-o} has a more significant impact on the overall reward.

In terms of **episode range**, it remained close to the expected interval for the cases A, B and C, evidenced in Tab. 8.21. Both episodes D and E were affected by the changes in the Eq. 8.10 and show lower intervals. Specifically for case D, the reward R_r was not enough to overcome the

exploration rewards but is closer to episodes A and C. In general, a purely exploratory episode (case A) is **closer** to an episode where only reaching was performed, which means reaching is a **sub-goal that is not enough to succeed**. It is also evidenced that reaching the object and pushing it out of the action or observation space is **worse than case A**. For case E, the conjunction of rewards R_r and R_p is enough to make up for all the other rewards. According to Tab. 8.21, the agent was able to **optimise the trajectory** either by going straight to the object centre or by performing reaching and picking simultaneously. The optimisation has a relatively greater impact this time, as the **cumulative reward range is much smaller**.

Case	Description	cumulative reward Interval
A	no reaching or picking	] - 61; -20[
B	object out of action space (no reaching)	] - 385; -31[
C	object out of observation space (no reaching)	] - 63; -5[
D	reaching (might exit spaces)	] - 270; -2.29[
E	success: reaching + picking	] 0; 21[
expert demonstration		] 11.97; 123.75[

Table 8.21: Pick Reward 5 - Observed situations and respective cumulative reward range

One similar approach in other environments is applying **reward normalisation**. However, this is not compatible with this task's nature: there is no lower bound for episodes where the object jumps **out of observation space**, as it depends on when it happens and how far it has moved. Higher rewards are not compatible with the hierarchical approach of **adding sub-goals**: every time a sub-goal is added, the episodic reward will increase.

The following analysis divides the training procedure into two sections: one part includes steps until 400, and the other examines the remaining episodes. The success curve depicted in Fig. 8.9 shows these two behaviours, in which the success rate falls smoothly until episode 400 and increases until episode 650, with a gap between reaching and picking.

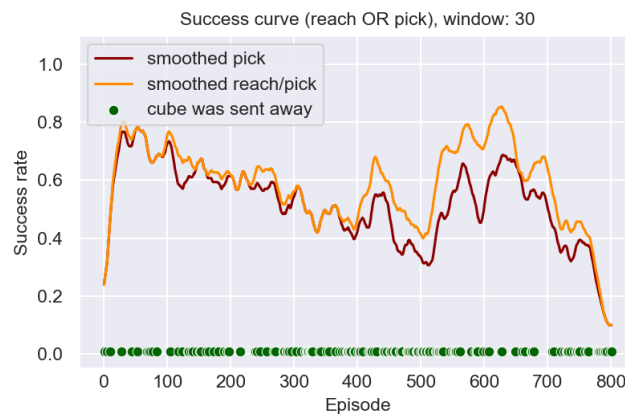


Figure 8.9: Success rate curve for pick agent trained anew, with ten expert demos and **reward function 5**. The function uses the gripper error as a dense reward, a bonus of +10 when the gripper reaches the object for the first time and +15 when the gripper successfully picks the object.

The case distribution is the following:

1. Between **steps 0 and 400**, the overall success rate is around 58.75%, according to Tab. 8.22, which are **better results** when compared to the previous outcomes. The residual episodes (9 out of 400) classified as D, ended with the object outside the observation or action spaces. The exploratory component was significant (case A), consisting of 5.5% of the episodes in which the success was compromised by the object moving inside the action space, such as episode 80 (Fig. F.1). A considerable fraction was made of cases B or C, representing 33.5% of the total episodes. Nonetheless, this is a clear improvement to the success rate, higher than 50% and results near 80% around step 50.

Case		Nr of occurrences	Percentage (0-100%)
D	A	22	5.50 %
	B	65	16.25%
	C	69	17.25%
	total	9	2.25%
	out of act space	6	1.5%
	out of obs space	3	0.65%
E		235	58.75%

Table 8.22: Pick - Case distribution for Reward 5, between episodes 0 and 399, inclusively.

2. Steps ranging **from 400 to the end** show two undesirable behaviours (Fig. 8.9), namely the difference between the reach and pick curves' discrepancy and the decrease in the success rate from episode 600 onward. The Tab. 8.23 shows that on 11.20% of the episodes, the agent failed to pick the object after reaching the desired pose (case D). However, not every episode ended with the object out of the workspace. Eight out of forty-five episodes registered the agent moving the object but reaching it correctly later in the episode. However, it failed to grasp the object. This is the situation depicted in Fig. F.2 and proves the agent can converge to success even after placing the object at a new pose.

Case		Nr of occurrences	Percentage (0-100%)
D	A	23	5.72%
	B	63	15.67%
	C	86	21.39%
	total	45	11.20%
	out of act space	22	5.47%
	out of obs space	17	4.23%
E		185	46.02%

Table 8.23: Pick - Case distribution for Reward 5, between episodes 400 and 801, inclusively.

In terms of the learning curve, cases where the object leaves the workspace are **more evident** and represent the negative spikes in Fig. F.3a. These spikes correspond to case B, according to Tab. 8.21. It is also observable that the episodic reward range is smaller and allows to **distinguish**

the episodes B and C more easily, according to each case. However, according to Fig. F.3b, the learning curve does not show signs of convergence yet, and the number of steps required to achieve success is increasing. The accuracy on generator data (Fig. F.4a) **oscillates** between 0.9 and 0.5, which suggests the model is not stable yet. The loss function (Fig. F.4b) **decreases faster** than previously and converges to 0.4, as expected. Despite the improvements, it is necessary to make it clear to the agent the differences between successful episodes, **to induce optimisation**.

8.3.1.6 Reward Function 6 - Inciting optimisation

The next reward proposal **decreases the rewards** for reaching or picking, while keeping $R_p > R_r$. This parameter tuning tries to **instigate optimisation**, as the number of steps will significantly impact the successful episodes. The new parameters are then evidenced at Tab. 8.24, and the respective equation is Eq. 8.11.

Parameter	Value
k_{pick}	1
R_r	+5
R_p	+10
P_o	$E_{g-o}(t)$
P_m	(NA)
P_c	(NA)

Table 8.24: Proposed parameters of Eq. 8.1 for Reward Proposal 6

$$r(t) = \begin{cases} 5 * \gamma_r(t) + 10 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (8.11)$$

Changing rewards R_r and R_p required ten new demonstrations. The trajectory acquisition followed the same instructions as before. The demonstrations lasted between 45 and 67 steps, requiring an average of 55 steps. The expert trajectories received episodic rewards between 6.973 and 10.335, resulting in an average of 8.762. The rewards R_r and R_p are smaller and allows the gripper error (E_{g-o}) to have **more impact on the overall reward**.

The reward ranges for the episodic rewards for each case are organised in Tab. 8.25. As expected, the difference from the previous reward has to do with cases D and E, given the changes in the reward parameters. Restricting the rewards R_r and R_p to lower values will also impact optimisation when other sub-tasks are added.

Case	Description	Cumulative Reward Interval
A	no reaching or picking	] - 61; -20[
B	object out of action space (no reaching)	] - 440; -24[
C	object out of observation space (no reaching)	] - 63; -4[
D	reaching (might exit spaces)	] - 325; -4[
E	success: reaching + picking	] - 6; 13[
Expert demonstrations		] 6.973; 10.335[

Table 8.25: Pick Reward 6 - Observed situations and respective cumulative reward range

Following the reward range study and after confirming the episodes covered the expected reward intervals, the next analysis is related to the training evolution. According to the success rate in Fig. 8.10, the analysis is divided into three parts:

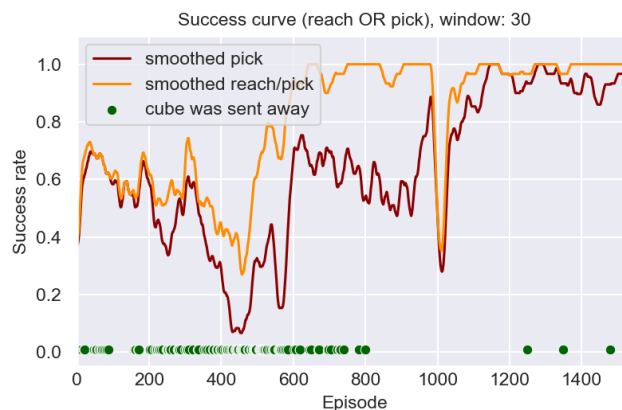


Figure 8.10: Success rate curve for pick agent trained anew, with ten expert demos and **reward function 6**. The function uses the gripper error as a dense reward, a bonus of +5 when the gripper reaches the object first and +10 when the gripper successfully picks the object.

1. Between **steps 0 and 450**, where the success rate starts around 60% and then decreases to 10% around episode 430. As previously, the agent **replicates** the expert behaviour at first, according to the log files, showing success in 48.22% of episodes (Tab. 8.26). Then, it starts exploring the gripper state action (y_c) and ends up projecting the object, as depicted in Fig. G.1. These situations that are classified as B and C, where the object leaves the action and the observation space, respectively, are substantial. These cases happened almost 40%, according to Tab. 8.26, and are most frequent towards the end of this part. This is observable as the number of green points increases towards episode 450, in Fig. 8.10. The last fact worth mentioning is the gap between reach and pick, which increases near episode 450 due to the collisions. The learning curve in Fig. G.5a supports this by decreasing and converging to cumulative rewards of -100 , in the range of cases B and D.

Case		Nr of occurrences	Percentage (0-100%)
D	A	17	3.78 %
	B	105	23.33%
	C	71	15.78%
	total	40	8.89%
	out of act space	25	5.56%
	out of obs space	10	2.22%
E		217	48.22%

Table 8.26: Pick - Case distribution for Reward 6, between episodes 0 and 449, inclusively.

2. From **step 450 to 1000**, the success curve rises again. According to Tab. 8.27, there are 54.36% of successful episodes. According to Fig. 8.10, the agent can reach the object nearly 100% of the time (despite the gap between reach and pick curves), without collisions between steps 800 to 1000. This time, the agent fails to pick the object (after reaching it) 34.18% of times. However, Tab. 8.27 shows that the object remains in the workspace at 119 out of 188 episodes. This is commonly caused by the object being pushed towards the action space's limits, such as episode 843 in Fig. G.2. According to Fig. G.5b, the successful episodes took longer. Successful episodes between 217 and 516 lasted between 50 and 450 steps. The longest episodes had the agent solving situations where the object moved. Specifically, episode 786 (Fig. G.3) shows the agent reaching the pose within 50 steps but picks it only at step 489. The agent might have been searching for multiple reaching rewards, as it places the gripper at less than 2cms. However, this shows the model can generalise and deal with objects moving away from the initial pose.

Case		Nr of occurrences	Percentage (0-100%)
D	A	8	1.45 %
	B	23	4.18%
	C	32	5.82%
	total	188	34.18%
	out of act space	38	6.91%
	out of obs space	31	5.64%
E		299	54.36%

Table 8.27: Pick - Case distribution for Reward 6, between episodes 450 and 1000, inclusively.

3. From **step 1000 to 1540**, the success rate rises again (Fig. 8.10.) However, the first thing noticed is the abrupt decrease in the success rate around episode 1000. Observing these episodes (such as episode 1000 in Fig. G.4), it is understood that the arm required recalibration, and the agent comes back to normal after that. As consequence of the the decrease in the success rate, the episodes 1000 to 1540 in the learning curve (Fig. G.5a) correspond to lower episodic rewards, which shows coherence over all analysis. Regarding GAIL performance, the discriminator accuracy decreases (Fig. G.6a) and the loss rises over 1 (Fig. G.6b), destabilising the model. After those outlier episodes, the model **converges to successful episodes**, and the success rate reaches 100%, between steps 1140 and 1340. The average success rate at this interval is 88.70%, according to the information in Tab. 8.28. The learning curve (Fig. G.5a) also converges to positive values between 4 and 10, and the episode length for successful episodes decreases, approaching the demonstration behaviour, according to Fig. G.5b.

Case		Nr of occurrences	Percentage (0-100%)
D	A	22	4.08%
	B	4	0.64%
	C	2	0.37%
	total	33	6.11%
	out of act space	1	0.19%
	out of obs space	1	0.19%
E		479	88.70%

Table 8.28: Pick - Case distribution for Reward 6, between episodes 1000 and 1540, inclusively.

By **reducing** the reward R_r and R_p , the agent **explores more** and ends up **avoiding collisions**. It can also **generalise** when the object moves away from the reset pose and optimises the episode later. It was essential to ensure all reward intervals **overlap** and are **sorted** from the undesirable cases to the successful ones. The gripper reward is a **smooth metric** that creates a **gradient** and guides the agent towards the object. This is only possible because the modelling had the action space **inside** the observation space.

8.3.1.7 Reward Function 7 - Trying to avoid collisions

The action that controls the finger's width (y_c) is **critical**. In other words, approaching the object with the **fingers closed** will prevent the agent from completing the first sub-task, either by not positioning the gripper near the object or colliding and projecting the object out of the workspace. Consequently, **all subsequent sub-tasks** will be compromised if the object is not grasped (even if the reaching sub-task is completed). So, the last experiment tries to **speed up convergence by penalising the agent when it closes the fingers earlier** than required. This reward hypothesis derives from the previous one (Subsection 8.3.1.6) and adds the penalty P_c , according to the parameters in Tab. 8.29.

Parameter	Value
k_{pick}	1
R_r	+5
R_p	+10
P_o	(NA)
P_m	(NA)
P_c	Eq.8.13

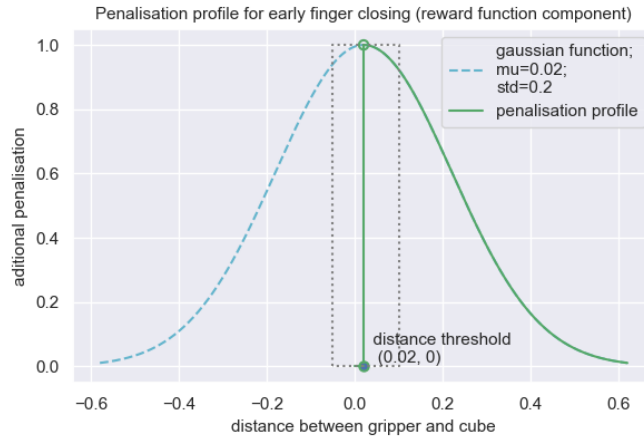
Table 8.29: Proposed parameters of Eq. 8.1 for Reward Proposal 7

The resulting reward equation corresponds to Eq. 8.12. The penalty P_c , which is part of the step reward, was related to the **gripper error** (E_{g-o}) and the **gripper state** (c). According to Eq. 8.13, P_c is applied if the gripper's fingers are **closed** ($c = 1$) during the initial steps. Its value follows a **Gaussian distribution**, as depicted in Fig. 8.11. In detail, if the gripper error is **lower** than $2cms$, the agent can close the fingers as it is within the accepted zone and $P_c = 0$. Otherwise, the closer the gripper, the more dangerous it is to close the fingers due to **higher collision probability**. So,

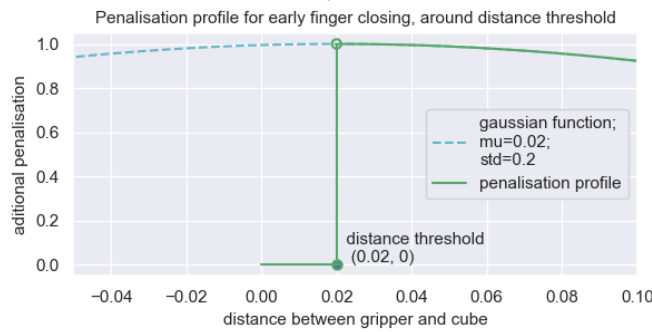
the Gaussian distribution will be centred in 0.02 ($\mu = 0.02$) and the standard deviation is $\sigma = 0.2$ to make the penalty insignificant at the reset ($E_{g-o} \approx 0.4 = 2 * \sigma$). Lastly, the penalty is normalised between 0 and 1.

$$r(t) = \begin{cases} 5 * \gamma_r(t) + 10 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)| - P_c(t), & otherwise \end{cases} \quad (8.12)$$

$$P_c(t) = \begin{cases} \frac{1}{0.2\sqrt{2\pi}} \exp -\frac{1}{2} \left(\frac{E_{g-o}(t) - 0.02}{0.2} \right)^2 * c, & |X_{gripper}(t) - X_{object}(t)| > 0.02 \\ 0, & |X_{gripper}(t) - X_{object}(t)| \leq 0.02 \end{cases} \quad (8.13)$$



(a) Penalisation is shaped as a Gaussian curve



(b) Penalisation around reach threshold

Figure 8.11: Penalisation profile for early gripper closing. The area withing grey dotted line on the figure a) is zoomed in on figure b)

The expert demonstrations acquired previously were reused in this new train as P_c does not influence them. Looking at the reward ranges at Tab. 8.30, it is observed that the reward intervals are **lower** in all cases. This is a direct consequence of the P_c . Even for successful episodes (case E), the lower limit of -240 shows the non-optimised episodes where the agent **closed the gripper**

inadvertently. The penalty P_c is **inversely proportional** to the gripper error E_{g-o} . For each episode, the cumulative reward decreases as the number of steps where $c = 1$ increases. However, **all reward ranges mixed up**, making it harder for the agent to learn the desired behaviour. This is clear for cases E, whose reward amplitude is 250, and only 7.2% of those values are specific to a successful episode.

Case	Description	Cumulative Reward Interval
A	no reaching or picking	] - 404; - 58[
B	object out of action space (no reaching)	] - 515; - 93[
C	object out of observation space (no reaching)	] - 400; - 8[
D	reaching (might exit spaces)	] - 470; - 22[
E	success: reaching + picking	] - 240; 10[
Expert demonstrations		]6.973; 10.335[

Table 8.30: Pick Reward 7 - Observed situations and respective cumulative reward range

The success rate curve in Fig. 8.12 is analysed now. Similarly to the previous approaches, the training is divided into three parts with similar behaviour:

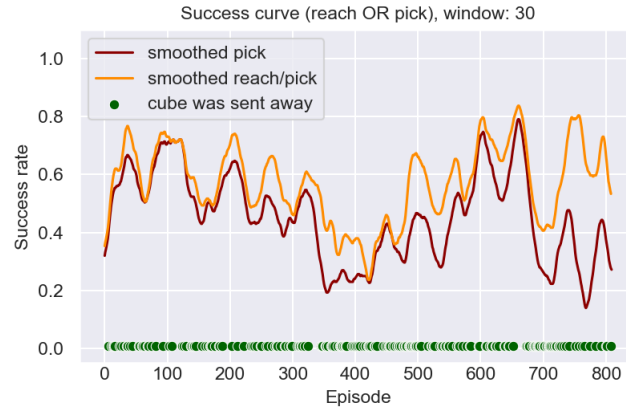


Figure 8.12: Success rate curve for pick agent trained anew, with ten expert demos and **reward function 7**. The function uses the gripper error as a dense reward, a bonus of +5 when the gripper reaches the object first and +10 when the gripper successfully picks the object. Whenever the object exits the observation space, the episode ends. An additional penalisation is applied every time the agent closes the gripper before completing reach.

1. Between **steps 0 and 400**, it is observed that the main differences were related to cases B and C. As stated in Tab. 8.31, the percentage of episodes where the object ended out of the action space (case B) decreased from 23.33% to 16.75%. However, the object exited the observation space more frequently, going from 15.78% to 20.25%. Even though the results show improvement in avoiding dangerous situations overall, the success rate had little impact, *i.e.* case E went from 48.22% 50.5%, compared to the corresponding period in Tab. 8.26). The oscillation is accentuated in the learning curve (Fig. H.2a), which is also related to the broad range of episodic rewards, as stated in Tab. 8.30.

Case		Nr of occurrences	Percentage (0-100%)
D	A	19	4.75 %
	B	67	16.75%
	C	81	20.25%
	total	37	7.75%
	out of act space	16	4.00%
	out of obs space	13	3.25%
E		202	50.5%

Table 8.31: Pick - Case distribution for Reward 7, between episodes 0 and 399, inclusively.

2. From **step 400 to 700**, the success rate (Fig. 8.12) starts high but falls abruptly around episode 700. There were 138 successful episodes at this stage, which means a success percentage of 46%, as shown in Tab. 8.32. It is also noticeable that cases where the object exits the observation space (case C) were more frequent (25.67%). The learning curve (Fig. H.2a) at the same interval depicts oscillations, and thus the model is far from convergence. According to Fig. H.2b, the successful episodes between episodes 202 and 340 lasted longer, on average.

Case		Nr of occurrences	Percentage (0-100%)
D	A	6	2.0 %
	B	50	16.67%
	C	77	25.67%
	total	29	9.66%
	out of act space	14	4.67%
	out of obs space	14	4.67%
E		138	46.0%

Table 8.32: Pick - Case distribution for Reward 7, between episodes 400 and 699, inclusively.

3. From **step 700 to 810**, the success rate decreases again to less than 50%, according to Fig. 8.12. The training shows more oscillations in the success rate than the results for the previous proposal, and there is a clear difference between situations D and E. According to Tab. 8.33, the agent performs reaching without picking (case D) on 33.64% of the episodes, and half of those had nothing to do with the object leaving the workspace. Instead, the agent moves the object around the action space, whose behaviour is represented by episode 751 (Fig. H.1). The trajectory plot (Fig. H.1b) shows the agent moved the object around the action space, and it could not generalise and solve the situation. The reward plot (Fig. H.1a) shows the oscillation on the reward curve caused by adding the penalty P_o . This might explain why the convergence is taking longer, as the reward plot is still far from convergence (Fig. H.2a). The successful episodes are also longer (episodes 340 to 374 in Fig. H.2b last, on average, more than 75 steps).

Case		Nr of occurrences	Percentage (0-100%)
D	A	13	11.82%
	B	9	8.18%
	C	17	15.45%
	total	37	33.64%
	out of act space	13	13.82%
	out of obs space	11	10.00%
E		34	30.91%

Table 8.33: Pick - Case distribution for Reward 7, between episodes 700 and 810, inclusively.

Despite the accuracy being over 80% (Fig. H.3a), the discriminator loss being between 0.2 and 0.3 and the generator loss being residual (Fig. H.3b), **this model does not perform better**. According to Fig. 8.13, reward function 6 appears to be closer to convergence. In terms of the **first sub-goal**, reward 6 presents higher accuracy, increasing steadily from episode 500 onward, achieving that over 90% of times. On the contrary, reward 7 shows significant oscillations between 0.4 and 0.8, which means the models has not been appropriately learnt.

In terms of the **second sub-goal**, both functions behave similarly until episode 600. However, the proposal 6 is converging while the proposal 7 is exploring, as suggested by the oscillations. So, if the agent cannot reach the object, it will not pick it. Overall, the penalty P_c does not help with the convergence and makes it more confusing to the agent.

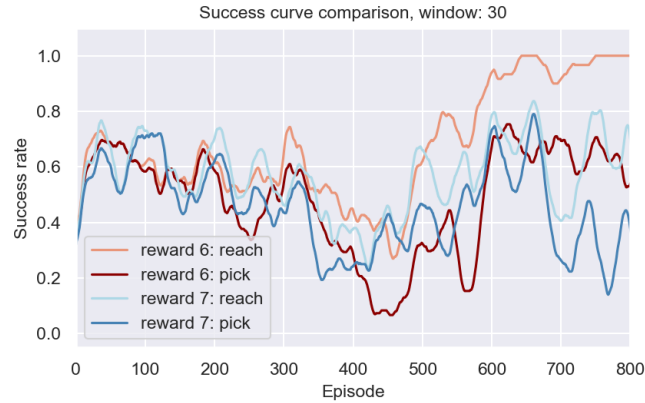


Figure 8.13: Comparison between pick reward functions 6 and 7, regarding success rate curve, trained anew, with ten expert demos. The light curves depict the successful reach and the dark curves the successful pick

8.3.1.8 Comparing all Reward Functions

Finally, we compare the reward functions regarding success rate evolution. For that, we plotted the seven picking curves on Fig. 8.14.



Figure 8.14: Pick - Comparison between **all reward functions**, regarding success rate curve, with ten expert demos

Success curve for **reward 4** (Subsection 8.3.1.4) shows the poorest results and corresponds to the situation where the episode ends if the object moves more than 2cms from the original pose ($|X_{object}(t_0) - X_{object}(t)| > 0.02$). The conclusion is that the **agent should learn how to interact with the environment, even if that corresponds to undemonstrated or harder situations**.

The **reward function 1** shows a steep decrease around episode 200. That situation used the penalty P_o every time the object flew outside the observation space. Also, the agent could not optimise successful episodes, as it had **little impact** on the episodic reward. Thus, it shows that penalties brought instability, and the rewards **must be as smooth** throughout the episode as possible. Even though P_c results in worse results, it is important to avoid observations out of the predefined limits. This is evident for **reward proposal 2**, which showed a volatile learning curve and generator losses over 0.2. Therefore, finishing the episodes earlier with the reward corresponding to the observation space's limits is the best combination studied to penalise the agent on cases C. For this, it was important to place the action space inside the bigger observation space.

Then, **reward functions 3, 5 and 6** are variations in which R_r and R_p are sequentially smaller. In those, the agent explored more and understood how to avoid collisions and the consequent abnormal observations. The key point was to ensure all reward ranges were sorted from the undesirable cases to the successful ones while overlapping as little as possible. The best results come from the situation 6 where the gripper error (E_{g-o}) had **greater impact**, *i.e.* for smaller R_r and R_p .

Even though **proposal 7** uses the same R_r and R_p as proposal 6, it shows poorer behaviour. Both reaching and picking learning curves show oscillations and instability. So, the penalty P_c (earlier finger closing) does not speed up convergence. Instead, it confuses the agent by **destroying**

the **smooth gradient** created by E_{g-o} to the gripper error component. According to Tab. 8.34, proposal 6 achieves the best performance, reaching 100% of success rate and finishing with 97%. The training lasted 1540 episodes, which took 23 hours approximately, mainly due to the inability of simulation parallelisation in Gazebo.

Reward Function	Max SR	Final SR	Number of Episodes
1	57%	23%	808
2	68%%	28%%	173
3	70%%	30%%	756
4	22%%	10%%	365
5	77%%	30%%	801
6	100%%	97%%	1540
7	79%%	45%%	810

Table 8.34: Pick Reward Functions - Success rate comparison

8.3.2 Traditional vs Sub-goal based Reward

In this study, we tested the **impact of a sub-goal methodology versus the traditional reward approach**. For that, it was considered an agent on the same task: picking the object. However, the environment did not assign an intermediate reaching reward (R_r). The so-called "traditional approach" (or vanilla reward) used only the gripper error and the bonus for picking, in the reward function of Eq. 8.14.

$$r(t) = \begin{cases} 10 * \gamma_p(t), & \gamma_p(t) = 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (8.14)$$

The traditional reward function promotes the trajectory as a whole. The results are depicted in Fig. 8.15, which show faster convergence for the sub-goal approach. The sub-goal approach achieves 100% on the first sub-goal and 70% on the second sub-goal for episodes over 600. At that moment, the traditional agent is still exploring the environment and analysing the demonstrated behaviour.

These results suggest that a sub-goal methodology that **divides a complex task into simpler sub-tasks** can boost convergence and achieve better results. The sub-goal procedure consists in assigning **intermediate positive rewards** when the agent completes one of the sub-tasks. Those rewards, however, should be given only once to avoid non-optimised behaviours. The following chapter will test this hypothesis on a three sub-goal task for validation.

8.3.3 Training from Scratch vs from Pre-learnt Tasks

After discovering the power of the sub-goal methodology in rewards and having the models perform more complex tasks sequentially, the idea of model hierarchisation came to light. The focus

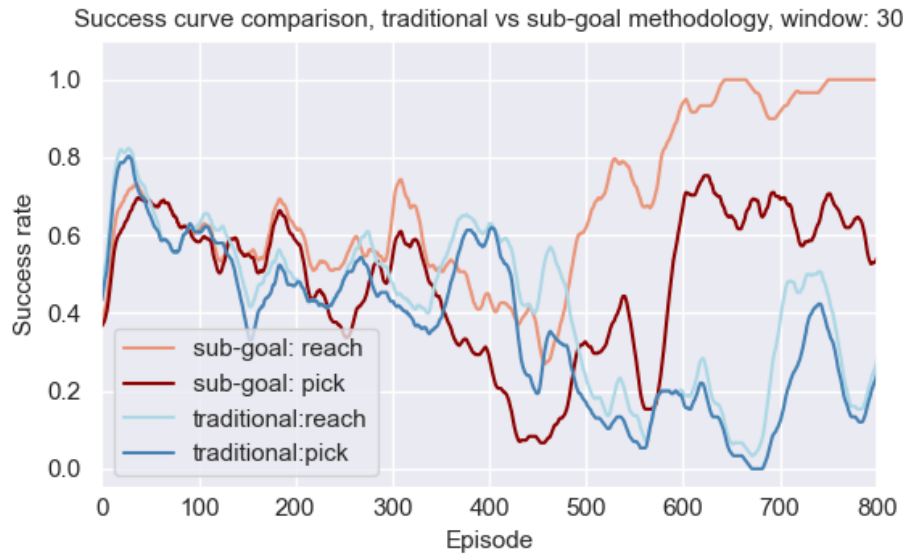


Figure 8.15: Two sub-goals. The success rate of the proposed sub-goal-based reward methodology (red) vs a traditional reward method (blue). The light colours represent the reaching sub-goal and the darker colours the picking.

of this next study was to reuse the reaching model (from Chapter 7) as the starting point for the agent to learn how to pick the object.

The [GAIL](#) algorithm had two neural networks: one for the discriminator and the other for the generator. So, in this retrain case study, we used both models that had been saved previously. Then, before starting the training, both were loaded. After, the training started and the agent didn't have to start from scratch.

For a reliable comparison, the pick environments used were the same, as well as the demonstration procedures. The results are pictured in Fig. 8.16.

It is observable that the hierarchical methodology does not perform well in this situation. The key differences between these two sub-tasks are:

- the agent is required to learn how to control the gripper state (Y_c), as the fingers are not stuck open;
- closing the fingers too soon will cause collisions which will cause dangerous situations;
- even without collisions, closing the gripper will prevent the agent from completing the episodes or even reaching the object;
- keeping the gripper open will not differentiate cases A and D from E.

So, closing the gripper at the correct time frame is a crucial skill that can be classified as antagonistic. On the one hand, closing the gripper is the only way to complete all the tasks. On the other hand, it brings unsucccess and destabilises the model. These results of reusing the reaching models on picking task were tested twice to assure the results' repeatability. Retraining brings no advantages in this situation, and further analysis will be held in the following two chapters.

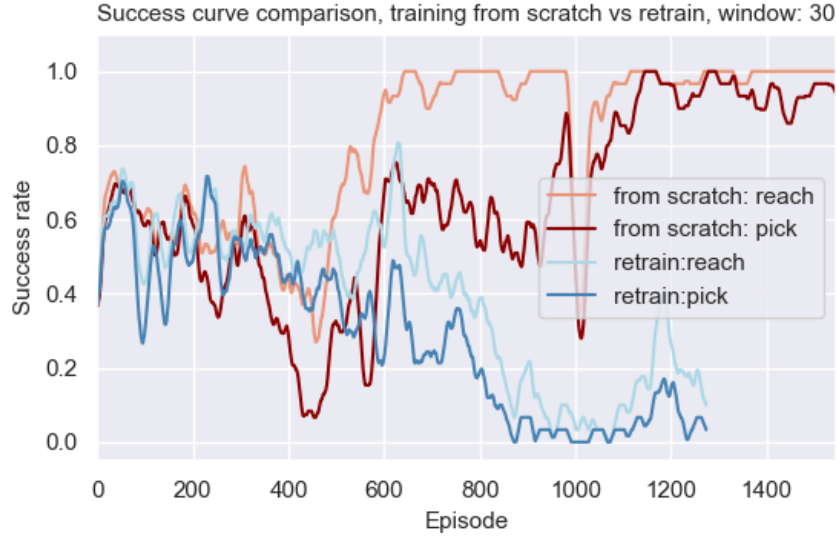


Figure 8.16: Hierarchical results- Comparison between the agent trained from scratch and pre-loaded models (retrain) on the reaching sub-task

8.4 Chapter Synopsis

This chapter presented the second experiment, which consists of reaching plus picking the parallelepiped object. Undoubtedly, this is the most difficult sub-task of them all: controlling the gripper opening causes frequent collisions. So, when choosing the correct reward function and corresponding parameters, we have analysed the occurrence of five different situations, summed up in Tab. 8.35.

Case	Description
A	no reaching or picking
B	object out of action space (no reaching)
C	object out of observation space (no reaching)
D	reaching (might exit spaces)
E	success: reaching + picking

Table 8.35: Observed situations summary.

First, we tested seven different combinations of the discretised reward function (Section 8.3.1). Those tests used ten expert demonstrations, recorded accordingly. Based on the success rate, the learning curves, discriminator accuracy and network losses, we opted for the parameter combination number six, repeated hereby in Eq. 8.15.

$$r(t) = \begin{cases} 5 * \gamma_r(t) + 10 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (8.15)$$

The function maintains the **gripper error** as a smooth dense reward. This is important to give the agent information on **where** to search for the object, alongside the information on the observation space (X_{object}).

Next, we concluded that the agent is deeply influenced by **penalties**. So, **over-penalising** it for projecting the object and inadvertently closing the fingers will only prevent exploration. Nevertheless, undesirable episodes ensure the skills are learnt, which make them crucial for understanding the environment. When analysing the trade-off between **allowing exploration** and **avoiding dangerous collisions and model losses**, we opted for simply **terminating the episodes earlier**, when the object leaves the observation space.

On the one hand, this will result in a penalty lower than the gripper error magnitude because the object is more distant. It is only possible because the action space is **inside** the observation space and **smaller**. On the other hand, we mitigate high losses by removing observations out of the expected intervals and episodes with tremendously low cumulative rewards. That final penalty, P_o , is not constant and depends on the gripper's error. The specific P_o does not erase the impact of R_r , such as in the situations classified as D.

By choosing the sub-goal rewards not so high, namely $R_r = 5$ and $R_p = 10$, there is an episode **normalisation**. The different situations (Tab.8.35) are in an adequate reward range that allows the agent to search **optimisation**. It was essential to ensure all reward intervals **overlap** and the cases were **sorted** from the undesirable cases to the successful ones. In terms of **generalisation**, we have observed that when the object moves away from the reset pose, the agent could chase it and did complete the episode.

These conclusions were derived from the comparison of the seven reward proposals. Then, after choosing the sixth function, we have analysed how sub-goal rewards related to the traditional modelling (Section 8.3.2). The sub-goal methodology showed enhanced performance after 500 episodes, where the success rate showed an increase between 20% to 60%. Opposed to the traditional methodology, the sub-goal approach shows a gap between one sub-task and two sub-tasks achieved. That suggests the agent can understand those two situations separately, allowing it to learn the sub-tasks despite not completing the main task. Overall, the sub-goal methodology that **divides a complex task into simpler sub-tasks** can boost convergence and achieve better results.

In terms of using the models knowing how to reach the object, reusing the models does not seem to be advantageous in this situation. The gripper state seems to be an unstable factor that influences the task success antagonistically. Further tests and analysis had to be carried out to come up with justified conclusions.

Chapter 9

Move Sub-Task

After picking the object, the agent must control the manipulator to **place the piece at the correct point**. This new sub-task is called move and requires the agent to handle the gripper's positioning, $Y_{gripper}$. At this task, the gripper orientation (ϕ) must be chosen by the agent.

In terms of complexity, the move sub-task is considered to be more straightforward. After picking the object, the agent is required to control the gripper positioning, which is less trouble than the fingers' state. The challenge here is to perform the three sub-tasks sequentially, so the maximum number of steps in an episode was incremented from $M = 500$ to $M = 600$.

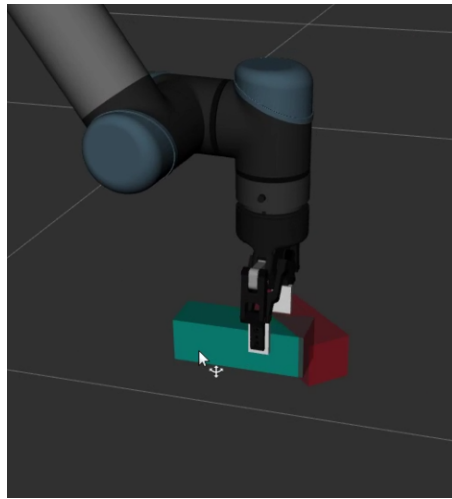


Figure 9.1: Sub-task 3 - Moving the object. The red shape represents the goal pose and the blue one is the real object.

This chapter explains how the move reward was built, based on the picking proposal and conclusions inferred. Then, we proceed to report how the expert demonstrations were recorded. In Section 9.3.1, we compare nine different proposals and choose the best parameters combination to a three sub-task mission. The best results are then compared to a sequential and traditional methodology, in Section 9.3.2. At last, the results of the pre-trained models are presented versus the behaviour learnt from an agent without previous knowledge.

9.1 Reward Function

According to the previous reward functions, the proposal to the move sub-task should also be based on the picking reward. So, we considered the pick reward function 6, which is repeated in Eq. 9.1:

$$r(t) = \begin{cases} 5 * \gamma_r(t) + 10 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (9.1)$$

Next, it was necessary to add two new components to the previous reward function:

- R_m , the reward for when the object is placed at the **correct point**, with an error smaller than $5cms$;
- E_{o-p} , the **error** between the **object** pose and the **desired** pose. According to Eq. 9.2, the error is calculated as the difference between the goal pose and the object's centre. This error has similar behaviour to the gripper error (E_{g-o}), which will create a smooth gradient throughout the space.

$$E_{o-p} = |X_{goal}(t) - X_{object}(t)| \quad (9.2)$$

In this example, the goal pose is $X_{goal} = (0.450, 0.130, 0.025)$, in which the height corresponds to the object's centre. The goal was as far from the object as possible without being placed at the action space's limits. The reward function to the move sub-task is then transformed into Eq. 9.3:

$$r(t) = \begin{cases} 5 * \gamma_r(t) + 10 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ R_m * \gamma_m(t), & \gamma_m(t) = 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & \sum_{\tau=t_0}^t (\gamma_r(\tau) + \gamma_p(\tau)) < 1 \cap X_{object}(t) \in S_o \\ k_m * |X_{goal}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (9.3)$$

Where: t_0 is the initial step, $M = 600$ is the maximum number of steps for a given episode, which was incremented because there are now three sub-tasks. S_o is part of the observation space corresponding to the object's pose (Tab. 6.1, Eq. 7.3). The γ_r indicates whether the gripper reached the object successfully at that step t (Eq. 7.4), γ_p takes the value 1 when grasping has been successful (Eq. 8.2), and γ_m gives information on the object placement (Eq. 9.4). The gain k_m corresponds to a constant that controls the object error's impact in the step reward.

$$\gamma_m(t) = \begin{cases} 1, & |X_{goal} - X_{object}| \leq 0.05 \cap \sum_{\tau=t_0}^t \gamma_p(\tau) = 1, \forall \sum_{\tau=t_0}^M \gamma_m(\tau) \leq 1 \\ 0, & otherwise \end{cases} \quad (9.4)$$

Similarly to the previous situation, the cumulative reward of a given episode will depend on the **episode duration**, the number of **sub-goals** completed, the values of R_m and k_m , the **gripper error** (E_{g-o}) and the **object error** (E_{o-p}).

9.2 Building Expert Trajectories

The expert demonstrations on the moving task include: approaching the object, positioning the gripper near the object's centre, picking the object and **moving the object to a new point**. This sequence of sub-tasks is represented in the diagram of Fig. 9.2.

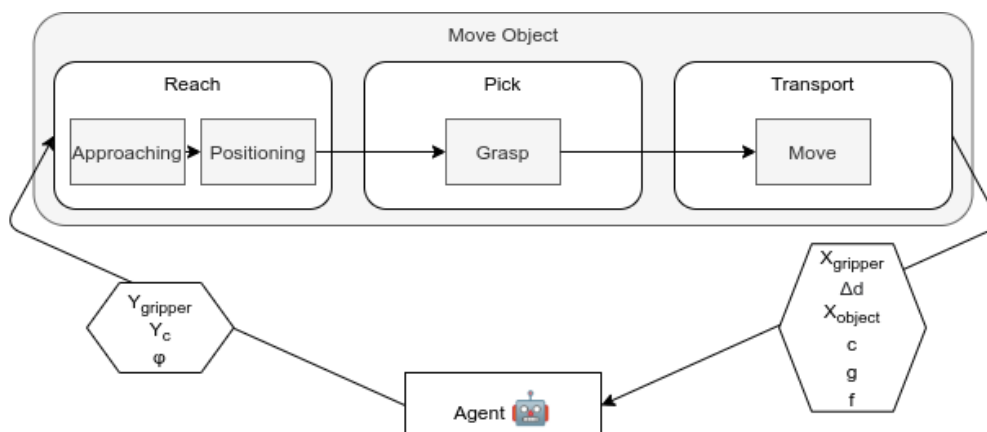


Figure 9.2: Moving the object - reaching, picking and moving sub-tasks. The action space is represented in the left diamond box and the observation space in the rightmost one. The object is attached to the gripper during the transportation phase.

All steps until the move sub-task follow the same rules defined previously. First, the gripper and the object start at the same points. Then, the gripper moves towards a random point near the object, reaches the object centre and picks it. All these steps are the same as before to promote continuity between the sequence of sub-tasks and enable retraining the models successively.

Regarding this sub-task, the demonstration includes moving the object up to $z_o(t) = z_o(t_0) * 1.5$ and then moving the object towards the goal. By **moving the object up**, the object will stop touching the ground, which eases the manoeuvre. However, it is not mandatory to do so. The gripper's fingers never touch the ground to avoid dangerous collisions that would damage the manipulator. That restriction is accomplished with the kinematics model that filters out those situations. Finally, the object error (E_{o-p}) is reduced until the object is placed at the goal point. At that moment, the reward will be R_m .

According to Fig. 9.3, These new demonstrations lasted longer than the previous ones. This increase in duration is expected, as a new sub-task was added that cannot be completed simultaneously. The variation between the number of steps in demonstrations was related to the approach phase, which significantly impacted the episode's optimisation. The demonstrations lasted between 48 and 73 steps, with an average of 63 steps per episode.

The trajectory of both gripper and object are represented in Fig. 9.4. There, we can observe the first sub-tasks, which are similar to the behaviour observed previously. After that, we see the object moving parallel to the gripper, which depicts this third sub-task of moving.



Figure 9.3: Required number of steps for expert demonstrations on the moving task

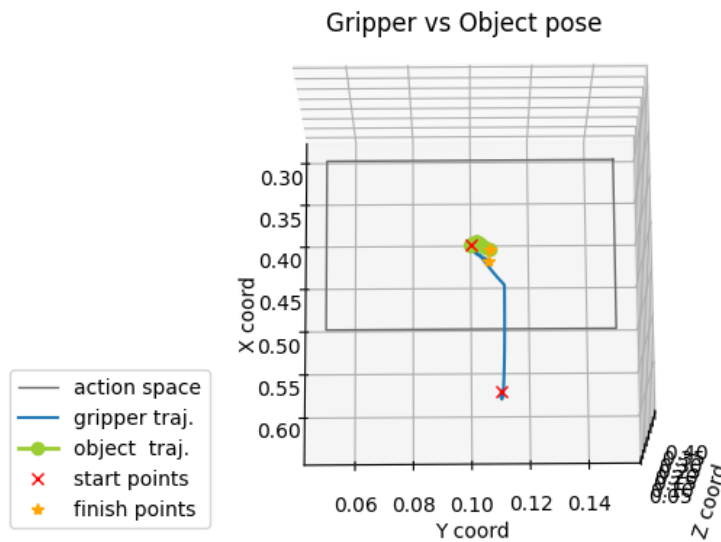


Figure 9.4: Expert demonstration 1 on the moving task. The blue curve represents the gripper trajectory, starting on the red cross and finishing on the yellow star. That gripper's trajectory includes 3 phases: the approaching phase, according to Eq. 7.5, plus the positioning, the picking and the object moving. When the object approaches the goal, the trajectory is complete. The object moves parallel to the gripper. The grey line defines the action space's limits, where the valid gripper points ($Y_{gripper}$) belong.

9.3 Experiments and Results

Similarly to the experiments carried out to the pick task (Section 8.3), we tested the task of reaching, picking and moving the object towards a new goal pose. These tests aim to study how reward discretization and model re-utilisation behave in complex tasks. First, in Section 9.3.1, we analyse the performance of **seven combinations** of the reward function (Eq. 9.3) and choose the best suited parameters. Then, in Section 9.3.2, we analyse that training performance compared to a **traditional modelling**, where only the last sub-goal is associated with a positive reward. In Section 9.3.3, to test the impact of **model reuse**, we evaluate the agent’s behaviour when providing it with the reaching and picking know-how.

9.3.1 Training from Scratch

After defining the reward function for the move sub-task (Eq. 9.3), we tested eight different combinations of parameters. According to Tab. 9.1, the variables consisted of the move gain k_m , the move reward R_m , the behaviour of the gripper state y_c and the observation Δ_d . For each combination, the success rate, the learning curve and the **GAIL** metrics were plotted. Those figures can be found in the Appendices I to Q.

Proposal	k_m	R_m	Action y_c	Observation Δ_d	Figures on Appendix ...
1	1	15	free	$\Delta_d = X_{goal} - X_{object} $	I
2	1	15	$y_c = 1$	$\Delta_d = X_{goal} - X_{object} $	J
3	1	15	$y_c = 1$	$\Delta_d = X_{goal}$	K
4	2	15	$y_c = 1$	$\Delta_d = X_{goal}$	L
5	0.5	15	$y_c = 1$	$\Delta_d = X_{goal}$	M
6	0.5	15	$y_c = 1$	$\Delta_d = X_{goal} - X_{object} $	N
7	0.5	18	$y_c = 1$	$\Delta_d = X_{goal} - X_{object} $	O
8	0.5	12	$y_c = 1$	$\Delta_d = X_{goal} - X_{object} $	P
9	0.5	8	$y_c = 1$	$\Delta_d = X_{goal} - X_{object} $	Q

Table 9.1: Parameters’ combinations tested for the moving task.

The **first proposal** lets the agent control the gripper state on every step. So, most times, the agent lets the object fall after picking it. The success rate in Fig. I.1 shows results never going over 10% while completing both reaching and picking. Beyond undoing the second sub-task, there’s the issue with the observation variables. As the Gazebo grasp fix plugin has some toleration, the object will remain grasped for some steps after unpicking it. This causes some incoherence during the subsequent steps and can confuse the models. On the one hand, the learning curve is unstable (Fig. I.2). On the other hand, the losses (Fig. I.3b) do not stabilise and decrease to less than 0.2, which is undesirable and breaks the balance between discriminator and generator.

The **second combination** uses the same parameters but restricts the gripper state, i.e., the agent can only control the orientation and pose after picking the object. As a consequence, the **object remains grasped while trying to complete the third sub-goal**. This simplification reduces the

exploration required to re-pick the object, which would require many steps or the addition of a new reward that would undermine optimisation. This combination shows improvements in the success rate after episode 600 (Fig. J.1). However, those are not sufficient, as they do not go over 20%. All the following functions **maintain** the gripper closed after picking.

Reward 3 tests the best method to teach the agent the goal pose. So, instead of providing the object error as Δ_d , the observation shows the real goal pose, resulting in $\Delta_d = X_{goal}$. This representation relates to the way the object is represented through real coordinates. Nonetheless, the reward remains the object error (E_{o-p}). Despite the short training, the agent does not seem to learn the behaviour, according to Figs. K.1 to K.3.

Reward functions 4 and 5 are variations of the previous. They change the impact of the object error on the episode. Respectively, the gains are double or half. The gain k_m impacts the episodic reward and influences the agent's perception of the object error (granularity). Specifically, **higher gains** will **penalise heavily** the agent during the moving steps, but will allow a **more precise** control. Smaller gains will work the other way around. When comparing rewards 3, 4 and 5, the best results obtained correspond to **reward 5**. It can be concluded that the gain $k_m = 0.5$ is the best trade-off.

The results obtained until the reward 5, suggest that the best representation to the agent is $\Delta_d = |X_{goal} - X_{object}|$. We understand that it might be easier for the agent to follow the error gradient than to find the trajectory through the gripper pose, object pose and goal pose, all separated. It is even clear when recalling the object is picked with an error between 0 and 2cms, making it tricky for the agent to control the object placement.

At this point, it was chosen to use $k_m = 0.5$, to **constrain the gripper state** (y_c) after picking the object and to represent the goal relatively to the object, having $\Delta_d = |X_{goal} - X_{object}|$. The final decision corresponds to the move reward R_m . So, in **functions 6 to 9**, we compared rewards in the range of the other bonus ($R_r = 5$ and $R_p = 10$). According to Tab. 9.1, rewards of 18, 15, 12 and 8 were studied. The **ninth reward function achieves good results**, reaching the success rate of 40% (Fig. Q.1). After episode 500, the success rate decreases, and both discriminator and generator become unstable: the accuracy decreases and the losses increase after round 100 (Fig. Q.3). So, from then on, the results must be discarded and correspond to undesirable behaviours. These reward function's parameters seem to be unstable.

The visual comparison between all functions is represented in Fig. 9.5, which supports the choice for the **reward proposal 7**. We consider this the best combination, as it achieves almost 60% of the success rate.

According to the rewards, we can extrapolate that the rewards associated with the sub-tasks should be **sequentially higher**, as the sequence increases. The rewards chosen were: $R_r = 5$, $R_p = 10$, $R_m = 18$. It is also important to refer that one of the issues observed was related to **re-calibration**. The dangerous situations associated with collisions introduced joint errors that could extend through several episodes. So, after each collision, we proceeded with the recalibration by restarting the simulation. All the models had to be re-spawn to try to overcome broken joints. However, sometimes that was not sufficient to overcome the problems, and a new train had to be

performed. The problems with the Gazebo re-spawn are debated in the community forum, and no solution has been found.



Figure 9.5: Move - Comparison between **all reward functions**, regarding success rate curve, with ten expert demos

9.3.2 Traditional vs. Sub-goal based Reward

The sub-goal methodology proved to be better in the two sub-task of reaching and picking an object (Section 8.3.2). According to Fig. 9.6, it is also better for a task involving three steps: reaching, picking and moving.

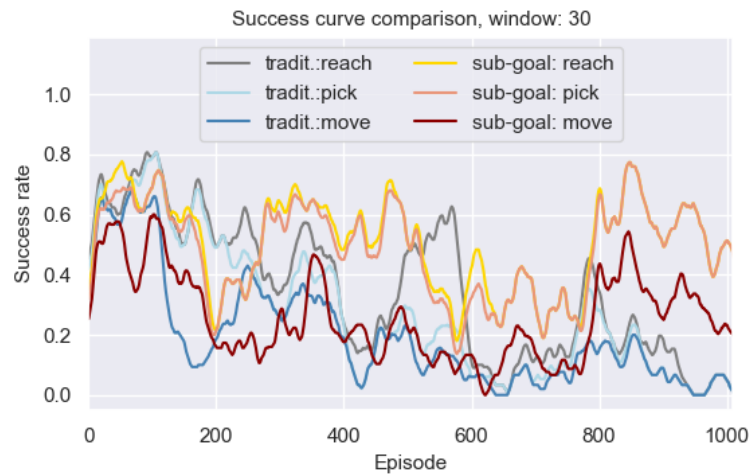


Figure 9.6: Three sub-goals. The success rate of the proposed methodology (red) vs the traditional method (blue).

The sub-goal approach shows a bigger separation between curves than in the traditional approach. So, it can be concluded that the agent **understands** each of those sub-tasks **separately**, allowing it to **optimise each of them**. The traditional approach gives no additional reward when the agent completes the first or second sub-tasks, only after moving the object correctly. So, it is expected to take longer to learn those.

During the majority of the episodes, the **proposed methodology shows better results on all sub-tasks**. It is more evident between episodes 200 and 800 that the picking is completed more often than in the traditional approach. Near episode 800, the sub-goal agent picks the object nearly 80% and moves the object correctly almost 60% of times. Comparatively, the traditional methodology completes the task with a similar frequency of 20%.

Near episode 1000, the traditional agent is confused with the sub-tasks order, and it is still exploring new situations differently from those demonstrated. At that time, the traditional agent goes directly to the goal pose without picking the object first. Episode 991 (Fig. 9.7) shows that behaviour, in which the gripper moves towards the goal, colliding with the object and projecting it outside the observation space.

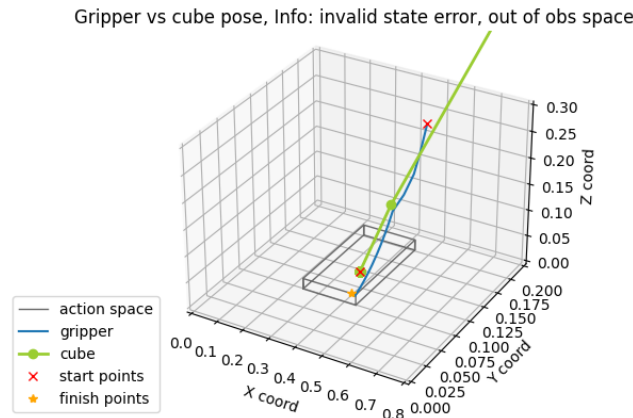


Figure 9.7: Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object trajectory shows the object exiting the observation space after colliding with the gripper. The agent guides the gripper to the surroundings of the goal pose without completing the other sub-tasks first.

9.3.3 Training from Scratch vs from Pre-learnt Tasks

In the previous chapter (Chapter 9), it was observed that retraining an agent that knew how to reach the object to learn how to pick it brought no advantage. Nonetheless, we tested the retraining approach on the moving task. One used the **reaching model** and the second used the **picking model**.

Retraining on the reaching model

The agent needs to learn how to control the gripper state between the reaching and moving models. Fig. 9.8 compares those situations, where the blue line represent the agent trained from scratch (combination 7 in Section 9.3.1) and the red represents the hierarchical results on the reaching sub-task. Similarly to what happened in the picking sub-task (Section 8.3.3), retraining the models brings no advantages. In fact, reusing the reaching models produces worse results.

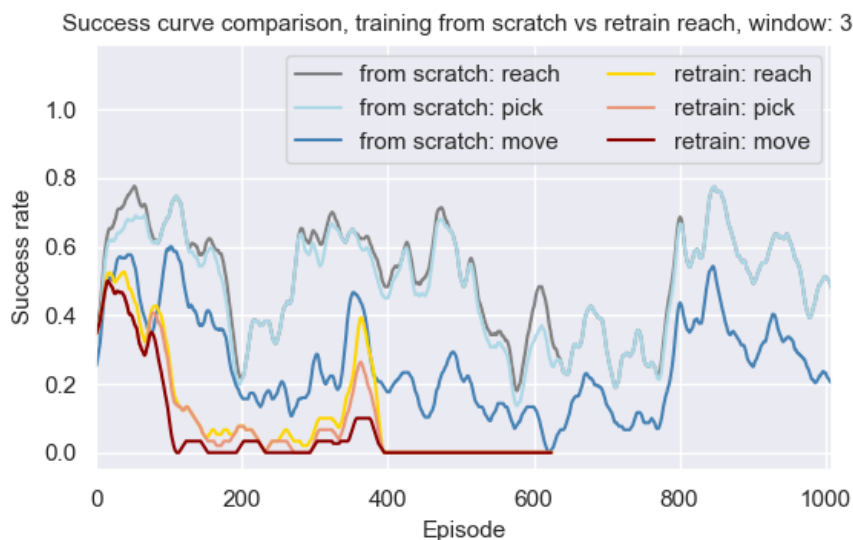


Figure 9.8: Hierarchical results- Comparison between the agent trained from scratch (blue) and the pre-loaded models (retrain) on the reaching sub-task (red)

Retraining on the picking model

From the already learnt picking to the moving sub-task, the agent needs to move the gripper again. Although the object stays open and the wrist orientation being controllable after picking, no new skills have to be learnt between these two models. The retraining results can be found in Fig. 9.9.

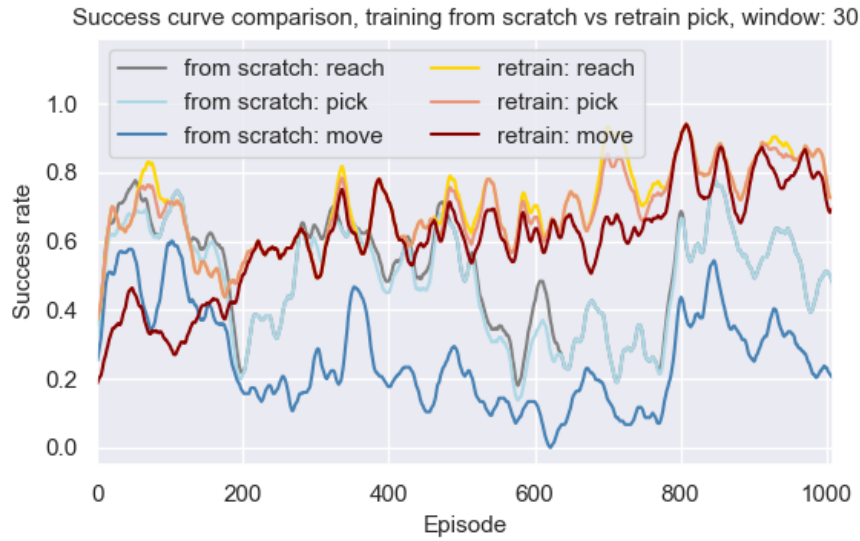


Figure 9.9: Hierarchical results- Comparison between the agent trained from scratch (blue) and pre-loaded models (retrain) on the picking sub-task (red)

First, we observe that retraining the agent does improve the performance, being more evident after episode 200. The **performance boost** is always higher than 20% after episode 200. The boost reaches 60% near episode 600.

In terms of **learning curve**, the retraining curve is steadily rising. The training from scratch shows the initial high success rate due to expert copying (episodes 0 to 200), decreases (episodes 200 to 600) and then starts improving (starting on episode 600). Given that, retraining the agent seems to give less emphasis on expert copy (there is no initial high success rate) and also seems to require less exploration (success rate does not go through a minimum phase). Regarding **convergence**, the hierarchical approach seems to converge around episode 800 with a success rate over 80%. On the contrary, the traditional agent appears more unstable and does not reach an approximately constant value.

Training the agent from scratch shows greater **separation between sub-tasks**. That gap is particularly noticeable between steps 200 and 600, where the success rate difference between moving and picking reaches 40%. However, that is different in the retraining approach. The gap between the moving sub-task and the previous two is only substantial during the first 200 episodes, with a gap of around 30%. The reason for that might be related to the **agent adaptation**. The model used in the retrain test was completing those sub-tasks nearly 100%, but the agent was not prepared to complete the moving sub-task. That confusion is clear until round 40 in Fig. 9.10, where the accuracy on the generator data (Fig. 9.10a) has spikes below 60% and the discriminator loss (Fig. 9.10b) is over 0.4. After the adaptation phase, the agent understands the sequence of three sub-goals and completes them more and more frequently.



(a) Discriminator accuracy on expert data, on generated data and overall discriminator accuracy (b) Discriminator loss, Generator loss and Generator value loss

Figure 9.10: Train Accuracy and loss, for Move retrained over Picking

Figure 9.11 gives information on the comparative **optimisation**. Following the instability of the learning curve, training from scratch does not follow a consistent decrease in the number of steps per successful episode. Instead, the blue line oscillates more frequently and the averaged curve (of a ten sample window) goes from 100 to 300 steps. The retrained model undergoes some instability at the first 100 successful episodes and decreases consistently after that. Around episode 300, there are required less than 50 steps to achieve success, which is less than the best demonstration in Fig. 9.3. So, the retrained model achieved better optimisation than the other. The red curve is also longer, which shows the absolute difference between the number of successes.

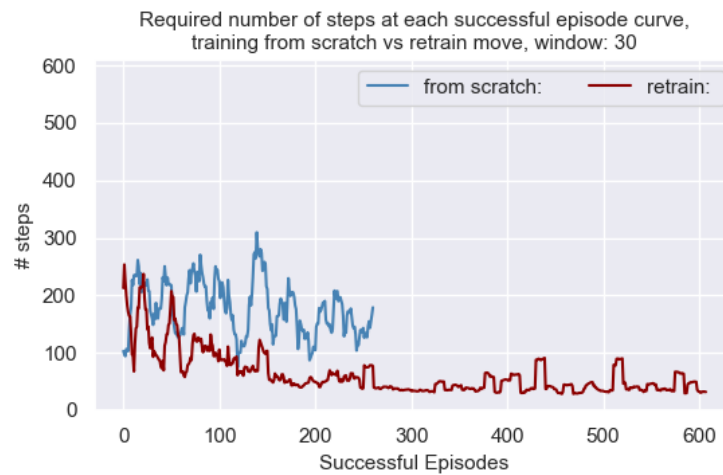


Figure 9.11: Required number of steps for expert demonstrations on the reaching task, retraining the move agent from the picking models

Nonetheless, the agent did not match the results regarding the previous sub-tasks. In Fig. 8.14, the chosen function achieves almost 100% on both reaching and picking, which does not happen.

9.4 Chapter Synopsis

On this chapter, we analysed the pick and place task which was discretised into three sub-tasks: reaching, picking and moving the object. These three steps complete the pick and place task and are the base of any other task the robotic manipulator might be requested to carry on. The task requires the skills of moving the gripper ($Y_{gripper}$) and its state (y_c).

To find the best suited reward function, the first analysis compared nine different combinations of the reward function (Tab. 9.1). All of those proposals followed the rationale of task discretization and led up to the Fig. 9.5. The best results corresponded to the function seven, presented again in Eq. 9.5.

$$r(t) = \begin{cases} 5 * \gamma_r(t) + 10 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ 18 * \gamma_m(t), & \gamma_m(t) = 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & \sum_{\tau=t_0}^t (\gamma_r(\tau) + \gamma_p(\tau)) < 1 \cap X_{object}(t) \in S_o \\ 0.5 * |X_{goal}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (9.5)$$

On the one hand, this function decreases the **sensitivity** of the object error, with a gain of $k_m = 0.5$. That is understandable that the rewards assigned after picking should not over penalise the previous achievements. The further in the task sequence, the more valuable are the actions and, thus, the agent should be less penalised. On the other hand, it is also showed that the sub-goal rewards should be **increasing**, namely, $R_r > R_p > R_m$. The partial rewards must persuade the agent to proceed to the next sub-task, instead of preferring the previous, which would not be the case if the rewards were not successively greater.

As to the **goal representation**, it is observed that the agent copes better with the object to goal representation, when compared to the goal coordinates. Also, the gradient after picking is created by that object error ($E_{o-p} = |X_{goal}(t) - X_{object}(t)|$, Eq. 9.2), which is smooth and demonstrative. This representation helps to cope with representation errors such as the encoders uncertainty, being even more evident on precision works.

The results that compared the **traditional versus the discretized approach** (Section 9.3.2) show, once again, advantages in terms of performance boost. For the sub-goal approach, the three curves are separated, which mean the agent can understand the three sub-goals separately. That can be well suited to optimise each of them individually. This methodology results in a performance boost of around 20%, after episode 800. It is also observed that, in the traditional approach, the agent tries to go directly to the goal pose, instead of picking the object first. That stresses the importance of recompensate the agent for the intermediate accomplishments.

When opting for the **model hierarchy and reusing** the previously learnt sub-tasks, there are two scenarios. The first, which uses the **reaching model** does not show advantages, similarly to what happened previously (Section 8.3.3). The model becomes unstable and rarely achieves success. However, when **reusing the picking model**, the performance boost is evident after episode

200 (increase of 20% – 60%). The learning curve is shaped differently and does not show the initial performance peak, as the agent does not need to reproduce the agent or explore that much. Instead, during the first episodes the agent adapts to the new sub-task and perfects its behaviour afterwards. Consequently, the **optimisation** is clearer and stable, where there are recorded episodes shorter than those demonstrated.

Chapter 10

Fit Sub-Task

The last sub-task is the **fitting task**, which is an additional step not considered in the typical pick and place chores. However, every robotic task can be visualised based on a pick and place task with some work afterwards, like using a tool. It represents the action of working and completing the assignment, making this task relevant. When the mission only involves moving parts around, the fit task can be automatically considered as terminated. For demonstration purposes and having in mind that this aims to build a product regardless of the uncertainties, the fit task consists of **matching the piece with the correct orientation and placement as the goal says so**. For example, when thinking of a product of a **puzzle**, it is essential to place each part in the correct location and the correct three-dimensional orientation. That is what this task represents.

Following the discretization and hierarchisation rationale, this way of modelling the task helps with training agent required to move parts around. Thus the fitting task could be **easily substituted** by another task, like insertion or screwing. That would also profit from the sub-goal reward methodology and the use of sub-task pre-trained models for training.

In terms of specific details, the object remains attached to the gripper after picking, like in the moving sub-task. Now, the agent must understand **how the gripper orientation (ϕ) works** and how it should be controlled to complete the objective.

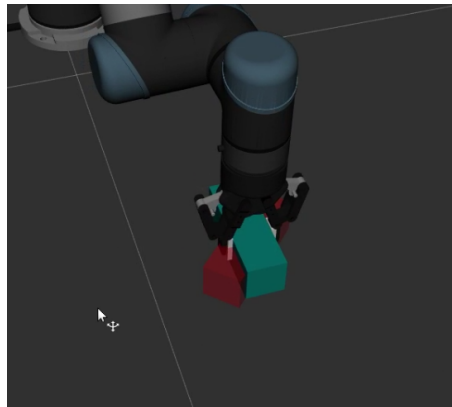


Figure 10.1: Sub-task 4 - Fitting the object. The red sub represents the goal pose and the blue one is the real object.

10.1 Reward Function

Given the better results for the sub-goal modelling and for consistency, the last reward function is derived from the moving equation (Eq. 9.3). Like the previous ones, this reward function assigns a specific reward when the fitting sub-task is completed, called R_f . The requirement to assign R_f is related to the observed variable fit (f in Tab. 6.1).

The fit measurement is calculated according to Algorithm 2. First, it is necessary to have the object dimensions, namely its width (W) and depth (D). It is also required to know the goal vertices (V_g), which are calculated based on the central point and the orientation. On each step t , it is required to observe the object's roll (θ) and pose in the xy plane. Then, object's vertices (v_i) are calculated based on its dimensions (W, D), pose ((x_o, y_o)) and roll (θ). At last, the fit is the ratio between the intersection shape's area ($A(V_{o \cap g})$) and the goal area ($A(V_g)$). The relation between areas can be observed in Fig. 10.2.

Algorithm 2: Calculate the Fit measure

```

1 get object's width  $W$ , depth  $D$  and goal vertices  $V_g$ ;
2 for  $t$  do
3   get object's roll  $\theta(t)$ ;
4   get object's pose in the  $xy$  plane  $(x_o, y_o)(t)$ ;
5    $R_{object} \leftarrow \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$ ;
6   for each vertex  $v_{oi}$  do
7      $v_{oi}^0 \leftarrow v_i * R_{object} + (x_o, y_o)(t)$ 
8   end
9    $V_{o \cap g} = V_o \cap V_g$ ;
10   $f = \frac{A(V_{o \cap g})}{A(V_g)}$ 
11 end
  
```

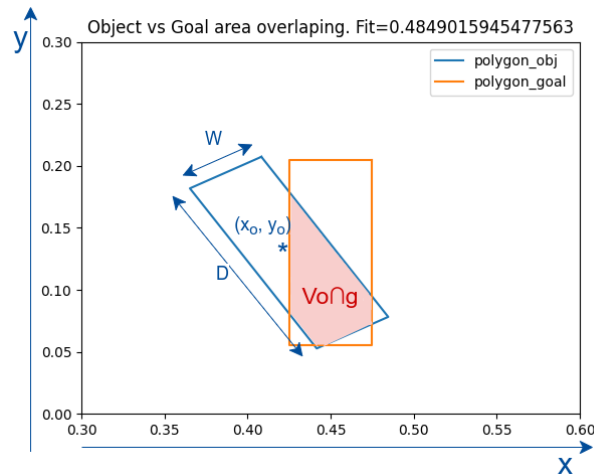


Figure 10.2: The intersection between object (blue) and desired goal (orange) is represented in red, as $v_{o \cap g}$. The object's coordinates in the xy plane are represented by (x_o, y_o) , the width by W and depth by D . This is a workspace's top view, whose x and y -axis correspond to the base frame.

So the **fit reward** (R_f) is assigned when the area overlap between the desired goal, and the actual object pose ($A(V_{o \cap g})$) is at least 75% of the goal's area. In other words, the additional reward is assigned when the fit measure is $f \geq 0.75$. This threshold value was considered to be appropriate for this task, and it was chosen empirically. The variable representing this constrain is γ_f , whose calculation respects Eq. 10.1:

$$\gamma_f(t) = \begin{cases} 1, & f = \frac{A(V_{o \cap g})}{A(V_g)} > 0.75 \\ 0, & \text{otherwise} \end{cases}, \forall \sum_{\tau=t_0}^M \gamma_f(\tau) \leq 1 \quad (10.1)$$

According to Eq. 10.1, there is no need for the other tasks to be completed beforehand. It does not disrespect the task discretisation and, in fact, tries to minimise the incoherence when the object is correctly placed, even when picking incorrectly. This aims to show the agent that not picking the object correctly will make it harder to complete the task and, thus, those episodes receive lower rewards. Nonetheless, if the reward R_m has not been assigned at the fit reward, they are assigned simultaneously. The final reward function is described at Eq. 10.2

$$r(t) = \begin{cases} 5 * \gamma_r(t) + 10 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ 18 * \gamma_m(t), & \gamma_m(t) = 1 \cap X_{object}(t) \in S_o \\ (R_f + 18) * \gamma_f(t), & \gamma_f(t) = 1 \cap \sum_{\tau=t_0}^t \gamma_m(\tau) = 0 \cap X_{object}(t) \in S_o \\ R_f * \gamma_f(t), & \gamma_f(t) = 1 \cap \sum_{\tau=t_0}^t \gamma_m(\tau) = 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & \sum_{\tau=t_0}^t (\gamma_r(\tau) + \gamma_p(\tau)) < 1 \cap X_{object}(t) \in S_o \\ k_m * |X_{goal}(t) - X_{object}(t)|, & \text{otherwise} \end{cases} \quad (10.2)$$

In the final reward function, t_0 is the initial step, $M = 600$ is the maximum number of steps for a given episode, which was incremented due to the increase in the number of sub-tasks. S_o is part of the observation space corresponding to the object's pose (Tab. 6.1, Eq. 7.3). The γ_r indicates whether the gripper reached the object successfully at that step t (Eq. 7.4), γ_p takes the value 1 when the grasping has been successful (Eq. 8.2), γ_m gives information on the object placement (Eq. 9.4) and γ_f (Eq. 10.1) signals if the object is positioned accordingly in terms of the desired area. The gain k_m corresponds to a constant that controls the object error's impact in the step reward.

According to the considerations on the sub-task modelling, each sub-task is associated with a specific positive reward. Those rewards are R_r , R_p , R_m and R_f , which correspond respectively to the reaching, picking, moving and fitting tasks. The remaining steps are assigned with the error, either the gripper or the object error. This provides the agent with smooth and consistent information throughout the entire episode and helps it converge to the product assembly.

As a clarification note, the goal pose does not change between episodes. The episode's optimisation will depend on the **episode duration**, the number of **sub-goals** completed, the **gripper error** (E_{g-o}) and the **object error** (E_{o-p}).

10.2 Building Expert Trajectories

For the fitting demonstrations, a script was created that included the following four stages:

1. reaching - includes approaching a pose near the object's centre and positioning the gripper;
2. picking the object, with a gripper error less than 2cms ;
3. moving the object towards the goal;
4. fitting the object into the goal shape, with the correct orientation, with an area overlap of over 75% .

These four tasks are represented sequentially in Fig. 10.3. The observation variables are in the rightmost trapezium, which includes the gripper pose ($X_{gripper}$), the object's error (δ_d), the object's pose (X_{object}), the gripper state (c), the grasping state (g) and the fitting measure (f). The leftmost trapezium represents the actions, namely the gripper desired positioning ($Y_{gripper}$), the gripper state (Y_c) and wrist orientation (ϕ). The steps representing the first three sub-tasks are the same as previously to make the demonstrations as continuous as possible and make it possible to retrain models over the pre-existent ones. To complete the fit sub-task, the agent needs to rotate the wrist and adjust the position accordingly.

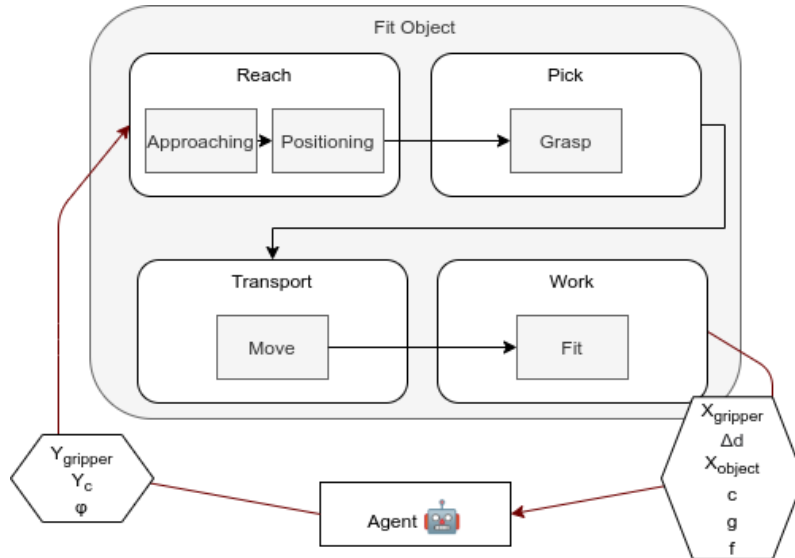


Figure 10.3: Fitting the object - reaching, picking, moving and fitting sub-tasks. The action space is represented in the left diamond box and the observation space in the rightmost one. The object is attached to the gripper during the transportation and fit phases.

Once again, the new demonstrations for this last sub-task lasted longer than the previous ones. This duration is expected, as a new sub-task was added and that cannot be completed simultaneously. The variation between demonstrations was related to the approach phase, which significantly impacted the episode's optimisation. The demonstrations lasted between 73 and 105 steps, with an average of 85 steps per episode.

The trajectory of both gripper and object are represented in Fig. 10.4. We can observe that the object trajectory is longer than the one in the move sub-task. On that, the object stopped at 5 cms from the goal. However, for a correct area fitting, the object needs to be much closer to the goal. During the object's movement, the gripper's trajectory is similar.

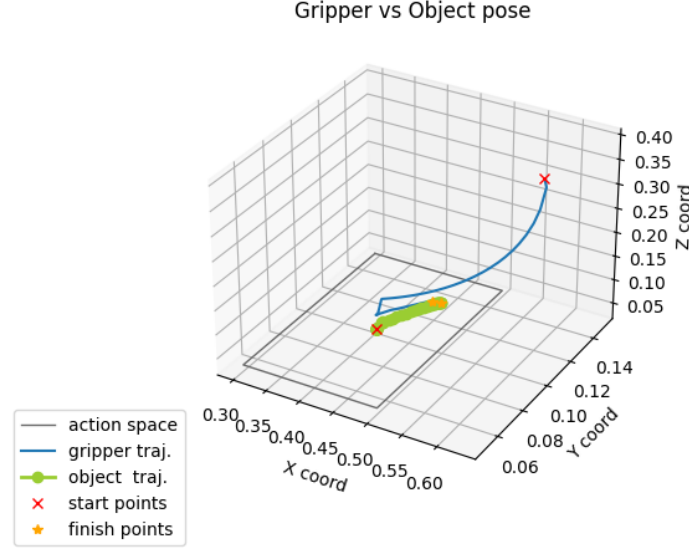


Figure 10.4: Expert demonstration 3 on the fitting task. The blue curve represents the gripper trajectory, starting on the red cross and finishing on the yellow star. That gripper's trajectory includes 3 phases: the approaching phase, according to Eq. 7.5, plus the positioning, the picking and the object moving/fitting. The object does move parallel to the gripper. The grey line defines the action space's limits, where the valid gripper points ($Y_{gripper}$) belong.

During that same demonstration, the reward and gripper state evolution is depicted in Fig. 10.5. There is an initial stage where the gripper (open) moves towards the object and the reward increases smoothly. Then, when the gripper error reaches 2cms, the agent receives the reward R_r and, after, the reward R_p for picking. There are a few steps until completing the moving sub-task, whose rewards correspond to half the object error and picks it. The fit measurement is negligible up to that step, according to the red line in Fig. 10.6. Then, from step 56 to 83, the agent must control the wrist orientation (blue line in Fig. 10.6) until the fit measure meets the threshold of 75%. The agent does not directly observe the wrist angle to avoid the transformations between the gripper and object. So, it might place the object correctly (observing the fit measurement), when controlling the gripper pose and wrist orientation simultaneously.

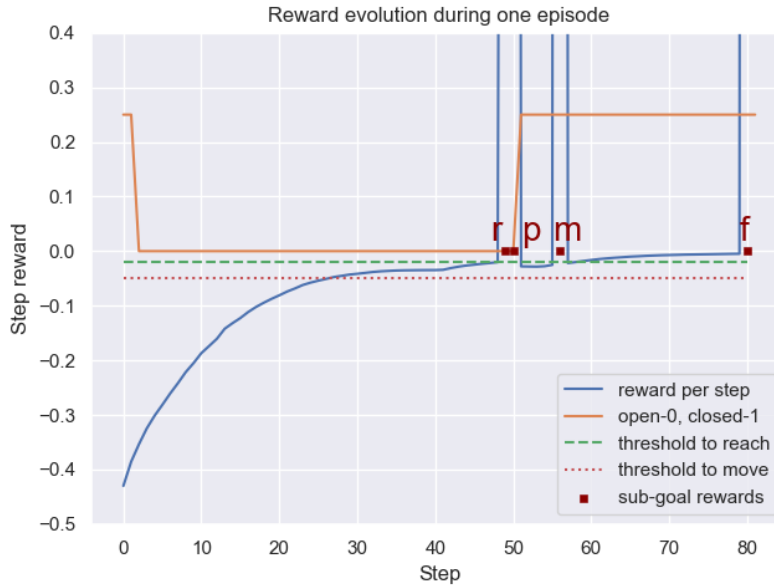


Figure 10.5: Expert demonstration 3 on the Fitting task. The blue curve represents the step rewards, the orange curve depicts the gripper state. The four points represent the sub-task rewards, respectively for reaching (r), picking (p), moving (m) and fitting (f) the object. Given the correspondence between the gripper error and the step reward for the first half of the task, the green line represents the threshold of 2cm to reaching/picking. Despite decreasing the error sensitivity on the object's error, the red line corresponds to the threshold for moving.

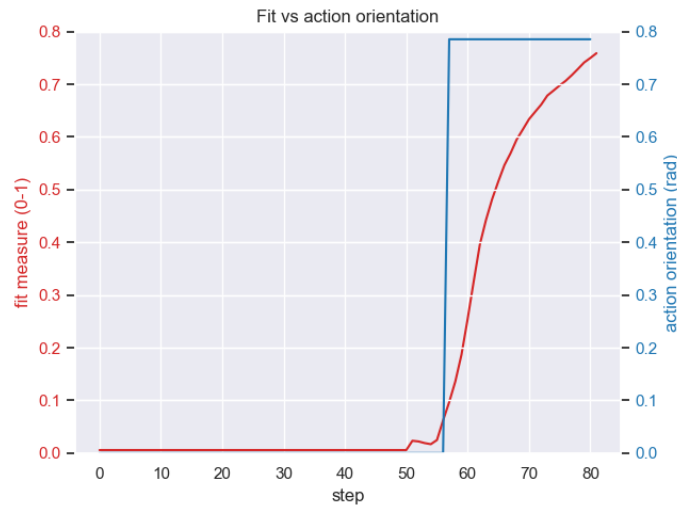


Figure 10.6: Expert demonstration 3 on the Fitting task. The graph represents the evolution of the fit measurement (red) and the orientation action (ϕ). Until receiving the reward for moving, the fit is almost zero as there's little overlap between the goal and the object. After requesting the wrist to rotate and during that movement, that measure starts increasing. The "s" shape is related to the joint [PID](#), which rotates slower when it approaches the suggested orientation.

10.3 Experiments and Results

Regarding the complete modified pick and place, the agent was required to reach, pick, move and fit the object in the goal pose. Again, we start by analysing the best suited **gains** for reward function (Eq. 10.2), derived from the moving reward function (Eq. 9.5). This gain study is carried out in Section 10.3.1. The following section, Section 10.3.2, inspects how the **model reuse** impacts the training performance.

10.3.1 Training from Scratch

Following the same strategy, the reward function for the fitting sub-task (Eq. 10.2) was adjusted experimentally. This time, four different values were tested for the R_f reward, as it was the only novelty when evolving from the moving sub-task. The experiments tested R_f successively higher, according to Tab. 10.1. The analysis between functions was based on the success curve plotted in Fig. 10.7 and individual metrics, namely the sub-task success rate, learning curve and **GAIL** training behaviour (Appendices R to U).

Proposal	R_f	Figures on Appendix ...
1	5	R
2	18	S
3	20	T
4	30	U
$R_r = 5$	$R_p = 10$	$R_m = 18$

Table 10.1: Parameters' combinations tested for the fitting task.

When analysing the comparative behaviour of the four reward functions in Fig. 10.7, it is observed that it was more difficult to achieve an acceptable success rates this time. The **first reward function** uses a low value for the last sub-goal, being the same as the reaching reward ($R_r = R_f$). For the first 800 episodes, the success rate remains low, under 20%, and the partial success rate (sub-tasks of reaching, picking and moving) also decreases, according to Fig. R.1. From episode 800 to 1300, the agent no longer repeats the expert's behaviour and moves the object without picking it. It suggests that the sum of all rewards does not compensate for the actions required to correctly fit the object in the goal. Consequently, the agent tries to minimise penalties and opts for performing only the moving and the fitting, where $R_r + R_p = 15 < R_m + R_f = 23$. However, as the agent does not pick the object correctly, it fails to fit it properly. This behaviour is then considered unstable, which is translated into the great oscillations in the learning curve (Fig. R.2). The lack of successful episodes has reduced the losses of the networks (Fig. R.3b), specifically the discriminator loss decreased below the desired 0.4. This makes the model diverge, where the generator loss starts rising and the discriminator expert is near 1, for rounds 170 onward. As a result, the agent rarely accomplishes the task.

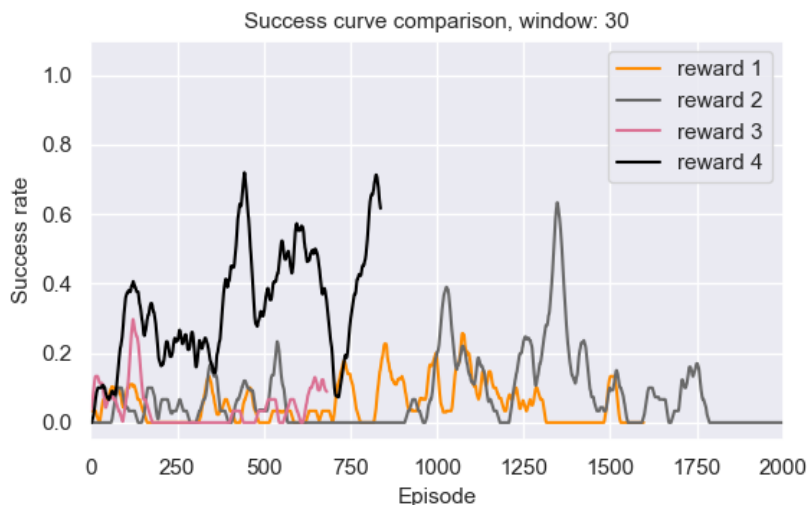


Figure 10.7: Fit - Comparison between **all reward functions**, regarding success rate curve, with ten expert demos

Then, the **second reward function** increases the fitting reward and considers it $R_f = R_m = 18$. This induced the success rate to increase over 60%, but that result came later in training, around episode 1400. Nonetheless, the results show the agent does perform better than previously but stops fitting while completing the moving nearly 70% of times (Fig. S.1). This implies the R_m is not high enough. We should also point out that the agent does not fail to reach and pick the object when correctly moves it. On the one hand, this shows the reward R_f is being adjusted in the right direction. On the other hand, it shows that the correct picking eases the fitting. In terms of the learning curve (Fig. S.2). Over episode 800, the success rate decreases to 0% and only the first three sub-tasks are performed. Consequently, the cumulative rewards decrease dramatically and the generator loss increases (after the discriminator's loss had gone lower than 0.2, Fig. S.3b).

The **third test** uses a reward value slightly higher than the one in the previous sub-task, namely $R_f = 20 > R_m = 18$. This model is not unstable as the previous ones, according to the accuracy and loss curves in Fig. T.3. Also, the learning curve (Fig. T.2) appears to be much more consistent, remaining higher than -100 and registering fewer outlier episodes. According to Fig. T.1, the agent starts exploring different combinations of sub-tasks. Despite following the rule of successively greater sub-rewards, the difference between the moving and fitting was small. This experiment shows that is not only important the absolute reward, but also the increment between consecutive sub-goal rewards. Despite the lower success rate compared to the second function between episode 200 and 700, it showed faster convergence and a peak of 30% near episode 200.

We tested a sub-goal reward of $R_f = 30$ in the **last reward function**, where the increment from moving was of 12, the higher according to Tab. 10.2. This combination corresponds to the best results, represented by the black curve in Fig. 10.7. According to Fig. U.1, after the first 400 episodes, the agent performs all four sub-tasks with the same frequency. Despite the task's

complexity, the agent could solve it with a success rate near 70% (around episode 450, Fig. U.1). The model's stability starts being an issue, as the learning curve (Fig. U.2) starts oscillating more towards the final episodes (700 onward).

$R_p - R_r$	$R_m - R_p$	$R_f - R_m$
$10 - 5 = 5$	$18 - 10 = 8$	$30 - 18 = 12$

Table 10.2: Sub-goal reward increments on the four sub-tasks

Having said so, we have opted for the fourth reward combination. Despite the concern about choosing rewards successively higher, we must retrain that the **reward increments** appear to impact training performance.

10.3.2 Training from Scratch vs from Pre-learnt Tasks

After adjusting the gains in the reward equation (Eq. 10.2), we performed two retrain tests to complete the conclusions regarding the picking and moving tasks. The reach model was not considered, as the previous results showed no advantages. Here, we have tried to teach the agent how to fit the object, reusing the models on the pick and the move tasks.

Retraining on the picking model

When using the picking model, we have tried to understand how the agent would react when there were two sub-tasks between the learnt procedures and the desired behaviour.

According to Fig. 10.8, the agent appears to diverge and forgets how to perform any of the sub-tasks. At episode 800, the agent is picking the object only 10% of times. This suggests the gap between tasks is undesirable in this situation and causes information loss.

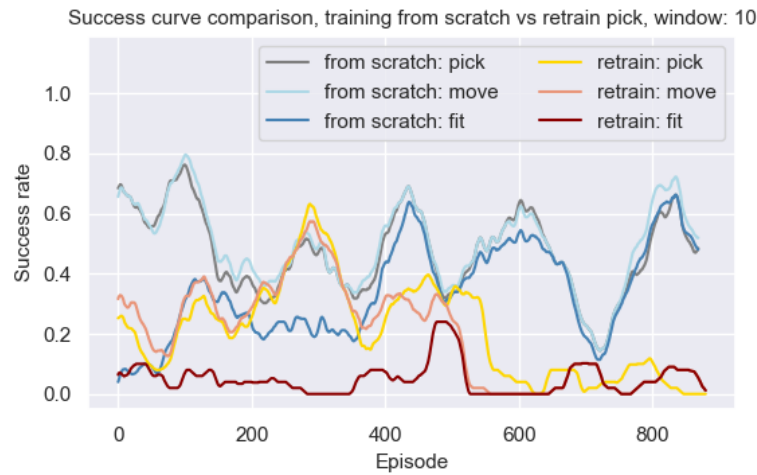


Figure 10.8: Hierarchical results - Comparison between the agent trained from scratch (blue) and the pre-loaded models (retrain) on the picking sub-task (red)

Retraining on the moving model

This time, the experiment uses the moving model to try to complete the object fitting. These sub-tasks are sequential, so there are no other sub-tasks in between like in the previous experiment.

As observed in Fig. 10.9, the agent takes longer to learn the difference between moving and fitting the object and appears to do so after episode 400. Until then, the agent is adapting to the task changes, as it did in Section 9.3.3. That adaptation is suggested by the separation between curves and the decrease in the moving frequency. After, the agent starts learning the sequence of sub-tasks and the success rate begins to increase, from 10% to 40% at episode 800. Nonetheless, the retrain behaviour (red lines) appears more stable than in the training from scratch (blue lines).

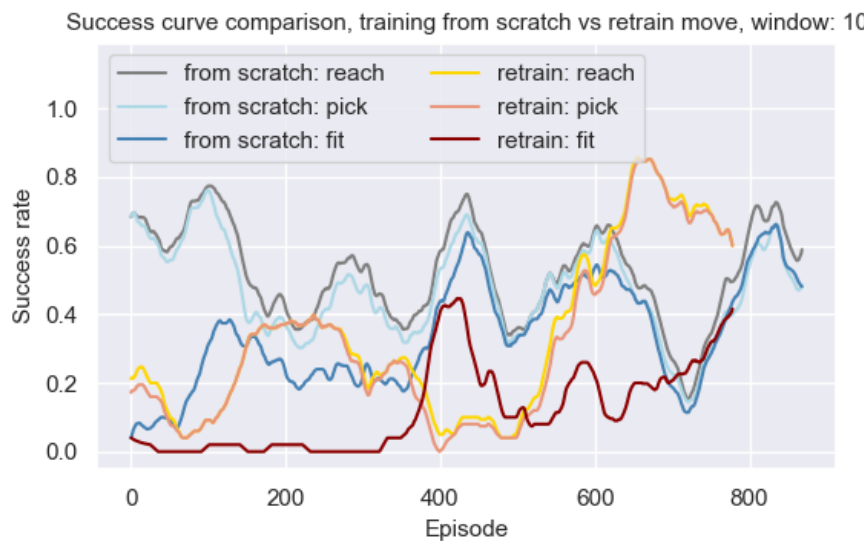


Figure 10.9: Hierarchical results - Comparison between the agent trained from scratch (blue) and the pre-loaded models (retrain) on the moving sub-task (red)

The curve's stability seems to come from the fact the agent had already explored many episodes. Consequently, it has an "idea" of the desired behaviour, despite not achieving it perfectly or with a high success rate. Regardless of the curve smoothness, the adaptation phase (the first 400 episodes in this case) seems to be related to "forgetting" incorrect information and acquiring the correct data. In fact, the agent could control the action space until then, but moving the wrist had little impact on the other sub-tasks. As a consequence, the agent retrained could have registered non-sense or erroneous information associated with that component.

10.4 Chapter Synopsis

In this Chapter, it was analysed the full modified pick and place task. To complete the task, the agent needs to control the **wrist orientation** after moving the object correctly. With the reward function on the move task as the starting point, the first experiment chose the best-suited gains for the dedicated sub-goal reward, R_f . The values tested (Tab. 10.1) analysed rewards smaller and greater than the previous sub-task rewards. The best results were achieved with the reward of $R_f = 30$, and the final equation is in Eq. 10.3.

$$r(t) = \begin{cases} 5 * \gamma_r(t) + 10 * \gamma_p(t), & \gamma_r(t) + \gamma_p(t) \geq 1 \cap X_{object}(t) \in S_o \\ 18 * \gamma_m(t), & \gamma_m(t) = 1 \cap X_{object}(t) \in S_o \\ (30 + 18) * \gamma_f(t), & \gamma_f(t) = 1 \cap \sum_{\tau=t_0}^t \gamma_m(\tau) = 0 \cap X_{object}(t) \in S_o \\ 30 * \gamma_f(t), & \gamma_f(t) = 1 \cap \sum_{\tau=t_0}^t \gamma_m(\tau) = 1 \cap X_{object}(t) \in S_o \\ |X_{gripper}(t) - X_{object}(t)|, & \sum_{\tau=t_0}^t (\gamma_r(\tau) + \gamma_p(\tau)) < 1 \cap X_{object}(t) \in S_o \\ k_m * |X_{goal}(t) - X_{object}(t)|, & otherwise \end{cases} \quad (10.3)$$

The results obtained in this section support two conclusions regarding the modelling of a sub-goal shaped task. The first is that the **sub-goal rewards must be increasingly bigger**, to encourage the agent to move on to the following sub-tasks. The second conclusion is that the absolute value is not the only factor. It is crucial to choose **reward increments successively greater**. The relation between increments could be linked with the relative cost to complete the next sub-task and to the cost-benefit perceived by the agent.

Regarding the retrain results, there was observed that **no advantages come from using models on non-consecutive sub-tasks**. In fact, reusing the pick model on the fitting task required much exploration and adaptation that delayed the convergence. On the other hand, reusing the move agent on the fitting task showed some advantages. Like what happened with the agent for the move sub-task retrained over the picking mode, it seems that **reusing models** results in **smoother behaviours** and **more stable success curves**. However, this last task required one new skill that has **not yet been acquired**, which **delayed the adaptation phase** and did not boost training performance. When retraining the models, there is a **trade-off** between the **exploration** required to acquire all the skills and the **adaptation** between incomplete mapping from actions to observations. In this case, the trade-off was not positive.

Chapter 11

Conclusions and Future Work

Complex tasks regarding robotic manufacturing processes hardly converge to acceptable results. When using intelligent approaches, the solutions become more robust, but their complexity remains a barrier to success and usually lack in terms of transferability to real implementations.

In terms of problem analysis and solution surveying, this project started by **studying** the definition of **RL** formulation, how **LfD** can boost performance and how useful can demonstrations be for a training agent. The analysis also included the situation where the human does not need to explicitly model the reward function, namely **IRL**, and how hierarchical structures can be applied to upgrade training techniques. Then, we opted to use the **GAIL** algorithm, an **IRL** algorithm that combines **RL** techniques with an expert, shortening exploration by directing the agent towards the desired actions following the expert's behaviour. **GAIL** is classified as a **model-free** apprenticeship learning technique and seeks to **scale** into high dimensional problems.

With this information, we could answer the first research question: *Is it possible to program a robot within a manufacturing environment, to perform its tasks using demonstrations and **IRL**?* The answer is affirmative, there are several examples of **LfD** algorithms applied to robotic tasks, such as the one employing the **GAIL**, by Ho and Ermon [42]. In fact, the **LfD** approach seemed to **catalyse the training performance**, producing better results overall.

In this project, we have **created a simulation scenario** for a modified pick and place task while testing a new way of modelling a task. The simulation scenario uses a UR5 industrial manipulator, a Robotiq gripper and a parallelepipedic object. The main goal is to teach the agent how to reach, pick, move and fit (sequentially) the object in the goal pose and with the correct orientation. The modelling considers the task accomplished when the goal and the object areas on the *xy* plane overlap more than 75%, simplifying the manipulation of non-homogeneous parts. Thus, the case study for hypothesis testing was also completed.

The **implementations** comprised developments on the **low-level**, in **ROS** and Gazebo. The **high-level** implementations followed the Open AI Gym framework to allow bench-marking between environments and **RL** algorithms. The bridge between these two components is the robogym framework [61], which eases simulation and real-world situations transferability. Choosing

robo-gym as the simulation framework seemed to be the best solution among the available simulators because it joined the advantages of the **RL world** and the **real world**.

The following objective was to define a set of **guidelines to discretise a task** into smaller sub-tasks and how to model that under the **RL** scope. The tests were divided into **four parts**, each of them consisting in adding the **new sub-task**, sequentially. For each sub-task, we have defined tests to choose the **best reward function** implementation, we have **compared** the sequential approach to the sub-goal based reward, and also, **compared** that with retraining models. Each training used appropriate demonstrations, which consisted in commanding the agent by manually sending the gripper coordinates, state and wrist orientation. The process of **recording demonstrations** was done with a script and involved saving the data in the specific formatting. The process of acquiring the demonstration can be simplified even more, making it easier for a human operator to hand guide the gripper, while connecting it to the simulation environment to save that information. So, it is safe to answer positively to the second research question: it is possible to demonstrate the manipulator's desirable behaviour, and it can be done flexibly with human supervision.

The third research question seeks a methodology to make the intelligent agent understand **discretisation**, for which we proposed the sub-goal shaped reward functions. The **task division** is related to different skills or different action control. The first thing to highlight is that sub-goal-shaped tasks are **natural** to humans and helps to craft the sequence of actions required. For a **dense** reward function, there must be a **base reward** with a behaviour similar to a gradient, such as the gripper or the object error. The sub-goals must be associated with **positive rewards**, increasingly higher as the order of the sub-goal increases. The sub-goal reward should only be applied once per episode to avoid repetitive actions by the agent to accumulate reward. The **magnitude** of the sub-goals should make the **reward range** as smaller as possible to promote optimisation. Regarding **penalties**, those must not prevent the agent from exploring desirable behaviours and cannot disrupt the gradient created by the base reward. Besides reward magnitude, we found out that **reward increments** are also important to the training performance. The difference between the previous sub-goal reward and the current must be greater than the previous difference. It can be linked to the relative cost associated with adding a new sub-task and extending the episode. The episode classification according to sub-goal completion is a crucial factor in improving the results and the respective reward range, as in "first sub-goal" or "all sub-goals" completed.

We were also able to verify that sub-goal-based reward functions show **better performance** than sequential modelling, in which the main approach is related to training convergence — the more sub-tasks, the greatest the improvement. Although a pick and place was used as the case study for both the reward guidelines and the sub-goal approach study, we believe these considerations also apply to other situations. These preliminary results can be used in problems with **different dimensions and complexities** if those can be **discretised** into smaller sub-tasks. For example, insertion or screwing tasks can take advantage of this sub-goal-based methodology when employing reward-based algorithms (**RL** scope).

Parallel to the task discretisation, we aimed to study if it was possible to **reuse models** with previously learnt concepts, according to the last research question. This feature can be tremen-

dously helpful when changing task configuration or skills' order. The tests compared the sub-goal results in an agent with no previous skills and the agent knowing how to perform previous sub-tasks. The results showed both advantages and disadvantages, which suggested an important trade-off. It seems that **reusing models** results in **smoother behaviours** and **more stable success curves**. However, when the new sub-task requires a new skill that has **not yet been acquired**, the **adaptation phase** takes longer. This adaptation phase corresponds to **GAIL**'s network adjustments necessary to adapt the interpretation of the actions yet unlocked. When this phase takes too long, there is no boost in the training performance. The trade-off is thus between the **exploration** required to acquire all the skills and the **adaptation** between incomplete mapping from actions to observations.

Specifically, when training the reach agent to the picking task, it needs to understand how to close the fingers. That action is antagonistic: on the one hand, closing the fingers is necessary to pick the object, but it will prevent the agent from reaching the object correctly and will create new undemonstrated situations. While retraining an agent brings no advantages in that situation, there was a clear improvement when the pick agent learns how to move the object. The advantages in performance are clearer when no new skills are required, like reaching and moving the object, which only needs positioning control. In that situation, the **GAIL** networks have been pre-adjusted in terms of weights, as expected.

Scientific Contributions

The resulting analysis and conclusions of this work led to two scientific paper submissions. The first contribution was related to the sub-goal methodology, which was proposed to the 20th International Conference on Advanced Robotics (<https://icar-2021.org/>). At this date, we are waiting for the results of acceptance, due in September 30, 2021. The second publication, regarding the retrain analysis, is ongoing and soon to be submitted to one of the following conferences:

- 2022 30th Mediterranean Conference on Control and Automation (MED)
- 2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)

Moreover, the implementations on the robo-gym framework were made available to the JOAN-NEUM RESEARCH - Robotics. The developments regarding the Robotiq gripper resulted in a pull request to the public repository.

Future Work

As future work, we intend to:

- use different reset and goal poses to validate generalisation;
- test the model, created in simulation, with the real UR5, assessing transferability to the real world;

- integrate different tasks such as screwing, and insertions to study the impact of the sub-goal methodology and retraining on different scenarios;
- try different [RL](#) based algorithms to understand in which extend does the conclusions scale to other methodologies;
- use several and different pieces to build a complex product, in which optimisation can have greater impact.

Appendix A

Training specifications

machine	CPU	GPU	RAM
Laptop	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz	Intel UHD Graph- ics 620	8.3 GB
Desktop	Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz	NVIDIA GeForce RTX 2080 Ti	16.7 GB

Table A.1: Machine's Specifications

General Configurations		
num_vec	Number of environments in VecEnv, must evenly divide gen_batch_size	1
parallel	If True, then use DummyVecEnv. Otherwise use SubprocVecEnv	True
max_episode_steps	If not None, then environments are wrapped by TimeLimit so that they have at most max_episode_steps steps per episode	500/600
Discriminator Net		
reward_type	If provided, then load the serialized reward of this type, wrapping the environment in this reward. This is useful to test whether a reward model transfers	DiscrimNet
expert_batch_size	Number of expert samples per discriminator update	60
n_disc_updates_per_round	Number of discriminator updates after each round of generator	4
Generator Net		
policy_class	two hidden layers of size 256	base.FeedForward256Policy
batch_size	PPO batch size	256
gae_lambda	Lambda for GAE-Lambda, for calculating the advantages of state-action pairs	0.95
gamma	Discount factor (1-1/H)	= 1-1/500
n_epochs	Number of epochs of interaction (equivalent to number of policy updates) to perform	200
ent_coef	Entropy regularization coefficient	0
learning_rate	learning rate used for all networks, it can be a function of the current progress remaining (from 1 to 0)	0.001

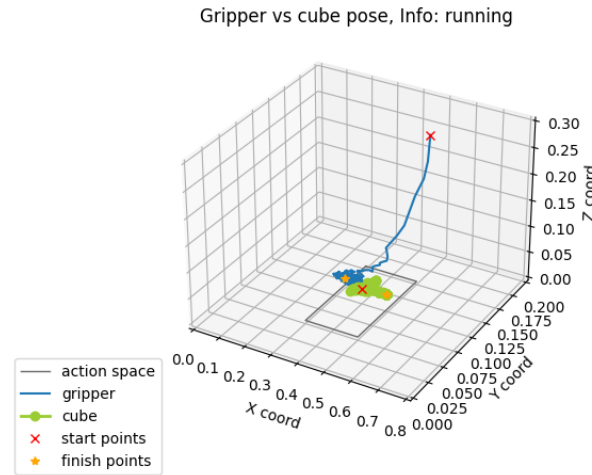
Table A.2: [GAIL](#) Training Parameters

Appendix B

Pick Results - Reward Function 1

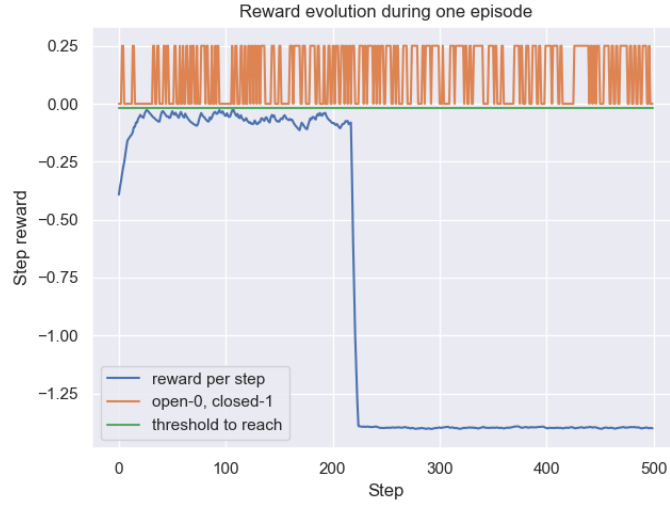


(a) Reward Function 1, Episode 167. Reward distribution per step (blue curve), when neither reaching or picking were successful. In this case, the reward maps directly into the distance between the gripper and the object's centre. The green line represents the threshold of 2cms, which the reward must hit for the agent to receive either R_r or R_p . The gripper state, given by the observation space variable c , is depicted by the orange line and provides information if the gripper is either open(low)/closed(high). As the open state comprises 25% of the possible finger's distance and because the fingers joints also have a PID controller, the constant permutation of the state c results in the fingers having a gap lower than the object's width. Moreover, the continuous oscillation suggests the agent was still exploring the gripper's state functioning, at that stage: the mapping between the continuous action and the actual boolean state, the influence of the finger's state on task.

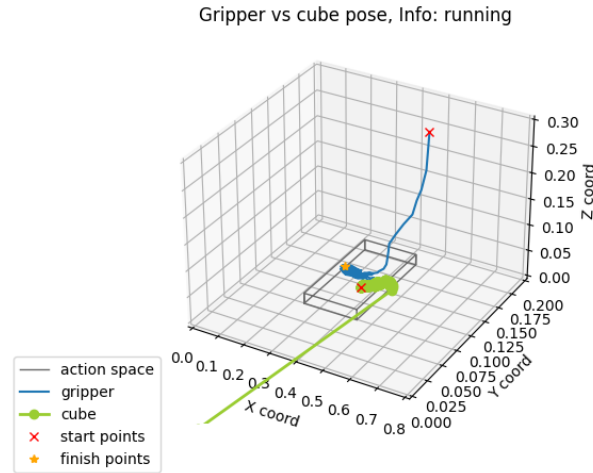


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. Both follow the direction of the x-axis, being the gripper always at the left of the object, suggesting the gripper is pushing it. The final poses are more than 2cms apart (approximately 15 cms), so none of the sub-tasks was completed. The object did not leave the action or observation space, so the agent did not receive additional penalties.

Figure B.1: Example on the reward proposal 1, when no reaching or picking is performed. This situation corresponds to the **episode 167** of the training, suggesting the gripper pushed the object away while closing too early. The cumulative reward for this episode is -54.485 , and the episode is classified as **case A**



(a) Reward Function 1 and Episode 314. The reward function and the coordinates' log suggest that the cube exited the action space at step 218, where there's an abrupt decrease in the step reward. According to the gripper state plot, the gripper was closed at that time, making it impossible to position the gripper at the object's centre. The gripper starts approaching the object and gets close to it around step 25 as the reward line approaches the threshold line. The behaviour until that step is similar to the observed during demonstrations. However, this episode accumulates such low rewards because of the rewards received from step 218 and onward. The object ends up in the position $X_{object} = (0.44040322; -1.29283178; 0.02397567)$. On the other hand, the gripper state is still not fully understood by the agent at this stage because it oscillates between open and closed, at a high frequency.

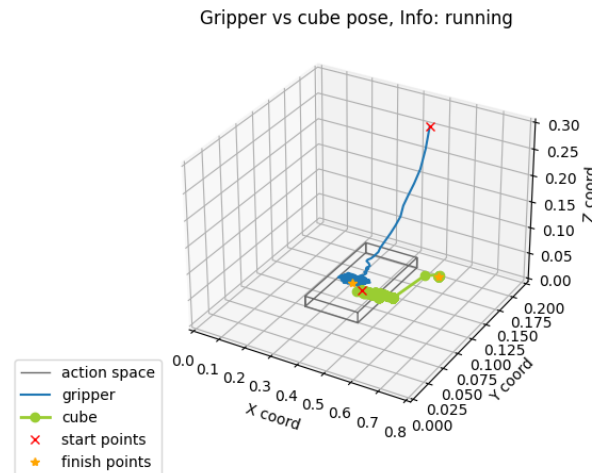


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object trajectory shows the object exiting the action space as the line extends to points $y < 0.01$. According to the blue line, the gripper collided with the object from behind, which suggests a wrong approach. The object did not leave the observation space. Hence the agent does not receive additional penalties

Figure B.2: Example on the reward proposal 1, when the object exits the action space by colliding with the gripper. The object flies away but remains inside the observation space. This situation corresponds to the **episode 314** of the training, suggesting the gripper projected the object by performing a wrong approach. The cumulative reward for this episode is -407.333 , and the episode is classified as **case B**



(a) Reward Function 1 and Episode 4. The reward function and the coordinates' log suggest that the cube exited the action space at step 271, ending the episode at $X_{object} = (0.58185959; 0.15100859; 0.02397567]$. The gripper starts approaching the object and gets close to it around step 30 as the reward line approaches the threshold line. This episode does not receive rewards as lower as the previous (Fig. B.2a) because it ends closer to the action space's limits.

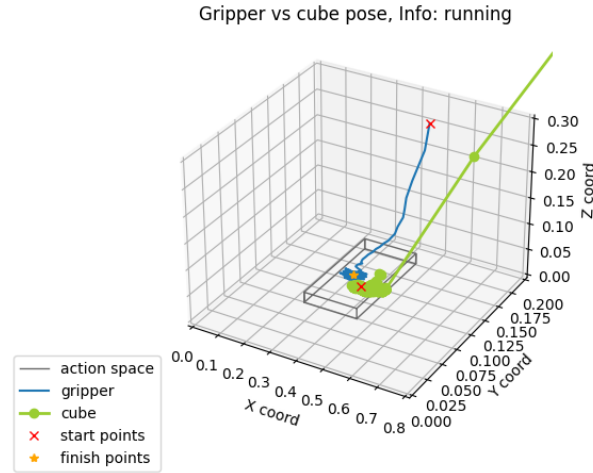


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object trajectory shows the object exiting the action space as the line extends to points $x > 0.5$. According to the blue line, the gripper collided with the object from the left, which suggests a wrong approach. The object did not leave the action or observation space. Hence the agent does not receive additional penalties

Figure B.3: Example on the reward proposal 1, when the object slides away and exits the action space. This situation corresponds to the **episode 4** of the training, suggesting the gripper projected the object by performing a wrong approach. The cumulative reward for this episode is -62.982 , and the episode is classified as **case B**

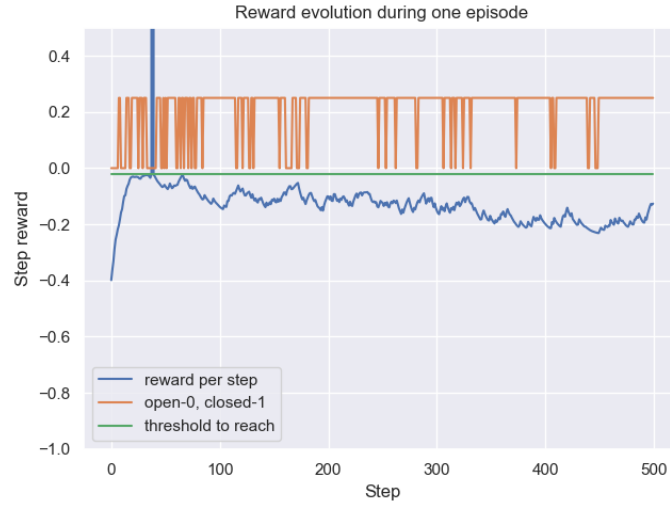


(a) Reward Function 1 and Episode 149. The reward function suggests that the cube was projected at step 201 due to the abrupt decrease in the reward (P_o). The gripper starts approaching the object and gets close to it around step 25 (the blue line approaches the green line), similar to the demonstration provided. Then, the gripper closes (orange line is high) without completing the approach to at most 2cms from the object centre. That created an undemonstrated situation, where the object was not at the starting position. The agent tried to solve that new situation, but with no success. The episode ended earlier due to a collision with the object projected out of the observation space, receiving the final reward of -10 .

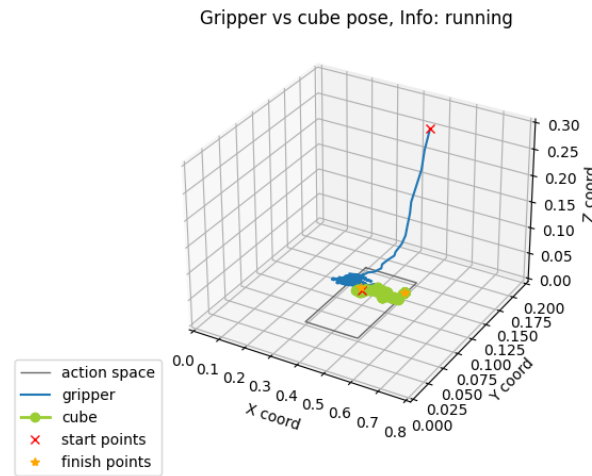


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object trajectory shows the object exiting the observation space as the green line depicts the projectile behaviour. However, the gripper trajectory is close to the demonstrated behaviour at the beginning, until step 25.

Figure B.4: Example on the reward proposal 1, when the object is projected out of the observation space. This situation corresponds to the **episode 149** of the training, suggesting the gripper performed a wrong approach. The cumulative reward for this episode is -27.715082 , and the episode is classified as **case C**. Episodes like this happen to introduce cumulative errors in the encoders by engine stalling.

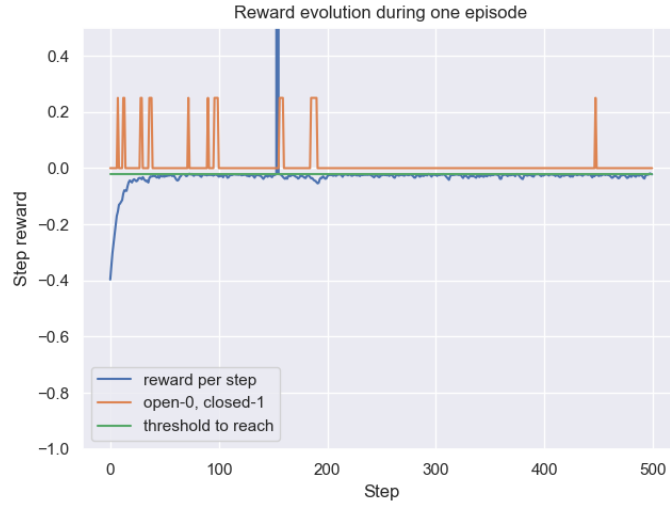


(a) Reward Function 1 and Episode 292. The gripper's fingers (orange line) seemed to close when positioning the gripper at one end of the object, moving it away at several instants (blue curve has horizontal segments). As the step reward approaches -0.2 , it is suggested the object was also pushed to the side, towards action space's limits

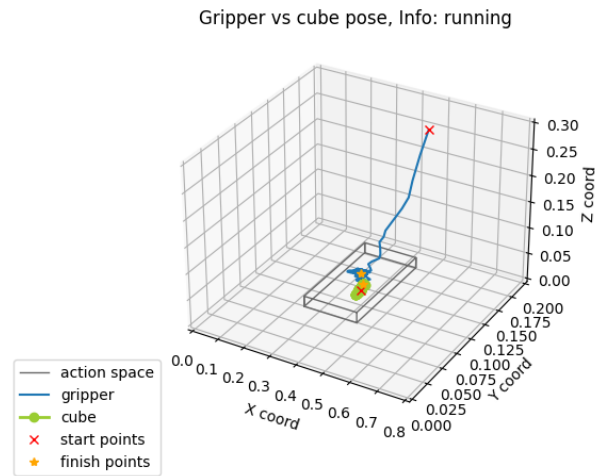


(b) The blue line overlaps the green line, which supports the premise of a successful reach. However, the cube finishes away from the gripper (star points are separated), meaning the gripper interacted with the object by pushing it away, undermining the picking subgoal.

Figure B.5: Example on the reward proposal 1, when the gripper successfully reaches the object but fails to pick it by pushing and moving the object around. This situation corresponds to the **episode 292** of the training. The cumulative reward for this episode is 33.480721, and the episode is classified as **case D**

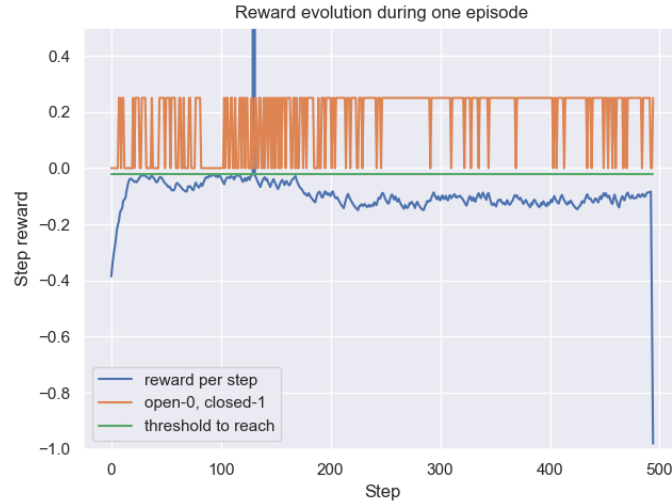


(a) Reward Function 1 and Episode 506. The agent successfully positions the gripper near the object centre for the entire episode, as the reward remains near the green line. Nonetheless, the agent hesitates to close the gripper, minimising the probability of achieving the goals.

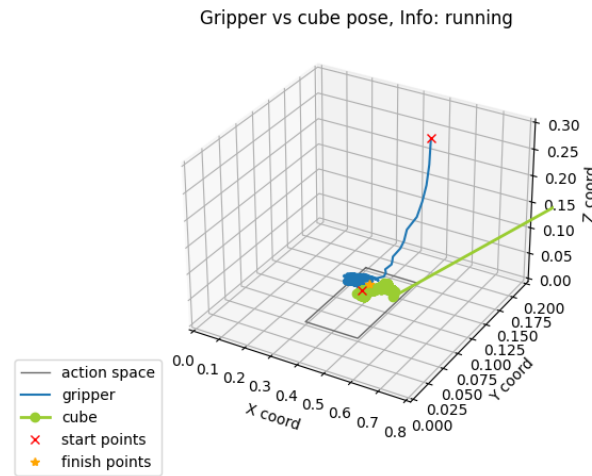


(b) Gripper approaches the object as both lines intersect, completing the reach. However, by analysing the episode logs, it is clear that the agent remained slightly above the accepted positions ($z_g > z_o$) the gripper error was greater than 2cms from the object's centre. When at the accepted points, the agent failed to close the fingers.

Figure B.6: Example on the reward proposal 1, when the object is reached, but the agent fails to grasp it by hesitating to close the fingers. This situation corresponds to the **episode 506** of the training, whose cumulative reward is 83.664498 and the episode is classified as **case D**.



(a) Reward Function 1 and Episode 382. The agent succeeds in reaching the object, even if doing so later in the episode (around step 130) when the reward received is positive. Nonetheless, the decrease in the reward value suggests the object was pushed away and the episode culminated in a dangerous collision, which is suggested by the final reward of -10 .

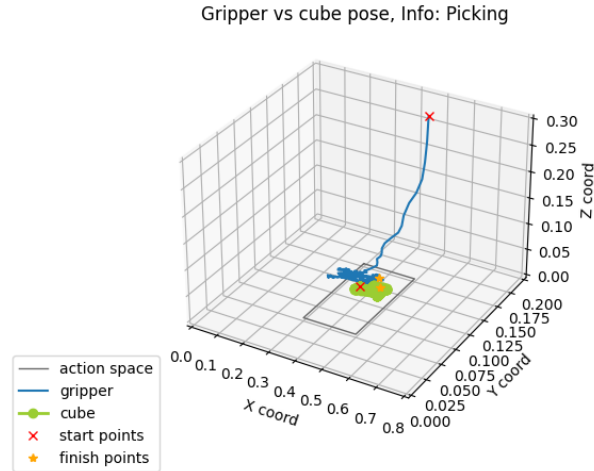


(b) Gripper trajectory overlaps the object trajectory suggesting the reach was successful. However, the green line spreads to the right suggesting the cube was pushed to the side with previous encounters with the gripper from the left side. At the end of the episode, the object is clearly projected out of the observation space, and its finish point is not visible.

Figure B.7: Example on the reward proposal 1, when the object is reached, but the agent causes a collision that results in the penalisation of -10 at that final step (step number 496). This situation corresponds to the **episode 382** of the training, whose cumulative reward is 41.80532 and the episode is classified as **case D**.



(a) Reward Function 1 and Episode 370. The agent reaching the object at step 248 and picking the object at step 351. The agent moves the gripper near the object at an early stage but is not successful. This episode demonstrates where the agent is not optimised, and there are around 100 steps of slack between reaching and picking. The main reason for the episode to last longer might be related to the alternation between gripper's fingers states, moving the cube to different positions



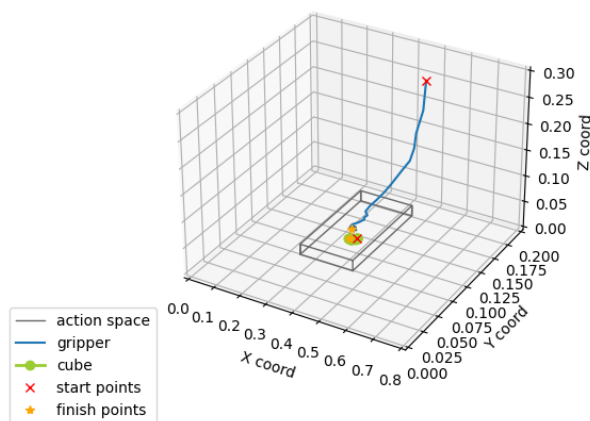
(b) Gripper trajectory overlaps the object trajectory suggesting the reach was successful. Even though the gripper performs many exploration steps, mainly in the x-direction, the agent can overcome the problem of moving the object. Although this is not an optimised episode, it demonstrates generalisation by picking the object at a new and undemonstrated position.

Figure B.8: Example on the reward proposal 1, when the agent succeeds in reaching and picking the object. The tasks are completed with approximately 100 steps of interval. This situation corresponds to the **episode 370** of the training, whose cumulative reward is 221.63355.



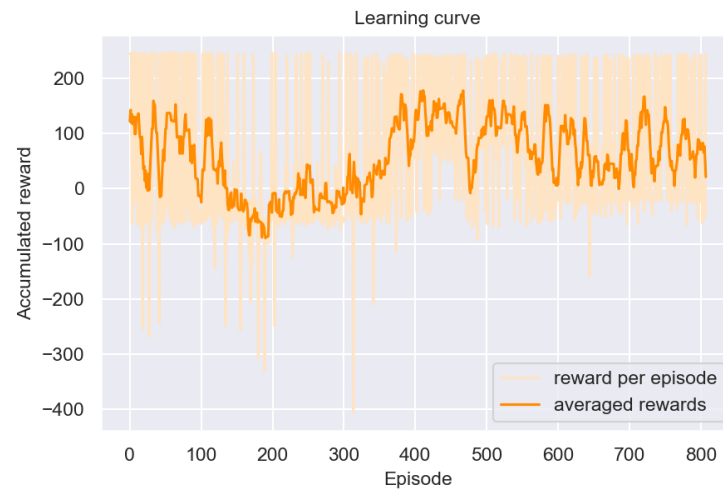
(a) Reward Function 1 and Episode 64. The agent succeeds in reaching and picking the object simultaneously. This behaviour is optimised when compared to the expert behaviour that performs those in consecutive steps

Gripper vs cube pose, Info: Reaching+Picking

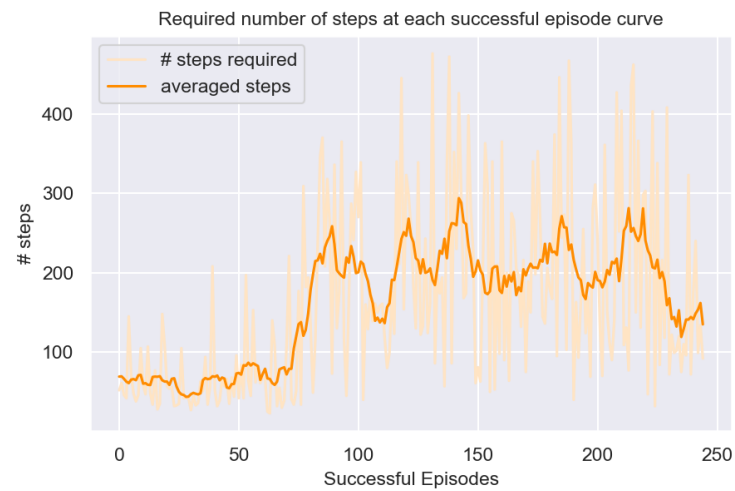


(b) Gripper trajectory overlaps the object trajectory and both finishing points match. This gripper trajectory is one of the smoothest recorded during training

Figure B.9: Example on the reward proposal 1, when the agent succeeds in reaching and picking the object. The tasks are completed simultaneously. This situation corresponds to the **episode 64** of the training, whose cumulative reward is 247.111281.



(a) Evolution of the cumulative reward per episode.



(b) Evolution of the required steps to achieved success.

Figure B.10: Pick- Training behaviour for Reward proposal 1



(a) Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 1

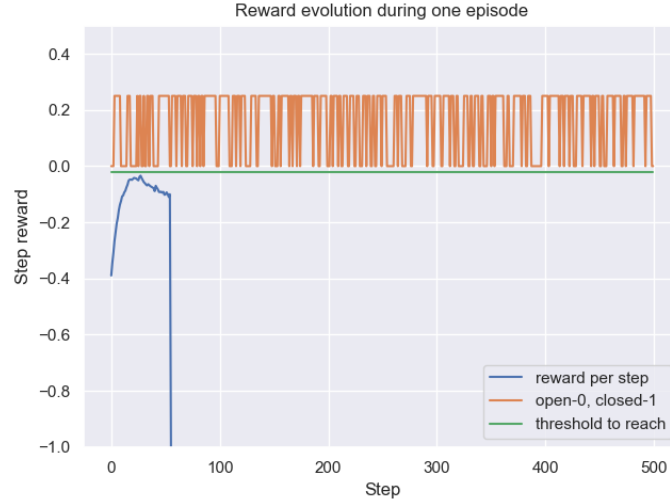


(b) Discriminator loss, Generator loss and Generator value loss, for Reward proposal 1

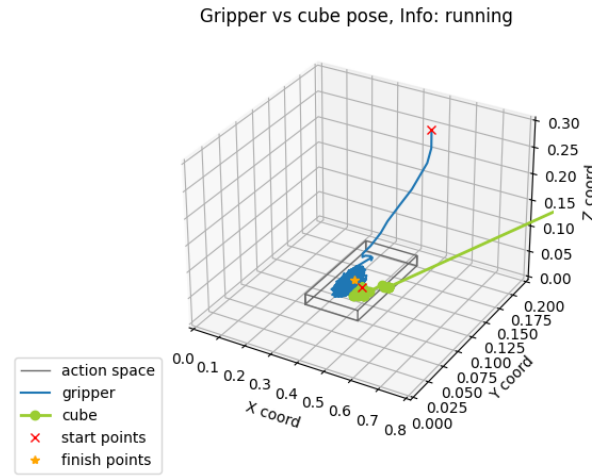
Figure B.11: Pick- GAIL training behaviour for Reward proposal 1

Appendix C

Pick Results - Reward Function 2

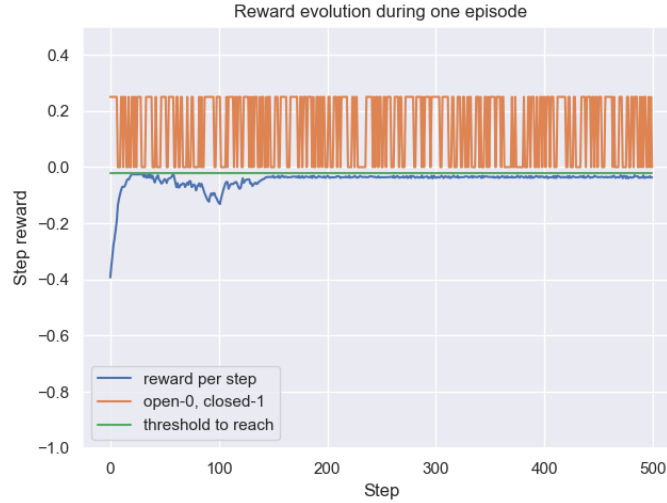


(a) Reward Function 2 and Episode 4. The reward function suggests that the cube was projected around step 50 due to the abrupt decrease in the reward. This time the step reward is not bounded and decreases to a value of a different magnitude (-5.656548). The episode does not end earlier, which is the main difference between reward functions 1 (Eq. 8.3) and 2 (Eq. 8.7).

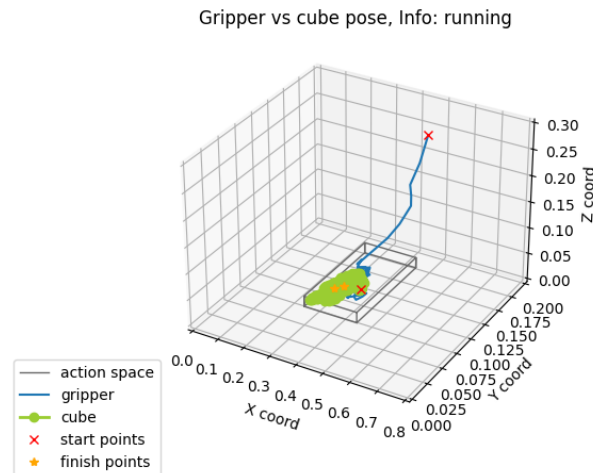


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object trajectory shows the object exiting the observation space as the green line depicts the projectile behaviour. The gripper trajectory suggests a lateral contact between them caused the collision.

Figure C.1: Example on the reward proposal 2, when the object is projected out of the observation space, and the episode does not end earlier. This situation corresponds to the **episode 4** of the training, suggesting the gripper performed a wrong approach. The cumulative reward for this episode is -2511.552437 , and the episode is classified as **case C**. Episodes like this happen to introduce cumulative errors in the encoders by engine stalling.



(a) Reward Function 2 and Episode 9. The reward function suggests that the cube was grasped around step 150 and despite opening and closing the fingers, the object remained attached to the gripper. This is the result of the high frequency of changing gripper state and the **PID** effect. Additionally, the Gazebo plugin keeps the object grasped for those fractions of time. Thus, the gripper error is approximately 5cm and constant for the rest of the episode.

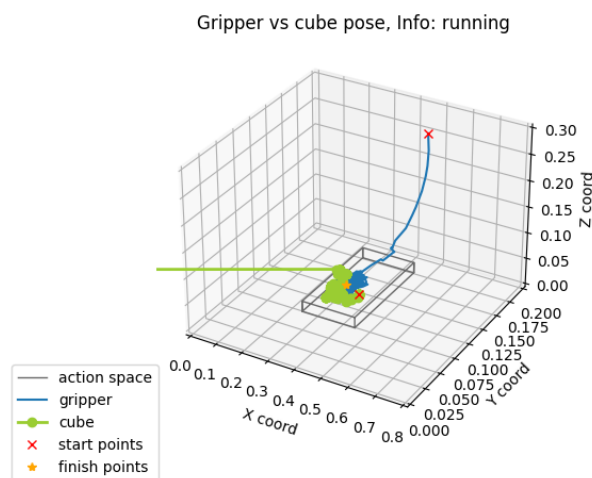


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object and gripper trajectory intersect along with several points, and the final points are much closed. Thus, this supports the idea that the object is grasped for the majority of the episode, at a wrong pose, which prevents the agent from achieving success.

Figure C.2: Example on the reward proposal 2 on **episode 9**, when the object remains grasped for the majority of the episode. This is a combination of 3 factors: the high frequency when changing gripper state, the effect of the **PID** in the finger joint and the Gazebo grasp plugin. The cumulative reward for this episode is -22.639519 , and the episode is classified as **case D**.



(a) Reward Function 2 and Episode 165. The reward function suggests that the cube was reached around step 30. Then, the gripper moved past the object centre and collided with it, around step 170. The following rewards were -11.25 , which ended up penalising the good cube approach.

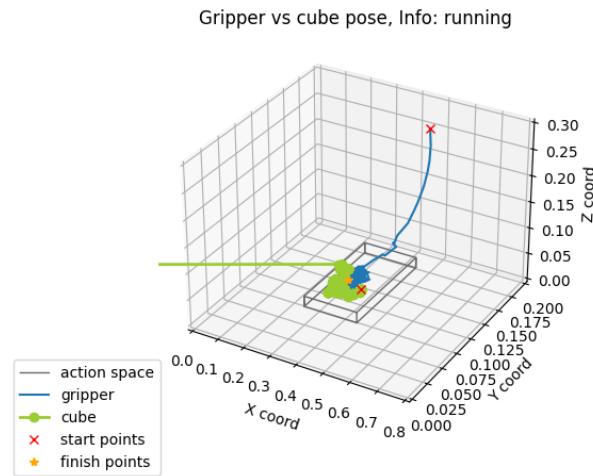


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object trajectory shows the object exiting the observation space as the green line depicts the projectile behaviour.

Figure C.3: Example on the reward proposal 2, when the object was successfully reached but ended out of the observation space. The cumulative reward for episode 165 is -3618.93 and corresponds **case D**.

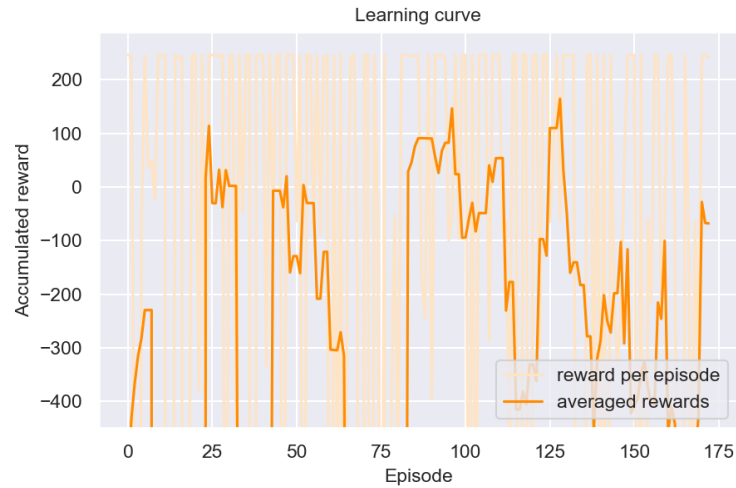


(a) Reward Function 2 and Episode 154. The reward function suggests that the cube was projected at step 90 and flies away until step 140. This collision was so dramatic that the object reentered the environment from then on, and the gripper remains to explore the workspace without the possibility to change that.

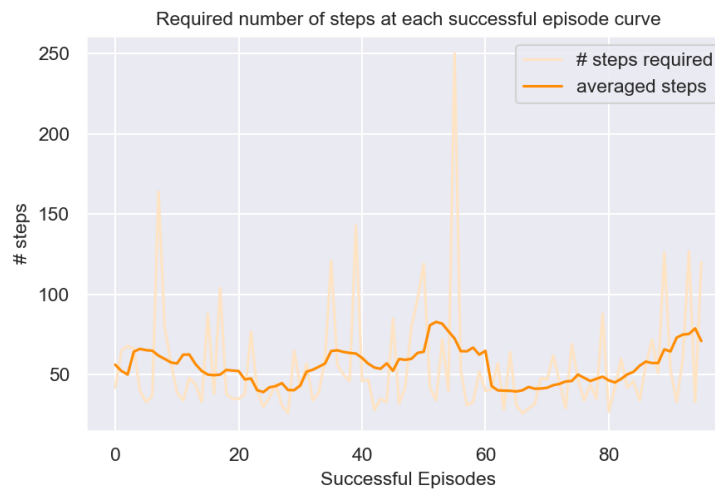


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object trajectory shows the object exiting the workspace and reenters at the origin, resulting in $X_{object} = (0;0;0)$.

Figure C.4: Example on the reward proposal 2, when there was a collision that caused the object to exit the workspace and reenter it at the origin. This episode is highly penalised by accumulating such low rewards for the max number of steps and introduces weird behaviours as object trajectory is unnatural. Nonetheless, the penalisation shows how undesirable that behaviour is. The cumulative reward for episode 154 is -3618.93 , and it is classified as **case B**.



(a) Evolution of the cumulative reward per episode.



(b) Evolution of the required steps to achieved success.

Figure C.5: Pick- Training behaviour for Reward proposal 2



(a) Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 2

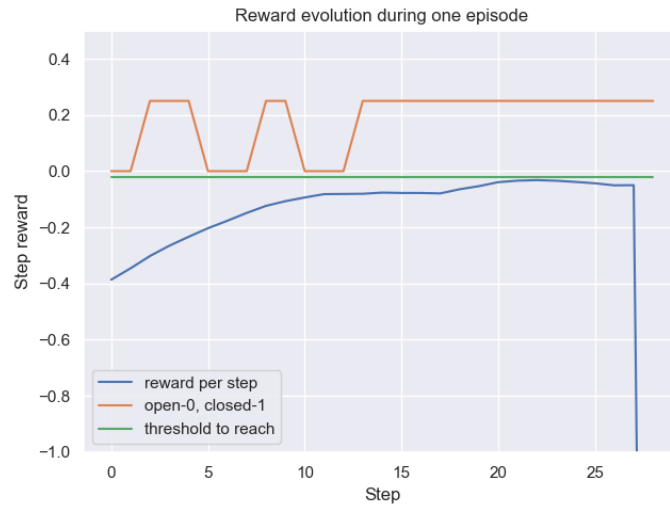


(b) Discriminator loss, Generator loss and Generator value loss, for Reward proposal 2

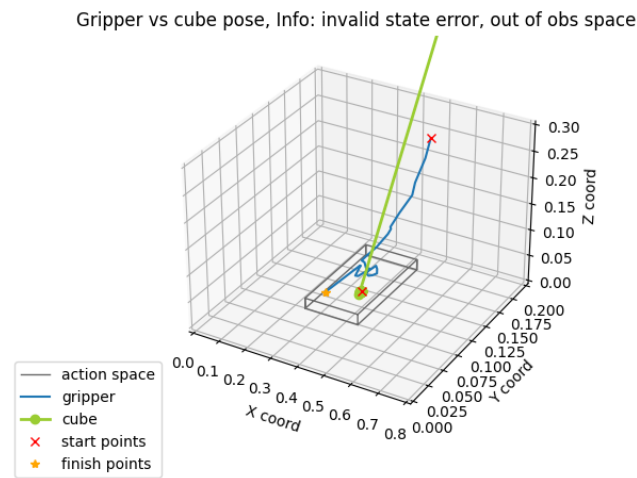
Figure C.6: Pick- GAIL training behaviour for Reward proposal 2

Appendix D

Pick Results- Reward Function 3

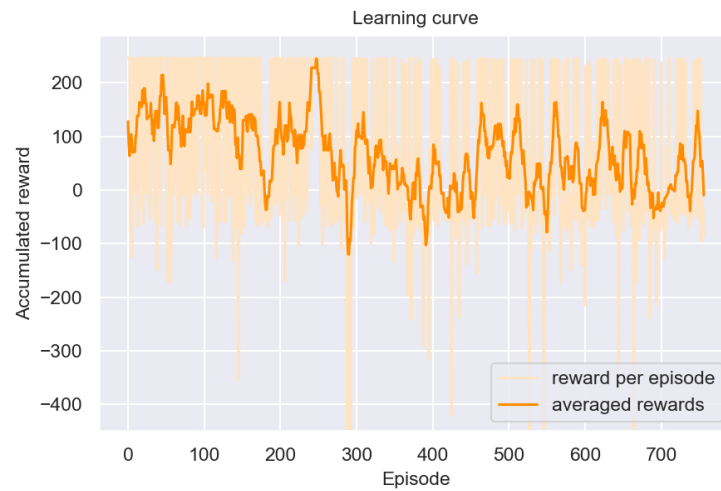


(a) Reward Function 3 and Episode 344. The reward function suggests that the agent chose a correct and smooth trajectory until step 26. Then, as the gripper was closed, there was a collision, and the reward curve decreases abruptly.

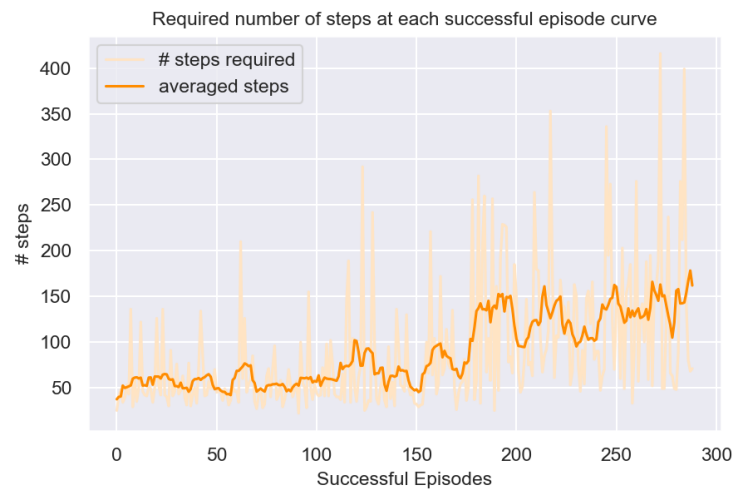


(b) Gripper (blue) vs object (green) trajectories for an unsuccessful episode. The object trajectory shows the gripper performing a smooth and precise trajectory for the first half of the episode. Then, near the object, the trajectory depicts a weird behaviour representing the collision, and the object flies into the air.

Figure D.1: Example on the reward proposal 3, when there was a collision caused by the wrong state of the gripper, *i.e.* the gripper was closed despite performing a correct trajectory. The cumulative reward for episode 344 is -9.38 , and it is classified as **case C**, as the object exits the observation space

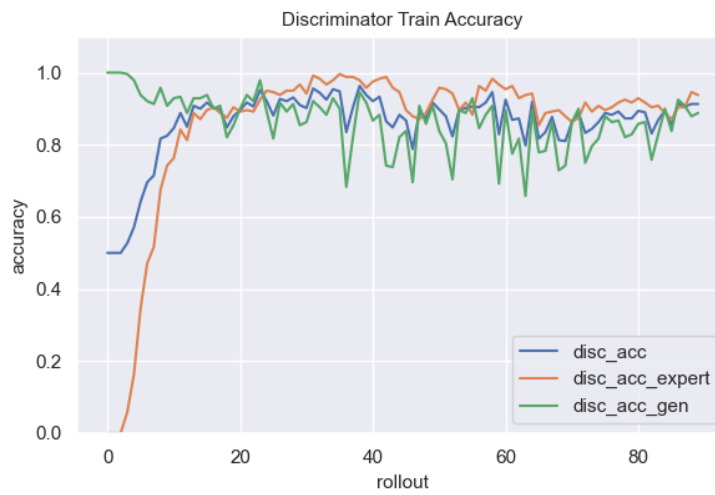


(a) Evolution of the cumulative reward per episode.



(b) Evolution of the required steps to achieved success.

Figure D.2: Pick- Training behaviour for Reward proposal 3



(a) Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 3

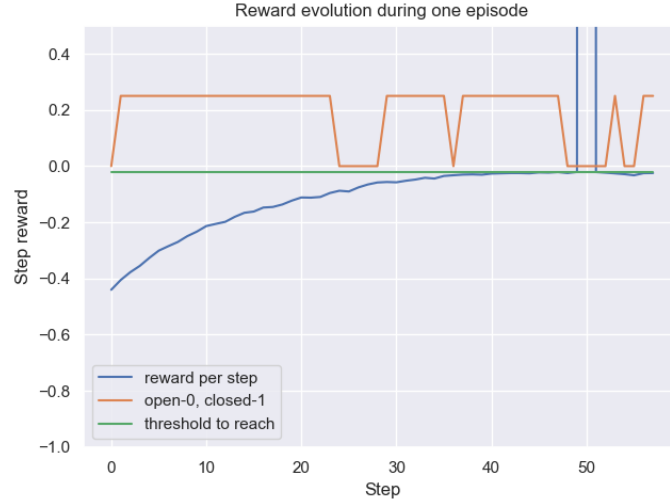


(b) Discriminator loss, Generator loss and Generator value loss, for Reward proposal 3

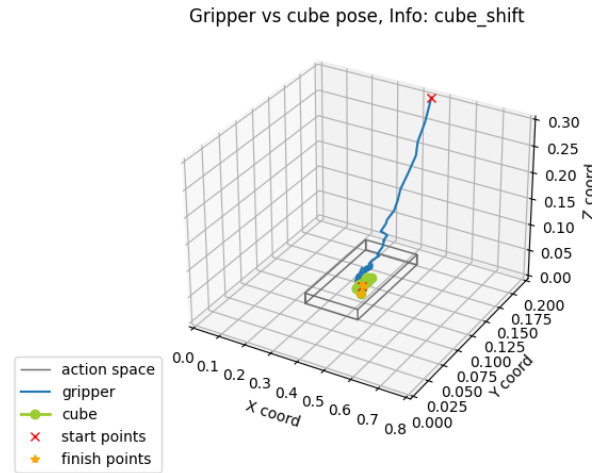
Figure D.3: Pick- GAIL training behaviour for Reward proposal 3

Appendix E

Pick Results- Reward Function 4

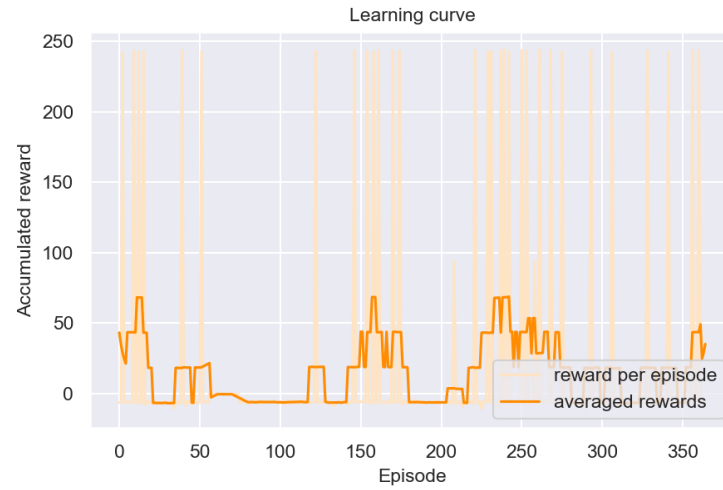


(a) Reward Function 4 and Episode 259. The reward function suggests the agent approached the object, reached it on step 50 but did not pick it, closing the fingers twice on the following steps. At final steps, both gripper and object were almost 2cms apart, as the reward curve is close to the threshold line.

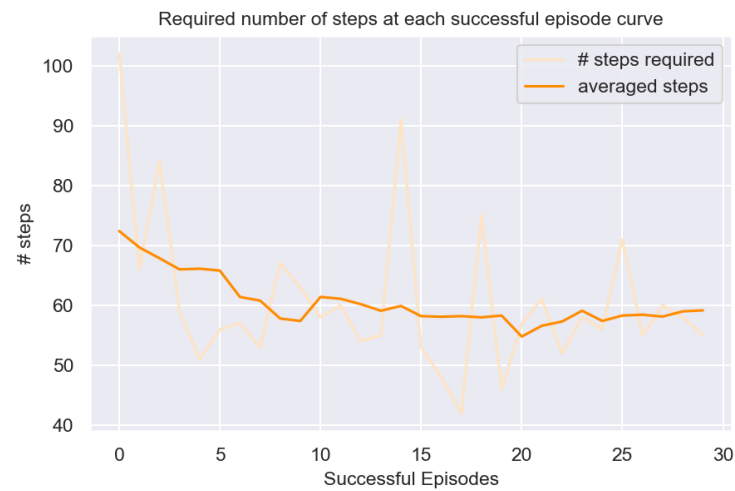


(b) Gripper (blue) vs object (green) trajectories for an episode where the object is moved. The gripper approaches the object, and both perform parallel trajectories. Both finish points are close, which suggest the gripper ended up close to the object.

Figure E.1: Example on the reward proposal 4, when the episode ended earlier as a way of penalising the agent for moving the object wrongly before picking it correctly. The reward curve and the trajectory suggest that the gripper was close to the desired pose and would perform picking in some steps. The cumulative reward for episode 259 is 93.397, and it is classified as **case D**, as the object performs picking but no reaching

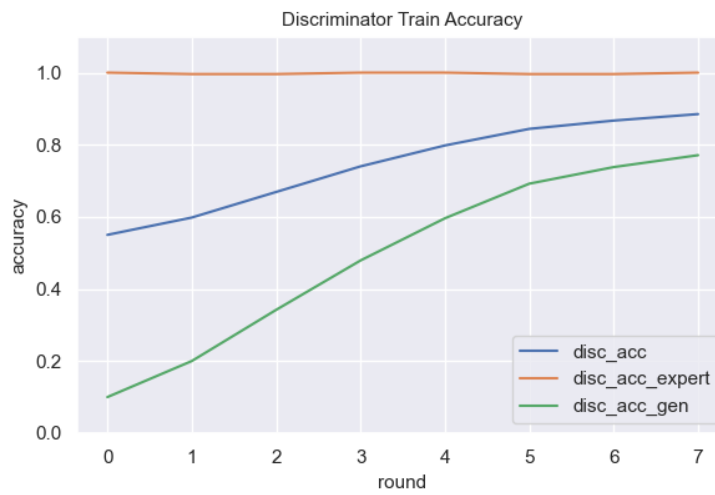


(a) Evolution of the cumulative reward per episode.

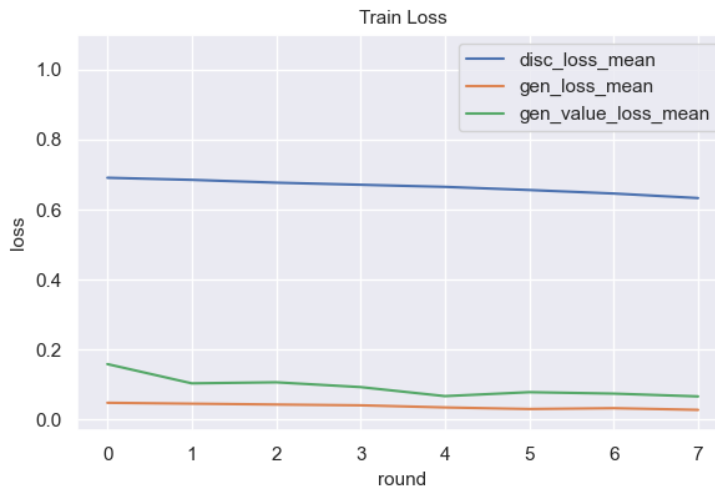


(b) Evolution of the required steps to achieved success.

Figure E.2: Pick- Training behaviour for Reward proposal 4



(a) Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 4

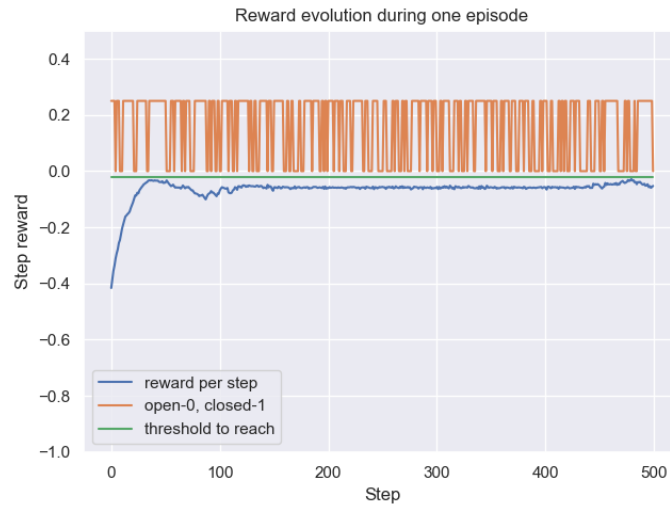


(b) Discriminator loss, Generator loss and Generator value loss, for Reward proposal 4

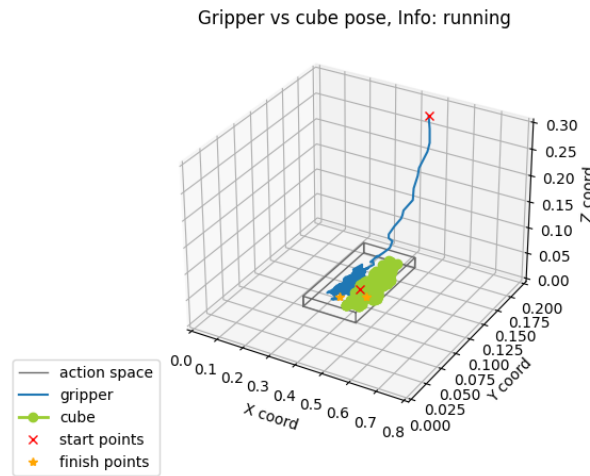
Figure E.3: Pick- GAIL training behaviour for Reward proposal 4

Appendix F

Pick Results- Reward Function 5



(a) Reward Function 5 and Episode 80. The reward function suggests the agent approached the object's centre around episode 30 but moved away during the next steps as the reward curve is approximately constant at $step_reward \approx -0.5$. This suggests the object was grasped near episode 100 and then moved around by the gripper. Near episode 450, the reward decreased, suggesting the agent released the object and approached its centre. However, this was not enough to complete the episode before the 500 steps are used.

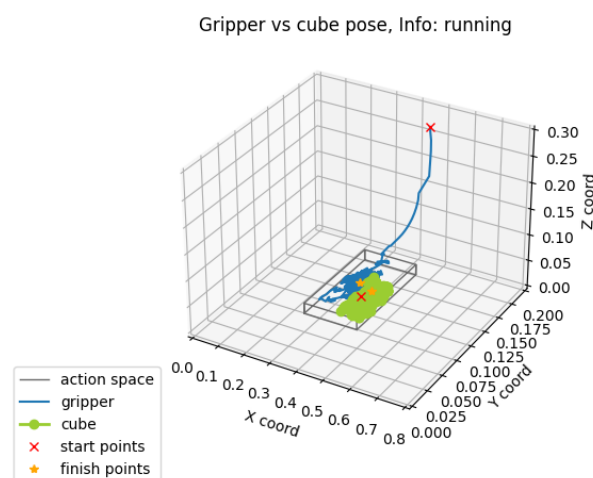


(b) Gripper (blue) vs object (green) trajectories for an episode where the object is moved. Despite the start and finish points being depicted as relatively close, the green curve extends all over the action space. The gripper trajectory is also parallel to the object trajectory, which suggests they were attached for most of the episode.

Figure F.1: Example on the reward proposal 5, when the object was attached for the majority of the episode. Both gripper and object moved around the action space and prevented the agent from performing either reaching or picking. The cumulative reward for episode 259 is -32.306 , and it is classified as **case A**, as the agent performs neither picking nor reaching, remaining inside the action space



(a) Reward Function 5 and Episode 609. The reward function suggests the agent approached the object and moved away several times during the episode. The agent only performed the reaching at step 340. then the agent moved past the point and failed to pick the object before the episode reached 500 steps.

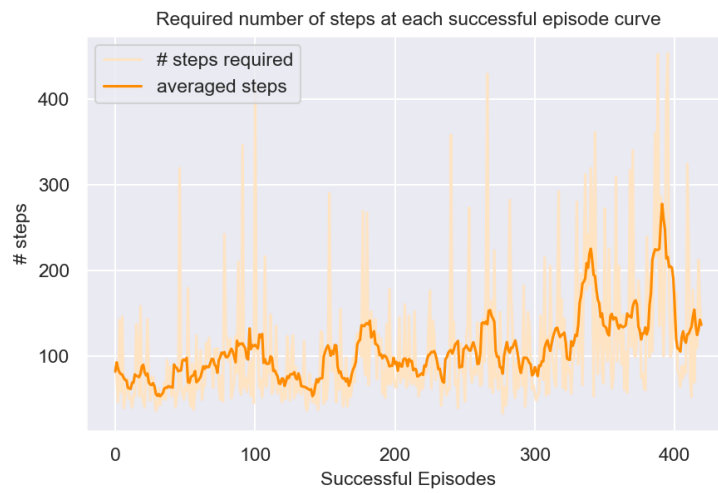


(b) Gripper (blue) vs object (green) trajectories for an episode where the object is moved. Both object and gripper trajectories are spread over the action space, more or less parallel to each other. It suggests they were attached during many steps.

Figure F.2: Example on the reward proposal 609, where the agent learnt how to reach the object (at episode 340) after moving it away. However, the agent failed to pick the object at the correct position. The cumulative reward for episode 609 is -20.985 , and it is classified as **case d**, as the agent only completes reaching.



(a) Evolution of the cumulative reward per episode.



(b) Evolution of the required steps to achieved success.

Figure F.3: Pick- Training behaviour for Reward proposal 5



(a) Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 5



(b) Discriminator loss, Generator loss and Generator value loss, for Reward proposal 5

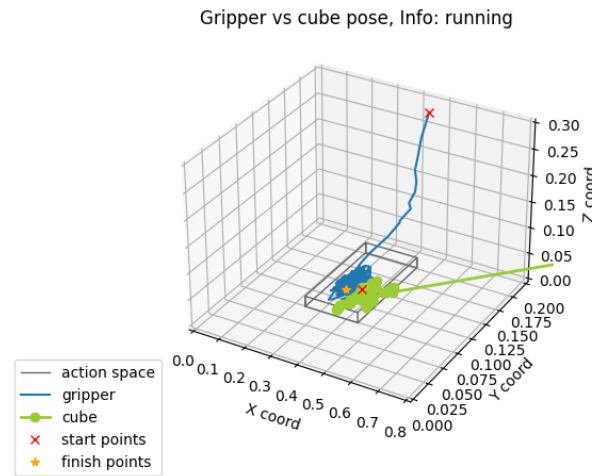
Figure F.4: Pick- GAIL training behaviour for Reward proposal 5

Appendix G

Pick Results- Reward Function 6

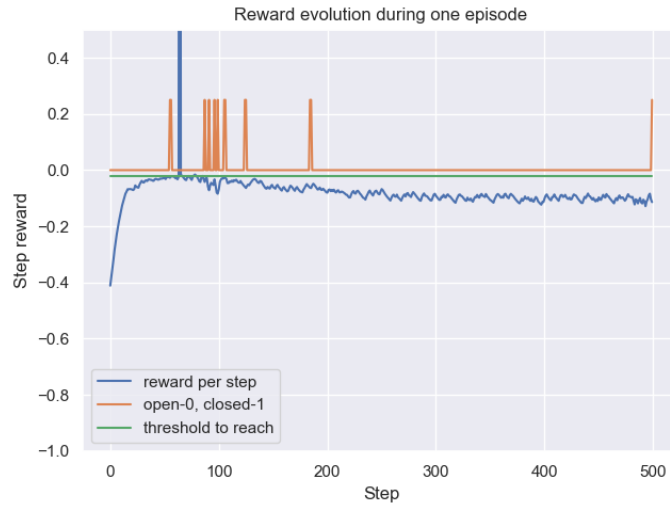


(a) Reward Function 6 and Episode 8. The reward function suggests the agent tried to approach the object's centre until step 25 but performs so with the fingers closed (orange curve is "high"). This results in the object being pushed throughout the action space. At step 108, a collision sent the object out of the action space before the agent could reach it properly. For the remaining 400 steps, the part stood at around 1m from the gripper.

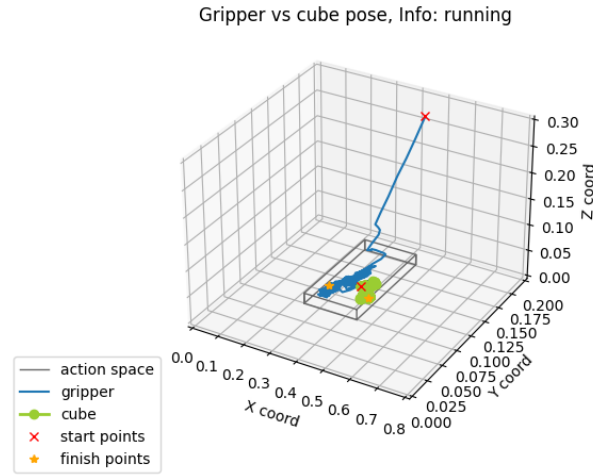


(b) Gripper (blue) vs object (green) trajectories for an episode where the object moved around the action space but ended out of it. The gripper and object trajectories do not look alike, which means the gripper pushed the object instead of grasping and moving it around. At last, the object flies away from the workspace.

Figure G.1: Example on the reward proposal 6, when the agent starts exploring the action variable y_c . In this situation, the agent approaches the object correctly. However, it performs so with the fingers closed and thus pushes it to the side. Ultimately, a collision at step 108 projects the object out of the action space, and the agent accumulates low rewards for the remainder of the episode. The cumulative reward for episode 8 is -382.46 , and it is classified as **case B**, as the agent causes a collision and the object ends out of the action space.

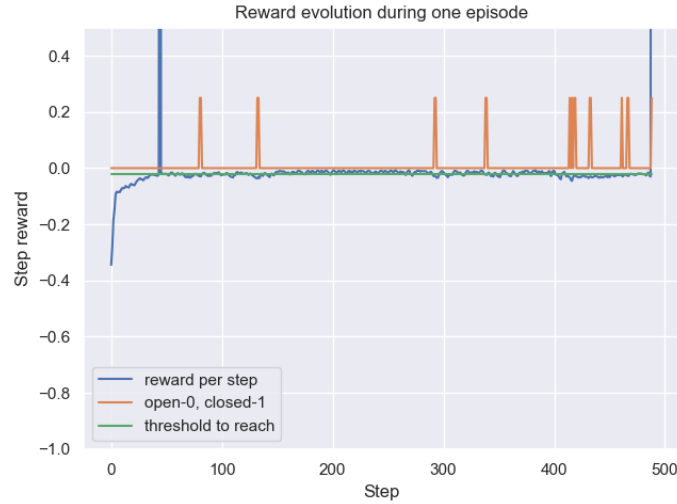


(a) Reward Function 6 and Episode 843. The reward function suggests the agent tried to approach the object's centre with the fingers correctly open. It achieves so at episode 63 but fails to close the fingers. When the agent tries to grasp the cube, between episodes 87 and 125, the object was moved and not correctly picked. For the remaining steps, the agent tries to reach the object's centre again without closing the fingers, suggesting that it has been learnt.

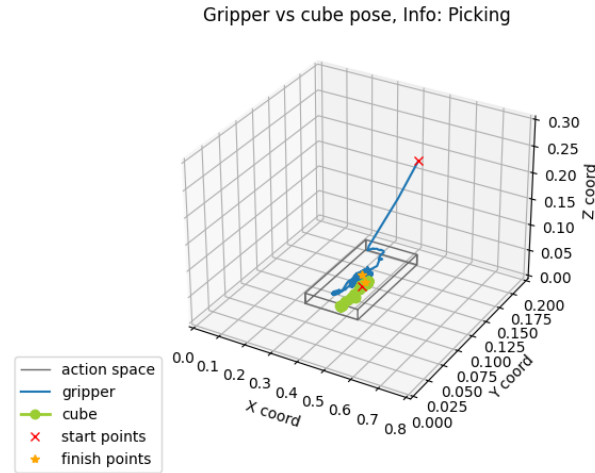


(b) Gripper (blue) vs object (green) trajectories for an episode where the object moved around the action space. The gripper trajectory and the object trajectory overlap, confirming the object has been reached. The object appears to move because its initial and final points do not match. Nonetheless, the object remains in the action space, and the agent successfully avoids a collision. However, the gripper and the object end the episode on opposite points of the action space, confirming the episode has not been successful.

Figure G.2: Example on the reward proposal 6, when the agent fails to pick the object even after reaching it. The unsucces is not linked with the object exiting the action/observation space. Instead, it moves to the action space's limit. Nonetheless, it is evident that the agent knows closing the gripper earlier will result in unsucces. The cumulative reward for episode 8 is -35.876 , and it is classified as **case D**.

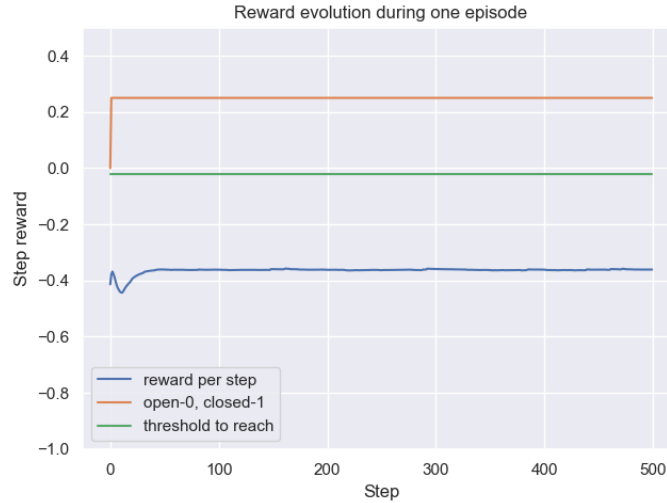


(a) Reward Function 6 and Episode 786. The reward function suggests the agent successfully reaches the object at step 42. However, it only closes the fingers at step 80, not at the desired pose. The gripper is opened at less than 2cms from the object centre for the majority of the subsequent steps. The agent successfully grasps the object at step 489.

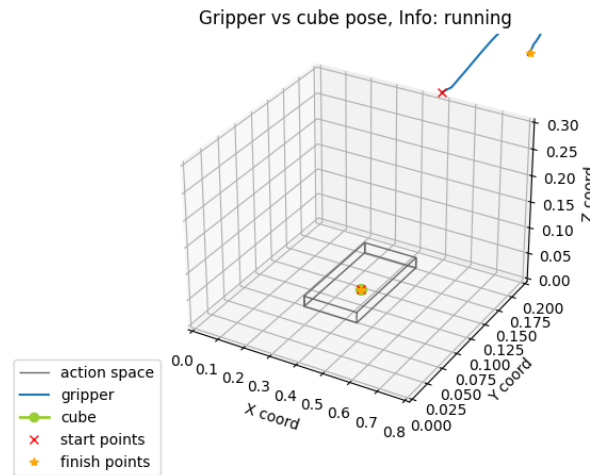


(b) Gripper (blue) vs object (green) trajectories for an episode where the object moved around the action space. The gripper trajectory and the object trajectory are similar, suggesting the object moved with the gripper, at least part of its trajectory. The final points are close and support the conclusion that the object has been correctly picked.

Figure G.3: Example on the reward proposal 6, when the agent picks the object later in the episode. Despite the episode being far from the optimal behaviour, the agent can overcome situations where the object is pushed and moved around the workspace. By closing the fingers very few times, the agent understands that it is a critical action. Episode 786 is penalised for lasting too many steps resulting in a cumulative reward of 3.3018, being classified as **case E**.



(a) Reward Function 6 and Episode 1000. The reward function shows a rapid change in the gripper error at the start. However, the distance between the object and the gripper remains approximately constant through the episode. Also, the finger joint is closed for all steps. This suggests a simulator error where the agent wasn't able to control the robotic arm.

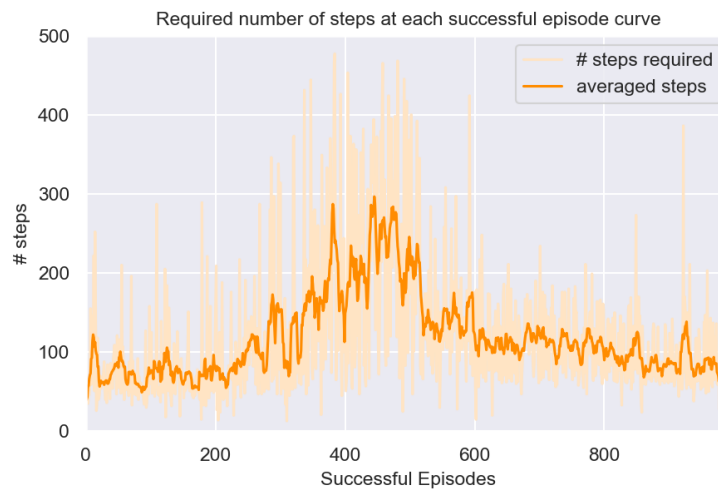


(b) Gripper (blue) vs object (green) trajectories for an episode where the object remained at the start position, but the gripper appears to be broken, and the trajectory is nothing but the expected.

Figure G.4: Example on the reward proposal 7, when the robotic arm was not correctly calibrated, and thus the agent wasn't able to control it. This episode shows the importance of performing regular re-calibrations throughout the training process. The episode 1000 received the cumulative reward of -182.16 and is classified as **case A**, despite being a clear **outlier**.

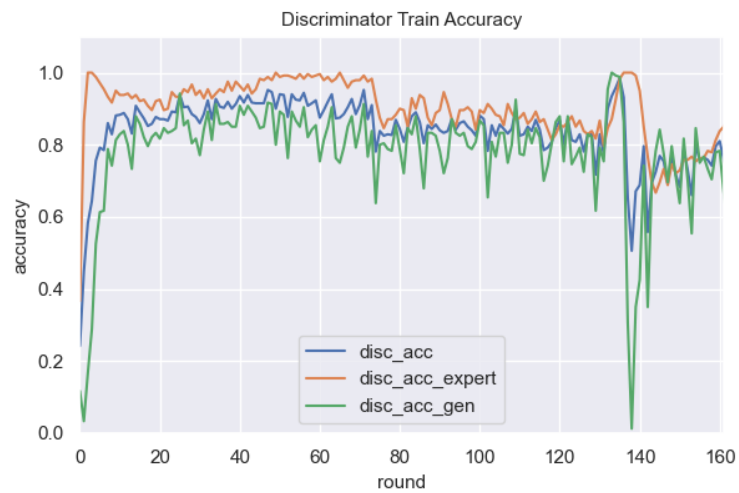


(a) Evolution of the cumulative reward per episode.



(b) Evolution of the required steps to achieved success.

Figure G.5: Pick- Training behaviour for Reward proposal 6



(a) Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 6

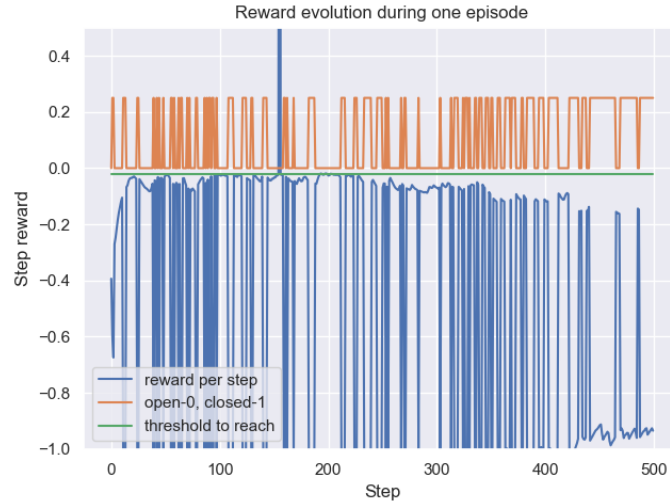


(b) Discriminator loss, Generator loss and Generator value loss, for Reward proposal 6

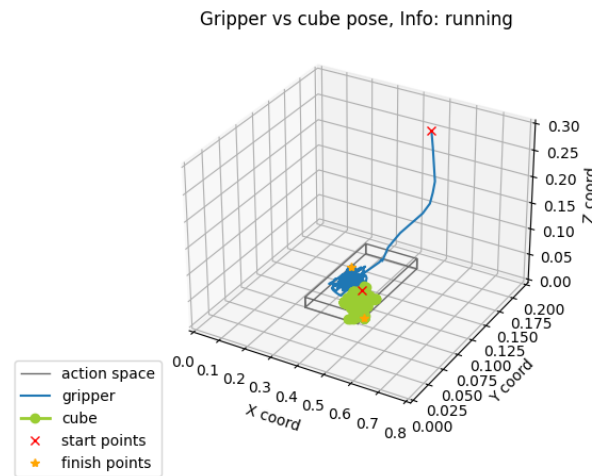
Figure G.6: Pick- GAIL training behaviour for Reward proposal 6

Appendix H

Pick Results- Reward Function 7

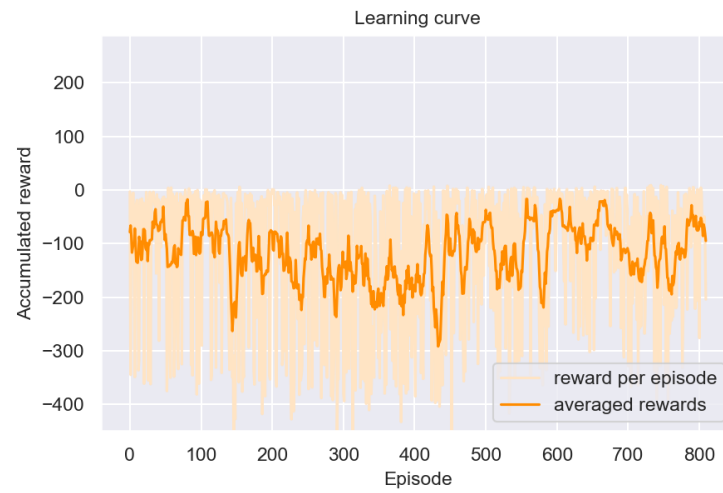


(a) Reward Function 7 and Episode 751. The reward function combines the gripper error and the penalty for closing the fingers without positioning the gripper correctly. The agent reaches the object at step 154, receiving a positive reward ($R_r = 5$). However, the rewards at each step vary between -1.1 and -0.02 . The main cause for such variation is related to the additional penalty P_c . This adds some instability to the training.

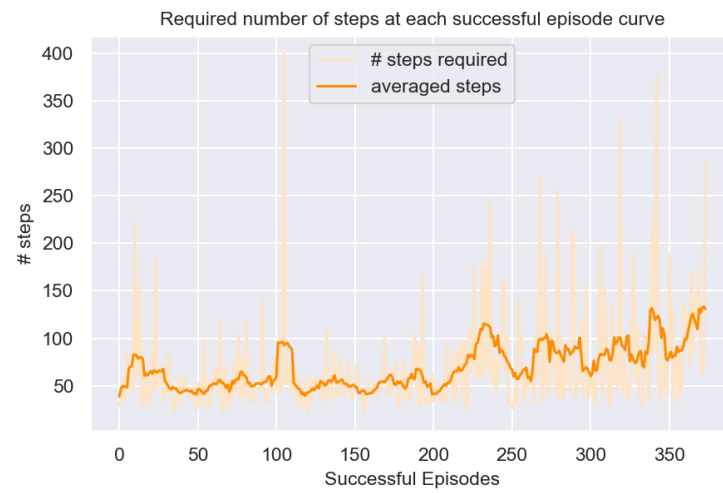


(b) Gripper (blue) vs object (green) trajectories for an episode where the object moves around the action space. The agent reaches the object but fails to pick it correctly.

Figure H.1: Example on the reward proposal 7, when the agent reaches the object without picking it correctly. The agent fails to complete the episode because it couldn't overcome the fact that the object was no longer at the reset pose. Moreover, the reward curve shows a less stable behaviour because the step rewards constantly oscillate due to penalty P_c . The cumulative reward for episode 8 is -213.996 , and it is classified as **case D**.



(a) Evolution of the cumulative reward per episode.



(b) Evolution of the required steps to achieved success.

Figure H.2: Pick- Training behaviour for Reward proposal 7



(a) Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 7



(b) Discriminator loss, Generator loss and Generator value loss, for Reward proposal 7

Figure H.3: Pick- GAIL training behaviour for Reward proposal 7

Appendix I

Move Results- Reward Function 1

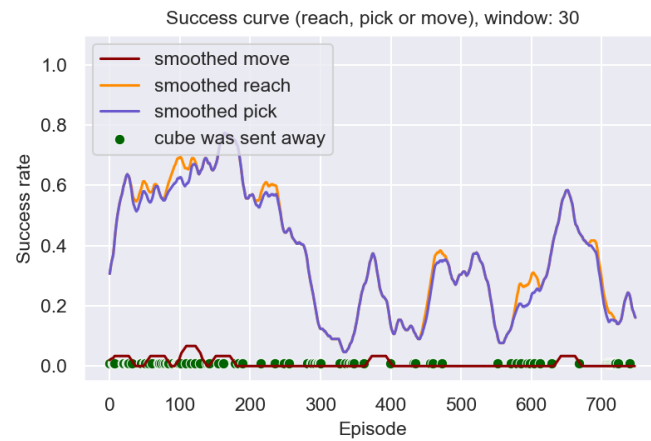


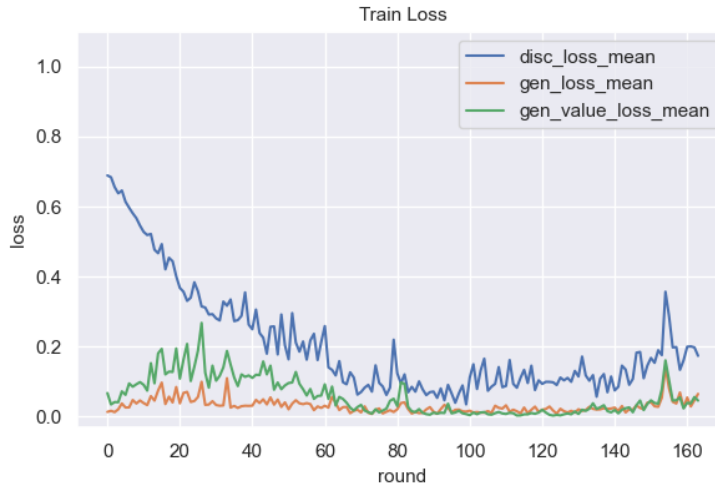
Figure I.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 1.



Figure I.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 1



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 1



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 1

Figure I.3: Move- GAIL training behaviour for Reward proposal 1

Appendix J

Move Results- Reward Function 2

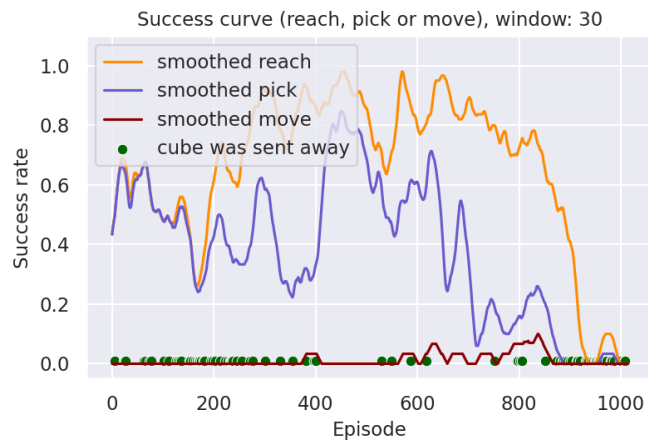


Figure J.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 2.



Figure J.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 2



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 2



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 2

Figure J.3: Move- GAIL training behaviour for Reward proposal 2

Appendix K

Move Results- Reward Function 3

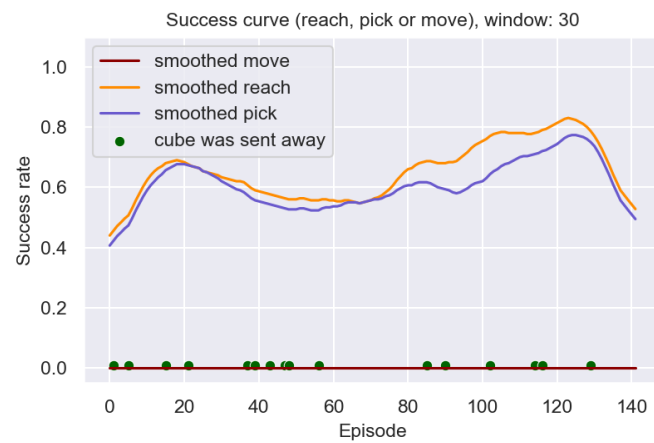
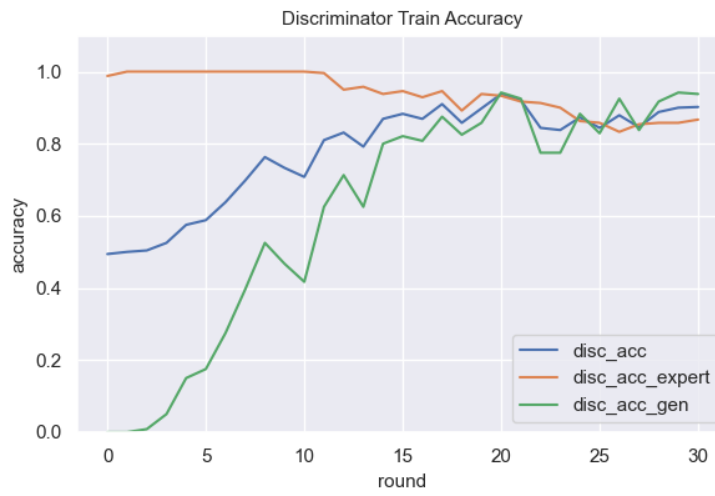


Figure K.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 3.



Figure K.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 3



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 3



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 3

Figure K.3: Move- GAIL training behaviour for Reward proposal 3

Appendix L

Move Results- Reward Function 4

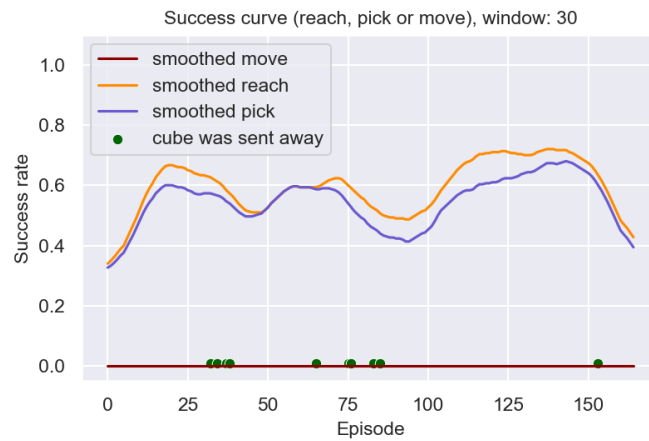


Figure L.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 4.

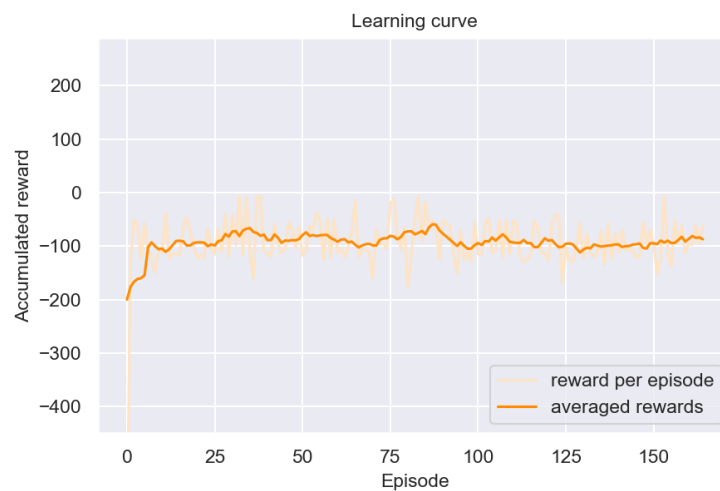
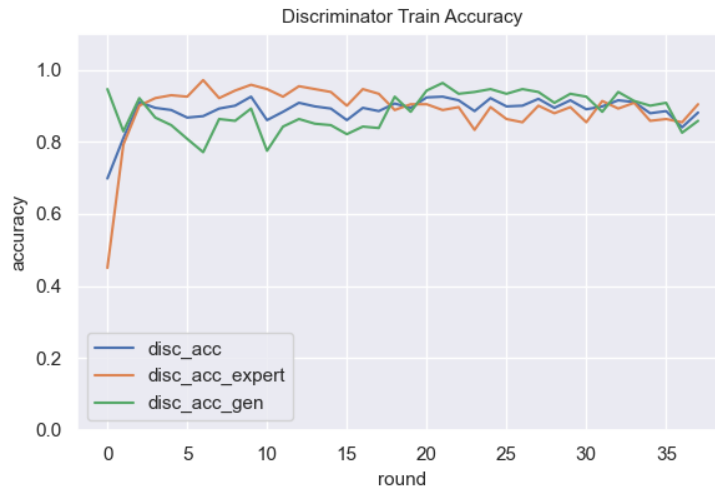


Figure L.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 4



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 4



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 4

Figure L.3: Move- GAIL training behaviour for Reward proposal 4

Appendix M

Move Results- Reward Function 5

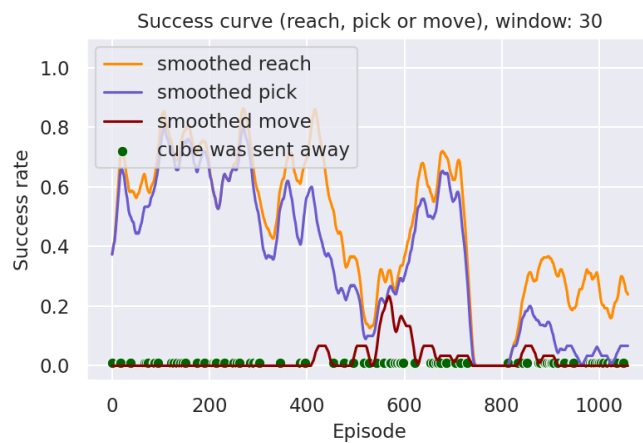
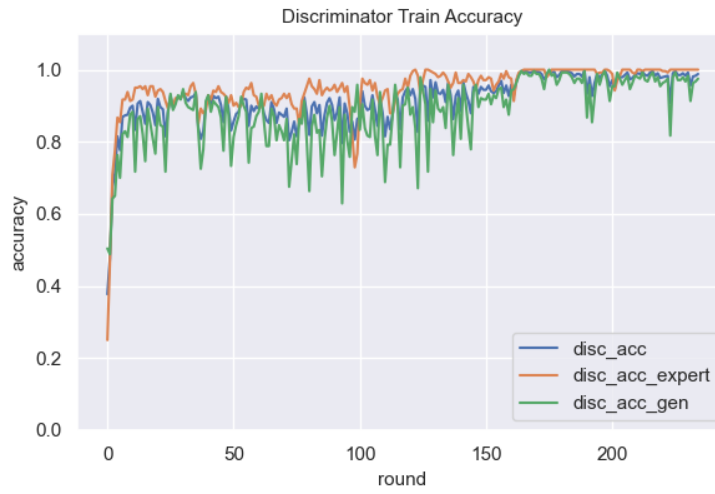


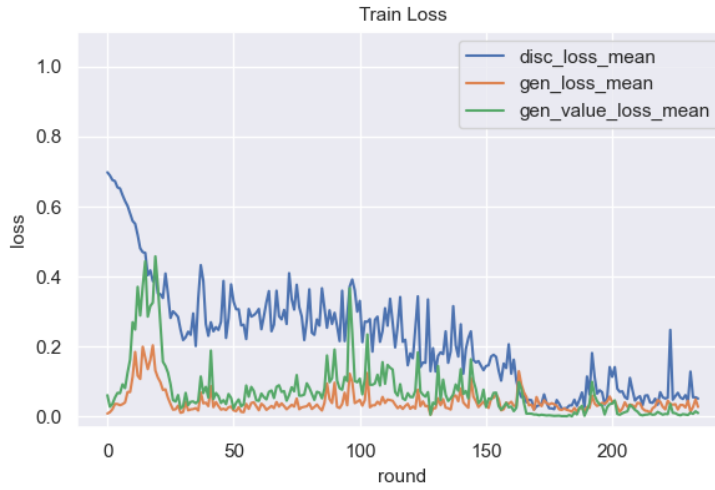
Figure M.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 5.



Figure M.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 5



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 5



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 5

Figure M.3: Move- GAIL training behaviour for Reward proposal 5

Appendix N

Move Results- Reward Function 6

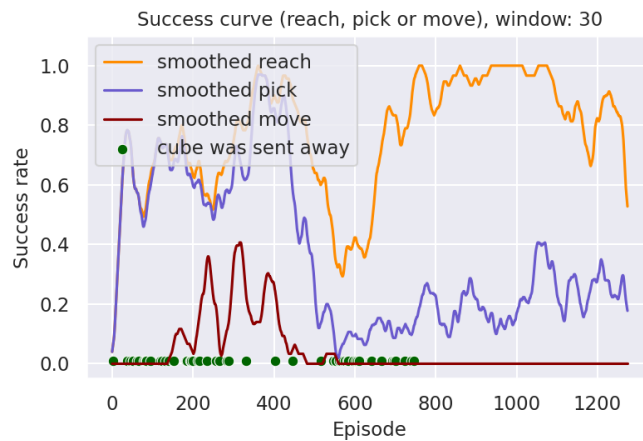


Figure N.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 6.

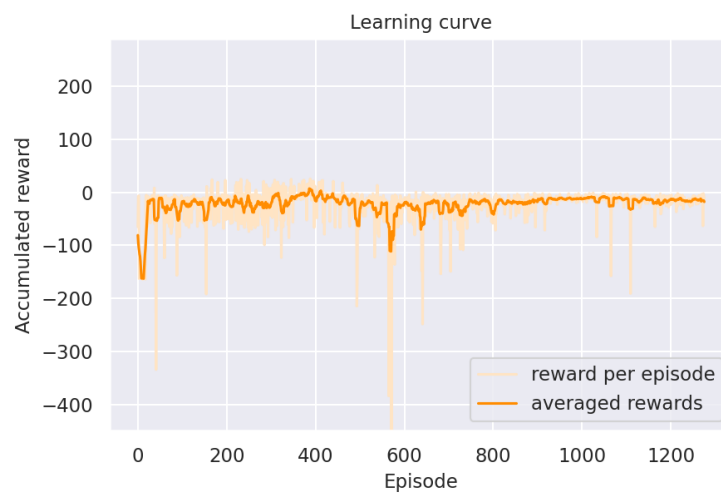
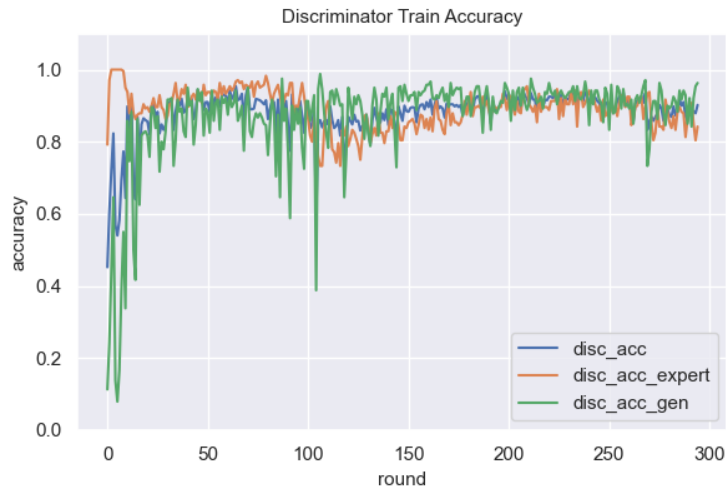


Figure N.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 6



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 6



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 6

Figure N.3: Move- GAIL training behaviour for Reward proposal 6

Appendix O

Move Results- Reward Function 7

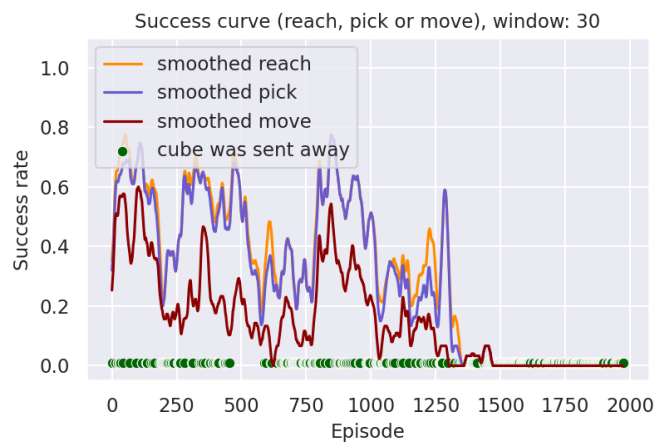


Figure O.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 7.

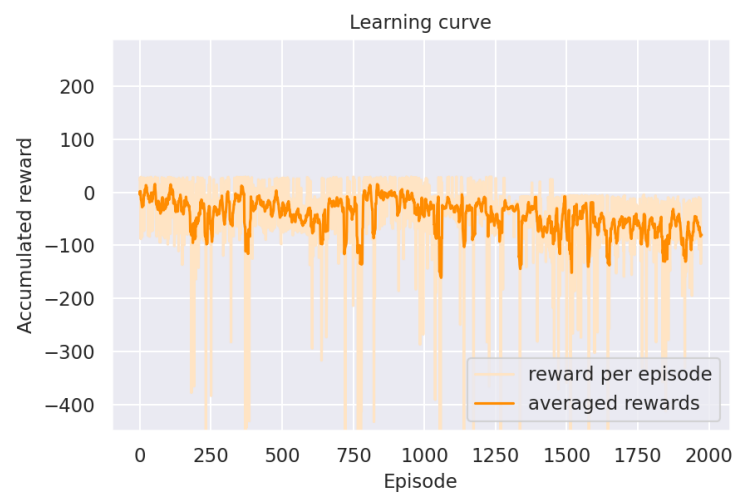
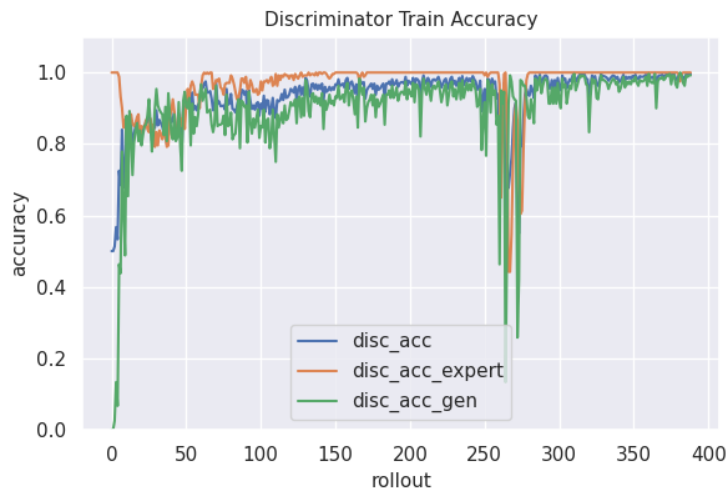
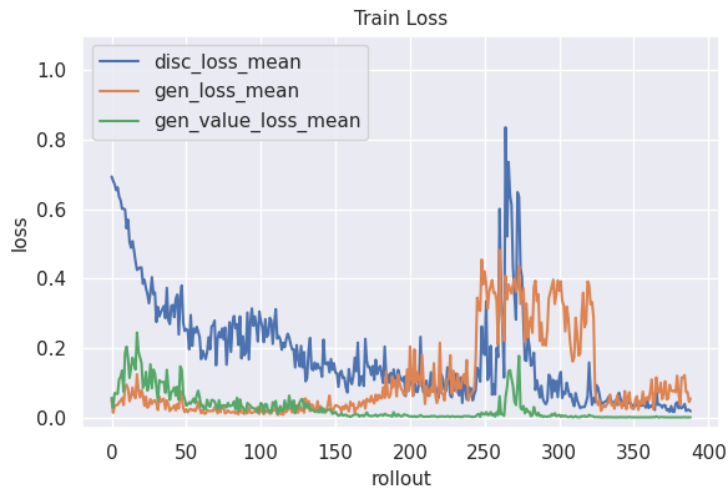


Figure O.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 7



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 7



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 7

Figure O.3: Move- GAIL training behaviour for Reward proposal 7

Appendix P

Move Results- Reward Function 8

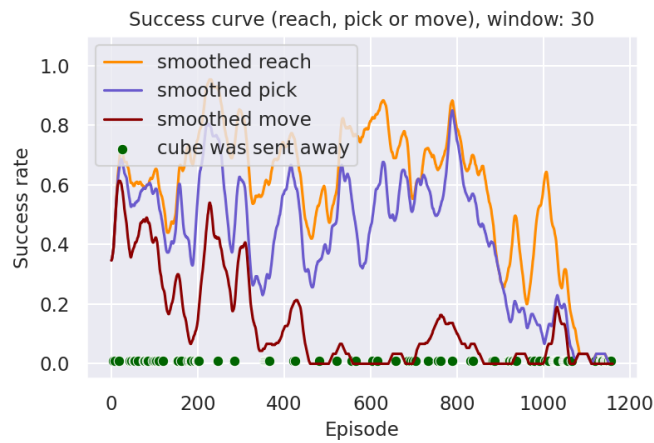
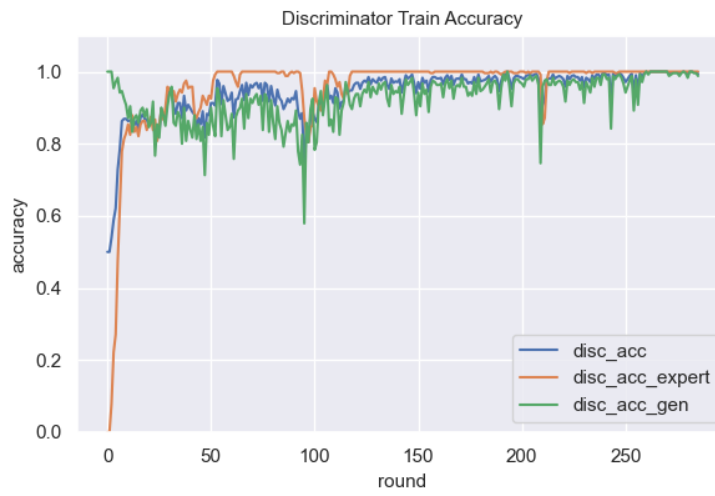


Figure P.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 8.



Figure P.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 8



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 8



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 8

Figure P.3: Move- GAIL training behaviour for Reward proposal 8

Appendix Q

Move Results- Reward Function 9

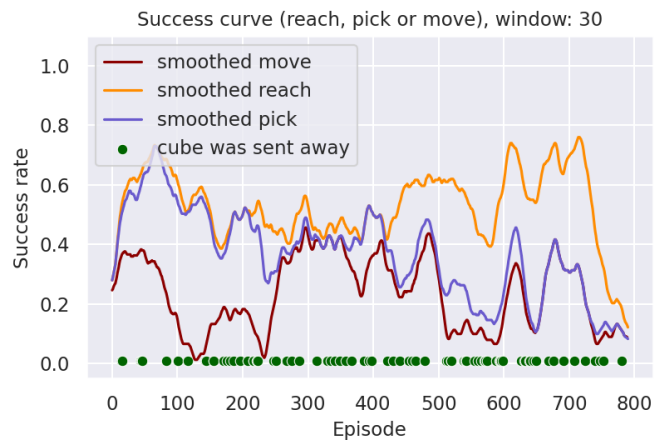
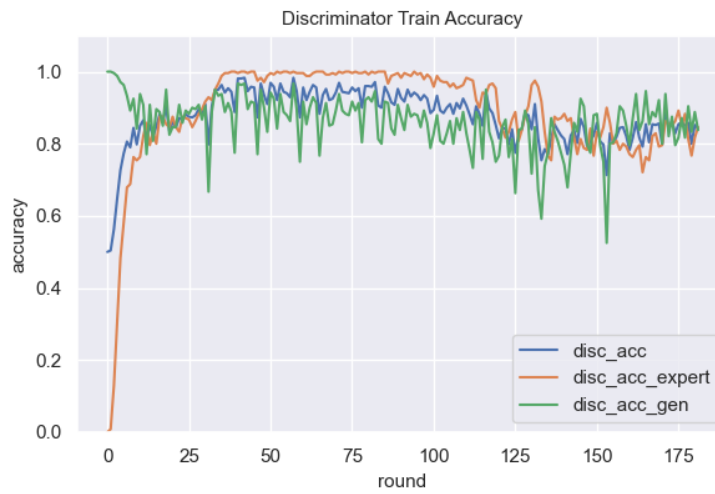


Figure Q.1: Move- Success rate curve for move agent trained anew, with ten expert demos and reward model 9.



Figure Q.2: Move- Evolution of the cumulative reward per episode, for Reward proposal 9



(a) Move- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 9



(b) Move- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 9

Figure Q.3: Move- GAIL training behaviour for Reward proposal 9

Appendix R

Fit Results- Reward Function 1

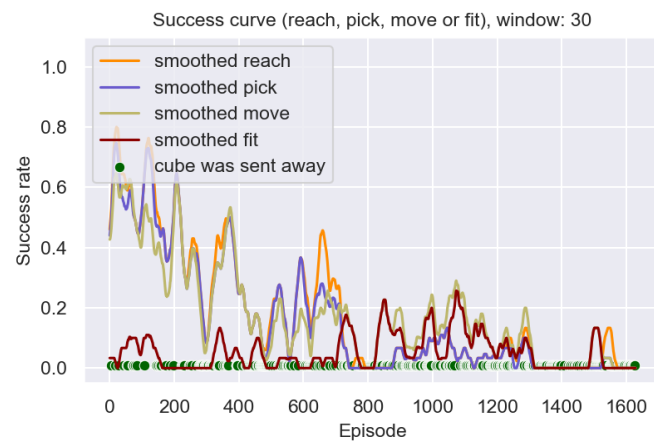


Figure R.1: Fit- Success rate curve for fit agent trained anew, with ten expert demos and **reward model 1**.

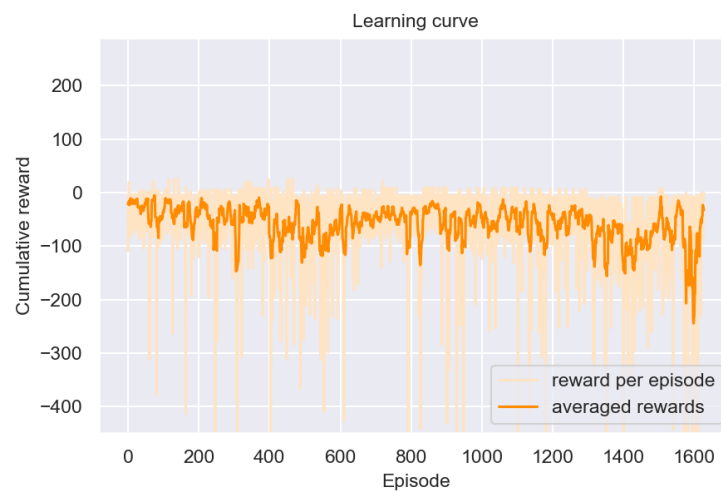
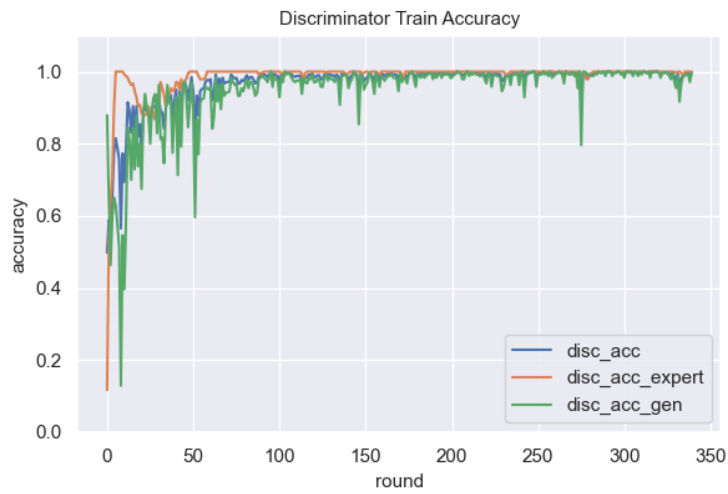
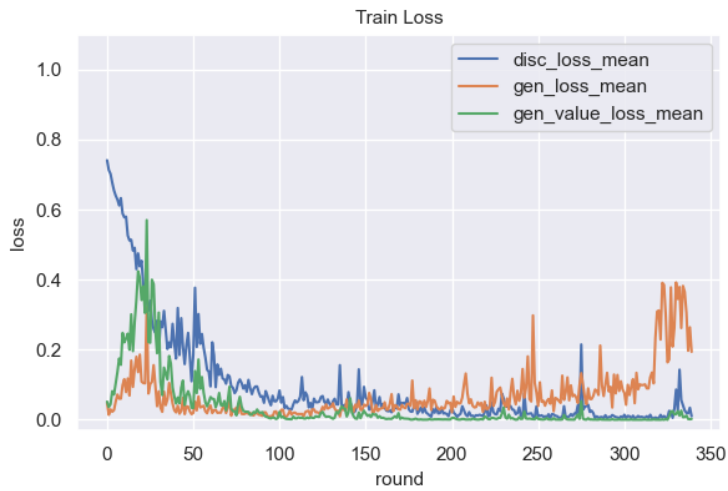


Figure R.2: Fit- Evolution of the cumulative reward per episode, for Reward proposal 1



(a) Fit- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 1



(b) Fit- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 1

Figure R.3: Fit- GAIL training behaviour for Reward proposal 1

Appendix S

Fit Results- Reward Function 2

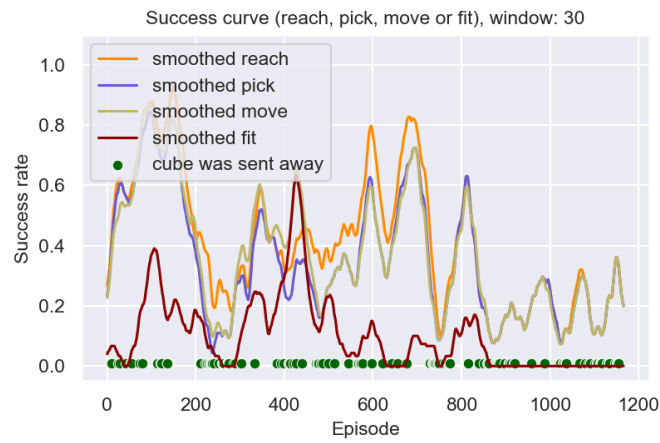
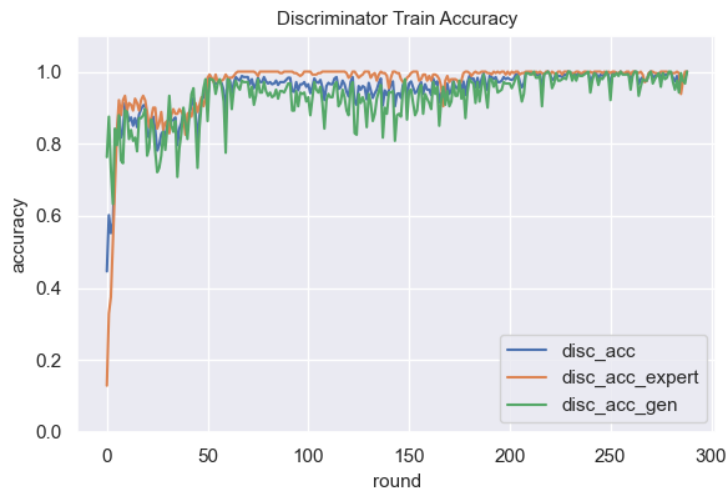


Figure S.1: Fit- Success rate curve for fit agent trained anew, with ten expert demos and **reward model 2**.



Figure S.2: Fit- Evolution of the cumulative reward per episode, for Reward proposal 2



(a) Fit- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 2



(b) Fit- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 2

Figure S.3: Fit- GAIL training behaviour for Reward proposal 2

Appendix T

Fit Results- Reward Function 3

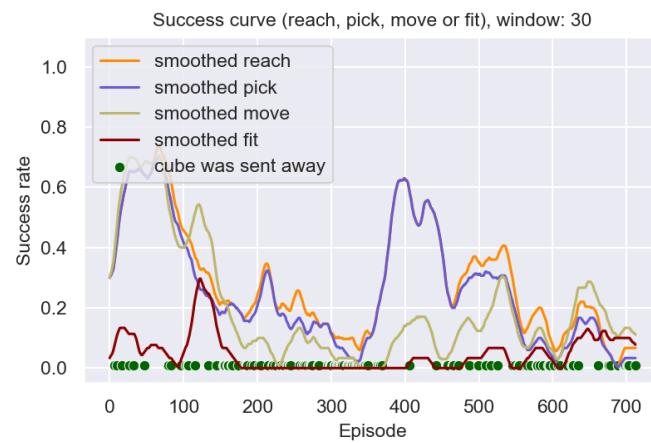


Figure T.1: Fit- Success rate curve for fit agent trained anew, with ten expert demos and **reward model 3**.

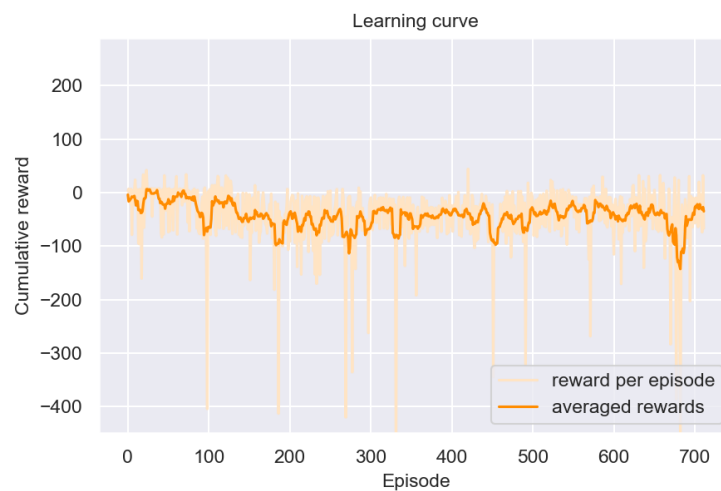
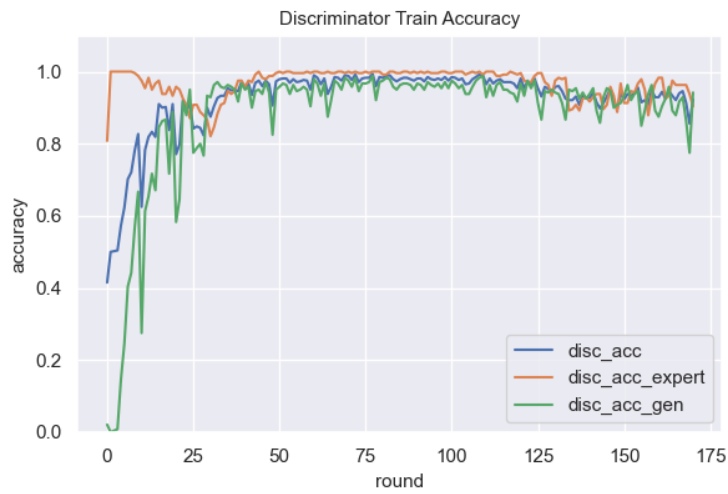


Figure T.2: Fit- Evolution of the cumulative reward per episode, for Reward proposal 3



(a) Fit- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 3



(b) Fit- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 3

Figure T.3: Fit- GAIL training behaviour for Reward proposal 3

Appendix U

Fit Results- Reward Function 4

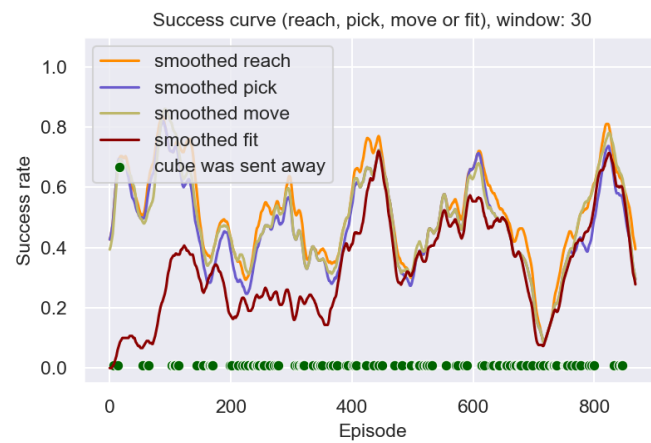


Figure U.1: Fit- Success rate curve for fit agent trained anew, with ten expert demos and **reward model 4**.

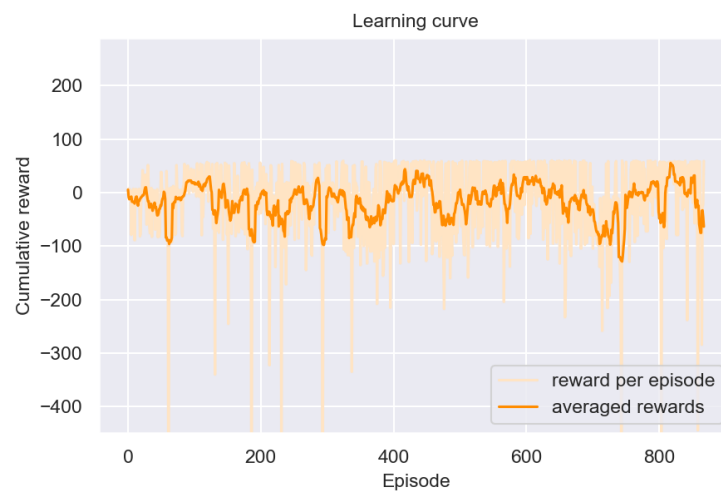


Figure U.2: Fit- Evolution of the cumulative reward per episode, for Reward proposal 4



(a) Fit- Discriminator accuracy on expert data, discriminator accuracy on generated data and overall discriminator accuracy, for Reward proposal 4



(b) Fit- Discriminator loss, Generator loss and Generator value loss, for Reward proposal 4

Figure U.3: Fit- GAIL training behaviour for Reward proposal 4

References

- [1] Ruchi Goel and Pooja Gupta. Robotics and industry 4.0, 2020. URL: https://link.springer.com/chapter/10.1007/978-3-030-14544-6_9, doi:10.1007/978-3-030-14544-6_9.
- [2] German Carro Fernandez, Sergio Martin Gutierrez, Elio Sancristobal Ruiz, Francisco Mur Perez, and Manuel Castro Gil. Robotics, the new industrial revolution. *IEEE Technology and Society Magazine*, 31:51–58, 2012. doi:10.1109/MTS.2012.2196595.
- [3] Khalid Hasan Tantawi, Alexandr Sokolov, and Omar Tantawi. Advances in industrial robotics: From industry 3.0 automation to industry 4.0 collaboration. Institute of Electrical and Electronics Engineers Inc., 12 2019. doi:10.1109/TIMES-ICON47539.2019.9024658.
- [4] Manal Aissam, Mohammed Benbrahim, and Mohammed Nabil Kabbaj. Cloud robotic: Opening a new road to the industry 4.0, 2019. URL: https://doi.org/10.1007/978-981-13-2212-9_1, doi:10.1007/978-981-13-2212-9_1.
- [5] Rongshen Lai, Wenguang Lin, and Yongming Wu. Review of research on the key technologies, application fields and development trends of intelligent robots. pages 449–458. Springer International Publishing, 2018.
- [6] Paulo Leitao, Flavia Pires, Stamatis Karnouskos, and Armando Walter Colombo. Quo vadis industry 4.0? position, trends, and challenges. *IEEE Open Journal of the Industrial Electronics Society*, 1:298–310, 10 2020. doi:10.1109/ojies.2020.3031660.
- [7] Antoni Grau, Manna Indri, Lucia Lo Bello, and Thilo Sauter. Industrial robotics in factory automation: From the early stage to the internet of things. volume 2017-January, pages 6159–6164. Institute of Electrical and Electronics Engineers Inc., 12 2017. doi:10.1109/IECON.2017.8217070.
- [8] Bojana Bajic, Aleksandar Rikalovic, Nikola Suzic, and Vincenzo Piuri. Industry 4.0 implementation challenges and opportunities: A managerial perspective. *IEEE Systems Journal*, pages 1–14, 9 2020. doi:10.1109/jsyst.2020.3023041.
- [9] Federica Ferraguti, Andrea Pertosa, Cristian Secchi, Cesare Fantuzzi, and Marcello Bonfè. A methodology for comparative analysis of collaborative robots for industry 4.0. pages 1070–1075. Institute of Electrical and Electronics Engineers Inc., 5 2019. doi:10.23919/DATE.2019.8714830.
- [10] D. Surdilovic, G. Schreck, and U. Schmidt. Development of collaborative robots (cobots) for flexible human-integrated assembly automation - vde conference publication. pages 1–8, 2010. URL: <https://ieeexplore.ieee.org/document/5756781>.

- [11] Simona Bukantaite. Factors of smart production processes modernization. Institute of Electrical and Electronics Engineers Inc., 6 2020. doi:[10.1109/HORA49412.2020.9152832](https://doi.org/10.1109/HORA49412.2020.9152832).
- [12] Loris Roveda, Mauro Magni, Martina Cantoni, Dario Piga, and Giuseppe Bucca. Human–robot collaboration in sensorless assembly task learning enhanced by uncertainties adaptation via bayesian optimization. *Robotics and Autonomous Systems*, 136:103711, 2 2021. doi:[10.1016/j.robot.2020.103711](https://doi.org/10.1016/j.robot.2020.103711).
- [13] Kimoon Lee, Sungmoon Joo, and Henrik I. Christensen. An assembly sequence generation of a product family for robot programming. volume 2016-November, pages 1268–1274. Institute of Electrical and Electronics Engineers Inc., 11 2016. doi:[10.1109/IROS.2016.7759210](https://doi.org/10.1109/IROS.2016.7759210).
- [14] J. Huckaby and H. Christensen. Modeling robot assembly tasks in manufacturing using sysml. pages 1–7, 2014. URL: <https://ieeexplore.ieee.org/document/6840209>.
- [15] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3:233–242, 6 1999. doi:[10.1016/S1364-6613\(99\)01327-3](https://doi.org/10.1016/S1364-6613(99)01327-3).
- [16] Sonia Chernova and Andrea L. Thomaz. Special issue on robot learning from demonstration. *International Journal of Social Robotics*, 4:317–318, 11 2012. URL: <https://link.springer.com/article/10.1007/s12369-012-0165-8>, doi:[10.1007/s12369-012-0165-8](https://doi.org/10.1007/s12369-012-0165-8).
- [17] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Pres, Cambridge, MA, London, England, 1 edition, 1998.
- [18] Chrystopher Nehaniv and Kerstin Dautenhahn. Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications, 6 1999. doi:[10.1142/9789812792747_0007](https://doi.org/10.1142/9789812792747_0007).
- [19] Aude Billard, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. Robot programming by demonstration, 2008. URL: http://link.springer.com/10.1007/978-3-540-30301-5_60, doi:[10.1007/978-3-540-30301-5_60](https://doi.org/10.1007/978-3-540-30301-5_60).
- [20] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483, 5 2009. doi:[10.1016/j.robot.2008.10.024](https://doi.org/10.1016/j.robot.2008.10.024).
- [21] Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. pages 1315–1322, 2007. doi:[10.1145/1329125.1329407](https://doi.org/10.1145/1329125.1329407).
- [22] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. 2 1970.
- [23] Tetsunari INAMURA Masayuki INABA Hirochika. Acquisition of probabilistic behavior decision model based on the interactive teaching method. 2001.
- [24] Joe Saunders, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. volume 2006, pages 118–125. Association for Computing Machinery (ACM), 2006. doi:[10.1145/1121241.1121263](https://doi.org/10.1145/1121241.1121263).

- [25] John D. Sweeney and Rod Grupen. A model of shared grasp affordances from demonstration. pages 27–35. IEEE Computer Society, 2007. doi:[10.1109/ICHR.2007.4813845](https://doi.org/10.1109/ICHR.2007.4813845).
- [26] Stanislao Lauria, Guido Bugmann, Theodoris Kyriacou, and Ewan Klein. Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38:171–181, 3 2002. doi:[10.1016/S0921-8890\(02\)00166-5](https://doi.org/10.1016/S0921-8890(02)00166-5).
- [27] Monica N. Nicolescu and Maja J. Matarić. Experience-based representation construction: Learning from human and robot teachers. *IEEE International Conference on Intelligent Robots and Systems*, 2:740–745, 2001. doi:[10.1109/IROS.2001.976257](https://doi.org/10.1109/IROS.2001.976257).
- [28] U. Nehmzow, O. Akanyeti, Christoph Weinrich, Theodoris Kyriacou, and S. A. Billings. Robot programming by demonstration through system identification. pages 801–806, 2007. doi:[10.1109/IROS.2007.4399087](https://doi.org/10.1109/IROS.2007.4399087).
- [29] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. volume 47, pages 79–91, 6 2004. doi:[10.1016/j.robot.2004.03.003](https://doi.org/10.1016/j.robot.2004.03.003).
- [30] Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. pages 255–262, 2007. doi:[10.1145/1228716.1228751](https://doi.org/10.1145/1228716.1228751).
- [31] Antonio Chella, Haris Ćzindo, Ignazio Infantino, and Irene Macaluso. A posture sequence learning system for an anthropomorphic robotic hand. volume 47, pages 143–152, 6 2004. doi:[10.1016/j.robot.2004.03.008](https://doi.org/10.1016/j.robot.2004.03.008).
- [32] Daniel H. Grollman and Aude G. Billard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 4:331–342, 11 2012. URL: <https://link.springer.com/article/10.1007/s12369-012-0161-z>, doi:[10.1007/s12369-012-0161-z](https://doi.org/10.1007/s12369-012-0161-z).
- [33] Shuonan Dong and Brian Williams. Learning and recognition of hybrid manipulation motions in variable environments using probabilistic flow tubes. *International Journal of Social Robotics*, 4:357–368, 11 2012. URL: <https://link.springer.com/article/10.1007/s12369-012-0155-x>, doi:[10.1007/s12369-012-0155-x](https://doi.org/10.1007/s12369-012-0155-x).
- [34] Yasser Mohammad and Toyooki Nishida. Fluid imitation: Discovering what to imitate? *International Journal of Social Robotics*, 4:369–382, 11 2012. URL: <https://link.springer.com/article/10.1007/s12369-012-0153-z>, doi:[10.1007/s12369-012-0153-z](https://doi.org/10.1007/s12369-012-0153-z).
- [35] Brenna D. Argall, Brett Browning, and Manuela M. Veloso. Policy feedback for the refinement of learned motion control on a mobile robot. *International Journal of Social Robotics*, 4:383–395, 11 2012. URL: <https://link.springer.com/article/10.1007/s12369-012-0156-9>, doi:[10.1007/s12369-012-0156-9](https://doi.org/10.1007/s12369-012-0156-9).
- [36] Rainer Jakel, Sven R. Schmidt-Rohr, Steffen W. Rühl, Alexander Kasper, Zhixing Xue, and Rüdiger Dillmann. Learning of planning models for dexterous manipulation based on human demonstrations. *International Journal of Social Robotics*, 4:437–448, 11 2012. URL: <https://link.springer.com/article/10.1007/s12369-012-0162-y>, doi:[10.1007/s12369-012-0162-y](https://doi.org/10.1007/s12369-012-0162-y).

- [37] Stuart Russell. Learning agents for uncertain environments (extended abstract). pages 101–103. ACM Press, 1998. URL: <http://portal.acm.org/citation.cfm?doid=279943.279964>, doi:10.1145/279943.279964.
- [38] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. pages 1–8. Association for Computing Machinery (ACM), 2004. URL: <http://portal.acm.org/citation.cfm?doid=1015330.1015430>, doi:10.1145/1015330.1015430.
- [39] Nematollah Ab Azar, Aref Shahmansoorian, and Mohsen Davoudi. From inverse optimal control to inverse reinforcement learning: A historical review, 1 2020. doi:10.1016/j.arcontrol.2020.06.001.
- [40] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. 12 2016. URL: <http://arxiv.org/abs/1701.00160>.
- [41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63:139–144, 10 2020. URL: <http://www.github.com/goodfeli/adversarial>, doi:10.1145/3422622.
- [42] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. pages 4572–4580. Neural information processing systems foundation, 6 2016. URL: <https://arxiv.org/abs/1606.03476v1>.
- [43] Bernhard Hengst. *Hierarchical Reinforcement Learning*, pages 495–502. Springer US, Boston, MA, 2010. URL: https://doi.org/10.1007/978-0-387-30164-8_363, doi:10.1007/978-0-387-30164-8_363.
- [44] Hoang M. Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé. Hierarchical imitation and reinforcement learning. *35th International Conference on Machine Learning, ICML 2018*, 7:4560–4573, 3 2018. URL: <http://arxiv.org/abs/1803.00590>.
- [45] Manfred Huber. Hybrid architecture for hierarchical reinforcement learning. volume 4, pages 3290–3295, 2000. doi:10.1109/robot.2000.845170.
- [46] Masanao Obayashi, Kenichiro Narita, Takashi Kuremoto, and Kunikazu Kobayashi. A reinforcement learning system with chaotic neural networks-based adaptive hierarchical memory structure for autonomous robots. pages 69–74, 2008. doi:10.1109/ICCAS.2008.4694529.
- [47] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.
- [48] Boyao Li, Tao Lu, Xiaocan Li, Yinghao Cai, and Shuo Wang. An automatic robot skills learning system from robot’s real-world demonstrations. *Proceedings of the 31st Chinese Control and Decision Conference, CCDC 2019*, pages 5138–5142, 6 2019. doi:10.1109/CCDC.2019.8833143.
- [49] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Feature construction for inverse reinforcement learning. 2 2010.

- [50] Dizan Vasquez, Billy Okal, and Kai O. Arras. Inverse reinforcement learning algorithms and features for robot navigation in crowds: An experimental comparison. pages 1341–1346. Institute of Electrical and Electronics Engineers Inc., 10 2014. doi:10.1109/IROS.2014.6942731.
- [51] Daehyung Park, Michael Noseworthy, Rohan Paul, Subhro Roy, and Nicholas Roy. Inferring task goals and constraints using bayesian nonparametric inverse reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1005–1014. PMLR, 30 Oct–01 Nov 2020. URL: <https://proceedings.mlr.press/v100/park20a.html>.
- [52] Sanjay Krishnan, Animesh Garg, Richard Liaw, Lauren Miller, Florian T. Pokorny, and Ken Goldberg. Hirl: Hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards. 4 2016. URL: <http://arxiv.org/abs/1604.06508>.
- [53] Liting Sun, Wei Zhan, and Masayoshi Tomizuka. Probabilistic prediction of interactive driving behavior via hierarchical inverse reinforcement learning. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018-November:2111–2117, 9 2018. URL: <http://arxiv.org/abs/1809.02926>.
- [54] Naijun Liu, Tao Lu, Yinghao Cai, Boyao Li, and Shuo Wang. Hindsight generative adversarial imitation learning. 3 2019. URL: <http://arxiv.org/abs/1903.07854>.
- [55] Yiming Ding, Carlos Florensa, Mariano Phielipp, and Pieter Abbeel. Goal-conditioned imitation learning. *Advances in Neural Information Processing Systems*, 32, 6 2019. URL: <http://arxiv.org/abs/1906.05838>.
- [56] Armando Plasencia, Yulia Shichkina, Ileana Suárez, and Zoila Ruiz. Open source robotic simulators platforms for teaching deep reinforcement learning algorithms. volume 150, pages 162–170. Elsevier B.V., 1 2019. doi:10.1016/j.procs.2019.02.031.
- [57] Mirella Santos Pessoa De Melo, Jose Gomes Da Silva Neto, Pedro Jorge Lima Da Silva, Joao Marcelo Xavier Natario Teixeira, and Veronica Teichrieb. Analysis and comparison of robotics 3d simulators. pages 242–251. Institute of Electrical and Electronics Engineers Inc., 10 2019. doi:10.1109/SVR.2019.00049.
- [58] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. 6 2016. URL: <http://arxiv.org/abs/1606.01540>.
- [59] OpenAI. Getting started with gym, 2021. URL: <https://gym.openai.com/docs/>.
- [60] Lucas Nogueira. Comparative analysis between gazebo and v-rep robotic simulators, 2 2014. doi:10.13140/RG.2.2.18282.36808.
- [61] Matteo Lucchi, Friedemann Zindler, Stephan Mühlbacher-Karrer, and Horst Pichler. robogym – an open source toolkit for distributed deep reinforcement learning on real and simulated robots. *arXiv*, 7 2020. URL: <http://arxiv.org/abs/2007.02753>.
- [62] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola,

- Mathias Lüttke, and Enrique Fernández Perdomo. *ros_control*: A generic and simple control framework for ros. *The Journal of Open Source Software*, 2017. URL: <http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf>, doi:10.21105/joss.00456.
- [63] Matteo Lucchi and Friedemann Zindler. *robo-gym* github repository. URL: <https://github.com/jr-robotics/robo-gym>.
- [64] Universal robot github repository. URL: https://github.com/ros-industrial/universal_robot.
- [65] Jointtrajectory message. URL: http://docs.ros.org/en/melodic/api/trajectory_msgs/html/msg/JointTrajectory.html.
- [66] Daniel Ordonez. *Robotiq 2 finger grippers* github repository. URL: https://github.com/Danfoa/robotiq_2finger_grippers.
- [67] Xiangyu Chen. *Robotiq-ur5* github repository. URL: <https://github.com/cxy1997/Robotiq-UR5>.
- [68] utecrobotics. *ur5* github repository. URL: <https://github.com/utecrobotics/ur5>.
- [69] ros industrial. *Robotiq* github repository. URL: <https://github.com/ros-industrial/robotiq>.
- [70] Geonhee Lee. *ur reinforcement learning* github repository. URL: <https://github.com/Geonhee-LEE/ur-reinforcement-learning>.
- [71] Cristian Beltran. *ur3* github repository. URL: <https://github.com/cambel/ur3>.
- [72] Konstantinos Chatzilygeroudis, Nick Lamprianidis, Martin Günther, Stefan Kohlbrecher, Mikael Arguedas, and Yuki Onish. *Mimicjointplugin* github repository. URL: https://github.com/roboticsgroup/roboticsgroup_gazebo_plugins.
- [73] Jennifer Buehler. *Gazebo grasp fix plugin* github repository. URL: <https://github.com/JenniferBuehler/gazebo-pkgs/wiki/The-Gazebo-grasp-fix-plugin>.
- [74] Eitan Marder-Eppstein, Vijay Pradeep, and Mikael Arguedas. *actionlib* package summary. URL: <http://wiki.ros.org/actionlib>.
- [75] Sachin Chitta. *Gripperactioncontroller* class template reference. URL: http://docs.ros.org/en/hydro/api/gripper_action_controller/html/classgripper__action__controller_1_1GripperActionController.html.
- [76] Eitan Marder-Eppstein and Pradeep. *Simpleactionclient* class template reference. URL: https://docs.ros.org/en/diamondback/api/actionlib/html/classactionlib_1_1SimpleActionClient.html.
- [77] *Grippercommand* control message reference page. URL: http://docs.ros.org/en/melodic/api/control_msgs/html/msg/GripperCommand.html.
- [78] *Physics parameters- gazebo* tutorial. URL: http://gazebo.org/tutorials/tut=physics_params.

- [79] Imitation trajectorywithrew datatype. URL: <https://gym.openai.com/envs/FetchPickAndPlace-v1/>.
- [80] Env class. URL: <https://github.com/openai/gym/blob/267916b9d268c37cc948baf35606c665aac53ac/gym/core.py#L8>.
- [81] Box space definition. URL: <https://github.com/openai/gym/blob/master/gym/spaces/box.py>.
- [82] Steven Wang, Sam Toyer, Adam Gleave, and Scott Emmons. The imitation library for imitation learning and inverse reinforcement learning. <https://github.com/HumanCompatibleAI/imitation>, 2020.
- [83] Imitation quickstart github. URL: <https://github.com/HumanCompatibleAI/imitation/blob/master/examples/quickstart.py>.
- [84] Imitation trajectorywithrew datatype. URL: <https://github.com/HumanCompatibleAI/imitation/blob/ae8227073479afc1ebcb83b94991006cfe2fc0a2/src/imitation/data/types.py#L80>.