UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Head Office: Università degli Studi di Padova

Department of Information Engineering

---

Ph.D. COURSE IN: INFORMATION ENGINEERING
CURRICULUM: ICT
SERIES 34

# HUMAN-AWARE TASK AND MOTION PLANNING THROUGH DEEP REINFORCEMENT LEARNING

Thesis written with the financial contribution of STIIMA - CNR

**Coordinator:** Prof. Andrea Neviani
**Supervisor**: Prof. Stefano Ghidoni
**Co-Supervisor**: Nicola Pedrocchi

**Ph.D. student** : Giorgio Nicola

*Allora signori, abbiamo lavorato tutto il giorno... è il momento della locura.*
*Motore!!... E andiamo a girare signori... azione!!*
*Boris 3*

# Contents

# Abstract

This thesis investigates the application of Deep Reinforcement Learning to develop human-aware task and motion planners. Human-robot applications introduce a set of criticalities to the problem of Task and Motion Planning. Indeed, human-robot applications are non-determinism and highly dynamic; thus, it is necessary to compute plans quickly and adapt to an ever-changing environment. Therefore, this thesis studied the planning problem as a sequential decision-making problem modeled as Markov Decision Process solved via Reinforcement Learning. Markov Decision Processes are a possible answer to the problem of non-deterministic and dynamic environments. Indeed, on the one hand, are stochastic models, and on the other hand, rather than computing a complete plan at the beginning of each activity, step by step, the optimal action to perform is computed based on the current status of the environment. In particular, it is firstly investigated the task planning and the motion planning problems separately; subsequently, the combined problem is studied.

The proposed solutions proved to be able to compute quick and effective task plans, motion plans, task and motion plans in dynamic e non-deterministic applications like human-robot cooperation. In all the applications, it was noticed that the agent was able to identify hazardous situations and minimize the risk, for example, in task planning by choosing the task with lower failure probability or in motion planning by avoiding region of space with a high probability of collision. Furthermore, it was possible to ensure safety by combining human-aware Task and Motion Planning with current industry safety standards.

# List of figures

# List of tables

# List of Acronyms

**APF** . . . . . Artificial Potential Fields

**DPG** . . . . . Deterministic Policy Gradient

**DDPG** . . . . . Deep Deterministic Policy Gradient

**DDQN** . . . . . Double Deep Q Network

**DoF** . . . . . Degree of Freedom

**DQN** . . . . . Deep Q Network

**DRL** . . . . . Deep Reinforcement Learning

**HRC** . . . . . Human Robot Collaboration

**MDP** . . . . . Markov Decision Process

**ML** . . . . . Machine Learning

**MP** . . . . . Motion Planing

**NN** . . . . . Neural Network

**PDDL** . . . . . Planning Domain Definition Language

**PER** . . . . . Prioritized Experience Replay

**PFL** . . . . . Power and Force Limiting

**POMPD** . . . . . Partially Observable Markov Decision Process

**ProMP** . . . . . Probabilistic Movement Primitives

**RL** . . . . . Reinforcement Learning

**SSM** . . . . . Speed and Separation Monitoring

**TAMP** . . . . . Task And Motion Planning

**TD**3 . . . . . Twin Delayed DDPG

**TP** . . . . . Task Planning

*This is your last chance. After this, there is no turning back. You take the blue pill, the story ends, you wake up in your bed and believe whatever you want to believe. You take the red pill, you stay in Wonderland, and I show you how deep the rabbit-hole goes.*

Matrix, Wachowski Sisters

# 1 Introduction

In this Chapter, the problem of Task and Motion Planning applied to Human Robot Cooperation is introduced, taking into consideration current trends in both the industry and the research community. A comprehensive analysis of the state of the art of Task and Motion Planning in Human Robot Cooperation is presented, thus in next chapters only a brief focus on specific relevant aspects will be performed.

## 1.1 Motivations

In the last decade, the number of new industrial robots installed every year has more than doubled in Europe and the Americas, while in Asia and Australia, it has more than quadrupled, mainly thanks to the growth of China (source: International Federation of Robotics). However, one of the most significant limitations that still exist today to the application of robots in many industrial contexts and other fields, is the capability of the robot to interact safely with humans. Indeed, since a few years ago, robots were restricted to factories and physically separated with cages from humans since it was not possible to maintain a high level of productivity while ensuring safety for the people. In the last years, however, a new family of robots, the Cobots (collaborative robots), has quickly risen, opening new opportunities to develop new robotic applications. Unlike standard industrial robots, Cobots are specifically designed to enable robots and humans to work together, taking advantage of the potentiality of each, on one side the creativity and flexibility of humans, on the other side the efficiency and high productivity of robots. As it is shown in Figure 1.1, various levels of collaborations between human and robot are possible, each one requiring a different level of intrinsic safety features (i.e., sensors and algorithms internal to the robot). However, despite the cobots, most of the human-robot application consists either in the coexistence without physical separation but without sharing the workspace, or sequential collaboration where humans and robots sequentially share the same workspace, not simultaneously.

Instead, to achieve effective true collaboration between humans and robots, robots must be able to simultaneously share the same workspace with humans, work on the same parts with humans, and act accordingly and safely to the human partner.

Such a scenario requires the robot to choose the most appropriate action/task and be able to move accordingly to the human, which can be modeled first as a Task Planning problem and the second as a Motion Planning problem. However, the presence of the human introduces various difficulties. First, it is a highly dynamic environment since the human is constantly moving within the robot workspace. Second, the environment is highly non-deterministic since it is difficult to predict human behavior since many factors must be considered. For example, humans typically perform tasks based on personal preferences rather than in a fixed and common manner [1]. Therefore, the robot must be able to adapt to human personal preferences. Another factor that should be considered is the mutual coupling of robots and humans when performing tasks. For example, given a set of possible tasks that can be performed by both robot and human, which action one should perform strictly depends on the action the other partner is currently performing or is going to perform. Similarly, the human and robot movements are strictly coupled since the human movements influence the robot's one and vice versa [2]. For example, the robot must be able to avoid collision with the human, but similarly, the human tries to avoid collision with the robot and, at the same time, when possible, tries to minimize the possible interferences with the robot. The human can also accidentally behave more like a disturbance to the robot for multiple. For example, the human could perform actions that partially conflict with the robot task for

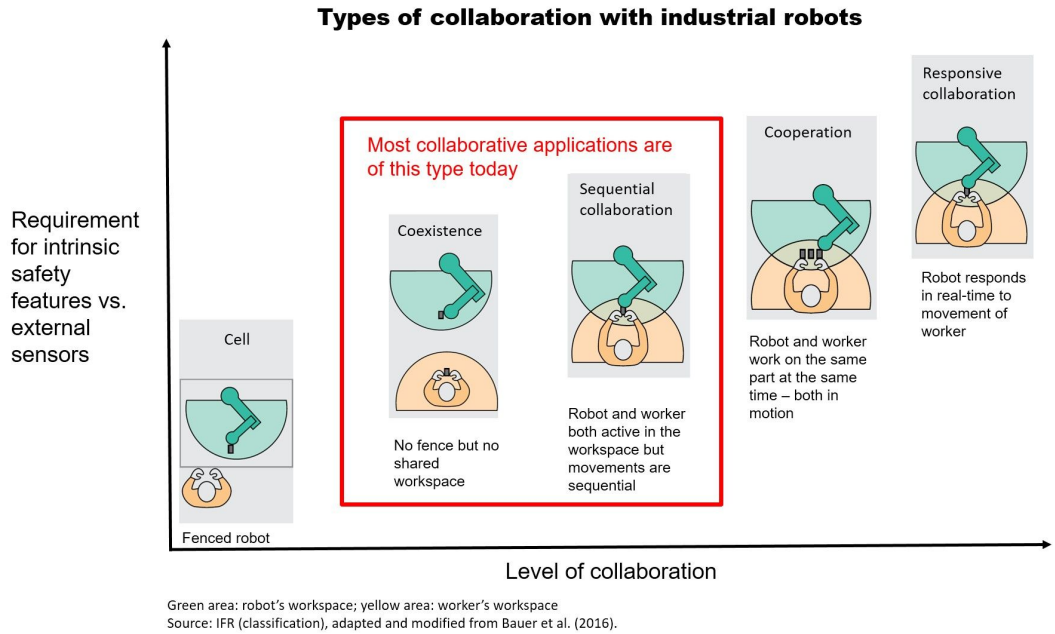



Figure 1.1: Possible levels of collaboration compared with the required level of safety, source: International Federation of Robotics

personal preferences. Finally, depending on the robot's behavior, the human could also decide to change his/her current task or movement to collaborate more effectively. However, such behavior might conflict with future planned actions by the robot. Last but not least, human trust in the robotic partner is a critical factor to consider [3].

On top of that, Motion Planning and Task Planning cannot always be considered independent problems. Indeed, to evaluate the optimality of an action, it must be evaluated how it is performed and, therefore, what movement is used. Thus, Task and Motion Planning must be studied together in what is called Task And Motion Plan (TAMP). When applied to deterministic environments, TAMP is an extremely complex problem since it requires solving simultaneously both Task Planning and Motion Planning that are NP-hard problems. The TAMP problem in a human-robot scenario becomes even harder since all the aforementioned criticalities must be considered, and on top of that, it is also necessary that the task and motion plan is computed quickly to avoid excessive idle time.

In conclusion, when planning actions and movements in scenarios where humans and robots must collaborate, the robot must compute quick and flexible plans that can adapt to sudden variations in the environment. Therefore, in this dissertation, we have decided to study Task Planning and Motion Planning problems as Markov Decision Processes (MDP). MDPs introduce two advantages; first, it is a mathematical framework that easily introduces the intrinsic stochasticity of human-robot collaborative environments. Second, in standard planning methods, the planner computes all the plans from the starting state to the goal

state, eventually considering human behavior and stochasticity. However, the longer the plan is, the less likely the human will behave as predicted, and therefore, the plan will probably fail. In this case, standard planning algorithms require the computation of a new plan that can be a time-consuming activity. Instead, in MDPs, a policy representing the optimal action for each state is computed offline. During the online phase, the robotic agent computes the optimal action for the next time step evaluating the policy, avoiding idle times for re-planning. Finally, Reinforcement Learning (RL), and in particular, Deep Reinforcement Learning (DRL), was used to solve the studied MDPs. Indeed, in the last years, DRL had proven the capability of solving problems that before were considered almost impossible, such as the game of Go [4]. Furthermore, DRL, since it implements Neural Networks (NN) that are considered universal function approximators, allows the use of multiple data sources with minimum variation in the training process. For example, it is possible to represent the MDP state as images or as vectors of discrete and continuous values and many more possibilities.

## 1.2 State of the Art

### 1.2.1 Task and Motion Planning

Solving high-level complexity tasks, such as assembly tasks, requires solving two types of problems on two distinct but deeply interconnected levels. The higher level, defined as Task Planning (TP), aims at finding the discrete sequence of actions/subtasks necessary to complete the desired task. On this level, symbolic reasoning is dominant, since the constraints of the problem are mainly logical. For example, every action is characterized by a set of preconditions that must be fulfilled to perform it and by a set of effects representing the scenario's evolution after the action. Meanwhile, the lower level, defined as Motion Planning (MP), consists in finding a free collision path for the robot necessary to complete the required action/subtask, and instead, it requires geometrical reasoning. Motion Planning and Task Planning are connected problems, since the solution of one problem affects the solution of the other. Let us consider a scenario where a robot must move various objects. A Task Planning action can make unfeasible a subsequent action from the Motion Planning point of view. For example, it could place one object interfering with another object that should be moved afterward. Vice versa, given an action with multiple Motion Planning solutions, one solution may reduce the execution optimality of a successive action. For example, an object could be picked with configuration robot configurations; however, one robot configuration may require a very long movement to reach the successive object to pick while another configuration may require a shorter movement.

Therefore, those two problems need to be addressed simultaneously as a single problem that spaces over different time scales and domains. Spacing over the time domain let switch from the entire task execution to the single action execution. Spacing over the domains allows investigating both discrete actions and continuous robot paths.

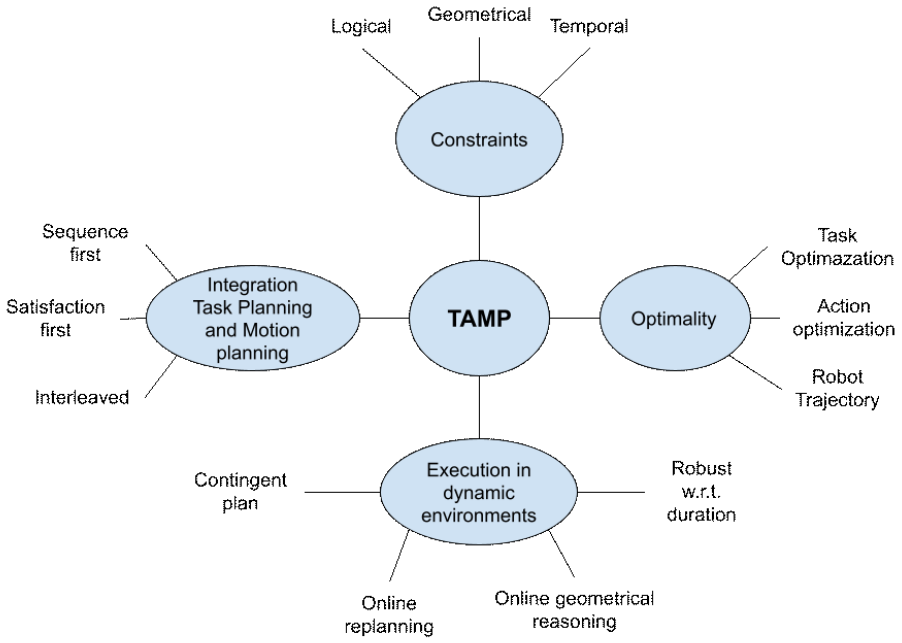



Figure 1.2: Classification of TAMP algorithms based on their features

As shown in Figure 1.2, task and Motion Planning methods in literature based on different categories that are:

- how the task planner and the motion planner are integrated;
- the constraints taken into consideration;
- the level of optimality they can achieve;
- how they behave in dynamic environments.

Constraints Three possible kind of constraints can be introduced in TAMP problem: *logical*, *geometrical* and *temporal*. The first two are the basic requirements for a TAMP problem, indeed the logic constraints describe the symbolic problem of the Task Planning meanwhile, the geometric constraints describe the Motion Planning problem to be solved. Some works introduce also temporal constraints [5, 6], e.g., some tasks might need to be performed before a certain moment (this is a very common requirement in industrial applications) or the duration of the execution of a task is not certain. Temporal constraints are typically dealt by introducing the Timelines task plans [6, 7] were the plan is a sequence of synchronized temporal behaviors.

Task planning and Motion Planning integration There are three methods to integrate task planner and motion planner in literature: *sequencing first*, *satisfaction first* and *interleaved*. The *sequencing* first approach first found a sequence of actions that satisfies the task level constraints, and subsequently checks the Motion Planning feasibility of each action [8, 9]. Thus, in case of a collision a new task plan solution is computed, eventually adding information or constraints from the motion planner, example of this approach are The *satisfaction first* approach instead solves the TAMP problem with an opposite approach: it precomputes the motion feasibility of each action, and it exploits this information to reduce the set of possible Task Planning solutions. This approach is extremely powerful in static environments, it is possible to compute offline the Motion Planning solution for each action. Indeed, the Motion Planning problem can be computationally extremely expensive to solve, solving it offline can reduce robot idle time due to ongoing planning. An example of this approach is Robosynth [10] which first generates a manipulation graph whose nodes represent robot configuration and all the possible position of the object that can be manipulated. Subsequently, those nodes are connected by arches corresponding to feasible motion plans. Finally, the manipulation graph combined with a semantic description of the desired task is used by the Satisfiability Modulo Theories (SMT) to find a feasible task and motion plan. The same approach is further expanded in [11] introducing policies to deal with dynamic environments. Finally, the interleaved integration uses the motion planner as a subroutine of the task planner, therefore the motion level feasibility of each task is verified only when the task is evaluated as part of the candidate solution. In [12] the authors proposes a modification of the Planning Domain Definition Language (PDDL) [13] to introduce as semantic attachment to evaluate motion feasibility during the Task Planning. In various works, Task Planning is addressed as a tree search problem where Motion Planning is used as an instrument to prune the tree that otherwise might be extremely big for example as in [14].

Optimality Optimality can be achieved on different levels: action, goal and robot trajectory. Action level optimization aims at finding the sequence of actions that minimize a desired cost function based only on symbolic reasoning. Goal optimization instead aims at achieving the optimal execution of a task introducing geometrical reasoning. For example, let's consider the action of picking a blue box while multiple blue boxes are available, goal optimization aims at finding the best blue box to pick. Finally, even the robot trajectory can be optimized. For example, the most convenient blue box should be picked with the most efficient robotic trajectory. Typically, the level of optimality the task and motion planner can achieve is a trade-off with the time available to compute the overall plan. However, various example of Task and Motion Planner that optimize both actions, goals and trajectories have been developed as [15, 16] In dynamic environments, where it is necessary to compute quickly the plan, feasibility is typically preferred over optimality.

Execution in dynamic environment The deployment of task and motion planners in dynamic environments introduce the necessity to develop plans that can deal with changing environments or uncertainty due to the presence of stochastic processes (e.g., a task might have a probability of failing or there is another agent that does not behave deterministically). In literature, various approaches can be found, the most simple is based on online re-planning of all the task and motion plan when unforeseen events occur. However, this approach is very little efficient in complex environments since it would cause excessive robot idle time. On the other hand, uncertainty can be efficiently dealt either developing contingency strategies [17, 18, 19, 20] or with online geometrical reasoning. Finally, in dynamic environments, also the action duration is likely not to be deterministic, and it raises a challenge when TAMP is applied to a scheduling problem. In those scenarios, a common approach is based on the introduction of Timelines [21], which can effectively describe the duration of the actions composing a plan.

### 1.2.2 Task and Motion Planning in Human Robot applications

An example of TAMP in dynamic environments concerning human-robot applications is detailed in [6]. Such scenario introduces other difficulties in addition to those mentioned in the previous sections. Indeed, the human introduces variability at both Task Planning and Motion Planning levels. On the Task Planning level, the human might perform the same task differently based on, for example, personal preferences or the current action performed by the robot. Thus, a task and motion plan previously computed may become unfeasible due to unexpected human actions. On the Motion Planning level, it is well known that robot motion affects human motion in human-robot shared workspaces [3]. Furthermore, the human movements are likely to affect the robot time execution of a required action. Therefore, a task and motion planner needs to be able to adapt to the human operator on both Task Planning and Motion Planning levels during the task execution. Therefore, TAMP in human-robot applications is tackled by introducing human-aware task planners and motion planners in the systems mentioned above, such as in [7, 22, 17]. In [7] Task Planning is handled by a centralized intelligence that computes and optimizes order, scheduling, and assignment of tasks between human and robot, taking into account temporal constraints and duration variability. In [23] task and Motion Planning is based on identifying of both human trajectories and human task plans. Once they are both inferred, the most effective robotic plan is chosen among a dataset, and the next goal position is sent to the motion planner that will generate safe robot trajectories based on the predicted human trajectory. Few existing works address the problem of TAMP in HRC scenarios, since it is already an extremely challenging problem without the presence of the human. Most of them usually study the two separated problems. Thus, in Sections 1.2.3 and 1.2.4 the principal Motion Planning and Task Planning methods applied to HRC will be detailed.

### 1.2.3 Motion Planning in Human Robot Cooperation

In literature several approaches exist to develop human aware motion planners. They can be divided into 3 categories:

- velocity scaling;
- reactive;
- predictive.

*Velocity scaling* methods first compute the motion plan not taking into consideration the current human position. To avoid collisions, they track the human position, and scale the trajectory velocity maintaining the geometrical path unchanged. Collision avoidance is achieved by stopping the robot when a collision is imminent; however, in many cases, it is only guaranteed that collisions can happen when the robot is at hold. Current industrial safety standards follow this principle, such as the Speed and Separation Monitoring (SSM) or the Power and Force Limiting (PFL) from the ISO TS 15066. In literature, various works propose different scaling methods. In [24], a combination of SSM and PFL is proposed to minimize the robot idle time. In [25], the Cartesian trajectory scaling is based on a Model Predictive Control that maximizes the distance between the robot and the human by taking advantage of the robot redundancy. In [26], velocity scaling is combined with the introduction of fail-safe maneuvers that allows the robot to stop, avoiding the workspace reachable by the human in the successive instants.

*Reactive methods* usually deform the current motion plan based on the current human position, typically introducing repulsive forces to avoid dynamic and fixed obstacles. One of the first noticeable methods is the Artificial Potential Fields (APF) [27], which generates an attractive potential field toward the desired goal and a set of repulsive potential fields for the obstacles. Thanks to its simplicity, APF is still very popular in the fields of mobile robots navigation [28] and unmanned aerial vehicles (UAV) [29]; however, it is not very commonly applied to robotic manipulators. Indeed, the main drawback of APF is that it can easily fall into local minima. Therefore, a very complex space like the joint space of a manipulator is not particularly suited. In [30], the authors propose the Elastic Strips framework, which performs collision avoidance by defining an elastic tunnel of free space around the candidate starting solution computed by the preferred motion planner. The candidate solution is subsequently deformed within the elastic tunnel based on a repulsive force to avoid collisions. In [31], the author presents a new method for collision avoidance in dynamic environments for manipulators based on projecting repulsive velocities in the null space of the end effector. It also imposes artificial forces on the control points of the robot links and translates them into joint velocity constraints. Finally, in [32] a library of Movement Primitive is implemented and each movement primitive describes the trajectory to perform a desired task. Movement primitives are first learned offline and then modified online via repulsive forces to

avoid collisions with obstacles, while Closed Loop Inverse Kinematics [33] is used to avoid self-collisions.

*Predictive motion planners* are based on the predicting the workspace occupied by the human in the future. The prediction is based on current human motion and/or the task the human is performing. In [34], a library of human arms movement is developed offline by means of Gaussian Mixture Models. Online early classification of the current human movement, i.e., classification from unfinished trajectories, is used to predict the occupied workspace. Therefore, a new plan is computed if the current robot motion plan is predicted to be in collision with the human. A similar approach is used in [22], where short-term human motions are regressed using sparse pseudo-input Gaussian process, then are used online to compute the optimal robot trajectory. In [35], in a scenario of autonomous driving in a crowd, a Partially Observable Markov Decision Process (POMDP) is used to predict the human goal and, therefore, to predict the human trajectory. Subsequently, a modified version of A* is used to compute the car trajectory while avoiding the human based on the predicted trajectory.

After the recent breakthroughs in the reinforcement learning community [36, 37], Deep Reinforcement Learning was applied to solve Motion Planning problems in dynamic environments. One of the most successful scenarios was navigation of mobile robots among crowds, as shown in [38, 39, 40]. It was also applied for Motion Planning of manipulators in dynamic environments in [41, 42]. The peculiarity of a DRL based motion planner is that the output is not a plan composed by a list of poses the robot has to reach but only the next action, for example, joints speed or joints acceleration.

### 1.2.4 Task Planning in Human Robot Cooperation

Typically, Task Planning in human-robot applications is formulated as an MDP and solved as a reinforcement learning problem. In [1], cross-training is exploited, a technique used to train human teams where the members iteratively switch roles between human and robot, combined with reinforcement learning to increase the overall human-robot team performance. In [22], instead, Q-Learning [43] is implemented; the policy is trained in a simulated environment combined with a human-aware motion planner and a probabilistic model describing the preferred sequence of human actions. Furthermore, a specific reward function is designed to minimize execution delays due to collision avoidance and maximize the probability that the robot performs a task from which the human can benefit. In [44], the authors formulate the problem as a Multi Agent Reinforcement Learning problem solved with Q-learning where one agent is the robot, and the other is the human. In detail, a Q-value function is learned for each agent; however, the human policy is trained starting from the human preferred action sequence. Subsequently, the human develops a policy that partially adapts to the robot and partially preserves the original preferred action sequence.

Other works model the collaborative task as a POMPD where the current human intention is modelled as a hidden latent variable of the state of the belief space [45, 46]. How-

ever, computing the solution of POMPD is typically extremely computationally expensive. Therefore, it is a common approach to use approximated solutions.

Finally, a different approach is introduced to consider the uncertainty of the execution time of all the tasks. Indeed, MDP-based methods can only introduce the execution time uncertainty in the reward function; therefore, balancing all the possible components of the reward functions is challenging. In [6] timelines from [21] are introduced combining Task Planning with scheduling.

*A good plan, violently executed now, is better than a perfect plan next week.*

George Patton

# 2 Reinforcement Learning

In this Chapter, the theoretical background of Reinforcement Learning is provided. First, the basic fundamental concepts are provided, and a simple taxonomy of Reinforcement Learning is described. Subsequently, the most influential model-free reinforcement learning techniques like Temporal-Difference Learning, SARSA, or Q-learning are described. Those techniques are the foundations of the current state-of-the-art model-free algorithm. Finally, the Deep Reinforcement Learning algorithm implemented in this thesis will be described. In particular, it will be studied Deep Q Networks with its extensions and Deterministic Policy Gradient with its extension Twin Delayed Deep Deterministic Policy Gradients.

## 2.1 Key Concepts

Reinforcement Learning (RL) is one of the main branches of machine learning alongside Supervised Learning and Unsupervised Learning, Figure 2.1. Given a task to solve, the objective of Reinforcement Learning is to find the optimal strategy (the policy) based on a feedback signal (the reward) that allows the agent to solve the required task. Differently from the other Machine Learning class of methods, it does not rely on a dataset of labeled examples like in Supervised Learning or unlabeled data like in Unsupervised Learning. Instead, it must explore the environment to collect experience in a trial and error fashion. In particular, Reinforcement Learning is based on a discrete time interaction between the agent and the surrounding environment as shown in Figure 2.2; the agent at a timestep $t$ performs an action $a_t$ on the environment based on the current state $s_t$, at the next timestep $t+1$ it receives a reward $r_{t+1}$, the feedback on the effects of the performed action on the environment, and the new current state $s_{t+1}$. This framework is mathematically described as a Markov Decision Process (MDP). A Markov Decision Process is a discrete-time stochastic control model used to model decision-making problems in stochastic environments where the outcomes are partially or totally dependent on the actions of a decision-making agent. An MDP is defined as the tuple $(S, A, P, r)$.

- $S$ is the set of possible states that fully describe the environment.
- $A$ is the set of possible actions the agent can perform in the environment.
- $P$ , called transition distribution, is the distribution over the next state $s_{t+1}$ from the current state $s_t$ and performing an action $a_t$, therefore we can write $P(s_{t+1}, s_t, a_t) =$

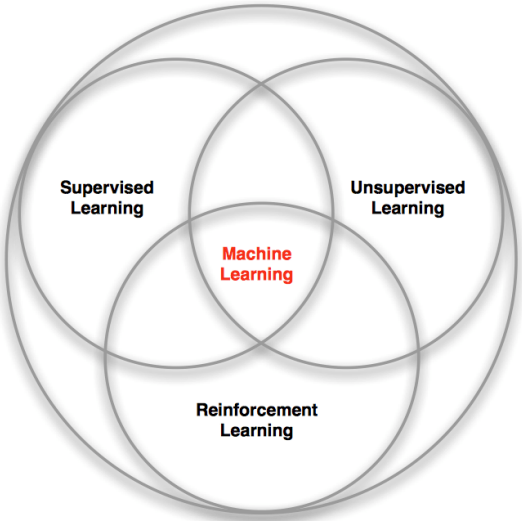



Figure 2.1: Machine Learnings and its branches

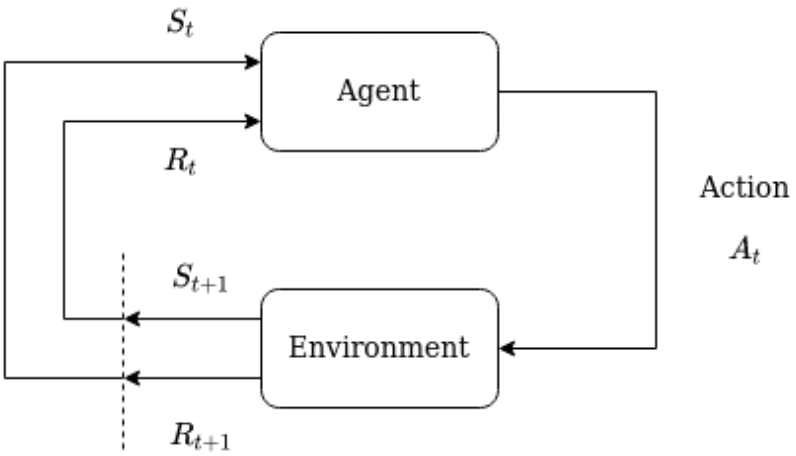



Figure 2.2: Reinforcement Learning scheme

$Pr(s_{t+1}|s_t, a_t)$. In other words, it describes the dynamic of the environment that can be stochastic or deterministic.

- $r$ is the reward received by the agent at every transition given the current state $s_t$, the next state $s_{t+1}$ and the performed action $a_t$, therefore $r = R(s_t, a_t, s_{t+1})$ where $R$ is the designed reward function.

An MDP has also the Markov property, therefore the transition probability depends only on the current state and not the history.

$$\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_1, \dots, s_t) \tag{2.1}$$

We define the trajectory $\tau$ as the sequence of states and actions

$$\tau = (s_0, a_0, s_1, a_1, \dots) \tag{2.2}$$

and we define the cumulated discounted reward,

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \,, \tag{2.3}$$

We define the policy $\pi$ as the function that maps from the state domain to the action domain. The objective is to find an optimal policy $\pi^*$ that maximize the expected reward

$$\pi^* = \text{argmax } J(\pi) = \text{argmax } \underset{\tau\sim\pi}{\mathbb{E}}[R(\tau)] \tag{2.4}$$

The major components of a reinforcement learning agent are the *Policy*, the *Value function* and the *Model*.

The *Policy* is the agent behaviour and maps from the current state $s_t$ to the action space. It can be deterministic $\pi(a_t|s_t) = a_t$ or it can be stochastic $\pi(s_t) = \mathbb{P}(a_t|s_t)$.

The *Value function* describes how good a state or a state-action pair is, by means of the expected cumulated reward. In particular, there are four main value functions:

- *On Policy Value Function*: It describes the expected cumulated reward given a state $s_t$ and if the agent acts according to the policy $\pi$.

$$V^{\pi}(s_t) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s_t] \tag{2.5}$$

- *On Policy Action-Value Function*: It describes the expected cumulated reward given the pair state $s_t$ and the action $a_t$, and if the agent acts according to the policy $\pi$.

$$Q^{\pi}(s_t, a_t) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s_t, a_0 = a_t] \tag{2.6}$$

- *Optimal Value Function*: It describes the expected cumulated reward given a state $s_t$ and if the policy $\pi$ and the agent acts according to the *optimal* policy $\pi^*$ in the environment.

$$V^{*}(s_t) = \mathop{\mathbb{E}}_{\tau\sim\pi^*}[R(\tau)|s_0 = s_t] \tag{2.7}$$

- *Optimal Action-Value Function*: It describes the expected cumulated reward given the pair state $s_t$ and the action $a_t$, and if the agent acts according to the *optimal* policy $\pi^*$ in the environment.

$$Q^{*}(s_t, a_t) = \mathop{\mathbb{E}}_{\tau\sim\pi^*}[R(\tau)|s_0 = s_t, a_0 = a_t] \tag{2.8}$$

The value and the action value functions have the same values when the same policy is applied, therefore:

$$\begin{aligned} V^{\pi}(s_t) &= Q^{\pi}(s_t, \pi(s_t)) \\ V^{*}(s_t) &= Q^{*}(s_t, \pi^*(s_t)) \end{aligned} \tag{2.9}$$

Given the optimal action-value function it is possible to compute the optimal action $a^*(s)$ as:

$$a^*(s_t) = \arg\max_{a}(Q^*(s_t, a)) \tag{2.10}$$

All the aforementioned value functions can be re-written based on the Bellman equation [47].

$$V^{\pi}(s_t) = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ s_{t+1}\sim P}}[r(s_t, a_t) + \gamma V^{\pi}(s_{t+1})] \tag{2.11}$$

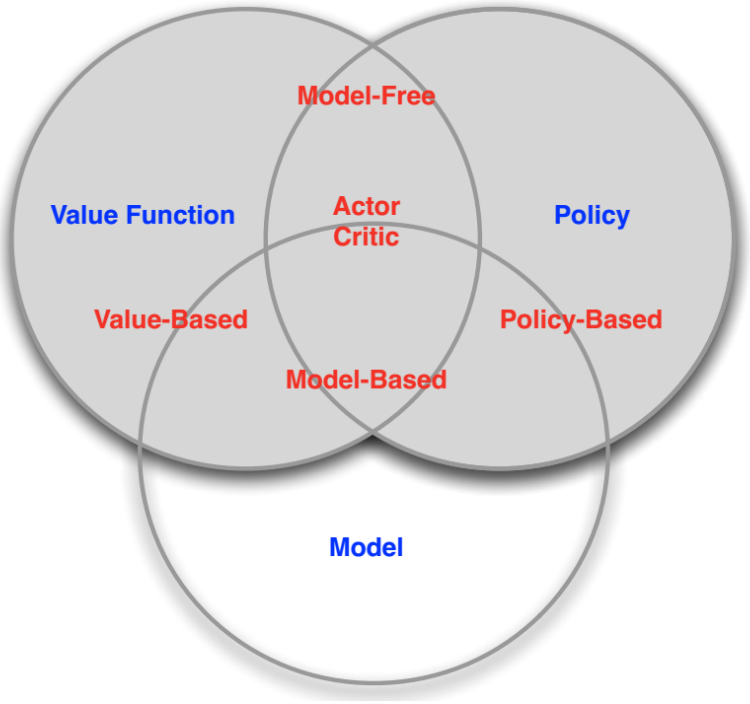



Figure 2.3: Classification of reinforcement learning algorithms

$$Q^{\pi}(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1}\sim P}[r(s_t, a_t) + \gamma \mathop{\mathbb{E}}_{a_{t+1}\sim\pi} Q^{\pi}(s_{t+1}, a_{t+1})]\,, \tag{2.12}$$

$$V^{*}(s_t) = \max_{a} \mathop{\mathbb{E}}_{s_{t+1}\sim P}[r(s_t, a_t) + \gamma V^{*}(s_{t+1})]\,, \tag{2.13}$$

$$Q^{*}(s_t, a_t) = \max_{a} \mathop{\mathbb{E}}_{s_{t+1}\sim P}[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^{*}(s_{t+1}, a_{t+1})]\,, \tag{2.14}$$

where $P$ is a shorthand for $\mathbb{P}(\cdot|s_t, a_t)$).

Finally, the model describes the environment, including the dynamic, i.e., given a state and an action, how it will evolve, and the reward after each transition. The model can be either learned or given at the beginning.

## 2.2 Taxonomy

In the literature, an enormous quantity of reinforcement learning methods have been developed, and they can be classified based on multiple characteristics as shown in Fig. 2.3. A classification can be made between those algorithms that use a model that can be learned or is given (*Model-Based*) and those algorithms that do not use a model of the environment (*Model-Free*). Model-based algorithms use the model of the environment to predict future steps. Typically, once the model is learned or given, they are much more sample efficient. However, the introduction of a model has various criticalities. First, in some scenario, a model cannot

be computed analytically, and therefore it must be learned. However, learning a model capable of representing the environment correctly can be an extremely time-consuming activity. Second, the policy may learn to take advantage of imperfections or loopholes in the model. Therefore, it can be troublesome to transfer the policy from a simulated environment used in the training phase to a real scenario.

Another classification can be made on the basis of how they compute the optimal actions. On the one hand, we have algorithms that use an implicit policy that is defined by a learned value function (*Value-based*). On the other hand, we have algorithms that directly learn an explicit policy (*Policy-based*). In between these classes, we have algorithms that learn both a value function and an explicit policy (*Actor-Critic*).

Algorithms can also be classified based on whether they learn from experience collected with the current policy (*On-Policy*) or they learn from experience not collected with the current policy (*Off-Policy*) but using a so-called behavior-policy. Off-policy algorithms have the advantage of achieving the optimal policy, while on-policy algorithms achieve sub-optimal policies since the policy must also maintain an exploratory behavior. Furthermore, off-policy algorithms are generally more sample efficient compared to on-policy algorithms.

In the following sections, for the sake of brevity, we will focus only on the model-free algorithms and in particular only on a family of algorithms called *Temporal Difference*, since only this family of algorithms has been used in this thesis. Model-free algorithms, mainly include 2 families of algorithms: *Monte Carlo* and *Temporal Difference* methods. The main difference between those two families lies on the frequency of update of the value functions and policy. Indeed, Monte Carlo methods updates the policy at the end of each episode, while Temporal difference methods update the value function and the policy in a step-by-step fashion.

## 2.3 Temporal-Difference methods

### 2.3.1 Temporal Difference Learning

One of the most influential techniques within the model-free algorithms is Temporal-Difference Learning (TD) [48]. In particular, it introduces the possibility of learning from incomplete episodes, thanks to bootstrapping, differently from Monte Carlo methods (MC), where it is required to complete the episode before updating the value function and the policy. In Monte Carlo methods, first, an episode is completed, all the transitions are stored, and subsequently, the following update rule to all the steps is implemented:

$$V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t)]\,, \tag{2.15}$$

where $G_t$ is the actual cumulated return following the state $s_t$ and $\alpha$ is a positive constant. Therefore, in order to know the exact value of $G_t$, it is necessary to wait for the completion of the episode.

In Temporal-Difference instead, the cumulated return is estimated thanks to the Value function (i.e., bootstrapping), therefore:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]\,. \tag{2.16}$$

We define $y$ as the *TD-target*, and $\delta$ as the *TD-error*.

$$y = r_t + \gamma V(s_{t+1})\,, \tag{2.17}$$

$$\delta = r_t + \gamma V(s_{t+1}) - V(s_t)\,. \tag{2.18}$$

This method is called TD(0), since it updates the value function based only on the current reward and looks 0 steps ahead. It is a special case of more general *n-step TD* where $n$ steps are used as described in the following equation.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})] \tag{2.19}$$

### 2.3.2 On-policy TD Control

SARSA [49] is one of the most influential applications of temporal difference to the control problem, and it applies TD(0) to learn the Q-value function in an *on-policy* fashion. In particular, the update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\,, \tag{2.20}$$

and the control algorithm is shown in Algorithm 2.1. As it can be noticed, the update is

**Algorithm 2.1** SARSA

Initialize $Q(s, a)\,\forall\, s \in S\,, \forall\, a \in A(s)$ and $Q(terminal\ state, a) = 0$
**repeat**
  Initialize $s_t$
  Choose an action $a_t$ derived from $Q(s_t, a)$ (e.g. $\epsilon$-greedy)
  **repeat** for each step episode
    Take action $a_t$ and observe $s_{t+1}$ and $r_{t+1}$
    Choose $a_{t+1}$ derived from $Q(s_{t+1}, a)$ (e.g. $\epsilon$-greedy)
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
    $s_t \leftarrow s_{t+1}$ and $a_t \leftarrow a_{t+1}$
  **until** $s_t$ is terminal
**until** Convergence is reached

performed at each time-step based on the quintuple of events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ that gives the name to the algorithm. Similarly to TD prediction, it is possible to extend SARSA to *n-step* SARSA and SARSA($\lambda$) [50].

### 2.3.3 Q-Learning

One of the major breakthrough in reinforcement learning was the implementation of TD control in an off-policy fashion [51] introducing the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]\,. \tag{2.21}$$

As it can be noticed the update rule in the Q-learning (2.21) differs from the update rule in SARSA (Equation 2.20) on how the future action for the update is computed. In SARSA, $a_{t+1}$ is computed based on the same behavioral policy used to explore (e.g. $\epsilon$-greedy) meanwhile in Q-Learning the target policy is used $\pi = \max_a Q(s_{t+1}, a)$.

**Algorithm 2.2** Q-Learning

Initialize $Q(s, a) \forall\, s \in S\,, \forall\, a \in A(s)$ and $Q(terminal\ state, a) = 0$
**repeat**
  Initialize $s_t$
  **repeat** for each step episode
    Choose an action $a_t$ derived from $Q(s_t, a)$ (e.g. $\epsilon$-greedy)
    Take action $a_t$ and observe $s_{t+1}$ and $r_{t+1}$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
    $s_t \leftarrow s_{t+1}$
  **until** $s_t$ is terminal
**until** Convergence is reached

Similarly to SARSA even Q-Learning can be extended to multistep and to eligibility traces.

### 2.3.4 Double Q-Learning

Algorithms like SARSA and Q-learning are affected by an overestimation of the action-value function, also known as the *maximization bias*, that caused by the target policy is obtained via maximization (2.21). For example, in Q-learning, the policy is greedy with respect to the action-value function, which is defined with a max operation. Therefore, a maximum overestimated value is used implicitly as an estimate of the maximum value, which can lead to a significant positive bias. Let's consider a single state $s$ where many actions can be taken whose true value $q(s, a)$ are zero, but the estimated value $Q(s, a)$, that is uncertain, is some distribution with positive and negative values. The maximum of the true values $\max_a\ q(s, a)$, is zero, meanwhile, the maximum of the estimated values, $\max_a\ Q(s, a)$, is a positive value, therefore a positive bias is present.

Double Q-learning [52] proposes to learn two independent estimators of the true action value function called, $Q_1(s, a)$ and $Q_2(s, a)$. Then, in order to estimate the value function, the maximizing action is computed with one estimator $A^* = \arg\max_a\ Q_1(s, a)$ while the

estimation is computed with the remaining estimator using the maximizing action from the first estimator $Q_2(s, A^*)$. The update rule for Double Q-learning became:

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha[r_{t+1} + \gamma\, Q_2(s_{t+1}, \arg\max_a Q_1(s_t, a)) - Q_1(s_t, a_t)]\,. \quad (2.22)$$

The pseudocode of Double Q-learning is shown in Algorithm 2.3

**Algorithm 2.3** Double Q-Learning

Initialize $Q_1(s, a)$ and $Q_2(s, a) \forall\, s \in S\,, \forall\, a \in A(s)$ and $Q_1(terminal\ state, a) = 0$ and $Q_2(terminal\ state, a) = 0$
**repeat**
  Initialize $s_t$
  **repeat** for each step episode
    Choose an action $a_t$ derived from $Q_1$ and $Q_2$ (e.g. $\epsilon$-greedy in $Q_1 + Q_2$ )
    Take action $a_t$ and observe $s_{t+1}$ and $r_{t+1}$
    With $0.5$ probability:
      $Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha[r_{t+1} + \gamma\, Q_2(s_{t+1}, \arg\max_a Q_1(s_t, a)) - Q_1(s_t, a_t)]$
    **else**
      $Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha[r_{t+1} + \gamma\, Q_1(s_{t+1}, \arg\max_a Q_2(s_t, a)) - Q_2(s_t, a_t)]$
    $s_t \leftarrow s_{t+1}$
  **until** $s_t$ is terminal
**until** Convergence is reached

## 2.4 Function Approximations and Neural Networks

The reinforcement learning algorithms presented in Section 2.3.2 and Section 2.3.3 are also called *tabular methods* since the value functions and the policy functions can be easily represented via tables. However, in most of the scenarios, the sets of states and actions are so large that representing them via tables and array is no longer practical, and when states and actions are continuous, tables cannot be used. For such reasons, *Function Approximators* are introduced to approximate the value functions and the policy. In particular 2 great families of function approximators *Linear* and *Non-Linear*. Linear methods approximate the function by the inner product between a vector of weights $\mathbf{w}$ and a vector of features $\mathbf{x}$.

$$v(s) \cong \hat{v}(s) = \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s) \quad (2.23)$$

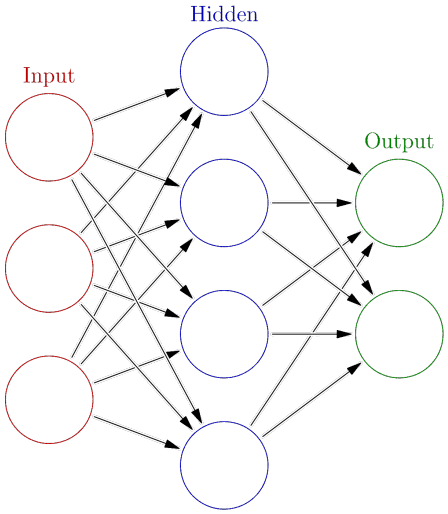



Figure 2.4: Scheme of a simple Neural Network

The feature vector is a vector composed by a set of basis functions chosen based on the properties of the studied problem, notable features polynomial functions, radial basis functions and Fourier basis.

Among the various non-linear function approximation methods, the most commonly used in RL are the Neural Networks (NNs). NNs are a computational model represented as a directed weighted graph whose units are called Neurons arranged in layers firstly studied in [53]. As shown in Figure 2.4, NNs can be divided into three parts, the first one is the input layer (the most left one), then there is the output layer (the most right one), and finally, there are all the hidden layers between the input and the output layer.

Each neuron is connected to some or all neurons of the subsequent layer that define some weighting of the unit's output at the start of the line (from left), to the unit's input at the end of the line. The most simple NN architecture is called FeedForward Neural Network, where each neuron is connected to all neurons of the successive layer.

In Figure 2.5 it is shown the schematics of the calculation that takes place in each neuron of the NN. First, a linear transformation and affine transformation, described by a set of weights $W$ and a bias unit $b$, are applied to a vector of inputs $x$.

$$v_k = \sum_{j=1}^{n} w_{kj} x_j + b_k \tag{2.24}$$

Subsequently, a non-linear activation function $\varphi(\cdot)$ is applied. Therefore, the output of each neuron is:

$$y_k = \varphi(v_k)\,. \tag{2.25}$$

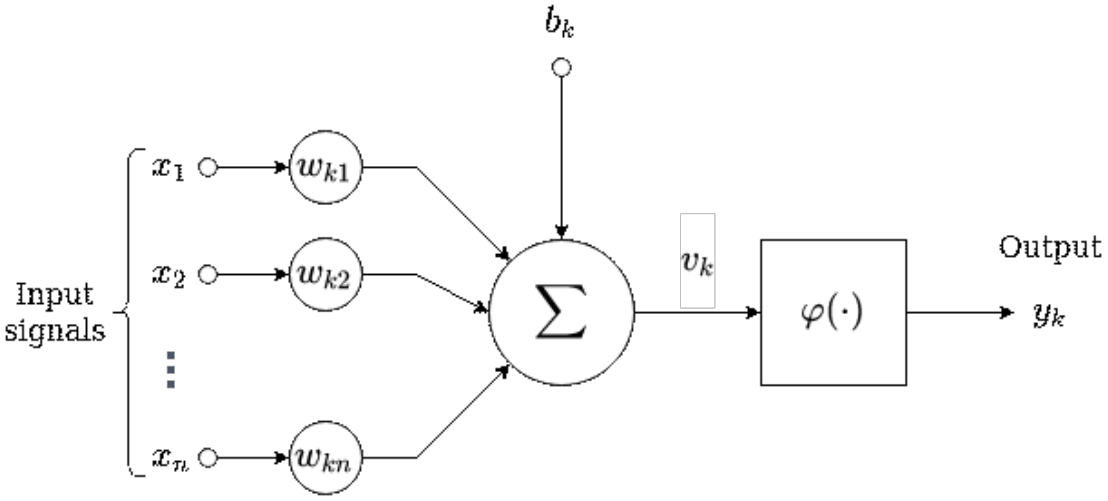



Figure 2.5: Schematic representation of the calculations that take place in a neural network neuron

The activation function denoted by $\phi(v)$ defines the output of a neuron in terms of the induced local field $v$. Activation function are typically non-linear since it is proven that linear activation function allow computing non-trivial problems. Some of the most commonly used activation functions are listed in Table 2.1

## 2.5 Deep Q-Learning and its extensions

The first algorithm to efficiently implement Neural Networks and in particular, Deep Neural Network for function approximation, was Deep Q-Learning [36], where an agent was trained to play 7 different Atari games using as input only the raw Atari frames. In particular, Deep Q-Learning is mainly based on Q-Learning [51], and the main contributions are the introduction of DNN for approximating the action-value function and the introduction of the Experience Replay Memory for training the DNN.

First, the update rule based on the Bellman equation used as an iterative update is defined as:

$$Q_{i+1}(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma \max_a Q_i(s_{t+1}, a)|s_t, a_t] \,. \tag{2.26}$$

Subsequently, the action-value function is approximated with neural networks described by the weights $\theta$, $Q(s, a; \theta)$ and called Q-network. The Q-network is trained by minimizing a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration $i$,

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{s,a\sim\rho(\cdot)} \left[(y_i - Q\left(s, a; \theta_i\right))^2\right] \,, \tag{2.27}$$

Table 2.1: List of commonly used activation functions

| Name | Function |
|---|---|
| Logistic (Sigmoid) | $f = \dfrac{1}{1+e^{-x}}$ |
| Hyperbolic tangent (Tanh) | $f = \dfrac{e^{2x}-1}{e^{2x}+1}$ |
| Rectified Linear Unit (ReLU) | $f = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Leaky ReLU | $f = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Softmax | $f = \dfrac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \text{ for } i = 1, \ldots, K$ |

where $y_i = \mathbb{E}[r + \gamma \max_a Q(s_{t+1}, a; \theta_{i-1})|s_t, a_t]$ is the target for the $i$-th iteration and $\rho(s, a)$ is a probability distribution over sequences $s$ and actions $a$ that we refer to as the *behaviour distribution* (for example an $\epsilon$-greedy strategy). Differentiating the loss function with respect to the weights, we arrive at the following gradient:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathop{\mathbb{E}}_{s,a\sim\rho(\cdot)} \left[ \left( r + \gamma \max_a Q(s_{t+1}, a; \theta_{i-1}) - Q(s_t, a_t; \theta_i) \right) \nabla_{\theta_i} Q(s_t, a_t; \theta_i) \right] . \tag{2.28}$$

Rather than computing the full expectations in the above gradient, it is often computationally convenient to optimize the loss function by stochastic gradient descent.

In contrast to approaches in literature like TD-Gammon [54] that are defined online since the update is performed only on the last transition, the authors used a technique called *experience replay* [55]. In particular, the agent experience at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$, in a dataset $\mathcal{D}$ with size $N$, $\mathcal{D} = (e_0, \ldots, e_N)$, also called *replay memory*. Subsequently, at each training step, a mini-batch of experience is sampled from the replay memory and used to update the Q-function as explained in (2.28). This approach has several advantages over standard online Q-learning. First is possible to sample the same experience multiple times and update the weights multiple times, allowing for greater sample efficiency. Second, learning directly from consecutive samples is inefficient due to the strong correlations between

the samples; randomizing the samples breaks these correlations and, therefore, reduces the updates' variance. Finally, when learning on-policy, the current parameters determine the next data sample that the parameters are trained on; therefore it can easily fall in local minima. Using experience replay memory, the behavior distribution is averaged over many of its previous states, smoothing out the learning and avoiding oscillations or divergence in the parameters.

In Algorithm 2.4 the complete pseudocode is shown.

**Algorithm 2.4** Deep Q-Learning with Experience Replay

Initialize Replay Memory with size $\mathcal{D}$
Initialize $Q$ with random weights $\theta$
**for** episode $= 1, \ldots, M$
  Initialize $s_1$
  **for** $t = 1, \ldots, T$
    With probability $\epsilon$ select $a_t$ random action otherwise $a_t = \max_a Q(s_t, a; \theta)$
    Take action $a_t$ and observe $s_{t+1}$ and $r_{t+1}$
    Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$
    Sample random mini-batch of transition from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{if } s_j \text{ is terminal} \\ r_j + \gamma \max_a Q(s_j, a; \theta) & \text{if } s_j \text{ is terminal} \end{cases}$
    Perform gradient descent step on $(y_i - Q(s_j))^2$ as shown in (2.28)
  **end for**
**end for**

After the introduction of Deep Q-Learning, various extensions have been proposed to increase performances. In the following sections, only the extension used in this paper will be analyzed.

### 2.5.1 Double Deep Q Networks

Similarly to Q-learning, DQN is as well affected by the maximization bias detailed in Section 2.3.4; indeed, the same maximization operation is performed. Therefore, in Double Deep Q-Networks [56] a similar approach to Double Q-learning is applied. In particular, as in Double Q-learning, two estimators of the Q-function are used that are represented by two networks, the online network, described by the weights $\theta$, to evaluate the greedy policy and target network, described by the weights $\theta^-$, to estimate its value. In Double Q-learning, the two Q functions were updated evenly and alternately used to evaluate the greedy policy and estimate the value function. Instead, in Double DQN, the two networks have fixed roles, the online network evaluates the greedy policy, and the target network estimates the value function. The update rule remains the same as DQN, with the only difference of introducing

the target network in the computation of the target $Y_t$

$$Y_t \equiv r_{t+1} + Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t), \theta_t^-) \,. \tag{2.29}$$

Finally, the target network weights are periodically updated by copying the Q-network weights.

### 2.5.2 Prioritized Experience Replay

In DQN, the experience is sampled uniformly from the replay memory; however, it is intuitive that some experience is more relevant than others, and therefore it should be sampled more frequently. The Prioritized Experience Replay (PER) [57] introduces a technique to increase the probability of sampling relevant transitions stored in the memory. In particular, the transitions are prioritized based on the TD-error $\delta_i$, and the prioritization is defined as,

$$p_i = |\delta_i| + \epsilon \,, \tag{2.30}$$

where $\epsilon$ is a small positive constant to avoid that if the TD-error of a transition is equal to zero, it will not be sampled again. The probability of sampling the $i$-th transition is computed as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \,, \tag{2.31}$$

where $\alpha$ is the prioritizing factor, if $\alpha = 0$ the sampling is uniform. However, the prioritized replay introduces bias because it changes the sampling distribution, and therefore changes the solution that the estimates will converge. Therefore, importance sampling weights $w_i$ are introduced to correct this bias.

$$w_i = \left( \frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta \tag{2.32}$$

The bias is fully compensated if $\beta = 1$; however, it was noticed that in the first stages of the training is beneficial to have a bias. Therefore, typically $\beta$ is linearly annealed from its starting value to 1.

### 2.5.3 Dueling Networks

Dueling networks were introduced in [58] to optimize the architecture of the NN used for estimating the action-value function by explicitly separating the representation of state values and action advantages. Indeed, it was noticed that in some scenarios where actions do not particularly affect the environment, it is more relevant to know which states are more valuable rather than knowing the effect of an action on each state.

Dueling networks, as shown in Figure 2.6, consist in a NN that at the end splits into two streams, one for the estimation of the value function $V(s)$ and the other for the estimation of

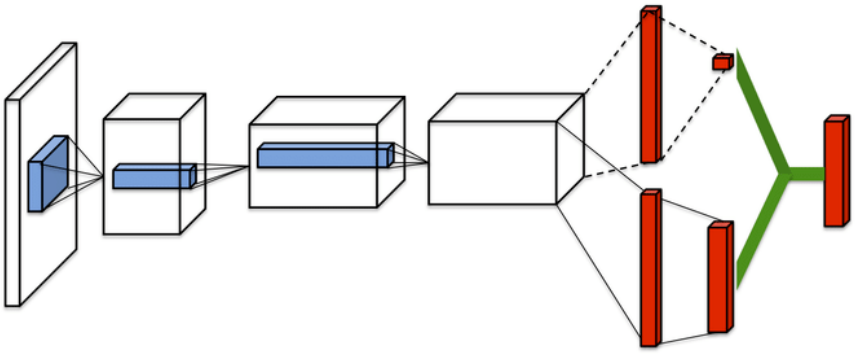


Figure 2.6: Dueling architecture

advantage function $A(s, a)$ which are used to estimate the action-value $Q(s, a)$. We define the weights of the common layers as $\theta$, the weights of the layers of the advantage function $\alpha$, and the weights of the layers of the value function $\beta$. Starting from the definition of the advantage function $A$:

$$A(s, a) = Q(s, a) - V(s)\,, \tag{2.33}$$

theoretically we can compute $Q(s, a)$ as:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)\,. \tag{2.34}$$

However, it is empirically proven to have poor performances. Indeed, given a unique value of $Q$, any combination of $V$ and $A$ whose sum is equal to $Q$ is valid. Therefore, it is chosen to impose the stream of the advantage function to have the average of the output for each action equal to zero and $Q(s, a)$ can be estimated as

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left( A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha) \right) \tag{2.35}$$

It is important to note that Equation 2.35 is part of the network representing the Q-function therefore it can be trained as a standard Q-network. As the dueling architecture shares the same input-output interface with standard Q networks, we can recycle all learning algorithms with Q networks (e.g., DDQN and SARSA) to train the dueling architecture.

## 2.6 Reinforcement Learning with Continuous Actions and the Policy Gradient Theorem

The algorithms described in the previous sections are value-based since the policy is defined implicitly based on the action-value function through the max operator applied to the value

function. Finding the maximum of the value function with discrete action states is straightforward, it requires evaluating the function for each possible action and picking the optimal action. Finding the maximum of the value function with discrete action states is straightforward. It requires evaluating the value function for each possible action and picking the optimal. This approach is no longer viable in continuous actions domains since an optimization algorithm would be required to find the optimal action, which would be highly time-consuming. Therefore, in those scenarios, the policy is defined as a differentiable function with weights $\theta$. The weights are updated based on the gradient of a performance measure $J$.

$$\theta_{t+1} = \theta + \alpha \nabla_\theta J(\theta) \tag{2.36}$$

However, multiple performance measures are available to evaluate the policy, such as:

- the start value in episodic environment, $J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$ ,
- the average value in continuing environment, $J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$ ,
- the average reward per timestep, $J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a|s) \mathcal{R}_s^a$ ,

where $d^{\pi_\theta}(s)$ is the stationary distribution of the Markov chain under the policy $\pi_\theta$.

The Policy Gradient Theorem [59] gives an analytical solution for any performance measure.

**Theorem 1 (Policy Gradient Theorem)** *For any differentiable policy $\pi_\theta(a|s)$, for any policy objective function $J = J_1, J_{avV}, J_{avR}$ the policy gradient is*

$$\nabla_\theta J(\theta) \propto \sum_s d^{\pi_\theta}(s) \sum_a q_\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

## 2.7 Deterministic Policy Gradient and its extensions

The policy gradient theorem analytically defines the gradient in the case of stochastic policies. In [60] it is proven that the policy gradient can also be obtained for any differentiable deterministic policy under the same assumptions of the stochastic policy gradient. In particular, it is also proven that the deterministic policy gradient is a particular case of the stochastic policy gradient when the variance $\sigma \to 0$. Therefore, we define the performance measure the expected discounted reward,

$$J(\mu_\theta) = \mathop{\mathbb{E}}_{s \sim \rho^\mu}[r(s, \mu_\theta)] \,, \tag{2.37}$$

where $\rho^\mu(s)$ is the discounted state distribution and $\mu_\theta$ is the deterministic policy parametrized as with the parameter vector $\theta$. Then the Deterministic Policy Gradient Theorem states:

**Theorem 2 (Deterministic Policy Gradient Theorem)** *if* $\nabla_\theta \mu_\theta(s)$ *and* $\nabla_a Q^\mu(s,a)$ *exist then deterministic policy gradient exist and is:*

$$\nabla_\theta J(\mu_\theta) = \mathop{\mathbb{E}}_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a_{\mu_\theta(s)}}]$$

Grounding on the Deterministic Policy Gradient Theorem, the off-policy actor-critic algorithm called Deterministic Policy Gradient (DPG) was developed. In particular, the critic, the action-value function $Q$, defined by the parameters $\theta^Q$ is learned via Q-learning and used to update the policy based on the Deterministic Policy Gradient Theorem. Thus, we can rewrite the action-value function in the case of deterministic policies as

$$Q^\mu(s_t, a_t) = \mathbb{E}\left[r_t(s_t, a_t) + \gamma\left[Q^\mu(s_{t+1}, \mu(s_{t+1}))\right]\right], \tag{2.38}$$

and the loss function to minimize is:

$$\begin{aligned} L(\theta^Q) &= \mathbb{E}\left[\left(Q^\mu(s_t, a_t|\theta^Q) - y_t\right)^2\right] \\ y_t &= r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}|\theta^Q)) \end{aligned}. \tag{2.39}$$

### 2.7.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient [61] (DDPG), similarly to DQN, was the first successful implementation of nonlinear function approximation, via NN, applied to DPG.

Similar to the case of Q-learning and DQN, it is not possible to directly use NNs as function approximators because the learning process tends to be unstable. Thus, various modifications inspired by the work of DQN are implemented, in particular *replay buffer* and *target network*. Replay buffer is used to store the transitions $(s_t, a_t, r_t, s_{t+1})$ and it used to sample mini-batches and perform stochastic gradient descent on Equation 2.39, and they become:

$$L = \frac{1}{N}\sum_i \left(Q(s_i, a_i|\theta^Q) - y_i\right)^2 \tag{2.40}$$

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t} \tag{2.41}$$

Target networks are introduced to increase stability; indeed, one of the main problems is that the network updated $Q(s, a|\theta^Q)$ is also used to compute the target value. Thus similarly to Double DQN, a copy of the actor and critic networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ respectively, are used for calculating the target values. The weights of these target networks

are then updated by having them slowly track the learned networks.

$$\begin{aligned} &\theta' \leftarrow \tau\theta + (1-\tau)\theta' \\ &\mu' \leftarrow \tau\mu + (1-\tau)\mu' \\ &\tau \ll 1 \end{aligned} \tag{2.42}$$

In conclusion, the pseudocode of DDPG is shown in Algorithm 2.5.

**Algorithm 2.5** Deep Deterministic Policy Gradient

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q$ and $\mu$ with weights $\theta^Q \leftarrow \theta^Q$, $\theta \leftarrow \theta^\mu$
Initialize replay buffer R
**for** episode = $1, \ldots, M$
Initialize a random process N for action exploration
Receive initial observation state $s_1$
**for** t = $1, \ldots, T$
Select action $a_t = \mu(s_t|\theta) + \mathcal{N}_t$ according to the current policy and exploration noise
Execute action at and observe reward rt and observe new state $s_{t+1}$
Store transition $(s_t, a_t, r_t, s_{t+1})$ in R
Sample a random minibatch of N transitions $(s_i, a_i, r_i, s_{i+1})$ from R
Set $y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1}|\theta)|\theta^Q)$
Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i Q(s_i, a_i|\theta^{Q'}))^2$
Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta_Q)|_{s=s_i, a=\theta(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\begin{aligned} &\theta' \leftarrow \tau\theta + (1-\tau)\theta' \\ &\mu' \leftarrow \tau\mu + (1-\tau)\mu' \end{aligned}$$

**end for**
**end for**

### 2.7.2 Twin Delayed Deep Deterministic policy gradient (TD3)

Actor-critic algorithms are affected by two criticalities:

- overestimation bias of the action value function;

- noisy policy updates caused by high variance in the estimation.

In Q-learning, the overestimation bias is caused by the maximization operator as detailed in Section 2.3. Instead, in actor-critic algorithms where the policy update is computed via the policy gradient theorem or the deterministic policy gradient, it is caused by the gradient; indeed, the gradient direction is a local maximizer. Noisy policy updates are caused by the fact that when function approximators are used in the Bellman Equation, estimations are never exactly, and each update leaves some amount of residual TD-error $\delta(s, a)$. Those, residuals if not addressed, tend to accumulate and create high variance (noise) in the policy updates.

In [62] a set of techniques to avoid or limit them are proposed. In order to solve the overestimation bias, Clipped Double Q-Learning for actor-critic is proposed. Two target networks with independent weights $\theta_1$ and $\theta_2$ are learned in Clipped Double Q-Learning. However, when computing the target $y$, only the target networks minimize the overestimation of the value function, consisting of taking the minimum between the estimates.

$$y_t = r_t + \gamma \min_{i=1,2} Q_{\theta_i}(s_{t+1}, \pi(s_{t+1})) \tag{2.43}$$

In order to solve the noisy policy updates, two solutions are proposed: delayed policy updates and target policy smoothing regularization. Delayed policy updates perform an update of the policy every $d$ updates of the target networks; thus, it is possible to train more the target networks before using their estimates to update the policy. Instead, the target policy smoothing regularization introduces a regularization term in order to prevent the target networks from overfitting. The regularization term is based on the intuition that given a state $s_t$, the expected reward values of similar actions should be similar as well. Thus, random noise is introduced at the action level during the target updates:

$$\begin{aligned} y &= r_t + \gamma Q_{\theta'}(s_{t+1}, \pi(s_{t+1}) + \epsilon)\,, \\ \epsilon &\sim \mathrm{clip}(\mathcal{N}(0, \sigma), -c, c)\,. \end{aligned} \tag{2.44}$$

All the aforementioned improvements are applied combined to DDPG in a new algorithm called Twin Delayed Deep Deterministic policy gradient (TD3) shown in Algorithm 2.6.

**Algorithm 2.6** TD3

Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$ with random parameters $\theta_1$, $\theta_2$, $\phi$
Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\mu' \leftarrow \mu$
Initialize replay buffer $\mathcal{B}$
**for** t = 1 to T
Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$ and observe reward $r_t$ and new state $s_{t+1}$
Store transition tuple $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{B}$
Sample mini-batch of N transitions $(s_t, a_t, r_t, s_{t+1})$ from $\mathcal{B}$
$\tilde{a} \leftarrow \pi_{\phi'}(s_{t+1}) + \epsilon \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), c, c)$
$y \leftarrow r_t + \gamma \min_{i=1,2} Q_{\theta_i}(s_{t+1}, \tilde{a})$
Update critics $\theta_i \leftarrow \text{argmin}_i \frac{1}{N} \sum_i (y - Q_{\theta_i}(s, a))^2$
**if** $t \bmod d$
Update $\phi$ by the deterministic policy gradient:

$$\nabla_\phi J(\phi) = \frac{1}{N} \sum_i \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$$

Update target networks:

$$\theta_i' \leftarrow \tau\theta_i + (1 - \tau)\theta_i'$$
$$\phi_i' \leftarrow \tau\phi_i + (1 - \tau)\phi_i'$$

**end if**
**end for**

*To succeed, planning alone is insufficient. One must improvise as well.*

Foundation, Isaac Asimov

# 3

# Task Planning in Non-Deterministic Environments solved via Reinforcement Learning

This Chapter proposes a general formulation to describe and solve task planning problems in non-deterministic environments via Reinforcement Learning. Specifically, the Task Planning problem is formulated as a Markov Decision Process to introduce non-determinism and enable to deal with dynamic environment effectively. The studied method is applied to a robotic object sorting scenario to prove the efficacy of the proposed method.

## 3.1 Introduction

In order to effectively cooperate with humans, robots must be able to perform tasks based on the current status of the environment, including the person, and react to unexpected events. The unexpected events can vary from changes in the desired goal to friendly/adversarial human interaction. For example, the human can either cooperate to complete the task or perform a conflicting task.

Thus, in this Chapter, we study the problem of Task Planning applied to non-deterministic and dynamic environments. Those scenarios raise a significant challenge to classic task planning algorithms. Typically, Task Planning is solved assuming the dynamic of the scenario is entirely predictable; thus, it is possible to create a tree describing the evolution of the environment for each action performed and apply graph search algorithms to compute the optimal plan. However, in non-deterministic environments, it is not possible to define a unique tree to compute the optimal plan. Thus, as detailed in Section 1.2.4, different works formulate the task planning problem as an MDP and solve it with Reinforcement Learning.

Similarly, we propose a formulation of Task Planning, and then we translate it into an MDP. Subsequently, we apply the proposed formulation to a case study depicting an object sorting, and we solve it via Deep Reinforcement Learning. In particular, we highlight that the proposed formulation is general and can be applied to different task planning applications. It is scalable since, differently from standard task planning, the running time to compute the optimal action is neglectable, even increasing the dimension of the problem. Finally, it can effectively deal with non-deterministic environments characterized by probabilistic dynamics and unexpected events.

## 3.2 Formulation

Let's consider a generic Task Planning environment $\mathcal{E}$ populated by $N_{obj}$ objects, we can describe $k$-th object by its state $s_k$ that is the minimum set of attributes $\boldsymbol{\lambda}_k$ describing the object properties.

$$s_k = \boldsymbol{\lambda}_k = \langle \lambda_{k,i} \rangle \quad \text{with } i \in \{1, \dots, N_{attr,k}\} \text{ and } k \in \{1, \dots, N_{obj}\} \tag{3.1}$$

The attribute $\lambda_{k,i}$ is a singular value describing one and only one property of the $k$-th object, and it is defined over the domain $D_{\lambda_{k,i}}$. For each $k$-th object we can define the domain of the state $S_k$, representing all possible object's states $s_k \in S_k$, as $D_k$ where:

$$D_{S_k} = D_{\lambda_{k,1}} \times \cdots \times D_{\lambda_{k,N_{attr,k}}} \tag{3.2}$$

We define that two objects $o$ and $o'$ are identical if and only if the sets of attributes $\boldsymbol{\lambda}_o$ and $\boldsymbol{\lambda}_{o'}$ are identical.

$$o = o' \iff \boldsymbol{\lambda}_o = \boldsymbol{\lambda}_{o'} \tag{3.3}$$

We define that two objects belong to the same class $\mathcal{C}$ if they have the same state domain $D_{S_o} = D_{S_{o'}}$, i.e., they share the same set of properties. Finally, we define the state of the environment, $s$, as the union of the state objects belonging to the environment $\mathcal{E}$.

$$s = \{s_k\} \quad \text{with } k \in \{1, \dots, N_{obj}\} \tag{3.4}$$

Analogously, it is possible to define the set of environment states $S$ whose domain is defined as:

$$D_S = D_{S_1} \times \cdots \times D_{S_{N_obj}} \quad \text{with } k \in \{1, \dots, N_{obj}\} \tag{3.5}$$

In the environment, a set of action $A$ can be performed that have the effect of modifying the environment state, described by the state transition function $\Gamma : S \times A \to S$. In other words, an action $a \in A$ has the effect of modifying one or more object attributes. In non-deterministic environments, the state transition function is a probabilistic, $\Gamma = \mathbb{P}(s_{t+1}|s_t, a)$, where $s_t$ and $s_{t+1}$ two generic successive environment states.

The objective is to find a sequence of actions $a$, such as it is possible to reach an environment goal state $s_{goal}$ from a start state $s_{start}$ while minimizing a cost function $f$. In deterministic environments, the sequence of actions can be described as a tuple called the plan $\mathcal{P} = \langle a_1, \dots, a_n \rangle$. In non-deterministic environments, the plan $\mathcal{P}$ cannot be computed, thus it is defined the policy $\pi : S \to A$ that maps from states to actions such as if iteratively applied allows to reach the goal state $s_{goal}$ given any state transition described by $\Gamma$. The policy $\pi$ can be computed by solving a MDP defined by the tuple $(S, A, R, P)$ where state $S$ and the action $A$ are identical to the previous ones, the transition probabilistic function $P$ is equivalent to the probabilistic version of $\Gamma$. Finally, the reward function $R : S \times A \times S \to \mathbb{R}$ is the counterpart of the cost function $f$. However, $f$ is defined over all the plan $\mathcal{P}$, while $R$ is defined over only a single state-action-state transition. In and MDP the objective is to maximize the cumulative discounted reward that can be interpreted as the negative cost function.

## 3.3 Object Sorting Environment

We apply the aforementioned formalization to a scenario of object sorting where object must be sorted based on a qualitative property and ordered based on a quantitative property. The objects are randomly placed in a region of space called "clutter" and should be sorted in regions called "clusters" based on the qualitative property.

Each object has 3 attributes: color, size and position; therefore the object state is defined as:

$$s_k = \langle \text{color}, \text{size}, \text{position} \rangle \,. \tag{3.6}$$

The color is the qualitative property, the size is the quantitative property and finally the position describe where the object is placed in the clutter or in the clusters. Thus, we define the

attributes domains:

$$D_{colors} = \{\text{colors}\} \tag{3.7}$$

$$D_{sizes} = \{\text{sizes}\} \tag{3.8}$$

$$D_{positions} = D_{clutter} \cup \left( \bigcup_{c=1}^{|D_{colors}|-1} D_{cluster_c} \right) \tag{3.9}$$

The positions' domain is defined by the union of the clutter domain, i.e., all positions in the clutter space, and the cluster domains, i.e., all the positions in the clusters, one for every color. We assume that the number of objects to sort is variable but limited to $K$, $N_{obj} \leq K$. Thus, the number of clutter and cluster positions is related to the maximum number of objects. The clutter and each cluster must be able to hold all the objects in the environment, therefore

$$|D_{clutter}| = |D_{cluster_c}| = K\,. \tag{3.10}$$

We assume that all clutter and clusters' free positions are occupied by a *null* object that have the peculiarity of having a *null* color and a *null* size. Thus including the *null*-objects, the total amount of objects is $K \cdot |D_{colors}|$. The domain $D_{sizes}$ include all the possible sizes plus the null size, similarly, $D_{colors}$ includes all the possible colors plus the null-color. As defined in Section 3.2 the environment state is:

$$s_{env} = \{s_k\} \quad \text{with } k \in \{1, \ldots, K \cdot |D_{colors}|\} \tag{3.11}$$

Thanks to the introduction of the *null*-object the dimension of the environment state $s_{env}$ is independent from the number of objects in the environment as long as they are less than or equal to $K$.

We assume that given a set of objects it can be required to sort only a subset of them, thus we introduce the goal state $s_{goal}$ that describes which objects should be sorted. The goal state represent only the set of objects to be sorted and similarly to the environment state we desire that its domain is independent from the current number of objects in the environment or to be sorted. Thus, the goal state is described the target state of each cluster, similarly to the environment state, free positions are considered occupied by the *null*-object.

$$s_{goal} = \{s^{goal}_{c,k}\} \quad \text{with } c \in \{1, \ldots, |D_{colors}| - 1\} \text{ and } k \in \{1, \ldots, K\} \tag{3.12}$$

In conclusion, the state $s$ of the Task Planning problem is derived as the tuple composed by the environment state and the goal state.

$$s = \langle s_{env}, s_{goal} \rangle \tag{3.13}$$

The proposed definition of the state $s$ has the advantage of having the same domain $D$ in-

dependently from any number of objects, shapes, or colors as long as they are less or equal to the predefined maximum number of objects, colors, and shapes. Therefore, the policy function $\pi$ to solve any task planning problem, within the domain, can be approximated by one set of parameters $\theta$. In other words, it is possible to define one single policy function to solve all task planning problems in the domain.

We define the set of actions $A$ as the set of pick and place actions to move and object from its current position to another position.

$$a = \text{move}(o_k, \text{pick position}, \text{place position}) \,. \tag{3.14}$$

Assuming that one position can be occupied by only one object at time, we omit to specify the object and identify it only based on its current position.

$$a = \text{move}(\text{pick position}, \text{place position}) \,. \tag{3.15}$$

Furthermore, we assume that the objects in the cluster are placed in a tower-like method, i.e., objects are always placed in the lowest empty position, only objects from the top occupied position can be picked, and no empty position there may be between two objects of the same cluster. Finally, we assume also that objects picked from the clutter can be placed only in one of the clusters and objects picked from clusters can be placed in other clusters or in the clutter. Thus, we define the set pick positions as:

$$\text{pick positions} = D_{clutter} \cup \{\text{clusters}\} \,, \tag{3.16}$$

meanwhile, the place positions are defined based on the pick position.

$$\begin{cases} \text{place positions} = \{\text{clusters}\} & \text{if pick} \in D_{clutter} \\ \text{place positions} = D_{clutter} & \text{if pick} \in \{cluster\} \end{cases} \tag{3.17}$$

Based on the action formulation, we define the state transition functions $\Gamma$. As described earlier, the action is a pick and place and therefore the only attribute that is modified is the position of an object. Finally, the scenario is non-deterministic and, in particular, picking an object from some region of environments introduces a failure probability.

We compute the policy $\pi$ by solving the equivalent MDP with same state, action and state transition function, and we introduce the reward function $R$.

For each performed state-action-state transition, a reward is given to the agent, according to the scalar function $R$ defined by the following equation:

$$R = r_{move} + r_{empty} + r_{stuck} + r_{fail} + r_{back} + r_{dist} + r_{succ} \tag{3.18}$$

where

- $r_{move}$ is the cost of a *move* action;
- $r_{empty}$ is the cost of an empty pick: the policy decides to pick from an empty cell;
- $r_{stuck}$ is the cost of picking an object from a cell and placing it back to the same cell;
- $r_{fail}$ is the cost of a failure in terms of either localization or picking: the detection is inaccurate, or the wrong object is moved, or the gripper fails to grasp the object;
- $r_{back}$ is the cost of moving an object from the top cell of one cluster to a cell of the clutter: when the agent takes this action, even for a temporary relocation, it should be penalized;
- $r_{dist}$ is the reward describing the distance between the attributes in the environment state and the goal state. In particular, is defined as:

$$\sum_{c=1}^{|D_{clusters}|-1} \sum_{k=1}^{K} \alpha \cdot d_{c,k} \tag{3.19}$$

  where $alpha$ is a vector of coefficients describing the relative importance of each attribute and $d_{c,k}$ is the vector of attributes distances between two elements of the clusters one from the environment state and the from the goal state.

- $r_{succ}$ is the reward assigned when the desired goal is achieved;

Except for $r_{succ}$, all the reward values are strictly negative.

## 3.4 Experimental evaluation

### 3.4.1 Environment

A Python environment has been implemented* depicting the formulation of Section 3.2. Of such environment, two scenarios have been considered. The former, $\epsilon_0$, is characterized by a maximum of $K = 10$ objects, the color domain $D_{color} = \{r, b\}$, and the size domain $D_{size} = \{s, m, l\}$. Similarly, the latter, $\epsilon_1$, has $K = 20$ objects, the color domain $D_{color} = \{r, g, b\}$, and the size domain $D_{size} = \{s, m, l\}$.

A Robot Operating System (ROS)-based [63] setup has been realized to validate both the environment and the policies. It asks both a simulated and a real agent to group objects of both $\epsilon_0$ and $\epsilon_1$ according to the *color*, and order each cluster depending on the size. The agent is a UR10 manipulator robot equipped with a magnet on its end-effector. A Microsoft Kinect v2 perceives the workspace, detects the objects, and recognize them. Gazebo is used

*Implementation: https://github.com/iaslab-unipd/iaslab-sorting-env

as simulator. The workspace is non-deterministic: a human operator can randomly add new objects, change the status of the environment, and update the goal. The robotic agent should accomplish the assigned task while dealing with these external disturbances. Moreover, it should correctly and promptly react to the failures it may encounter. As stated in Section **??**, the environment is discretized into cells and a different failure probability is assigned to each of them. With the main goal of testing the system, a higher failure probability is associated to the cells closer to the robot base and farthest from the localization system. Indeed, in those areas object detection and robot motion planning are expected to be less reliable.

### 3.4.2 Training

The policy was trained using the DQN [36] implementation of the DRL Stable-Baselines [64] library, with the following extensions: Double-Q learning [65], Duelling-DQN [66] and Prioritized Experience Replay [67]. The implemented exploration strategy is an epsilon-greedy one with linear decay over a window of length $N_t$ training steps.

Aiming at increasing the training speed, a custom-made curriculum learning strategy was implemented. It feeds the policy with a percentage of partially solved training episodes. The curriculum learning strategy depends on two parameters:

- the percentage of partially solved training episodes $e_s$, computed with respect to the total number of episodes provided to the policy;
- the percentage at which the goal of the partially solved episodes $g_s$ is already achieved.

Both parameters were linearly decreased along the window. The considered environment is characterized by tasks having a very high variability in complexity.

The complexity of an episode depends on the combination of two episode-specific parameters: the number of objects on the environment $K_E$, and the amount of them that should be ordered $K_G$, i.e., the number of objects that are included in the desired target clusters configuration (with $K_G \leq K_E$). This high variability in inter-episode complexity could be really problematic for an effective training of the policy if all the possible combination of those two parameters have equal distribution during the training. Indeed, the policy risks overfitting the easiest tasks (i.e., low $K_E$ and low $K_G$) thus being ineffective in solving the hardest ones (i.e., high $K_E$ and high $K_G$). To minimize the risk of overfitting, both $K_E$ and $K_G$ were sampled from 2 triangular distributions. The former having mode equal to the environment maximum number of objects $K$, while the latter having mode equals to $K_E$, the total objects available in the episode.

Two policies have been trained to solve $\epsilon_0$ and $\epsilon_1$, respectively. The two training exploited the same hyperparameters, except for the minimum number of training steps $N_t$: in $\epsilon_0$, $N_t = 2 \times 10^6$; in $\epsilon_1$, $N_t = 8 \times 10^6$. Table 3.1 reports the values of all adopted hyperparameters.

Figure 3.1 and Figure 3.2 present the results obtained from the test of the two policies trained to solve $\epsilon_0$ and $\epsilon_1$, respectively. To build those graphs, at each $50k$ training steps, the

Table 3.1: DQN hyperparameters used to train the policies

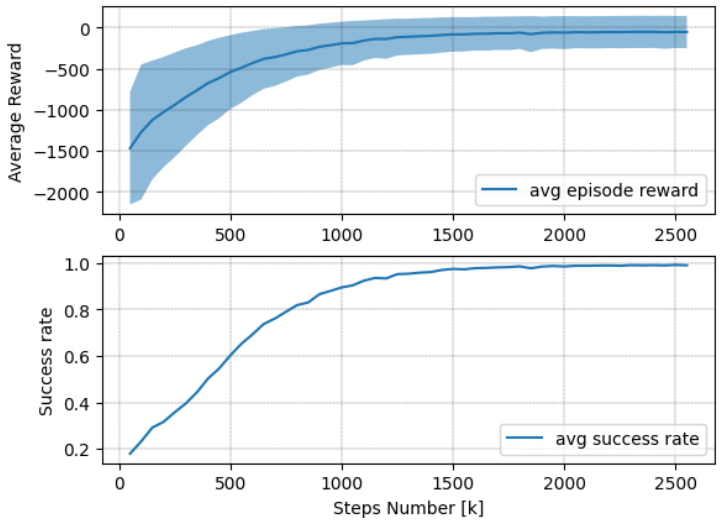
| hyper-parameters | |
|---|---|
| Discount factor | 0.92 |
| Learning rate | $10^{-4}$ |
| Batch size | 256 |
| Update target network freq. | 500 |
| Memory size | 100000 |
| Prioritized exp. replay $\alpha$ | 0.6 |
| Prioritized exp. replay $\beta$ | 0.4 |
| Network | $2 \times 128$ ReLu |
| Start exploration | 1.0 |
| Final exploration | 0.05 |

training has been stopped, the model stored and tested over $T$ ($T = 500$ for $\epsilon_0$ and $T = 100$ for $\epsilon_1$) episodes with respect to every combination of $K_E$ and $K_G$, and the results registered. Both for $\epsilon_0$ and $\epsilon_1$, Figure 3.2a and Figure 3.2b report the average validation results of the best model; Figure 3.1a and Figure 3.1b, instead, reports the average episode reward (plus/minus its standard deviation) and the average success rate curves. Finally, Fig 3.2a and Figure 3.2b show more in detail the results of the testing phase obtained for the best trained policy, once again for both $\epsilon_0$ and $\epsilon_1$. It can be noticed that the combinations with higher $K_E$ and $K_G$ obtained lower success rates, despite the higher number of training episodes visited in training, coherently with their expected higher complexity.

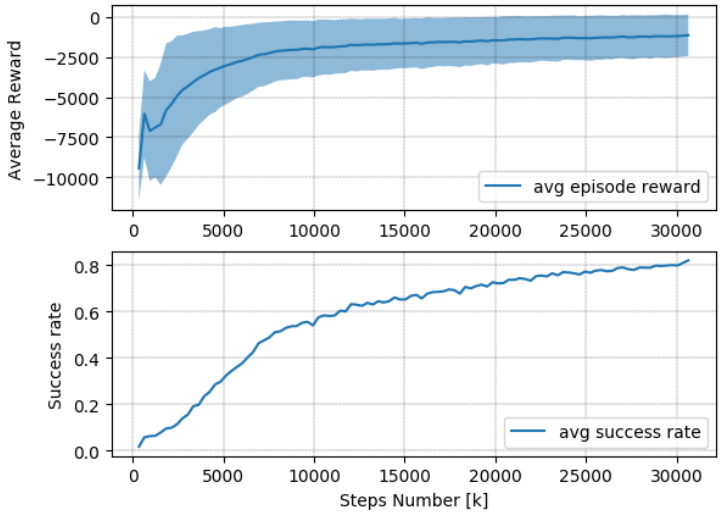
### 3.4.3 Tests and Results

Six tests have been designed, each one aiming at highlighting a specific contribution of the proposal.

- **Test 1**. $K_E = K_G = 5$. The agent is asked to sort all the available objects. For each object $o_k$, the pair $(\gamma_m, \lambda_n)$ is randomly chosen from those available (see Section 3.4.1);
- **Test 2**. $K_E = K_G = 10$. The assignment is the same as in **Test 1** but with twice the objects to be sorted;
- **Test 3**. $K_E = 10$, $K_G = 5$. The agent is asked to sort only 5 of the 10 available objects. Objects' properties are randomly chosen. Couples of objects with the same properties exist: one object is located on a cell with high failure probability, while the other stands on a low failure probability one;
- **Test 4**. $K_E = K_G = 5$. The assignment equals that of **Test 1** but when the goal is reached, a human operator adds an object, bigger than the one of the top of its target




(a)




(b)

Figure 3.1: Average episode reward and success rate obtained for $\epsilon_0$ **(a)** and $\epsilon_1$ **(b)**, respectively.

stack, and asks the policy to sort it. Thus, to reach the new goal, the policy should temporally relocate already sorted objects;

- **Test 5**. $K_E = 8$, $K_G = 5$. Objects' properties are randomly chosen. First, the user interferes by adding a new object without updating the goal: such object becomes an alternative. Once the goal is reached, the user modifies the goal, asking the robot to recheck already sorted objects;

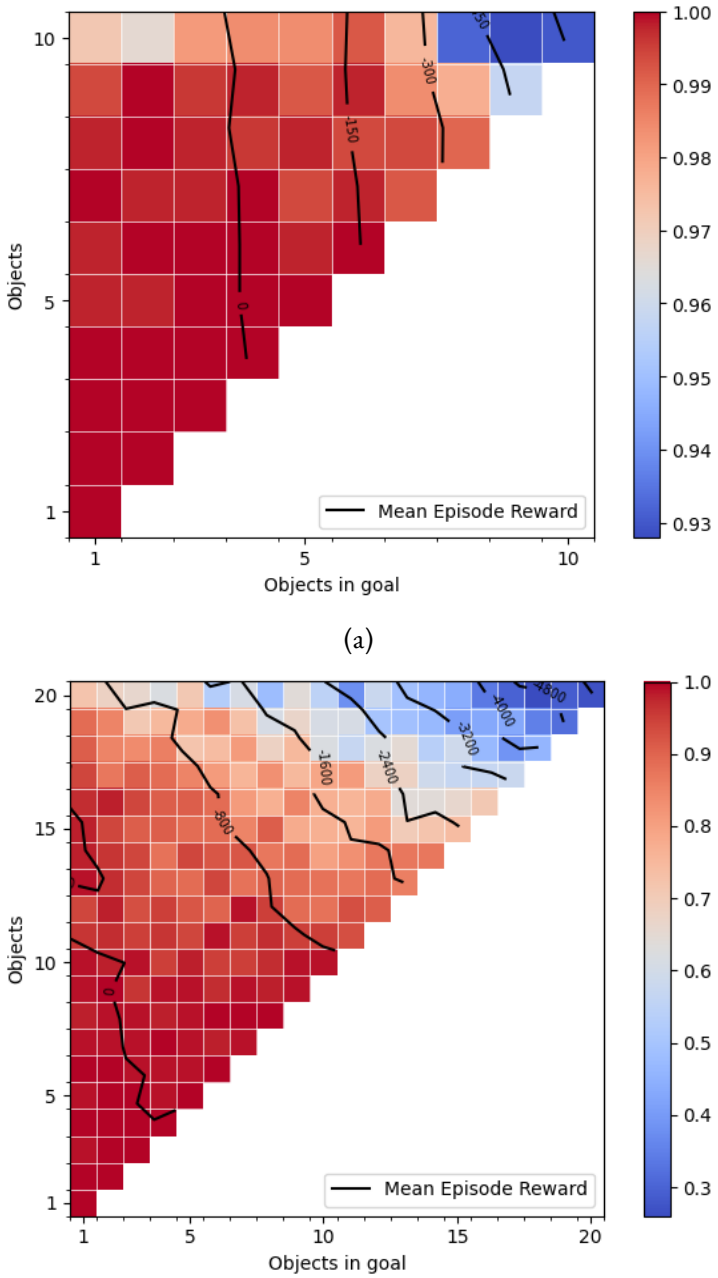



Figure 3.2: Average success and average reward for each possible combination of objects in the environment, $K_E$, and objects in the goal configuration, $K_G$. Results obtained for $\epsilon_0$ **(a)** and $\epsilon_1$ **(b)**, respectively.

- **Test 6**. $K_E = 8$, $K_G = 3$. Objects properties are randomly chosen, but the goal is composed of objects of the same category, i.e., the same color. Once reached, the target is updated including 3 other objects of another color.

As reported in Section 3.4.2, both $\epsilon_0$ and $\epsilon_1$ obtained similar results. Thus, only results of $\epsilon_0$ follow, subdivided according to the contribution they highlight.

- **Robustness to adversities**. Both in **Test 4** and **Test 5**, an external actor adds new objects to the scene. The policy correctly faces such adversities by relocating objects each time a goal update is requested. In all the cases, the minimum number of moves is employed to complete the assignment. Furthermore, the trained policy proves to successfully cope with and recover from failures occurred in both object grasping and localization.

- **Risk minimization**. While performing **Test 3** the policy chooses, whenever possible, to take those objects located on the lower failure probability cells. Such behavior shows that the policy trained on the formulated environment learns to minimize the failure risk, regardless of the criteria followed by the user.

- **Scalability**. **Test 1** and **Test 2** ask to solve the same task, but the latter requires the manipulation of a growing number of objects. The capability of the policy to correctly accomplish both assignments with the minimum number of moves proves the ability of the system to scale with the increasing dimensionality of the problem.

- **Generality**. The proper resolution of **Test 6** demonstrates that the system can generalize to increasingly complex tasks with new classification categories and ordering priorities: no re-training is required and no computational overhead results.

Figure 3.3 shows the simulated and real execution of **Test 4**. In both cases, a new object is added to the scene and the goal configuration is update. The agent adapts to the new setup and successfully completes the new assignment. Moreover, in the real world, the robot fails to grasp one object but it promptly recovers from it. Attached videos[†], instead, show all six tests performed in the real world.

## 3.5 Conclusions and future work

In this Chapter, a formulation for task planning in non-deterministic environments based on Markov Decision Process was proposed. Subsequently, the formulation is applied to a scenario of object sorting characterized by a non-deterministic environment due to probabilistic effects of the agent actions and the interaction with a human performing friendly or adversarial actions.

[†]Video: https://doi.org/10.5281/zenodo.3961813

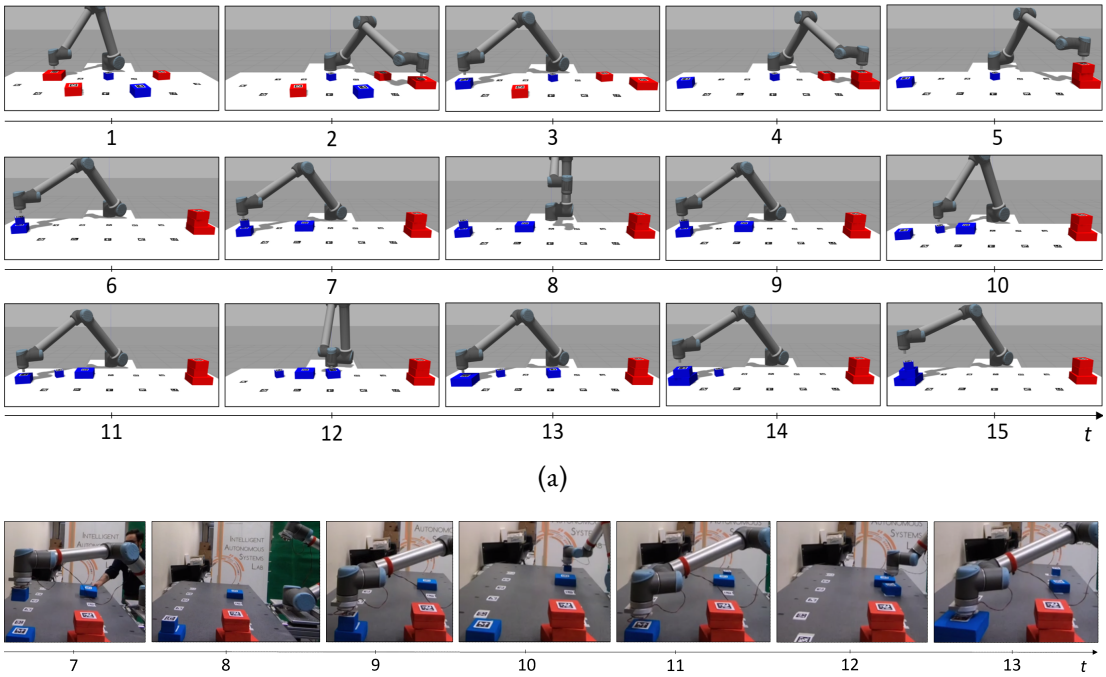



Figure 3.3: Video frames of **Test 4**: (**a**) Simulated solution sequence and (**b**) real experimental resolution. For brevity, (**b**) only shows frames for $7 \leq t \leq 13$. At time $t = 7$, a human operator adds a new large blue block. Once detected, the policy temporally relocates already sorted objects to complete the task.

Experiments performed both in simulation and in the real world required the robotic agent to sort a maximum of 20 objects, subdivided according to at most three colors, and ordered based on a maximum of three discrete sizes. During the execution of the assigned task, a human user interfered with the environment by adding objects and/or updating the desired goal. Obtained results demonstrate that the system learned to adapt to unpredictable external interventions, to recover from failures, and to correctly chose actions, whenever possible, that minimize the failure risk. Finally, the proposed system scales on the numbers of objects and assigned goals.

As future work, to demonstrate the generality of the proposed formulation, other sorting problems will be tested together with other DRL algorithms. Such new experiments will allow us to extensively assess also the scalability in terms of number of properties. Finally, partially-observable problems will be extensively investigated to push even forward the generality of the formulation.

*In preparing for battle I have always found that plans are useless, but planning is indispensable.*

Dwight D. Eisenhower

# 4

# Deep Reinforcement Learning applied to Motion Planning in Human Robot Shared Workspaces

In this chapter, a method to solve the problem of motion planning in human robot shared workspaces as a reinforcement learning problem is proposed. A brief analysis of the state art is the presented highlighting the limitations and showing the contribution of this chapter. The proposed method models a feedback motion planner as an MDP subsequently trained via DRL and a novel method to model the person during the training phase. The person is modelled via a pseudo-random occupancy model that is able to achieve subject independence and human task independence. Finally, the proposed method is evaluated both in simulation and on an experimental setup, and it is combined with the current industrial safety standard to highlight the increase of performance compared with current industry best practices.

## 4.1 Introduction

Human-robot collaboration in manufacturing is a valuable solution to increase the flexibility of the production tasks; however, the generation of safe robot trajectories granting high productivity is still a challenge. As we have detailed in Section 1.2.3, many approaches are used in literature to solve motion planning in human-robot shared workspaces; none of them seems to be sufficiently general to solve the problem completely. Motion planners that are based on predicting the occupied workspace by the human have various drawbacks. First, they require an extensive process to develop a stochastic model of the human behavior that can include the probability of performing a task [68, 45] or occupying a specific region of space [34], or both [22]. However, the human behavior stochastic model is subject-dependent and task-dependent. A model describing the probability of occupying a specific region of space is subject-dependent because it depends on the kinematic model of the person, i.e., it depends on the body shape of the person (height, arm length, etc.). Analogously, a model describing the probability of a task is subject-dependent because when multiple equally feasible tasks can be performed simultaneously, human personal preferences arise. Finally, they are task-dependent because those models require to define either the set of tasks the person can perform or the complete sequence of tasks that compose the manufacturing process; therefore, they hardly deal with unexpected behavior from the human. For example, in [34] the possible set of movements the human can perform is known a priori and used to perform online classification of the human movements; however, in a realistic scenario, it is unlikely to know all the movements and all the possible methods a task can be performed. Motion planners who develop velocity scaling strategies instead usually lack the flexibility required to be effective in environments with frequent close interaction between humans and robots, such as a collaborative human-robot assembly workstation. However, these planners usually can guarantee the person's safety since they abide by the current safety standard like ISO TS 15066. Finally, reactive motion planners can easily fall into local minima or perform excessively long trajectories (freezing robot problem) when deployed in cluttered environments or in the presence of many moving obstacles (e.g., humans).

As highlighted in Section 1.2.3, recent works [38, 39, 40] investigate the use of DRL for motion planning, primarily for mobile robot navigation in a dynamic environment. In these works, the environment in which the mobile robot moves is considered to be populated by decision-making agents (the humans) that try their best to avoid collisions with the robot by adopting social rules or the policy that is being trained in a self play fashion, or collision avoidance protocols specifically designed, such as ORCA [69]. However, those approaches to model the human can hardly be applied to human-robot shared workspace in an industrial scenario. First, the implemented collision avoidance protocols and the social rules are designed for social navigation, or mobile robotics cannot be used as industrial robotic manipulators. Second, it is impossible to apply the policy to the human that is currently trained because the robot and the human have very different kinematic models (e.g., different degrees of freedom). Furthermore, while it is commonly accepted in social navigation to adopt

a discrete action space model, this choice would not be appropriate for robotic manipulators, especially for industrial manipulators. A low level of accuracy and vibrations caused by discontinuities in control commands would negatively affect the resulting motion.

In this chapter is to develop a human-aware motion planner for a scenario of human-robot co-existence that does not suffer from the limits of the methods mentioned above while including their strength. In detail, it aims to develop a motion planner that is subject-independent, task-independent, and ensures safety. Therefore, it is investigated the usage of a feedback motion planner based on DRL for human-robot co-existence (i.e., workspace sharing) combined with a pseudo-random human occupancy model and combined with the current industrial safety standard SSM.

Specifically, the proposed approach models the feedback motion-planning problem as a Markov Decision Process and applies DRL to learn a policy capable of approximating the optimal feedback motion planner defined by the reward function. The human-task independence and subject-independence are given by training the DRL through a pseudo-random occupancy volume that models the human occupancy in the work area. Such occupancy model does not represent a specific set of human tasks (i.e., set of predefined patterns), that are characterized by high levels of heteroscedasticity [70], thus, challenging to correctly model. In contrast, the proposed pseudo-random occupancy model can address the problem of describing a large set of possible human tasks, including the high variance in their execution by people, in a compact way. In other words, the DRL is trained using a dataset of possible occupancy volumes that have been designed to map the possible interaction modalities statistically. Similarly, it is human-subject independent because the occupancy model can also model the space occupied by people with different body sizes. Finally, the occupancy model considers the human not as a decision-making agent, i.e., he/she acts independently from the robot, and he/she does not try to avoid collisions, and it makes the proposed method robust to avoid hazardous situations for the operators. In other words, the DRL learns how to avoid the operator also when he/she displays hazardous behaviors, e.g., moving towards the robot when it is moving. The pseudo-random occupancy volume models the human as a variable radius cylinder that moves across the workspace. The human direction is chosen randomly. Meanwhile, the speed and the radius are defined by parametric functions whose parameters are sampled at each episode.

It is also introduced a novel formulation MDP action to enhance smooth robot trajectories. A common approach in literature is to model actions as joint speed references for the robot controller; however, there is no guarantee that the controller will be able to reach the reference value within a single timestep of the MDP. Instead, we propose to define the action as a parametric continuous trajectory of the duration of a single timestep. It is also proven that it is possible to guarantee the kinodynamic feasibility of the parametric trajectory, imposing the joint limits.

A technique to boost the policy training, based on the discretization of the goal space during the training phase, is also proposed. The goal space is discretized as a 3D uniform

grid and the policy is trained only on the element of the 3D grid. After that, the policy is interpolated over the discretized goal space in order to reach any goal in the continuous space.

The trained policy is straightforwardly applied to the experimental scenario (i.e., without fine-tuning it), showing the efficacy and robustness of the proposed approach. Finally, it is shown that the proposed method is compatible with current safety standards and that it outperforms current industry best practices by increasing the efficiency of the cooperation between humans and robots. In particular, Speed and Separation Monitoring (SSM) from the ISO TS 15066 was implemented on top of the proposed feedback motion planner and of a standard motion planner.

## 4.2 Problem Formulation

### 4.2.1 Context

Consider a simulated environment where the agent (i.e., a robotic manipulator) needs to reach a set of targets placed in its workspace while avoiding collisions with fixed obstacles and human agents. Human agents are assumed to be performing an action independently from the robot motion, therefore the considered problems fit in the class of moving obstacles avoidance. Within this class of problems there is no collaboration between the human and the robot, therefore the robot simply needs to ensure that it will not enter into the volume occupied by the human, regardless from the action he/she is performing. This assumption freed us to from the need to model the human task, allowing us to simply model his/her volume occupancy as a vertical variable-radius cylinder moving in the shared workspace (i.e., the minimum radius cylinder fully containing the human shape, augmented with a proper safety factor). Furthermore, to endorse the independence of the robot task from the human task, both the motion laws and the changes in the cylinder radius are defined as pseudo-random.

## 4.3 Methodology

The proposed methodology consists in developing a simulated environment to train via DRL an agent (i.e., the robot) to perform a reaching task while avoiding collisions with a human. The adopted simplified human model consists in a pseudo random volume occupancy model, in particular the volume occupied by the human is modeled as a cylinder, whose radius and speed are defined by pseudo-random functions as shown in (4.2) and (4.1). The actions of the MDP are modeled as parametric trajectories whose coefficient are bounded in order to guarantee kinodynamic feasibilty. The agent will be trained to reach targets defined by sampling the desired continuous target region. If the user-target does not belong to the discrete training set, it is then computed by a first-order interpolate of the policy computed on the closest training targets.

### 4.3.1 Human Occupancy Model

Let's consider an industrial scenario where a robot has to perform a pick&place task while human co-workers share the same workspace performing parallel unknown tasks not synchronized with the robot. The main difficulty of such scenario is that the human movements can be hardly predictable, so it is extremely complex to define a model closely representing the human behavior in the specific scenario. Furthermore, the absence of knowledge about the task the human is performing makes it almost impossible to acquire a dataset of human movements to develop a stochastic model of the behavior.

Thus, a synthetic dataset of the human behavior is developed by sampling deterministic trajectories from pseudo-random functions. Such choice allows training the agent without the risk of biasing the policy due to the incompleteness or unbalanceness of an acquired dataset. On the one hand, this solves the problem of incompleteness because the number of possible sampled trajectories is extremely high within the range of trajectories described by the pseudo-random functions. On the other hand, it solves the problem of unbalanceness since the coefficients can be sampled either uniformly or with any desired distributions, therefore is possible to balance the dataset as desired. It is worth to notice that the pseudo random functions used to create the synthetic dataset have a very simple formulation compared to the approaches used in literature, considerably reducing the required effort for developing a new training environment. Using a synthetic dataset of human trajectories takes inspiration from various works in literature in the field of reinforcement learning where agents that share cooperative or adversarial objectives with humans are not trained with data based on humans — e.g., competitive games are trained against themselves in a so-called self-play fashion like in AlphaGo Zero [4] and OpenAI Five [71]. Specifically, self-trained AlphaGo Zero achieves better results against human players than the predecessor Alpha Go that was previously trained with a dataset of real matches. Looking at works related to navigation/motion planning, in [40, 39] "non-human" movement were used in training. In both cases, the policy was successfully transferred to real world scenarios. In particular, in [39] a combination of handcrafted strategies and the policy were used to control the human agents, while in [40] the human agents were controlled with ORCA (hand-crafted protocol for collision avoidance of mobile robots).

The human is modelled as a cylindrical variable radius collision volume moving in the horizontal plane with a variable speed. Both the velocity and the radius are defined by 2 pseudo-random functions approximating all the possible velocity and radius values combinations the human can reach in the proposed scenario.

The module of the velocity is defined as follows:

$$\begin{aligned} v(t) =& \frac{v_{max} + v_{min}}{2} + 0.5 v_r \sin(2\pi f_r^v t + \phi_{r_1}^v) + \\ & 0.3 v_r \sin(6\pi f_r^v t + \phi_{r_2}^v) + 0.2 v_r \sin(8\pi f_r^v t + \phi_{r_3}^v) \,. \end{aligned} \tag{4.1}$$

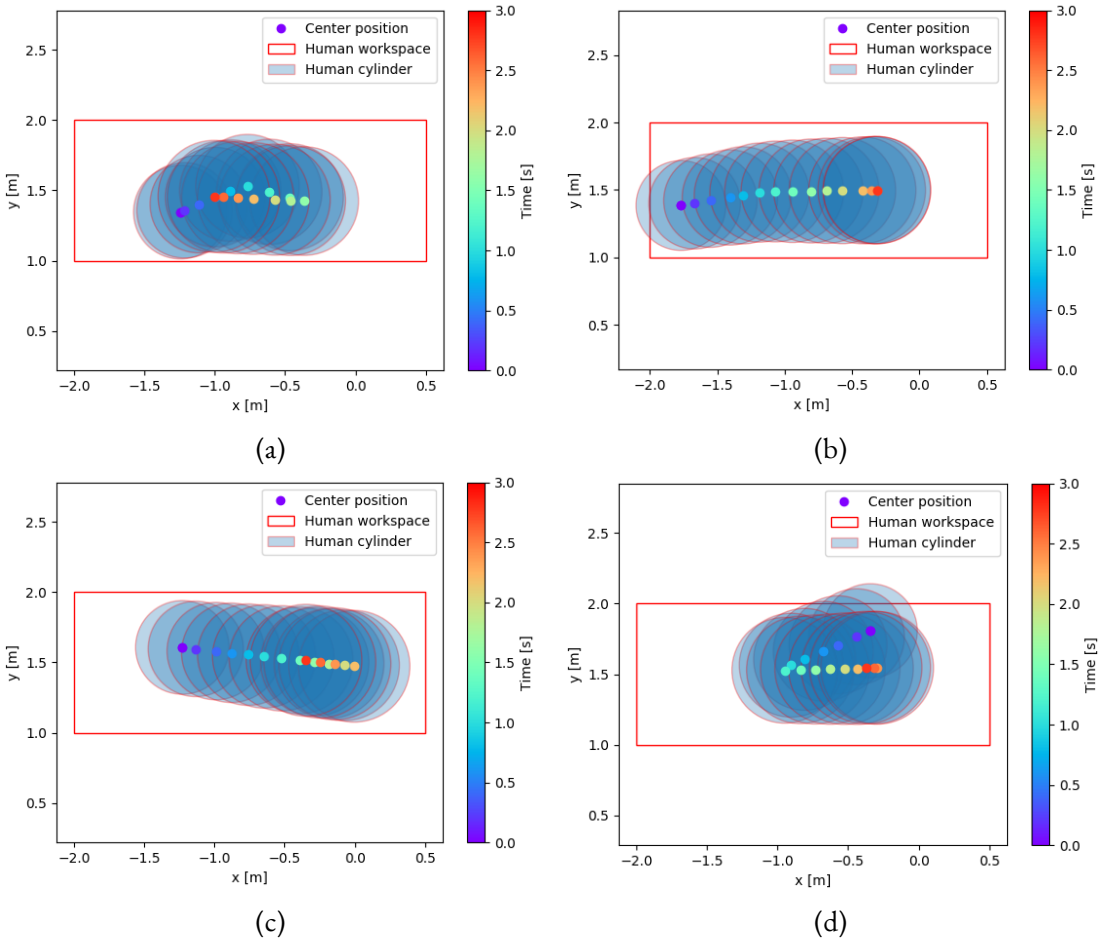



Figure 4.1: Examples of trajectories from the human occupancy model. The blue shaded region represents the workspace occupied during the trajectory, and the colored dots the position of the center of the cylinder. The color of the dot represents the time of the trajectory.

In particular, $v_{min}$ and $v_{max}$ are respectively the minimum and the maximum walking speed, $v_r$ is a random value within the walking speed range, $f_r^v$ is a random frequency within a range and $\phi_{r_1}^v, \phi_{r_2}^v, \phi_{r_3}^v$ are random phases. If the velocity value is below the minimum velocity it is set to zero and if it is above $v_{max}$ is set to $v_{max}$ . The human at the beginning of each episode starts from a random position and walks along a straight line toward a random target position; it has a small probability to stand still for the entire episode. When the human is within a distance tolerance from the target position, a new random target position is chosen, however it has a small probability to stop on the last position for the rest of the episode.

The cylinder radius function is defined, similarly to the velocity function, as:

$$\begin{aligned} r(t) =& \frac{r_{max} + r_{min}}{2} + 0.5 r_r \sin(2\pi f_r^r t + \phi_{r_1}^r) + \\ & 0.3 r_r \sin(6\pi f_r^r t + \phi_{r_2}^r) + 0.2 r_r \sin(8\pi f_r^r t + \phi_{r_3}^r) \, . \end{aligned} \tag{4.2}$$

In particular, $r_{min}$ and $r_{max}$ are the minimum and maximum values among with the radius is clipped, $r_r$ is a random value in the radius range, $f_r^r$ is a random frequency within a range and $\phi_{r_1}^r, \phi_{r_2}^r, \phi_{r_3}^r$ are random phases. The pseudo-random functions used to model the human velocity and radius have the property to be continuous and to be composed by high and low frequency fluctuations in order to mimic the variability of the human movement. Figure 4.1 shows four examples of trajectories and the occupied workspace by the human occupancy model applied to the experiment in Section 4.4.1.

The human state is defined as the Cartesian position of the human $(x, y)$, the speed $(v_x, v_y)$ and its radius $r$.

$$s_h = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ r \end{bmatrix} \tag{4.3}$$

The proposed human model complies with the Markov property required to model the environment as an MDP, therefore it assumes that the cylinder speed and radius (i.e., the human speed and space occupied) do not depend on the history of the state i.e., they depend neither on the history of robot behavior nor on the history of the human behavior. Indeed, we can write the evolution of the next human state from its current human state only as:

$$\mathbf{s}_h' = \begin{bmatrix} x_h \\ y_h \\ \dot{x}_h \\ \dot{y}_h \\ r_h \end{bmatrix} = \begin{bmatrix} x(t) + v_x(t)\Delta t \\ y(t) + v_y(t)\Delta t \\ v_x(t+1) \\ v_y(t+1) \\ r(t+1) \end{bmatrix} \, . \tag{4.4}$$

As it can be noticed, the proposed human model makes the policy training independent from the human operator and from the task he is performing. Indeed, the velocity function is suitable to generate an infinite number of different velocity profiles, all compatibles with human movements. Moreover, the radius function is suitable to represent humans with different shape and dimensions representing possible different tasks (e.g., walking, carrying object etc.) or just unpredictable actions. Thus, the proposed human occupancy model allows to obtain a policy effective not only on a small of set predetermined situations but generally effective, in any human-robot shared workspace application.

### 4.3.2 Action Formulation

Common state-of-the-art approaches define the action $\mathbf{a}_t$ that a robot performs to reach a target either as an instantaneous joint speed command or a change in joint speed (i.e., an average joint acceleration). The former does not consider the initial position and velocity of the agent: the robot controller ensures their continuity over time. The latter guarantees continuity with the initial conditions but does not include the joint position and velocity limits. Eventually, based on the reward function formulation, the agent may learn not to exceed them.

Differently from the literature, given the trajectory $\tau$ that the robot has to follow to reach a target, we propose to model $\mathbf{a}_t$ as the parametric sub-trajectory $\tau_t \in \tau$ to be performed during the $t$-th time step, with $t \in [0, T]$ and $\Delta x$ the duration of the time step itself. Thus, $\tau$ is defined as the following concatenation of sub-trajectories $\tau_t$:

$$\tau = \{\tau_0, \tau_1, \dots, \tau_T\} = \{\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_T\}. \tag{4.5}$$

If the agent has $n$ Degrees of Freedom (DOFs), each sub-trajectory $\tau_t$ is a vector of $n$ sub-trajectories, one per joint. Thus,

$$\mathbf{a}_t = \begin{bmatrix} a_t^0 \\ \vdots \\ a_t^n \end{bmatrix} \tag{4.6}$$

To ensure each joint sub-trajectory is continuous and observes the joint limits, we model it as a polynomial function with the form

$$a_t^n = \sum_{i=0}^{N} c_{n,i}^t x^i \quad N \geq 2, \quad 0 \leq x \leq \Delta x. \tag{4.7}$$

where $N$ is the degree of the polynomial function, and $c_{n,i}^t$ is the $i$-th coefficient of the polynomial function, defined at time step $t$ for the $n$-th DOF. To guarantee position and velocity continuity, the minimum degree of the polynomial function is equal to 2. It is the minimum requirement to create feasible trajectories. Nevertheless, it is possible to set higher-order joint position time derivatives.

Polynomial functions provide multiple advantages:

- We can easily decide the grade continuity and smoothness of the final trajectory by imposing the degree of the polynomial function.
- We can control not only the acceleration but also higher time derivatives of the position, such as jerk and snap.
- It is easy to compute the maximum and minimum joint position (and its derivatives) to check compliance with the limits.

As a result,

$$\mathbf{a}_t = \begin{bmatrix} a_t^0 \\ \vdots \\ a_t^n \end{bmatrix} = \begin{bmatrix} c_{0,N}^t x^N + c_{0,N-1}^t x^{N-1} + \cdots + c_{0,1}^t x + c_{0,0}^t \\ \vdots \\ c_{n,N}^t x^N + c_{n,N-1}^t x^{N-1} + \cdots + c_{n,1}^t x + c_{n,0}^t \end{bmatrix}. \tag{4.8}$$

Imposing the starting conditions of each sub-trajectory (e.g., initial position and velocity), we can write:

$$\mathbf{a}_t = \begin{bmatrix} c_{0,N}^t x^N + c_{0,N-1}^t x^{N-1} + \cdots + c_{0,2}^t x^2 + \dot{q}_0^t x + q_0^t \\ \vdots \\ c_{n,N}^t x^N + + c_{n,N-1}^t x^{N-1} + \cdots + c_{n,2}^t x^2 + \dot{q}_n^t x + q_n^t \end{bmatrix}. \tag{4.9}$$

The action vector $\mathbf{a}_t$ becomes the vector of coefficients $\mathbf{c}_t$:

$$\mathbf{c}_t = \left[ c_{0,N}^t, c_{0,N-1}^t, \ldots, c_{n,N}^t, c_{n,N-1}^t, \ldots \right]. \tag{4.10}$$

Without loss of generality, we can drop the subscript $t$:

$$\mathbf{c} = \left[ c_{0,N}, c_{0,N-1}, \ldots, c_{n,N}, c_{n,N-1}, \ldots \right]. \tag{4.11}$$

We still cannot guarantee the compliance with the joint limits. To overcome this limitation, we define a set of inequalities for each DOF $q_n$. Taking into consideration a generic DOF $q$, the following system results:

$$\begin{cases} q_{max} \geq c_N x^N + c_{N-1} x^{N-1} + \cdots + c_2 x^2 + \dot{q} x + q \geq q_{min} \\ \dot{q}_{max} \geq N c_N x^{N-1} + (N-1) c_{N-1} x^{N-2} + \cdots + 2 c_2 x + \dot{q} \geq \dot{q}_{min} \\ \ddot{q}_{max} \geq N(N-1) c_N x^{N-2} + N(N-1)(N-2) c_{N-1} x^{N-3} + \cdots + 2 c_2 \geq \ddot{q}_{min} \\ \vdots \end{cases} \tag{4.12}$$

The goal is finding the range of values of $\mathbf{c}$ for which (4.12) is satisfied for the duration $x$ of the sub-trajectory, with $0 \leq x \leq \Delta x$. For this computation to be unambiguous, the polynomial function must be at most one degree higher than that of the boundary conditions (e.g., if the initial conditions are position, velocity, and acceleration, the degree of the starting condition is 2, and that of the polynomial function should be at most equal to 3). To verify that the joint limits are verified, it is necessary to compute the maximum and minimum values parametric over $c$. Computing the maximum and minimum of a function requires computing the roots of the derivative, and in our case they must be computed analytically since are parametric with respect to $c$. Therefore, to ensure their analytic computation, the degree of the polynomial function must be at most equal to 5 meaning a derivative of order 4 (the Abel-Ruffini Theorem states that there is no solution in radicals to general polynomial

equations of degree five or higher with arbitrary coefficients.).

For simplicity, we solve the system for a second degree polynomial function with position, velocity, and acceleration as the starting conditions. The same approach can be applied to more complex scenarios. In this context, $c_2$ becomes the only unknown coefficient, and we can refer to it as $c$. Therefore, (4.12) evolves into

$$\begin{cases} q_{max} \geq cx^2 + \dot{q}x + q \geq q_{min} \\ \dot{q}_{max} \geq 2cx + \dot{q} \geq \dot{q}_{min} \\ \ddot{q}_{max} \geq 2c \geq \ddot{q}_{min} \end{cases} \quad \text{with } 0 \leq x \leq \Delta x\,. \tag{4.13}$$

Starting from

$$q_{max} \geq cx^2 + \dot{q}x + q \geq q_{min} \quad \text{with} \quad c \neq 0\,, \tag{4.14}$$

we verify the maximum and minimum values by analytically computing the roots of the derivative parameterized with respect to $c$ as

$$x = -\frac{\dot{q}}{2c}\,. \tag{4.15}$$

We replace this value in (4.14):

$$q_{max} \geq c\frac{q^2}{4c^2} - \frac{\dot{q}^2}{2c} + q \geq q_{min} \quad \text{if} \quad 0 \leq -\frac{\dot{q}}{2c} \leq \Delta x\,, \tag{4.16}$$

and obtain the following result:

$$\frac{q^2 - 2\dot{q}^2}{4(q_{max} - q)} \leq c \leq \frac{q^2 - 2\dot{q}^2}{4(q_{min} - q)} \quad \text{if} \quad 0 \leq -\frac{\dot{q}}{2c} \leq \Delta x\,. \tag{4.17}$$

The condition $0 \leq -\frac{\dot{q}}{2c} \leq \Delta x$ lets check the value is within the joint limits only if it maximum/minimum falls within the duration of a time step $\Delta x$. According to the values of $\dot{q}$, such a condition becomes:

$$\begin{cases} -\frac{\dot{q}}{2c} \leq \Delta x \cup c < 0 & \text{if } \dot{q} \geq 0 \\ -\frac{\dot{q}}{2c} \leq \Delta x \cup c > 0 & \text{if } \dot{q} < 0 \end{cases} \tag{4.18}$$

To ensure that the value at the end of the time step is still within the joint limits, we also impose

$$q_{max} \geq c\Delta x^2 + \dot{q}\Delta x + q \geq q_{min}\,. \tag{4.19}$$

The result follows:

$$\frac{q_{max} - \dot{q}\Delta x - q}{\Delta x^2} \geq c \geq \frac{q_{mim} - \dot{q}\Delta x - q}{\Delta x^2}\,. \tag{4.20}$$

For the second and third inequalities of (4.13) instead the derivative has no roots and therefore it is only necessary to verify that at $x = \Delta x$ the joint limits are not exceeded. In conclusion, we can write the following system of inequalities.

$$\begin{cases} c \leq \frac{q^2 - 2\dot{q}^2}{4(q_{min} - q)} & \begin{cases} -\frac{\dot{q}}{2c} \leq \Delta x \cup c < 0 & \text{if } \dot{q} \geq 0 \\ -\frac{\dot{q}}{2c} \leq \Delta x \cup c > 0 & \text{if } \dot{q} < 0 \end{cases} \\ c \geq \frac{q^2 - 2\dot{q}^2}{4(q_{max} - q)} & \begin{cases} -\frac{\dot{q}}{2c} \leq \Delta x \cup c < 0 & \text{if } \dot{q} \geq 0 \\ -\frac{\dot{q}}{2c} \leq \Delta x \cup c > 0 & \text{if } \dot{q} < 0 \end{cases} \\ c \leq \frac{q_{max} - \dot{q}\Delta x - q}{\Delta x^2} \\ c \geq \frac{q_{mim} - \dot{q}\Delta x - q}{\Delta x^2} \\ \frac{\dot{q}_{max} - \dot{q}}{2\Delta x} \geq c \geq \frac{\dot{q}_{min} - \dot{q}}{2\Delta x} \\ \frac{\ddot{q}_{max}}{2} \geq c \geq \frac{\ddot{q}_{min}}{2} \end{cases} \tag{4.21}$$

As a result, the acceptable values of $c$ will be in the range

$$\begin{aligned} c_{max} = \max & \left( \frac{q^2 - 2\dot{q}^2}{4(q_{min} - q)}, \frac{q_{max} - \dot{q}\Delta x - q}{\Delta x^2}, \frac{\dot{q}_{max} - \dot{q}}{2\Delta x}, \frac{\ddot{q}_{max}}{2} \right) \geq c \geq \\ & \min \left( \frac{q^2 - 2\dot{q}^2}{4(q_{max} - q)}, \frac{q_{min} - \dot{q}\Delta x - q}{\Delta x^2}, \frac{\dot{q}_{min} - \dot{q}}{2\Delta x}, \frac{\ddot{q}_{min}}{2} \right) = c_{min} \end{aligned} . \tag{4.22}$$

Finally, the policy outputs an action $a \in [-1, 1]$ and $c$ is computed as:

$$c = \frac{c_{max} + c_{min}}{2} + a\frac{c_{max} - c_{min}}{2} \tag{4.23}$$

### 4.3.3 State and Reward

We define the state vector $\mathbf{s}$ of the proposed MDP as the concatenation of the agent state $\mathbf{s}_a$, the state of each i-th human $\mathbf{s}_{h_i}$ and the agent's target position $\mathbf{s}_t$.

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}_a \\ \mathbf{s}_{h_0} \\ \dots \\ \mathbf{s}_{h_n} \\ \mathbf{s}_t \end{bmatrix} \tag{4.24}$$

The reward function $r$ takes into account 3 objectives:

- reaching the target position;
- avoiding collisions;

- minimizing the required actions;

The reward function is therefore the sum of a distance reward $r_d$, a collision reward $r_c$ and an action reward $r_a$ denoted as:

$$r_d = -c_d ||\mathbf{x}_{\text{agent}} - \mathbf{x}_{\text{target}}||_2 \,, \tag{4.25}$$

$$r_c = \begin{cases} -c_c & \text{if collision} \\ 0 & \text{if no collision} \end{cases} , \tag{4.26}$$

$$r_a = -c_a ||\mathbf{c}||_2 \,. \tag{4.27}$$

In particular, $\mathbf{x}_{\text{agent}}$ and $\mathbf{x}_{\text{target}}$ are the agent and target Cartesian positions; $c_d, c_c, c_a$ are positive constants and $\mathbf{a}$ is the agent action. The training episode terminates successfully if the Cartesian distance between target and agent is below a threshold value $d_t$ and the agent is not in collision. In this case, a positive success reward $r_s$ is added.

$$r = \begin{cases} r_d + r_c + r_a + r_s & \text{if success} \\ r_d + r_c + r_a & \text{if not success} \end{cases} \tag{4.28}$$

### 4.3.4 Policy Interpolation

Given a set of targets defined over a continuous space (e.g., reaching any position within a certain region of space), a possible solution is to train the policy over goals sampled from a distribution defined over that space. However, this solution has the drawback of requiring a lot of samples in order to train sufficiently everywhere in the space. A more sample efficient solution, is instead to train over a limited number of sampled goals and to interpolate the policy on these sampled goals. In this paper a linear interpolation on the sampled training goals arranged as an evenly spaced 3D grid is implemented, as shown in Figure 4.2. The black dots represent the training targets, meanwhile the red dot is the desired goal position $p_g$. The first step to compute the policy value $\pi_g$ is to find the smallest cube, whose vertexes are the training targets, that include the desired goal position (the cube with red dotted edges). Then the policy value in $p_g$ (namely $\pi_g$) is computed as a linear interpolation on an evenly spaced grid, i.e., trilinear interpolation, described in (4.29):

$$\begin{aligned} \pi_g = &\pi_{000}\left(1-\frac{p_x}{l}\right)\left(1-\frac{p_y}{l}\right)\left(1-\frac{p_z}{l}\right) + \pi_{100}p_x\left(1-\frac{p_y}{l}\right)\left(1-\frac{p_z}{l}\right) + \\ &\pi_{010}\left(1-\frac{p_x}{l}\right)p_y\left(1-\frac{p_z}{l}\right) + \pi_{001}\left(1-\frac{p_x}{l}\right)\left(1-\frac{p_y}{l}\right)p_z + \pi_{101}p_x\left(1-\frac{p_y}{l}\right)p_z + \\ &\pi_{011}\left(1-\frac{p_x}{l}\right)p_y p_z + \pi_{110}p_x p_y\left(1-\frac{p_z}{l}\right) + \pi_{111}p_x p_y p_z \,. \end{aligned} \tag{4.29}$$

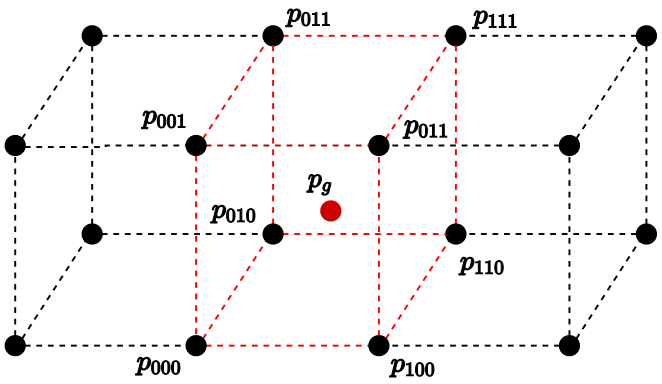



Figure 4.2: 3D grid of training targets (black dots) with a target goal (red dot), the red dotted square is the smallest cube, with as vertices the training target, that include the desired goal position $p_g$.

In detail, $\pi_{ijk}$ is the values of the policy in the trained target $p_{ijk}$, i.e., setting target position in $\mathbf{s}_t$ equal to $p_{ijk}$; $(p_x, p_y, p_z)$ is the distance vector between the desired goal position $p_g$ and the training target $p_{000}$ and $l$ is the cube side length.

## 4.4 Experiments and Results

### 4.4.1 The scenario

The proposed method was applied to the following real-world application (Figure 4.8a). An anthropomorphic manipulator (UR10) must reach, from a home position, a target (the sphere) randomly placed inside a box-shaped region of space having dimensions $1.0 \times 0.4 \times 0.4\,m$ (i.e., goal space) while one person (the cylinder) walks in and out of the robot workspace.

A simulated environment matching the described application was developed (Figure 4.3b). Following the formulation proposed in Section 4.3.1 a Human Occupancy model was included, the state vector $\mathbf{s}$ composed as in (4.30) and the reward function defined as (4.28). In particular, the agent state $s_a$ is defined as the robot joints position and joints speed, the human state $s_h$ is defined as in (4.3) and the target state $s_t$ is defined as the target Cartesian coordinates.

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}_a \\ \mathbf{s}_h \\ \mathbf{s_t} \end{bmatrix} = \begin{bmatrix} \text{robot joint positions} \\ \text{robot joint speeds} \\ \text{human position } (x, y) \\ \text{human speed } (\dot{x}, \dot{y}) \\ \text{human cylinder radius} \\ \text{target position } (x, y, z) \end{bmatrix} \tag{4.30}$$

We chose the action as a second order polynomial function for each joint, thus we can write,

based on Section 4.3.2:

$$\mathbf{a} = \begin{bmatrix} c_1 x^2 + \dot{q}_0 x + q_0 \\ \vdots \\ c_5 x^2 + \dot{q}_5 x + q_5 \end{bmatrix} \tag{4.31}$$

$$c_i = \frac{c_{max,i} + c_{min,i}}{2} + a\frac{c_{max,i} - c_{min,i}}{2} \quad a \in [-1, 1] \tag{4.32}$$

The other relevant parameters of the environment are detailed in Tab 4.1.

Table 4.1: List of all environment parameters

| Environment parameters | |
|---|---|
| action type | joint speed command |
| command frequency | $50\ Hz$ |
| distance threshold $d_t$ | $0.1\ m$ |
| max. joint speed | $\pm 1\ rad/s$ |
| min. robot human speed $v_{min}$ | $0.3\ m/s$ |
| max. human speed $v_max$ | $1.0\ m/s$ |
| min. human radius $r_min$ | $0.2\ m$ |
| max. human radius $r_max$ | $0.5\ m$ |
| success reward $r_s$ | 20 |
| collision reward $c_c$ | 5 |
| distance reward $c_d$ | 1 |
| action reward $c_a$ | 0.1 |

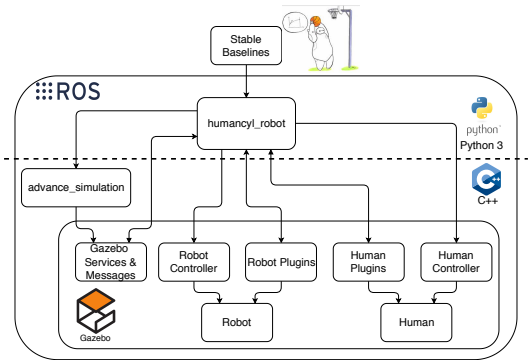
### 4.4.2 Training and Software architecture

Among the different reinforcement learning a Deep Deterministic policy Gradient (DDPG) [61] implemented by the library Stable-Baselines [64] has been used. The only difference, compared to the DDPG, implemented in [61] is the introduction of Prioritized Experience Replay Memory to increase the sample efficiency. The hyperparameters are detailed in Table 4.2.

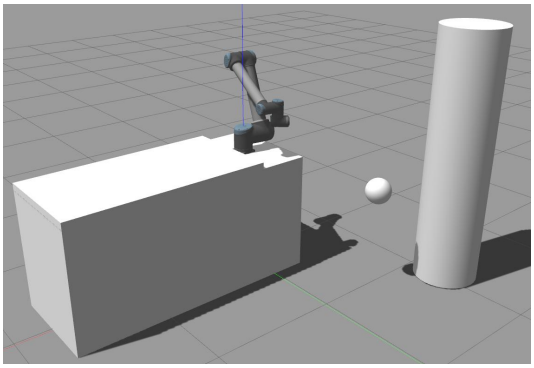
Figure 4.3a shows the system architecture* developed for training on the simulated environment is ROS and can be divided in 3 parts:

- **Stable-Baselines** The DRL library Stable-Baselines handle the Reinforcement Learning algorithm and communicate exclusively with the environment description (humancyl_robot) by sending the agent actions at each timestep and receiving the corresponding rewards $r_i$ and states transition $s_{i+1}$.

*code available at: https://github.com/giorgionicola/rl_gazebo




(a) The software architecture developed for the training phase. The DRL algorithm was implemented by using the library Stable-Baselines while the training environment is fully ROS based written in Python 3 and C++



(b) Image of the simulated environment in Gazebo physics simulator

Figure 4.3: The simulated environment developed to train the policy

- **humancyl_robot** The environment description. Its main functionalities are: i) observing the environment (i.e., the physics simulation) and computing the state from the observations ii) publishing the agent action (i.e., the robot joint speed commands) to the robot controller iii) defining the behaviour of all the non-agent elements of the simulation (e.g., the human movements) iv) advancing and resetting the simulation.

- **Gazebo** Gazebo is the physics simulator of the environment. As it can be noticed in Figure 4.3a, the physics simulation includes controllers for both the robot and the human and a set of plugins to add the required functionalities to the environment (ROS services for collision detection, setting and asking joint positions and velocities). Finally, Gazebo services and messages allows to control the simulation (start, pause, reset, advance $n$ steps).

The training results are detailed in Figure 4.4, where the evolutions of the average episode reward, the success rate, the collision rate and the time-over rate (i.e., the percentage of time the episode ended without either collisions or the robot reaching the target) are shown. The aforementioned curves were obtained by testing the policy every $5 \times 10^4$ steps on 20 episodes per target, for a total of 1080 test episodes. The training phase required 47 hours on a pc with CPU Intel(R) Core(TM) i7-7700HQ and a GPU Nvidia GeForce GTX 1050. The hyperparameters were chosen based on a grid search training the agent for a limited time, 500k steps. The grid search was limited to a small subset of the hyperparameters and for the remaining defaults values were used. The grid search was limited only to the learning rates, the mini-batch size and the ratio between rollout steps and training steps.

Table 4.2: List of DDPG hyper-parameter

| Hyper-parameters | |
|---|---|
| Training steps | $4 \times 10^6$ |
| Training episode length | $5\ s = 250$ steps |
| Discount rate | $0.99$ |
| Actor learning rate | $10^{-3}$ |
| Critic learning rate | $10^{-4}$ |
| Memory size | $2 \times 10^5$ |
| Minibatch size | $128$ |
| Target update factor $\tau$ | $0.001$ |
| Rollout steps | $60$ |
| Training steps | $1$ |
| Policy Network | $400 \times 10$ ReLU |
| Critic Network | $400 \times 10$ ReLU |
| Ornstein-Uhlenbeck $\mu$ | $0.0$ |
| Ornstein-Uhlenbeck $\theta$ | $0.3$ |
| Ornstein-Uhlenbeck $\sigma$ | $0.15$ |

### 4.4.3 Simulated validation

The simulated validation focused on answering four questions:

- What are the success and collision rate on the training targets?
- How much the overall success rate decreases introducing interpolated policy?
- Does an episode ended for time-over mean that the agent is stuck and cannot reach the desired target or it means that more time is required?
- Is the policy able to generalize over different human trajectories not described by the pseudo-random functions in (4.1) and (4.2)?
- Does the human speed have effects on the success rate?
- Is the policy able to generate continuous smooth trajectories?

In order to answer the first question, the policy has been tested $100$ times on each trained target position, for a total of $5400$ test episodes. The policy achieved an overall success rate of $91.76\%$. It should be noticed that an episode is accounted as successful only if the robot reaches the target without any collision. A deeper analysis is shown in Figures and Figure 4.5 and Figure 4.6, representing the success rate and the collision rate for each target. Both images are three horizontal slices of the target region at the $3$ different targets heights

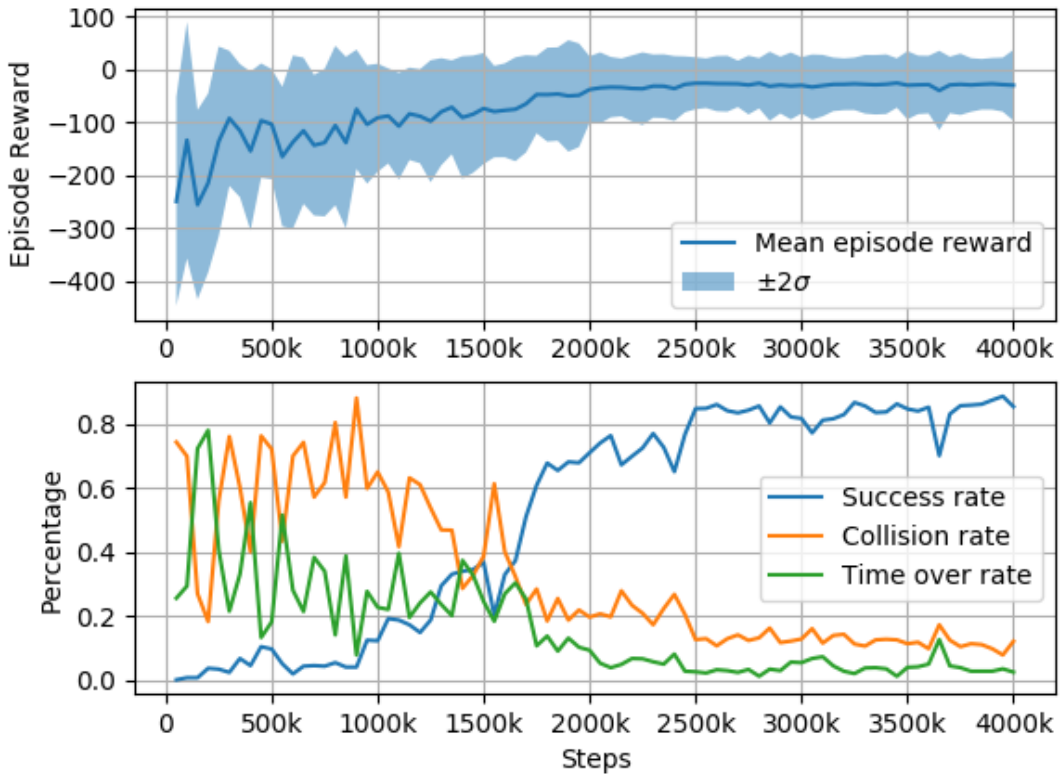



Figure 4.4: Learning curves: upper figure shows the episode reward; lower figure shows success rate, collision rate and time-over rate

($z = 0.8, 1.0, 1.2$). It can be noticed that the policy has high success rates and low collision rates everywhere but two regions of space:

- the lower right corner at height $z = 0.8$.
- the upper left corner at height $z = 0.8$.

Such results can be easily explained by pointing out that those two regions are the most difficult to reach. Indeed, the first one is extremely close to the table while the second is almost in the center of the human workspace, so it is the region with the highest probability of intersecting a human trajectory.

In order to answer the second question, the interpolated policy was tested over $5000$ random targets. The result, in Table 4.3, shows that the success rate slightly decreases, moving from $91.76\%$ to $89.74\%$. Such result confirms that the decrease in success rate is minimum compared to the benefit of passing from a discrete goal state to a continuous goal state. From a qualitative point of view the usage of the interpolated policy has caused the positive effect

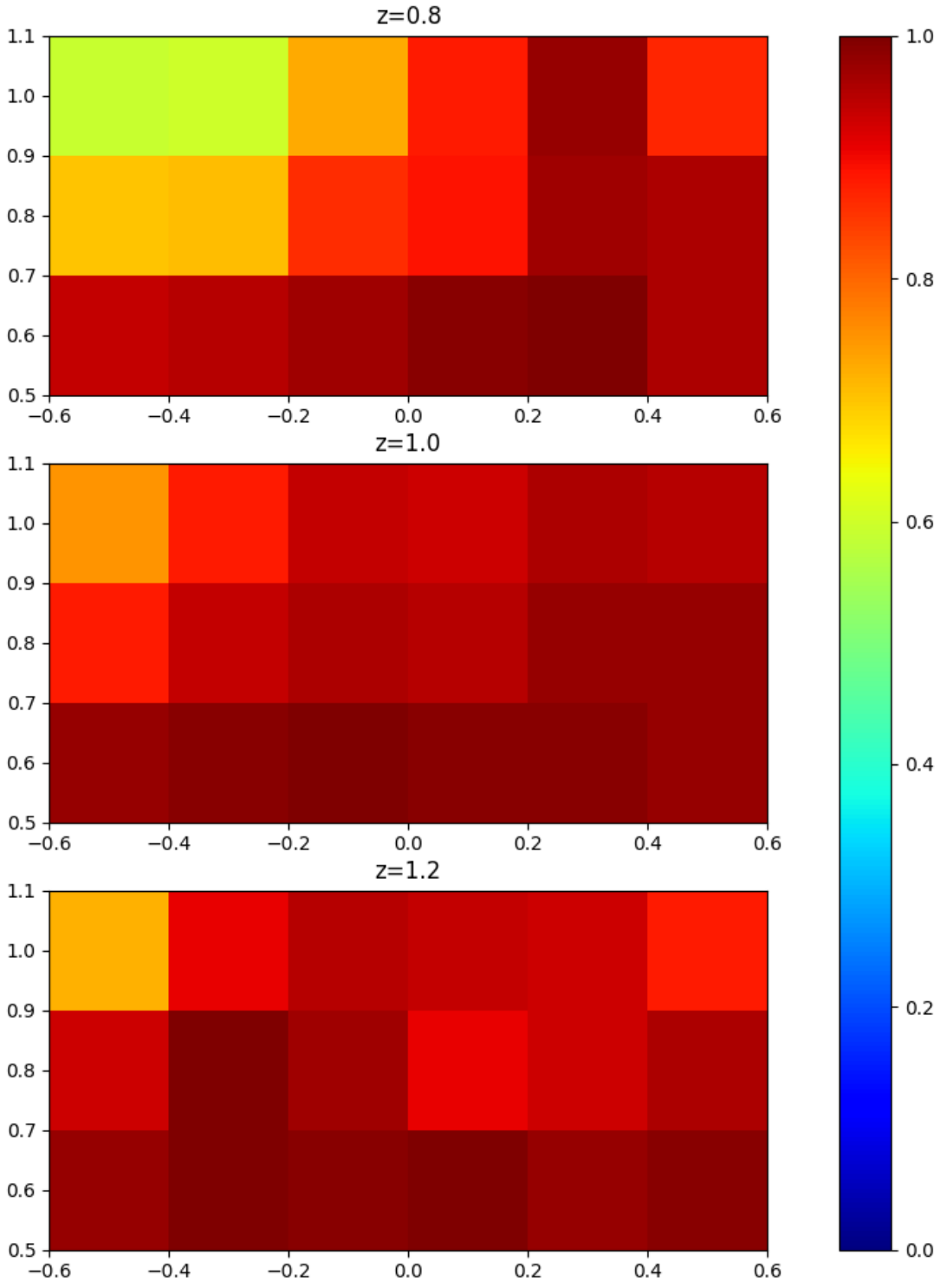



Figure 4.5: Analysis of the success rate for each training target

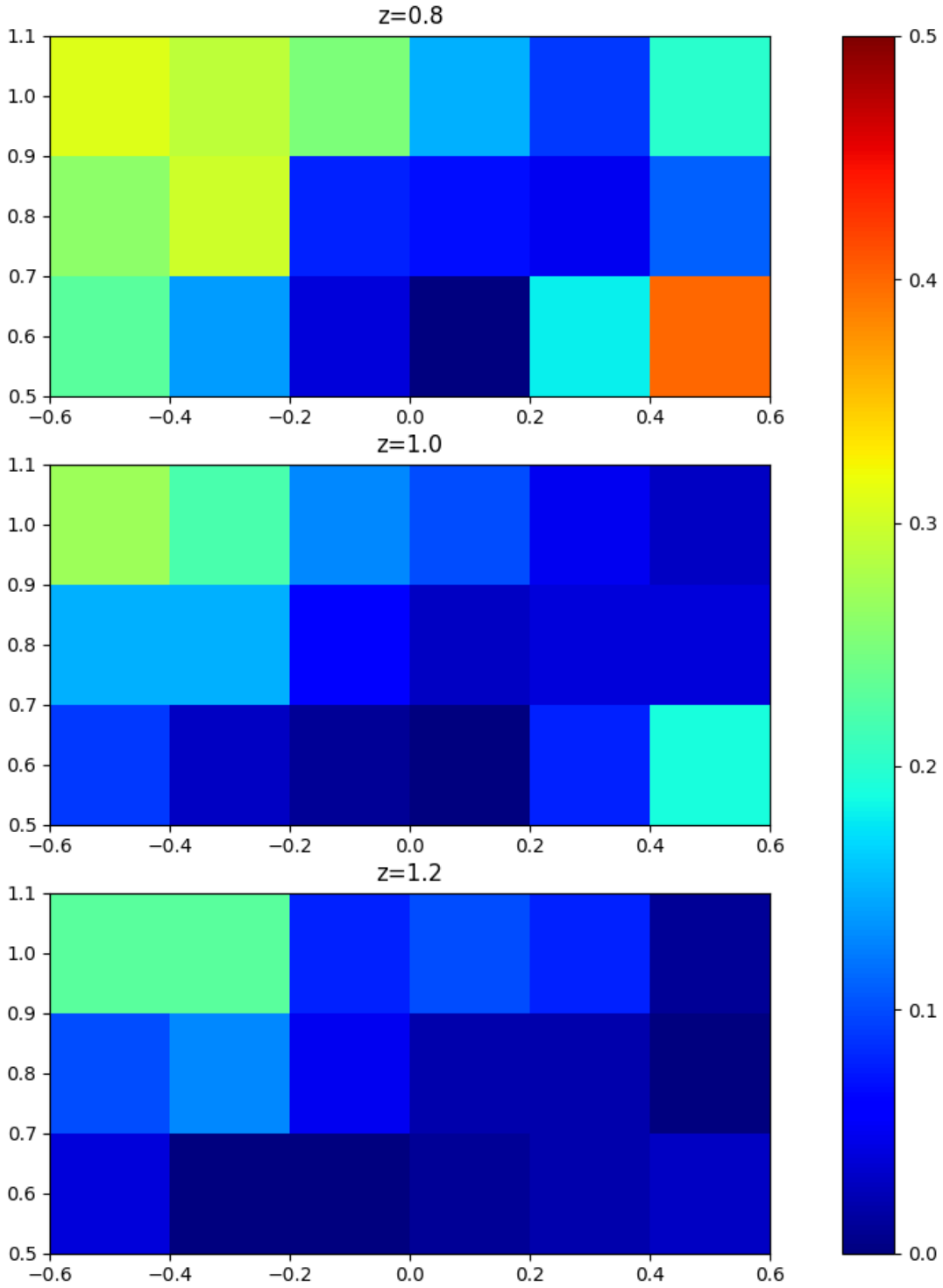



Figure 4.6: Analysis of the collision rate for each training target

of making the robot movements less jerky and smoother. The weighted average over the 8 surrounding training targets might indeed cause a regularization effect on the policy, thus reducing spikes in the policy function, explaining the empirical observations.

Table 4.3: Results of the interpolated policies

| Time $[s]$ | Success rate | Collision rate | Time-over rate |
|---|---|---|---|
| 5.0 | 89.74% | 5.5% | 4.76% |
| 25.0 | 92.6% | 6.18% | 1.22% |

The source of the so-called Time-over rate was investigated. Such parameter account for the episodes where the robot fails to reach the target within the defined time limit, but it successfully avoids collisions for the whole episode duration. Indeed, it might happen that the human interferes with the robot movements for more than 5 seconds, for example standing still in the area of the target for a long period of time. From Table 4.3 it can be noticed that by increasing the test episode length from 5 seconds to 25 seconds the time-over rate drops to $1.22\%$, the success rate raises to $92.6\%$, while the collision rate does not increase significantly. This result shows that, in a limited number of cases, the policy required more time to find a free path to the target due to the characteristics of those specific episodes.

We tested the robustness of the proposed approach to different functions used to model the human velocity and radius. In particular, instead of using the pseudo-random functions shown in (4.1) and (4.2) they were set fixed to a random value at the beginning of each episode. Even in this case, the policy was tested 100 times for each target for a total of $5400$ episodes and no significant variation of the success rate was observed.

Subsequently, it was analyzed the effect of the human speed on the success rate of the trained policy. The policy was tested in 3 cases where the human speed was fixed at 3 different values ($0.25\,m/s$, $0.5\,m/s$, $1\,m/s$) for 1000 episodes. No significant variation of the success rate was observed, showing that the policy was able to successfully generalize over the range of the human speed used in training.

Finally, to verify if the policy was able to generate continuous and smooth trajectories an example of a trajectory performed by the robot is reported in Figure 4.7. It can be noted the absence of any discontinuity at joint position and joint velocity.

### 4.4.4 Experimental Validation

The experimental setup developed to assess the in-field capabilities of the proposed approach is depicted in Figure 4.8a and Figure 4.8b. In particular, the latter shows the main ROS nodes required to interface and interconnect the hardware devices with the trained DRL model. Differently from the simulated environment, the position of the target and the state (i.e., position, velocity and radius) of the human-occupied cylindrical volume are no longer a-priori known, thus those quantities need to be estimated. To do so, an RGB-D camera (Microsoft Kinect v.2) was integrated in the system, and its pose with respect to the global reference

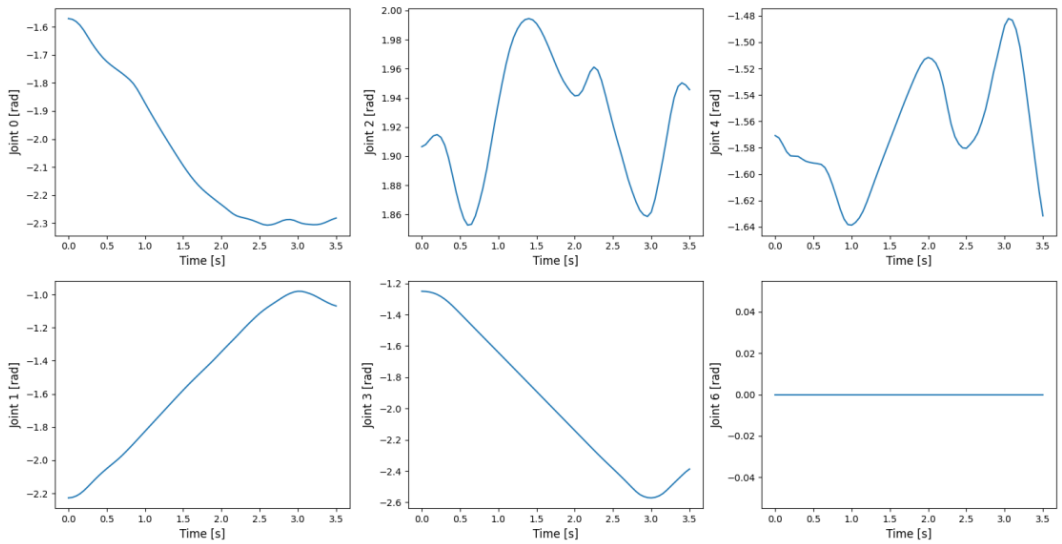
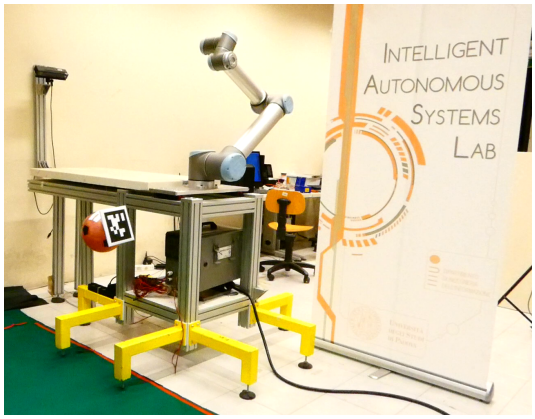


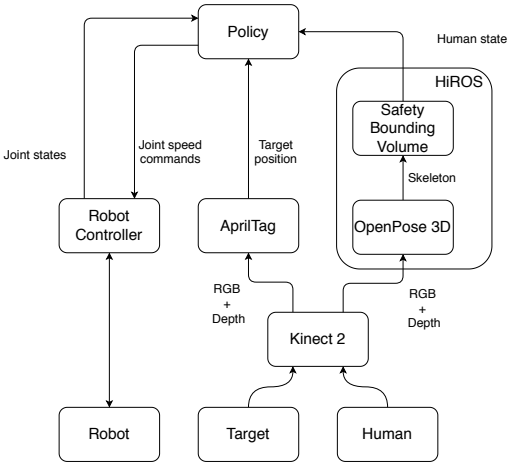

Figure 4.7: Example of a trajectory obtained with the trained policy.

frame calibrated using the `easy_handeye` ROS package [72]. For each experiment, the target is defined by placing in a random location within the goal space region a balloon, hanged to the roof, with an AprilTag marker attached on. Once positioned, an AprilTag detector node [73] estimates the target 3D position from the RGB image and the pointcloud provided, at each time frame (30 $Hz$), by the Kinect.

The human occupied volume, modeled as a variable-radius moving cylinder, is estimated from the Kinect RGB-D images according to an internally-developed human tracking pipeline relying a single-camera improved version of [74] (HiROS macro-block in Figure 4.8b). For this work, HiROS has been configured to be composed by three subsequent nodes: i) a configurable ROS wrapper for OpenPose, ii) a 3D projector, and iii) a safety bounding volume extractor. The former (i) takes as input the RGB image and estimates the 2D position of 25 human body key-points defined by the OpenPose `body_25` model, most of them coincident with the principal human joint centers. Those key-points, identified in the 2D image coordinates, are then projected by the following node (ii) in the 3D space through the depth image and expressed with respect to the global reference frame. Finally, the last node (iii) computes the so-called human safety-bounding volume, i.e., the minimum volume (m) containing all the 3D key-points of the detected person, thus well representing the volume occupied by the person itself. The computation is based on an SVD approach which computes the minimum-volume bounding box, then translated into a vertical cylinder containing the bounding box, multiplied by a safety factor of 1.1 . Following this approach, the radius of the safety cylindrical volume, the 3D absolute position and velocity of its centroid, and the target position are computed in real time at approximately 30 $Hz$.




(a) The physical experimental setup composed by a robot UR10 (the agent), a balloon with an AprilTag representing the target and a RGB-D camera Kinect




(b) The software architecture. The Policy receives as input the joints state, the target and human positions, and it produces as output the joint speed command that are sent directly to the Robot Controller

Figure 4.8: The experimental setup used to evaluate the policy

### 4.4.5 Results

The policy was tested on a total of 16 experimental runs each time with a random target position. In the first 8 runs, without the human interfering with the robot, the success rate was 100%. In the remaining 8 runs the human interfered with the robot, in particular, human interference with the robot consisted in walking/standing between the robot and the target or walking in proximity of the target. The policy was able to successfully complete the task in 7/8 runs, in the failed run however, the robot successfully managed to avoid collision with both the human and the fixed obstacles, like in all other experimental runs. There are various explanations to this result, first of all the behaviour of the human in simulation and the real world is different. Indeed, in simulation the human does not react to the robot's movements even in case of an imminent collision, meanwhile the real person, even if instructed appropriately to avoid the robot as minimum as possible, instinctively reacts to the robot movements avoiding collision when it is perceived as almost sure. However, such explanation would confirm the strength of the conservative approach in the design of the human occupancy model used in training. Secondly, it is possible that collisions were not experienced due to the limited number of experiments with respect to the high number of simulated episodes.

During the experimental evaluation, the following robot behaviors were noticed. First, the robot was able to recognize if the human was walking toward it and, in that case, the robot was able to maintain a safety distance from the person by moving in the opposite direction. Second, when the human was standing between the robot and the target, the robot moved

away from the person and it also started to move towards a zone not reachable by the human. This means that the policy gave priority to avoid any possible collision with the human rather than minimizing the distance from the target.

The average computation time of the interpolated policy with a CPU Intel(R) Core(TM) i7-7700HQ and a GPU Nvidia GeForce GTX 1050 is 6.9 $ms$ that is an acceptable value compared to the upper control frequency limit given by the maximum frame rate of the Kinect camera of 30 $Hz$.

Finally, it was noticed that the robot movements were less smooth in the real experiment compared to the simulated one. The authors believe that there are multiple reasons: first, the UR10 robot controller and the simulated one are not identical; second, in the simulated environment there are no delays and measurement noises from the human tracking systems. Indeed, there are no delays because all the required computations are performed while the simulation is paused and the measurements are not noisy because the human tracking system is modelled as ideal.

### 4.4.6 The Policy Combined with industrial safety standards

In order to demonstrate the applicability of the proposed approach in a real scenario, the method has been tested along safety standards. Specifically, a velocity scaling based on Speed and Separation Monitoring (SSM) from the ISO TS 15066 (as implemented in [75]) has been deployed. The velocity scaling introduces a speed override factor that decreases the joint speed commands according to two factors: i) the distance between the human and the robot; ii) the robot speed towards the person. The velocity scaling decreases the trajectory execution till completely stopping when the distance between the robot and the person are below a minimum safety distance. Therefore, it guarantees that collision between human and robot can happen only when the robot is hold, and it is appliable to any the motion plan regardless from the algorithm used to compute the plan. As a remark, once SSM is used, the quality of a feedback motion planner relies on the actual intervention of SSM. Intuitively, more SSM reduces the speed, less performing the motion planner is.

Among the most common motion planners, we selected RRT* [76] (Optimal Rapidly exploring Random Tree) as the benchmark for our methodology. Both the policy and RRT* have been tested on the same set of 30 human trajectories, performed by three people, consisting of a person crossing the robot workspace and passing between the robot and its target. For both RRT* and the policy, the same minimum safety distance of 0.3 $m$ was set.

In both cases, no collisions were experienced, and the person did not have to modify his/her trajectory to avoid the robot. However, some differences can be noticed looking at the speed override and at the total time required to reach the target as shown in Table 4.4. In particular, it can be easily noticed that the required time to reach the target is lower for the policy. This result can be explained by looking at the amount of time the safety override factor was below 50 (i.e., robot at 50% of the maximum speed) and was equal to 0 (i.e., the robot is hold). In both cases, the policy spent less time at reduced speed or stationary because

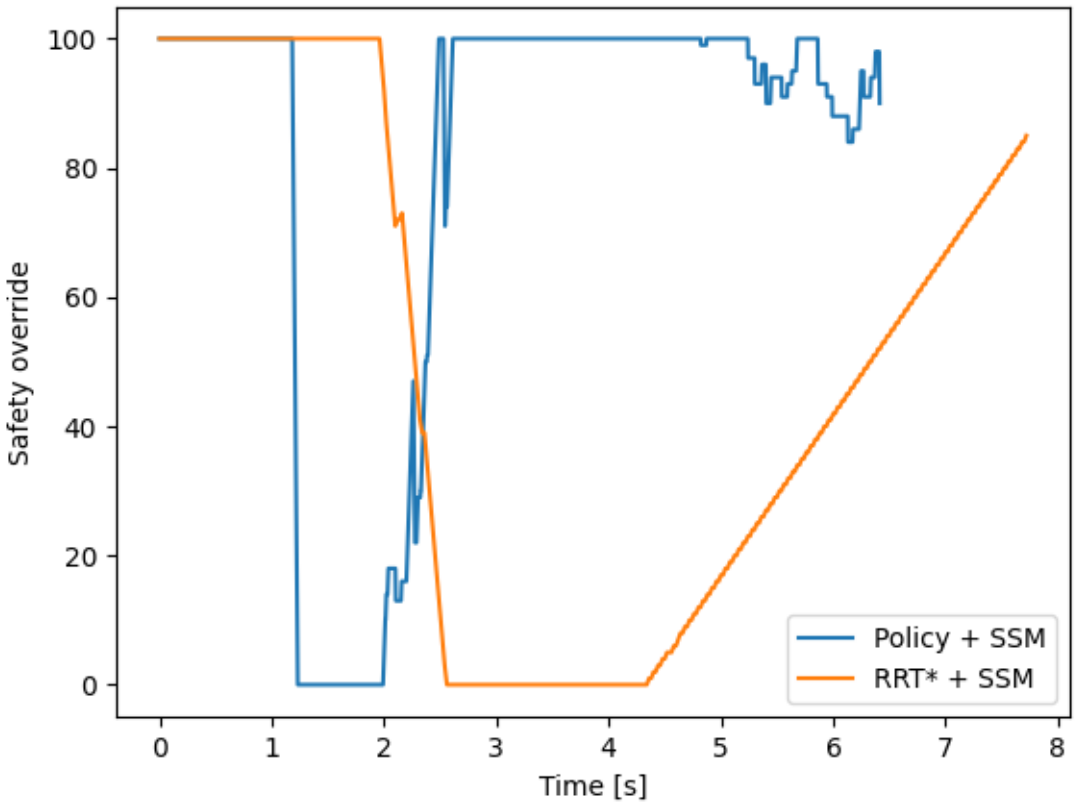



Figure 4.9: Comparison of the values safety override factor for the trained policy (blue) and a non-human aware motion planner like RRT* (orange). When the safety override is equal to 0 the robot is at rest and when it is equal to 100 the robot is at full speed.

the trajectory computed by the policy actually avoided the person meanwhile, the trajectory computed with RRT* did not avoid collisions, thus, it needed to wait for the person to move away from the robot trajectory. Furthermore, in about 30% of the cases the safety override factor with the policy did not reach zero, differently from RRT* where it always reached zero.

Table 4.4: Comparison between the Policy combined with SSM and RRT* combined with SSM, mean time plus standard deviation

| | Time | Safety override$< 50$ | Safety override$= 0$ |
|---|---|---|---|
| Policy | $5.74 \pm 1.04\ s$ | $0.21 \pm 0.2\ s$ | $1.30 \pm 0.73\ s$ |
| RRT* | $8.082 \pm 1.65\ s$ | $1.42 \pm 0.69\ s$ | $4.30 \pm 1.82\ s$ |

In Figure 4.9 an example of the safety override factor for the Policy and RRT* in the studied scenario is shown. As it can be noticed not only the total time required and the amount

of time spent at reduced/zero speed is lower but also the safety override factor increases faster for the policy.

In conclusion, the policy combined with industrial safety standard like SSM from ISO TS 15066 is able to completely avoid collisions, and it is more efficient than the current industrial standard based on non-human aware motion planner plus a safety standard like SSM.

## 4.5 Conclusions and future works

In this chapter, a DRL based motion planning approach for human-robot shared workspace applications is proposed. Such methodology, compared to most of the methods in literature, does not require any a-priori knowledge about the human behavior. Furthermore, it is proposed a novel formulation of the MDP action. The action is defined as a parametric polynomial function at the joint level, whose coefficients are computed by the policy. It demonstrated that for any polynomial trajectory with degree 5 or lower, it is possible to compute the range of coefficients that guarantee the trajectory not to exceed the joint limits, if it exists. The policy has been trained on a simulated environment on a limited number of targets and subsequently, by means of trilinear interpolation of the policy over the discretized training goal region, the robot proved to be capable to reach any target within such region showing a neglectable performance drop. The tests on the simulated environment shown success rate close to 92%. The policy has been successfully applied to the real world experimental setup without retraining or fine-tuning, highlighting the generality and robustness of the methodology proposed. Finally, in order to prove that the proposed approach can be applied in real world scenarios, it was combined with the current industrial safety standard and compared with a standard motion planner with the same safety standards. The comparison showed that in both cases no collision occurred, however, the proposed motion planner was able to reach the target faster and required less intervention of the safety measures, proving that the proposed approach is more effective than the current industrial best practice.

At the current stage, one of the limits of the proposed method is that the maximum number of people sharing the robot workspace must set during the training phase. The next step of our research will aim at removing the aforementioned prerequisite. A possible solution could be deploying a policy for each person inside the robot workspace and then averaging the action based on the distance between human and robot. Another possible solution could be to develop a policy based on recurrent neural networks such as in [39]. The grid-based interpolation of the policy will be substituted with an interpolation over a low discrepancy sequence of the training targets, such as the Halton sequence, in order to avoid that the policy learns undesired spatial patterns. Finally, the model will be extended, the human collision bounding volume will be substituted with a more realistic model and the robot will be trained to perform more complex task (e.g., pick and place or inspection of products) and also cooperative tasks such as object hand-over. The authors also plan to use the proposed method as a tool to simulate and study the effects of human robot cooperation on manufacturing processes. For example, the model could be used to evaluate in simulation the robot

execution time in human-robot shared workspace scenarios.

*No plan of operations extends with any certainty beyond the first contact with the main hostile force.*

Helmuth von Moltke the Elder

# 5

# Feedback Task and Motion Planning in Human-Robot Cooperative Scenario

In the previous Chapters, Task Planning and Motion Planing have been studied as separate problems. In this Chapter, instead, we combine Motion Planning with Feedback Task Planning. The motion planning is grounded on the results from Chapter 4. Meanwhile, the Task Planning compared with Chapter 3 divide the symbolic information of a task from its geometrical information. In this Chapter, we analyze how to properly use the geometric information in a non-deterministic and dynamic environment.

## 5.1 Introduction

The previous chapters proposed methods to perform task planning in non-deterministic environments in the presence of humans, Chapter 3, and compute safe motion plans, Chapter 4. In this Chapter, instead, we present a method to combine both motion planning and task planning. Indeed, as described in Chapter 1, task planning and motion planning in human-robot cooperative scenarios should be studied as a unique problem.

Let's consider the same scenario of object sorting proposed in Chapter 3. In that scenario, the robot was required to sort a set of desired objects, and in particular, multiple objects with the same attributes were available to solve the same task (e.g., two small red cubes were available, and only one was requested). In those cases, the policy chose which category of object (i.e., set of attributes) to pick and which object to pick among the ones with the same attributes. In other words, the policy was required to solve simultaneously both the symbolic level (i.e., choose the class of equivalent objects to pick) and the geometric level (i.e., choose one object within the class). In many robotic applications, the solution to the geometric level of the task planning is determined as the task that is considered optimal from the motion planner's point of view. In Chapter 2, a different approach was implemented. The task feasibility was encoded as a failure probability in the transition probability function, and the trajectory's quality was introduced in the reward function as a mathematical function. That formulation provided a good level of abstraction of the motion planning, simplifying the simulating environment. However, as the scenario becomes more complex (e.g., introducing other agents), it becomes impractical to describe the outcome of the motion planning routine for the desired task in terms of state transition probability and the reward functions. Therefore, in those complex cases, it becomes necessary to completely simulate the chosen task, including the motion planning execution.

In this Chapter, we propose to decouple the symbolic and the geometric problems. The symbolic level is solved as in Chapter 3; meanwhile, the geometric problem is solved by extending the approach to motion planning described in Chapter 4. As in Chapter 4, the motion planning problem is defined as a sequential decision-making problem described by an MDP and solved via DRL, and an actor-critic algorithm is used to train the motion planning policy. The feedback task plane is performed, evaluating the learned critic function with respect to the various tasks and choosing the one that maximizes it. Indeed, the critic function is trained to fit the expected cumulative reward that, on the one hand, is the objective function the policy has to maximize; on the other hand, it describes the quality of solution provided by the policy and, in our case, the quality of the trajectory. For example, low values of expected reward mean high chances of a collision or that it may require a very long trajectory. Thus, choosing the task that maximizes the expected cumulative reward of the motion planning is chosen the task with the overall best solution. Furthermore, the critic function can be easily evaluated online, allowing to modify the selected task online based on the current environment status.

Let's consider a symbolic task that can be completed by performing one task a goal $t$ among a set of geometric tasks $\mathcal{T}$, for example in a pick&place activity the robot instead of picking a specific object might be required to pick one object that has certain properties and multiple objects satisfy the desired properties. Thus, the problem is to select the optimal goal $t^*$ among $\mathcal{T}$ that maximize a fitness function $f$.

$$t^* = \underset{t}{\operatorname{argmax}}\, f(t) \quad t \in \mathcal{T} \quad \mathcal{T} = [t_0, \ldots, t_n] \tag{5.1}$$

The main difficulty of the problem formulated is that $f$ in most of the cases is not straightforwardly dependent from the geometric task, meanwhile it is function of the motion plan $\mathcal{P}(t)$ to reach $t$. Thus, in order to find the optimal task, it is necessary to compute the optimal motion plan for each task and then choose the best one.

$$t^* = \underset{t}{\operatorname{argmax}}\, f(\mathcal{P}^*(t)) \quad t \in \mathcal{T} \quad \mathcal{T} = [t_0, \ldots, t_n] \tag{5.2}$$

After modelling the motion planning problem as an MDP like in Chapter 4 we use as fitness function the expected cumulative discounted reward Therefore we define the vector $\mathbf{R}$ composed by the expected cumulative discounted reward composed for each geometric task $t_i$:

$$\mathbf{R} = [R_{t_0}, \ldots, R_{t_n}]\,. \tag{5.3}$$

Then we select the optimal task as:

$$t = \underset{t}{\operatorname{argmax}}\, \mathbf{R}\,. \tag{5.4}$$

To compute $R$ we use a critic function that in actor-critic algorithms is learned during the training phase. The critic function can be modeled as the Value function $V(s)$ or as the Action-Value function $Q(s, a)$, and we can compute $R$ as:

$$R = V(s) = Q(s, \pi(s)) \tag{5.5}$$

In order to evaluate the $R$ with respect to the tasks we define the state $s$ as the concatenation of the state of the environment, that includes the agents and eventual dynamic obstacles, and the task state, i.e., the task position.

$$s_t = \begin{bmatrix} s_{env} \\ s_{task} \end{bmatrix} \tag{5.6}$$

Thus, the vector $\mathbf{R}$ becomes:

$$\begin{aligned}\mathbf{R} = [R_{t_0}, \ldots, R_{t_n}] = [Q(s_{t_0}, \pi(s_{t_0})), \ldots, Q(s_{t_n}, \pi(s_{t_n}))] = \\ = [V(s_{t_0}), \ldots, V(s_{t_n})]\end{aligned}, \tag{5.7}$$

Theoretically, the same approach can be applied to policy learned with any reinforcement learning algorithm, not only to those based on actor-critic. However, it would require to acquire a dataset of trajectory for each task with to learn the critic function, described either as $Q(s, a)$ or $V(s)$. Instead, in actor-critic algorithm the expected reward is already learned during the training phase, thus it can be directly applied.

The feedback task plane is not implemented during the training phase, otherwise it would cause an effect similar to the maximization bias affecting Q-learning as described in Section 2.3.4. In standard Q-learning, the maximization bias causes the agent to take suboptimal actions both in training and in testing, because the Q-function is overestimated thanks to the maximization operator used to estimate the target value. Analogously, the feedback task plane is performed with a greedy strategy with respect to the expected reward via a maximization operator. If the feedback task plane was applied during the training, the policy would be affected by a bias. Indeed, once some tasks are found to be easier, those tasks will be chosen more frequently independently if they are really optimal.

This bias has two negative effects. First, the policy will perform extremely well on the tasks that are easier because they are chosen more frequently, instead, it will perform badly on harder tasks since not it was not collected enough experience. Second, it affects also the training of Q-value function, since the agent collects more training samples on the successfully solved tasks, meanwhile it collects few training samples on the tasks that are not solved yet. Thus, it causes that the estimation of $R$ is accurate for the successfully solved tasks and inaccurate for the others. We defined this problem as the *task maximization bias*. It should be noticed, that the task maximization bias does not have the effect of learning only to solve the goals that are easier and therefore, that could be the optimal choice. Instead, it biases the policy training on those goals that have already at least a partial solution. Let's consider the limit case where a policy and a critic function are pre-trained to solve only the targets that in reality are suboptimal. If we start the training of the policy and implement the feedback task plane, only the suboptimal tasks that already trained would be chosen, meanwhile, the policy would not train on the other tasks that in reality are optimal.

Summarizing, the task maximization bias can cause overfitting on suboptimal tasks already solved and underfitting on optimal tasks that are not solved yet and could be optimal. In conclusion, to avoid the task maximization bias the policy is trained without feedback task plane, instead the tasks are sampled from a uniform distribution.

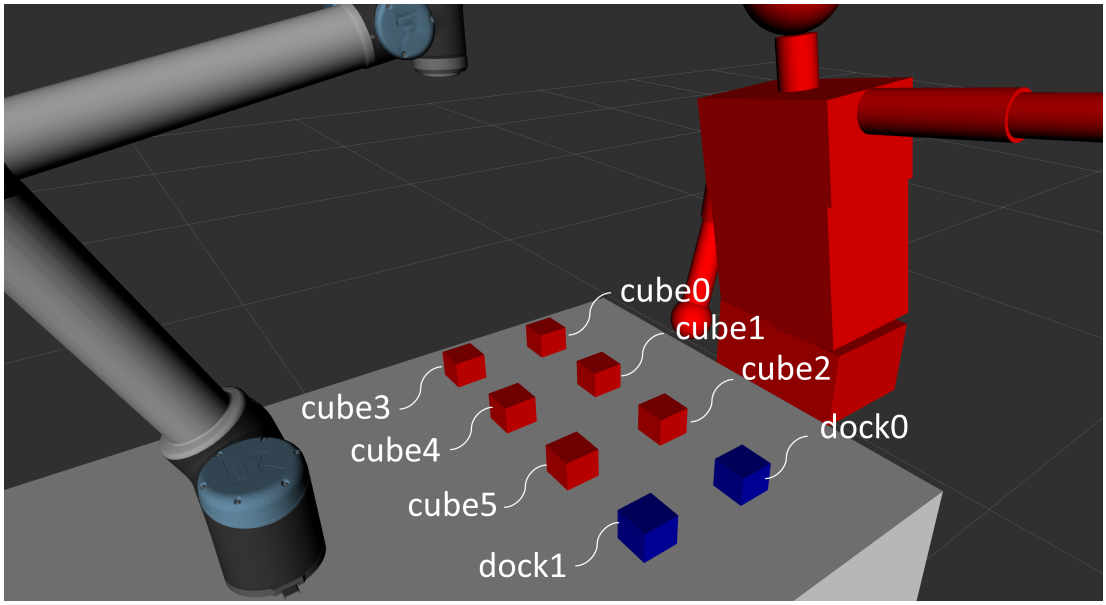



Figure 5.1: The scenario represents a human-robot cooperative pick and place application. The human performs a set of tasks in s cyclic manner, moving from a home position to the red cubes and to the blue cubes. The robot moves to the red and blue cubes from a home position or from one of the red/blue cubes

## 5.3 Experiments

In this section, the online feedback task plan is applied to a human-robot cooperative scenario combined with the feedback motion planner developed in Chapter 4. Firstly, the human-robot cooperative application is described, and it is formalized into an MDP. The formulation of the state of the action and the reward function is described. Subsequently, it is described the pipeline used to simulate a person performing pick and place tasks. The pipeline included firstly acquisition of human movements via a motion capture system, the conversion to a library of Probabilistic Movement Primitives, and a set of data augmentation techniques to avoid overfitting during the training. Afterward, the training phase is described, including the DRL algorithm used and the curriculum learning strategy implemented. Therefore, the training results are presented and an extensive evaluation of the feedback task planner is presented.

### 5.3.1 Scenario

The studied scenario is a human-robot cooperative pick and place application shown in Figure 5.1. In such scenario the human moves in a cyclic manner from a home position to a random red cube and then to the dock station represented as the blue cube, the human

can also perform the task in the opposite direction (dock→cube→home). Meanwhile, the agent, an anthropomorphic robot (UR10), is trained to reach any cube and all the docking stations. The agents can start from multiple positions: 1) a home position, out of the human workspace; 2) from a red cube; 3) from a docking station (blue cubes).

Similarly to the works presented in Chapter 4 and Chapter 3 the training is performed in a simulated environment. In particular, a human model is introduced, and its movements are based on a dataset of pick and place trajectories.

The dataset of trajectories performed by the human was acquired via a IMU (Inertial Measurement Unit) based motion tracking system. In particular, a set of Xsens MT Awinda were used, and the IMU readings were obtained via [77]. The IMU readings after an optimized inverse kinematics algorithms performed by the simulator OpenSim [78, 79] were converted in joint angles of the developed 18 DoFs human model. Subsequently, the joint trajectories are converted to a library of Probabilistic Movement Primitives.

An episode finishes under the following circumstances:

- Collision: a collision between robot and human or between robot and environment;
- Success: the robot end effector is within a distance of $0.1\ m$ from the target, the angle between the end effector axes and a vertical line is below $0.2$ radiant and the end effector speed is below $0.1\ m/s$;
- Timeout: the simulation lasts 10 seconds without experiencing collisions, an unfeasible trajectory or success.

Unlike most works, a specific simulator was developed instead of using a physics simulator like Mujoco [80], Gazebo [81], or PyBullet [82]. Indeed the simulators mentioned above are physics simulators, while in a motion planning application, it is necessary to simulate only the kinematics and to perform the collision detection. For this reason, the kinematic simulation was developed in Python 3, and the collision detection was performed with the library GPU-Voxels [83].

We present the relevant parameters of the environment in Table 5.1.

### 5.3.2 State, Action and Reward

The state vector $\mathbf{s}$ is defined as the concatenation of the robot joint position and speed, the human joints position and speed and the target position:

$$\mathbf{s} = \begin{bmatrix} \text{robot joints position} \\ \text{robot joints speed} \\ \text{human joints position} \\ \text{human joints speed} \\ \text{target position} \end{bmatrix} . \tag{5.8}$$

We chose the action as a second order polynomial function for each joint, thus we can write, based on Section 4.3.2:

$$\mathbf{a}_t = \begin{bmatrix} c_1^t x^2 + \dot{q}_0 x + q_0 \\ \vdots \\ c_5^t x^2 + \dot{q}_5 x + q_5 \end{bmatrix} \tag{5.9}$$

$$c_i^t = \frac{c_{max,i}^t + c_{min,i}^t}{2} + a\frac{c_{max,i}^t - c_{min,i}^t}{2} \quad a \in [-1, 1] \tag{5.10}$$

In this scenario only the first 5 joints of the robot UR10 are controlled, the last joint, that correspond to the rotation along the tool axis, is set constant to zero since it has no influence on the successful completion of an episode.

We define the reward function shown in (5.11) as the sum of a distance reward $r_d$, an end effector orientation reward $r_\theta$, a collision reward $r_c$, a reward for every timestep $r_{step}$ and finally a reward for successfully finishing an episode $r_{success}$. In detail, the distance reward $r_d$ and the reward for end-effector orientation $r_\theta$ and are proportional to the variation of distance/orientation from the target value (cube/docking station position for $r_{dist}$ and end effector axis vertical orientation for $r_\theta$). The collision reward is a constant negative reward whenever a collision is detected. Similar to the collision reward, the unfeasible trajectory reward is a negative constant value obtained when one of the robot joints exceed the position/velocity limit. At each time step elapsed, a constant negative reward $r_{step}$ to encourage the agent to complete the task as fast as possible. When the end effector is within a tolerance distance $d_{tcp\ speed}$ from the target, a negative reward proportional to the tcp speed is added to promote the agent to reach the target with speed equal to zero. Finally, if the episode terminate successfully, a positive reward $r_{success}$ is added.

$$r = r_d + r_\theta + r_c + r_{step} + r_{tcp\ speed} + r_{success} \tag{5.11}$$

with

$$r_d = c_d(d - d_{old}) \qquad r_\theta = c_\theta(\theta - \theta_{old})$$

$$r_c = \begin{cases} -c_c & \text{if collision} \\ 0 & \text{if no collision} \end{cases} \qquad r_{step} = -c_{step}$$

$$r_{tcp\ speed} = \begin{cases} -c_{tcp\ speed} v_{tcp\ speed} & \text{if } d \leq d_{tcp\ speed} \\ 0 & \text{if } d > d_{tcp\ speed} \end{cases} \qquad r_{success} = \begin{cases} c_{success} & \text{if success} \\ 0 & \text{if no success} \end{cases}$$

We would like to point out the design of the distance reward $r_d$ of the angle reward $r_\theta$. Typically, the reward function used for solving a reaching task is modelled as a negative reward, proportional to the distance. However, this formulation has 2 major disadvantages, first it is possible that in some scenario the agent, in order to minimize negative rewards, learns how

Table 5.1: List of all environment parameters

| Environment parameters | |
|---|---|
| command frequency | $20\,Hz$ |
| distance threshold $d_t$ | $0.1\,m$ |
| angle threshold $\theta_t$ | $0.2\,rad$ |
| max. joint speed | $\pm 0.5\,rad/s$ |
| success reward $c_{success}$ | $1$ |
| collision reward $c_c$ | $-1$ |
| distance reward $c_d$ | $0.5$ |
| angle reward $c_\theta$ | $0.5$ |
| step reward $c_{step}$ | $0.02$ |
| joint tcp speed $c_{tcp\ speed}$ | $-1$ |

to stop prematurely the episode "committing suicide". Indeed, in environments where the positive rewards are very sparse and negative rewards are very common, the agent can easily learn actions that cause the episode to end prematurely to collect as little as possible negative rewards. Second, two trajectories with the same duration and same length can have different cumulated reward based on the amount of time spent at different distances. Therefore, it induces in the agent the knowledge that spending an amount of time close to the target is better than spending the same amount of time far from the target. However, in our scenario such behaviour is not desirable since it would encourage the robot to spend time in proximity of the of target even if it is not reachable because of the presence of the human, increasing the chances of a collision. Our formulation, instead, model the reward for the distance and the orientation of the tool as a positive reward proportional to the decrease of distance/angle toward the target position/orientation. Thus, on one hand, the agent is encouraged to move towards the target to collect positive rewards even in environments where negative rewards are predominant; on the other hand, the cumulated reward of the distance is no longer dependent from the amount of time spent at different distances. Indeed, when the agent is not moving, it does not receive any positive or negative reward.

All the relevant environment parameters including reward coefficient are shown in Table 5.1.

### 5.3.3 Human Trajectory Acquisition and Data Augmentation

In order to simulate the movements of a human in the training environment, a human model with 18 Degrees of Freedom (DoFs) is introduced, shown in Fig. 5.2. The developed model has the following DoFs:

- 2 translational and 1 rotational at the pelvis;
- 3 rotational between pelvis and abdomen;

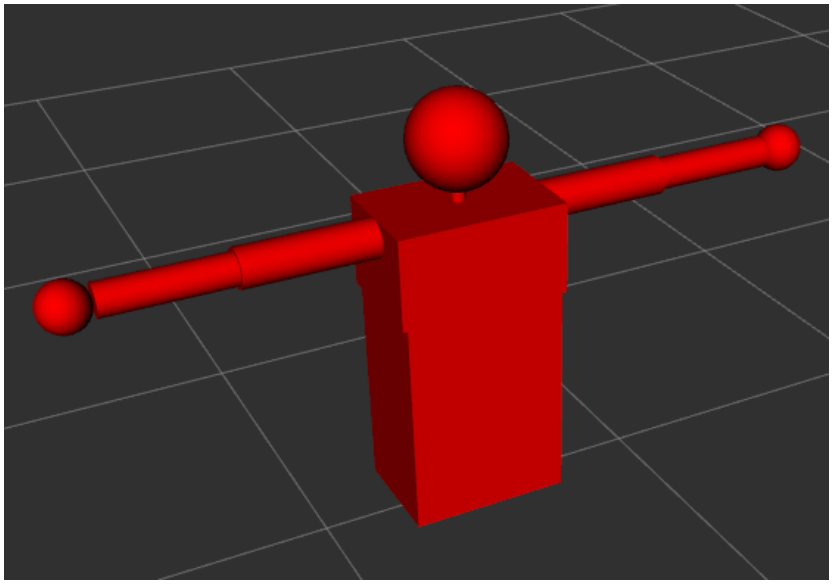


Figure 5.2: The 18 DoFs human model implemented

- 2 rotational between the sternum and each clavicle;
- 4 rotational per arm, divided as 3 in the shoulder and 1 in the elbow.

Subsequently, a dataset of a person performing a set of tasks in the studied scenario is acquired and used to create a library of Probabilistic Primitives Movements (ProMP) [84] that represents all the tasks the human performs in the scenario.

During the training phase, the human randomly selects a task from the library and generates a trajectory by sampling the ProMP library. To prevent the agent from overfitting the acquired dataset of trajectories, data augmentation was applied by:

- changing the trajectory execution time;
- adding variable random pause between consecutive tasks;
- exploiting the redundancy of the human arm to maintain the hand trajectory while modifying the elbow position;
- random starting position of the human, in other words the episode can start with the human that has already partially completed the selected task;
- reverting human trajectory, i.e., executes the trajectory in the opposite direction.

In order to create new human movements with same hand trajectory and different elbow position, a set of via points are created and the corresponding ProMP is conditioned to pass

through those points. Such approach allows to greatly enlarge the set of human trajectories while maintaining the properties of the original movements.

### 5.3.4 Training

As detailed, in Section 5.2 the reinforcement algorithm must be actor-critic and among this class of algorithms it was chosen TD3 [62] (see Section 2.7.2 for a detailed description of the algorithm) despite the fact that it uses as critic function, and therefore, is more computationally expensive to select the goal since also the policy must be evaluated for each goal. However, TD3 not only is one of the state of the art algorithms, but also it addressed the problem of bias overestimation of the critic function as detailed in Section 2.6, thus it is the algorithm that provide the most accurate estimate of the true critic function. However, differently from the original implementation, as exploration strategy it was implemented Parameter Space Noise [85].

In order to facilitate the training and increase its speed, a curriculum learning strategy was developed. As detailed in Section 5.3.1, the agent is trained to reach any cube/dock starting from any cube/dock or a home position. Thus, at the beginning of each episode, a random target and a random starting position are sampled. The curriculum learning strategy developed consists in starting each episode with the distance between the robot and the target sampled from a triangular distribution. The triangular distribution is defined between 0 (i.e., the robot is on the target) and 1 (i.e., the robot is in the sampled starting position). At the beginning of the training phase, the distribution has the mode close to 0 (i.e., the robot is close to the target), the lower limit equal to the mode and the upper limit equal to 1. During the first 500k training steps, the mode of the distribution move progressively toward the upper limit until they are equal. In the successive 500k training steps the lower limit moves toward the upper limit until after 1 million steps from the beginning of the distribution collapse to a single the value of 1, and therefore, the robot start the episode exactly on the sampled starting position. The implemented strategy, allows reducing the difficulty of the episodes and therefore allowing the agent to experience successful episodes even with low-quality policies.

In Table 5.2 the main TD3 hyperparameters used are reported.

The training results are detailed in Figure 5.3, where the evolution of the average episode reward is shown. The aforementioned curves were obtained by testing the policy every $50 \times 10^4$ steps on 200 random episodes. It can be observed that the agent was able to achieve a success rate of 75% after $2 \times 10^6$ steps. The failures were divided as follows, collision rate 25% and timeout rate 0%.

### 5.3.5 Feedback Task Planner Evaluation

To prove the effectiveness of the feedback task plane, first was tested similarly to Section 5.3.4 on 1000 episodes allowing the robot to choose between a random number of random targets (between 2 and 3). The results showed an increase of the success rate from 75% to 90%. In

Table 5.2: List of TD3 hyper-parameters

| Hyper-parameters | |
|---|---|
| Training steps | $4 \times 10^6$ |
| Training episode length | $20\ s = 400$ steps |
| Discount rate | 0.99 |
| Actor learning rate | $10^{-4}$ |
| Critic learning rate | $10^{-4}$ |
| Memory size | $5 \times 10^5$ |
| Minibatch size | 64 |
| Target update factor $\tau$ | 0.001 |
| Policy training frequency | 3 |
| Parameter space noise $\sigma$ | 0.5 |

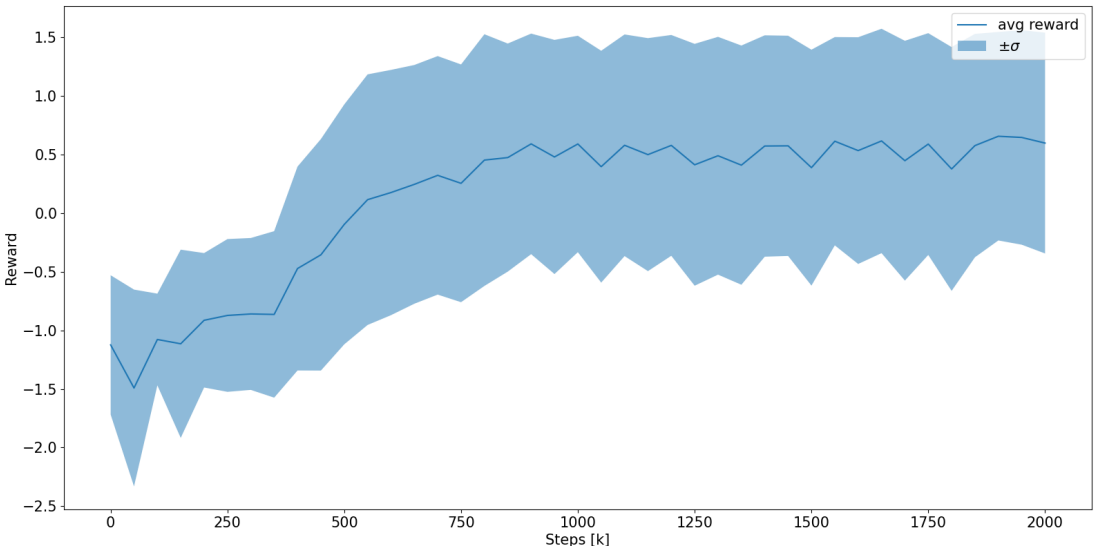



Figure 5.3: Evolution of the average episode reward during the training

order to prove that the proposed method is more effective than a simple heuristic like, for example, the distance, a set of specific tests were developed with the objective to highlight specific behaviors learned by the agent during the training. The tests, briefly summarized in Table 5.3, aim at verifying that the selection of the task based on the critic function is able to choose the optimal task, with a trade-off between the probability of colliding with the human and the required time to reach a target, i.e., the distance. For each test, a set of two possible tasks are available to the robot meanwhile the human can either stand still in proximity of one of the available goals to the robot (case studies 1 and 2, type: Static), or he/she performs a movement that will interfere with the robot movement toward one of the available goals (case studies 3 and 4, type: Dynamic).

Table 5.3: Summary of the tests

| Case study | Type | Robot start | Robot goals | Human start | Human goal |
|---|---|---|---|---|---|
| 1 | Static | cube5 | {cube0, cube4} | cube4 | None |
| 2 | | cube2 | {cube0, cube1} | cube1 | None |
| 3 | Dynamic | cube5 | {cube0, cube1} | cube3 | dock1 |
| 4 | | cube3 | {cube1, dock0} | dock1 | cube0 |

- **Case study** 1 We start with a trivial proof of concepts in terms of human interference. We want to prove that if both targets are reachable with different degree of human interference, the closest target that also minimize the human interference will maximize the expected reward, and it will be chosen.

  The robot starts from *cube5* and should move towards one of the targets of {*cube0*, *cube4*}. At the same time, the human's hand is stationary in proximity of *cube4*, thus both *cube0* and *cube4* are reachable without collisions, however *cube0* requires the robot to cross with the human differently from *cube0*. We expect the robot to select and reach *cube4* since it is the closest and it minimizes interferences with the human.

  As shown in Figure 5.4, already at time step $t = 0$, the robot perceives that *cube4* is much easier to reach compared to *cube0*. The Q-value of *cube4* keeps increasing until the robot successfully reaches it, meanwhile, the expected reward for *cube0* decreases since the robot get closer to the human, therefore it is more likely to have a collision.

- **Case study** 2 In this experiment, it is verified that the agent is able to choose the target not only based on the distance as a simple heuristic. Instead, it is able to evaluate the risk of collision and prefer targets that further but less hazardous.

  The robot starts from *cube2* and is assigned to {*cube0*, *cube1*}. At the same time, the human's right hand is steady on *cube1*. Therefore, *cube0* is totally free, meanwhile *cube1* it is almost completely covered by the human and to reach it the robot must

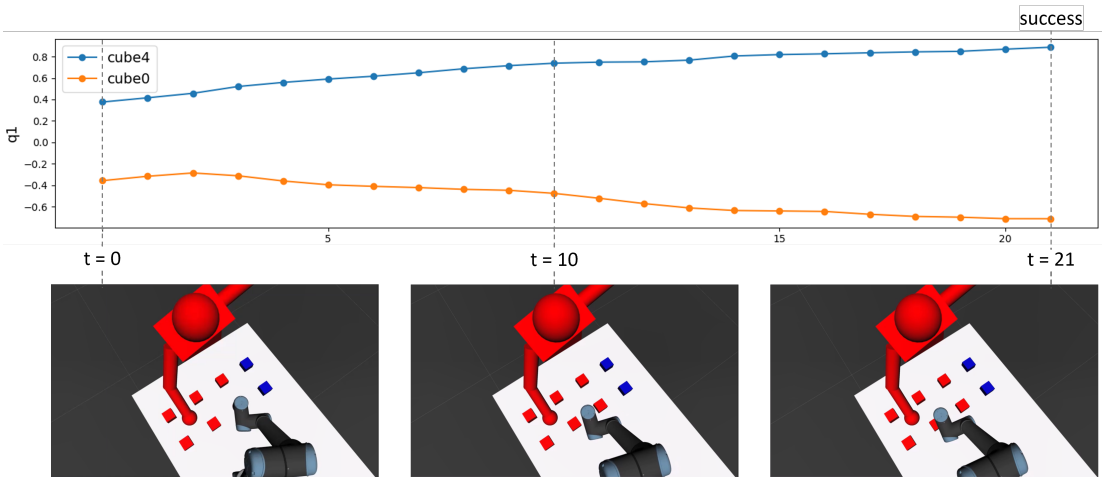



Figure 5.4: Case study 1.

come very close to the human with a very high risk of colliding if the human starts moving. It should be remembered that the robot does not know that the human will stand still indefinitely, instead it expects him/her to start moving again sooner or later. As the human is occupying *cube1* (that is the optimal target in terms of distance), the robot should automatically select *cube0* and move toward it.

As shown in Figure 5.5, at the beginning the robot is very close to the human thus it is perceived a high risk of collision and both expected rewards have very low values. However, *cube0* since it is completely free, has an expected reward slightly higher, and it is chosen. As the robot moves toward *cube0* circumventing the human, *cube1* is extremely close, and it is perceived that it is reachable and therefore is chosen ($t = 13$). However, eventually *cube0* is recognized a safer option indeed the expected reward between $t = 13$ and $t = 45$ keeps increasing, thus it is chosen even if is the furthest. The overall length of the trajectory is $1.2\ s$ (60 steps) and the agent spend $0.62\ s$ in the attempt of reaching *cube1*. We consider the overall duration of the trajectory satisfying.

- **Case study** 3 In this case, it was studied if the robot is able to select the correct task even if the person is moving. In a dynamic scenario, we expect that the target to chosen changes over the time based on the current level of interference between the robot and the person.

  The robot starts from *cube5* and is assigned to {*cube0*, *cube1*}. At the same time, the human's right arm moves from *cube3* to *dock1*. The human movement is chosen to obstruct for a certain amount of time one of the robot targets, in particular *cube0* that is the closest target that otherwise would be chosen. Furthermore, the human movement require the robot to perform collision avoidance to reach any of the targets.

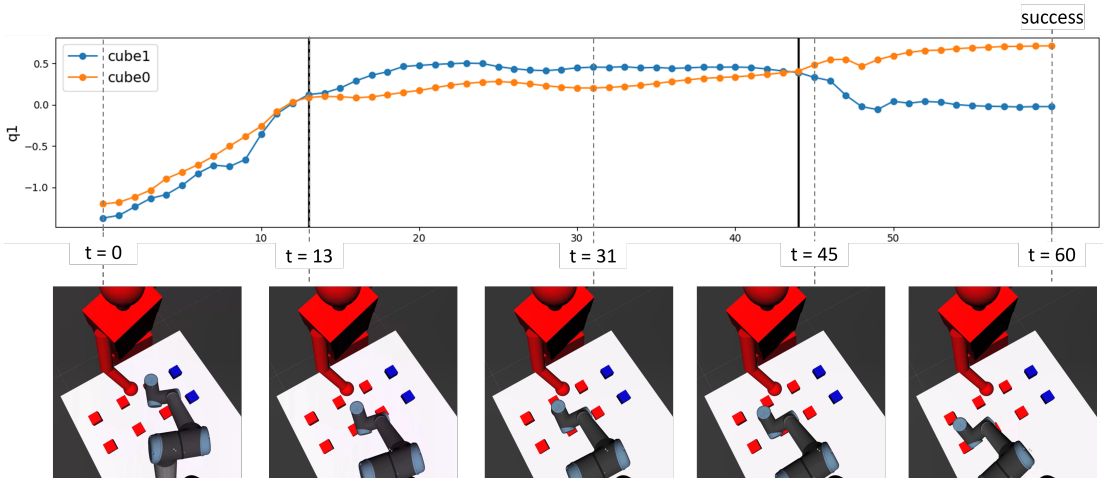



Figure 5.5: Case study 2.

*cube1*.

As shown in Figure 5.6, at $t = 0$, the robot the human's right arm occludes *cube1* that is the closest target, therefore the furthest target, *cubeo*, is chosen. At the same time, the human starts moving toward the robot to reach *dock1*. To avoid the operator, the robot extends its trajectory, and the expected reward decreases. At $t = 10$, the robot has completely avoided the human's arm, which in turn has reached *dock1*, and resumes its trajectory towards *cubeo*. At this point, *cubeo* and *cubeo* are both free and at similar distances from the robot and therefore they both have extremely similar expected rewards. At $t = 20$, has gotten closer enough to both targets that the difference in distance is evident and it chooses the closest one, *cubeo*, until it reaches it.

- **Case study** 4 In this last test, we want to stress that even in dynamic environment, the task choice is based on the human motion and on a trade-off between the target distance and the level of possible interference.

  The robot starts from *cube3* and is assigned to {*cube1*, *docko*}, at the same time, the human's right arm moves from *dock1* to *cubeo*.

  As shown in Figure 5.7, at the beginning, the robot starts moving towards the closest goal *cube1*, indeed, even if partially occluded by the human, it is possible that it will be freed. Once the human start moving toward *cubeo*, it completely obstructs the robot's path to *cube1* and at $t = 10$ the robot changes target, choosing *docko* that is further but free. Soon after the human reaches *cubeo* and he/she does not obstruct *cube1* anymore, however, the agent does not choose again *cubeo* even if it is the closest target. Instead, the robot keeps moving toward *docko*. Indeed, *cubeo* is a hazardous target since it is very close to both the hand and the abdomen of the human, therefore if

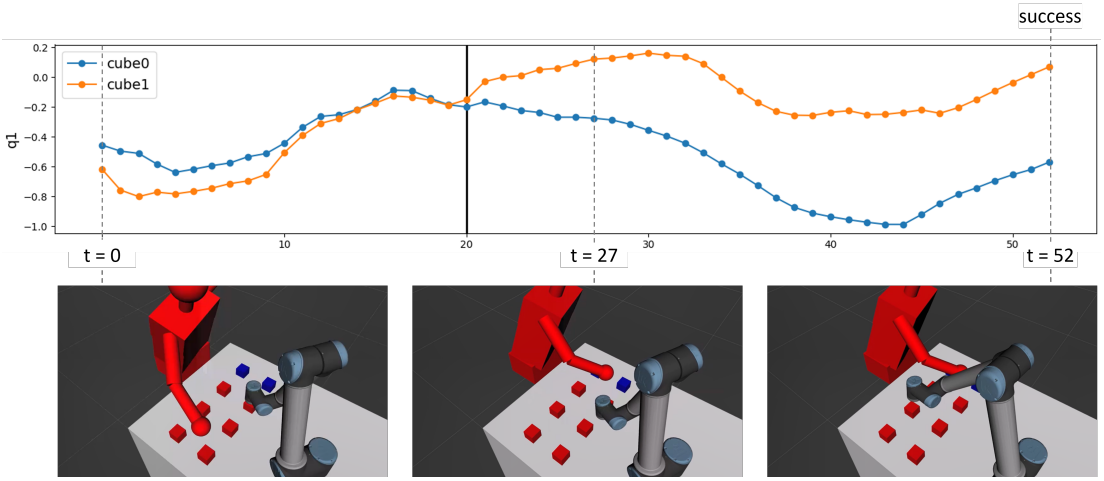



Figure 5.6: Case study 3.

he/she resumes moving, the robot has very little space to maneuver to avoid collisions. Such reasoning is confirmed by the fact that at $t = 21$, the expected reward for *docko* is much higher than the expected reward for *cube1*. In this case study, the robot is able to prefer safety to distance from the target.

## 5.4 Conclusions

In this chapter was proposed a novel method to integrate geometric task planning and motion planning. Grounding on the work of Chapter 4, the motion planning problem is modeled as an MDP, and the expected cumulative discounted reward is used to select the optimal geometric task. The expected cumulative discounted reward is computed using a critic function represented as NN and learned during the agent training phase. Thus, the expected cumulative reward is computed quickly, and the geometric task can be selected at a very high frequency. Subsequently, the approach was applied to a simulated human-robot cooperative scenario. In this scenario, both the human and the robot perform pick and place tasks while sharing the same workspace. The human movements in the simulation are based on an acquired dataset of human trajectories performing the same pick and place tasks. The dataset is converted into a library of probabilistic movement primitives and combined with custom-made data augmentation techniques to prevent the policy from overfitting. The robot was able to learn an effective policy to reach any target from any starting position, achieving an overall success rate of 75%. Finally, we tested the feedback task plane. First, when the agent can choose between multiple random geometric tasks, the success rate increases from 75% to about 90%. Second, we studied a set of specific case studies to analyze how the geometric

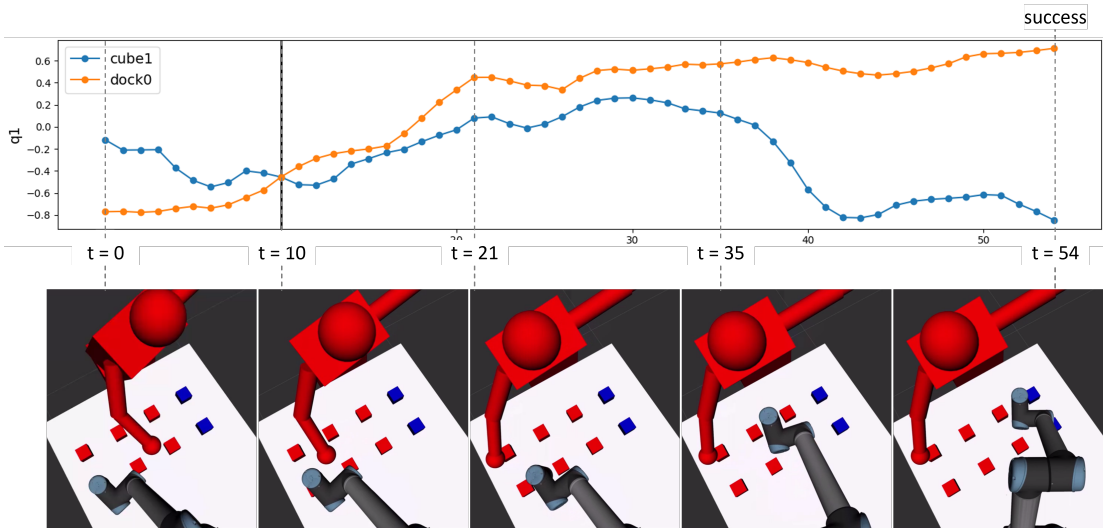



Figure 5.7: Case study 4

tasks were selected. In particular, it was noticed that the agent was able to select tasks that minimized the probability of collisions or, when it was safe, the closest target.

The proposed method has shown promising results in a highly complex scenario; however, it cannot completely avoid collisions. One possible solution could be an extensive fine-tuning of the reward function. Otherwise, similarly to Chapter 4, it is possible to combine the policy with the current state of the art of industrial safety standards like the Speed and Separation Monitoring or the Power and Force Limitation from the ISO TS 15066. Another limitation of the proposed solution is that the policy cannot deal with multiple peoples, and it is tailored to a specific human partner represented by a human model with unique dimensions (e.g., height or arm's length). The first problem could be solved by introducing a formulation of the state of the MDP with multiple humans, like in Chapter 4. Otherwise, it could be computed the action for every human, and the performed action is a weighted average of the actions computed previously. To introduce the possibility of generalizing over human models, the human state could be redefined by introducing the model's dimensions (e.g., links dimensions). Otherwise, the state could include different data sources like, for example, pointclouds.

Finally, in this work, the plan of the symbolic task is assumed given and not modifiable. However, even the symbolic task planner should take into consideration geometric informations. Therefore, we also plan to use the information of the expected cumulative reward from motion planner also to choose among symbolic tasks.

*And where does the newborn go from here? The net is vast, and infinite.*

Ghost in the Shell (1995), Mamoru Oshii

# 6 Conclusions and Future Works

## 6.1 Conclusions

This thesis studied the problems of Task planning, Motion planning and Task and Motion planning in dynamic and non-deterministic environments, with particular attention to human-robot cooperation. All those problems have been solved as sequential decision-making problems described as Markov Decision Processes. Indeed, Markov Decision Processes allow dealing with the two main criticalities of human-robot cooperation, non-determinism, and dynamic environments. On the one hand, non-determinism is considered, since Markov Decision Processes can effectively describe environments characterized by stochastic dynamics. On the other hand, they can easily handle dynamic environments since they compute the optimal action step by step instead of planning over a finite horizon or the complete task like classic motion planning or task planning techniques. Finally, Reinforcement Learning, particularly Deep Reinforcement Learning, was used to solve the studied Markov Decision Processes. Indeed, in the last decade, the combination of Deep Learning with Reinforcement Learning has allowed solving problems that before considered almost impossible, like the game of Go or achieving superhuman performances in most of the games of the Atari benchmark. Reinforcement Learning relies on a trial and error approach to solve the problems; however, in most scenarios, it is not feasible or safe to allow the robot directly to learn the most effective policy in the actual scenario, for example, the robot could collide and damage or even worse injure the human partner in human-robot application. Thus, for each of the studied problems, a simulated environment was developed to train the agent safely.

In Chapter 3, Task Planning in non-deterministic environments was studied. First, a general formulation of the Task Planning problem was proposed, and then it was applied to solve a sorting problem characterized by friendly and adversarial events and probabilistic execution of the planned actions. In particular, the friendly and adversarial events represent the possible outcomes of the interaction of a human partner in the same environment. The

human in such a scenario might be collaborating with the robot at completing the sorting task, or might be executing an unrelated task conflicting with the current task assigned to the robot. The proposed method was tested on multiple applications and transferred to a real-world application. It proved to be able to avoid actions with low success probability and react to unpredicted events without idle time like standard task planners that require computing a new plan.

In Chapter 4, Reinforcement Learning was used to develop a Feedback Motion Planner to allow a human and industrial robot manipulator to share the same workspace. The robot, after extensive training in the simulated environment, was able to avoid collision and at the same time perform a reaching task in almost the totality of the cases without collisions. The formulation allowed the robot to generate continuous and smooth trajectories and guaranteed the trajectory feasibility concerning the joint limits (position, velocity, acceleration, etc.). Thus, it was no longer necessary to verify that the robot trajectories were dynamically feasible. The learned policy was successfully transferred from the simulated environment to the real-world application, highlighting the solution's robustness. Finally, the learned policy was combined with current state o the art industrial safety standards to enhance the system's safety. The proposed solution proved to be more effective than current industry best practices.

Chapter 5 presents an application of combined Task and Motion Planning based on Reinforcement Learning, studying the problem of geometrical reasoning in dynamic and stochastic environments. Indeed, in many applications, choosing the optimal geometric task that solves the desired symbolic task is achieved by planning the solution for every symbolic task and then choosing the one that maximizes the desired metric. However, it is often non-practical in dynamic environments since it requires excessive computation time, and heuristics are often used even if they provide suboptimal solutions. Thus, it was solved the motion planning problem with a feedback motion planner trained via Deep Reinforcement Learning, similarly to Chapter 3. Subsequently, the expected reward, learned during the training phase as a Neural Network, is used to select the geometric task based on the current state of the environment. The proposed approach allows reducing computation time to select the geometric task at the level that is possible to select it at a very high frequency. When combined with the proposed feedback motion planner, it is also possible to change the task on the fly smoothly. Furthermore, a novel formulation of the action based on parametric functions was proposed. The approach was successfully applied to a simulated human-robot cooperation scenario involving a human and a robot performing pick and place activity in proximity. The studied application highlighted the capability of the proposed approach to deal with highly non-deterministic, cluttered, and high dynamics environments.

In conclusion, a methodology to solve task and motion planning in human-robot cooperative applications, based on Deep Reinforcement Learning. The proposed method was able to solve studied problems when the expected human behavior was known, like in Chapter 5, and when it was unknown, like in Chapters 3 and 4. Deep Reinforcement Learning proved

to be a highly flexible method capable of solving a wide variety of problems, even with very different formulations. For example, in Chapters 3 and 5, it was used to solve motion planning problems described by continuous actions and states, while in Chapter 4, it was used to solve task planning problems with discrete actions and states.

## 6.2 Future Works

As future works, it would be interesting to investigate some open questions. First of all, the human behavior (motion or task execution) was considered independent of robot behavior in all the studied scenarios. This choice introduced a conservative approach; for example, in motion planning applications, the robot was also trained to avoid collisions even when the person performed dangerous movements (i.e., high risk of collision). However, when people cooperate, they can adapt to each other to achieve a higher common goal, for example, the overall assembly time in an industrial assembly. For example, humans can adapt movements to minimize the risk of colliding or avoid interfering; similarly, they can choose which task to execute based on the task the other people are performing. A model describing the human-robot interaction at motion and task levels is necessary to introduce the reciprocal adaptation between humans and robots. Nevertheless, the study of human-robot interaction on the motion and task levels is still an open problem. For example, in [44], a method integrates an adaptive human behavior based on defining it as a combination of following their habits and performing the optimal action for the common task. However, the proposed model is arbitrary and does not guarantee to be general. Human-robot in interaction in motion has been studied in various works [3]. Typically, those works analyzed the overall effect on the joint task, described by metrics like the idle time or the time to complete the task, or analyzed the level of trust in the robot based on different movement strategies. However, no analysis on how human movements are affected by robot movement is proposed. Instead, in [86] the same problem is studied in the case of human-human cooperation. However, there is no guarantee that the studied solution can be transferred to a human-robot application.

In this thesis, only model-free were implemented; however, it could be interesting to investigate other reinforcement learning methods like Model-Based or Hierarchical methods. The Model-Based methods typically can reduce the required training time thanks to the introduction of models to predict the dynamics of the environment. Hierarchical Reinforcement Learning promises to solve more complex problems and to reuse the knowledge learned in related problems thanks to hierarchical policies. Indeed, while Reinforcement Learning focuses on learning a policy to solve a single task, Hierarchical Reinforcement Learning focuses on decomposing single tasks in subtasks, each one solved with a specific policy. Related problems can share some subtasks, and therefore, it is possible to reuse the policy for the specific subtask.

To conclude, the proposed approach to task and motion planning is considered promising. On the one hand, we expect that new methods from the field of Human-Robot Interaction will help describe human behavior in human-robot scenarios. The description of an adaptive

human behavior will enable to achieve true collaboration between humans and robots, similarly to human-human collaboration. On the other hand, the reinforcement learning community continuously develops new methods (e.g., algorithms, exploration strategies, etc.) to solve problems faster and more challenging problems. Thus, in the future, we will be able to solve problems that nowadays either require too much training time or are not solvable because too complex.

# References


[1] S. Nikolaidis and J. Shah, "Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy," in *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2013, pp. 33–40.

[2] M. Koppenborg, P. Nickel, B. Naber, A. Lungfiel, and M. Huelke, "Effects of movement speed and predictability in human–robot collaboration," *Human Factors and Ergonomics in Manufacturing & Service Industries*, vol. 27, no. 4, pp. 197–209, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/hfm.20703

[3] A. D. Dragan, S. Bauman, J. Forlizzi, and S. S. Srinivasa, "Effects of robot motion on human-robot collaboration," in *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2015, pp. 51–58.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[5] S. Edelkamp, M. Lahijanian, D. Magazzeni, and E. Plaku, "Integrating temporal reasoning and sampling-based motion planning for multigoal problems with dynamics and time windows," *IEEE Robotics and Automation Letters*, vol. 3, pp. 3473–3480, 2018.

[6] S. Pellegrinelli, A. Orlandini, N. Pedrocchi, A. Umbrico, and T. Tolio, "Motion planning and scheduling for human and industrial-robot collaboration," *CIRP Annals*, vol. 66, no. 1, pp. 1–4, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0007850617300951

[7] M. Faroni, M. Beschi, S. Ghidini, N. Pedrocchi, A. Umbrico, A. Orlandini, and A. Cesta, "A layered control approach to human-aware task and motion planning for human-robot collaboration," in *IEEE Int. Conf. on Robot and Human Inter. Comm.*, Naples (Italy), 2020.

[8] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in

*2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 639–646.

[9] T. Lozano-Perez and L. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3684–3691, 2014.

[10] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki, "Smt-based synthesis of integrated task and motion plans from plan outlines," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 655–662.

[11] Y. Wang, N. T. Dantam, S. Chaudhuri, and L. E. Kavraki, "Task and motion policy synthesis as liveness games," in *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'16. AAAI Press, 2016, p. 536–540.

[12] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, *Semantic Attachments for Domain-Independent Planning Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 99–115. [Online]. Available: https://doi.org/10.1007/978-3-642-25116-0_9

[13] "Pddl - the planning domain definition language," 1997.

[14] C. R. Garrett, T. Lozano-Perez, and L. Kaelbling, "Backward-forward search for manipulation planning," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6366–6373, 2015.

[15] C. Zhang and J. A. Shah, "Co-optimizing task and motion planning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4750–4756.

[16] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *IJCAI*, 2015.

[17] A. Akbari, M. Diab, and J. Rosell, "Contingent task and motion planning under uncertainty for human–robot interactions," *Applied Sciences*, vol. 10, no. 5, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/5/1665

[18] G. Milliez, R. Lallement, M. Fiore, and R. Alami, "Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring," *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 43–50, 2016.

[19] K. Darvish, B. Bruno, E. Simetti, F. Mastrogiovanni, and G. Casalino, "Interleaved online task planning, simulation, task allocation and motion control for flexible human-robot cooperation," *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pp. 58–65, 2018.

[20] K. Darvish, E. Simetti, F. Mastrogiovanni, and G. Casalino, "A hierarchical architecture for human–robot cooperation processes," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 567–586, 2021.

[21] N. Muscettola, "Hsts: Integrating planning and scheduling," 1993.

[22] J. S. Park, C. Park, and D. Manocha, "I-Planner: Intention-Aware Motion Planning Using Learning Based Human Motion Prediction," 2016. [Online]. Available: http://arxiv.org/abs/1608.04837

[23] Y. Cheng, L. Sun, C. Liu, and M. Tomizuka, "Towards efficient human-robot collaboration with robust plan recognition and trajectory prediction," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2602–2609, 2020.

[24] N. Lucci, B. Lacevic, A. M. Zanchettin, and P. Rocco, "Combining speed and separation monitoring with power and force limiting for safe collaborative robotics applications," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6121–6128, 2020.

[25] M. Faroni, M. Beschi, and N. Pedrocchi, "An mpc framework for online motion planning in human-robot collaborative tasks," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1555–1558.

[26] M. Althoff, A. Giusti, S. B. Liu, and A. Pereira, "Effortless creation of safe robots from modules through self-programming and self-verification," *Science Robotics*, vol. 4, no. 31, 2019. [Online]. Available: https://robotics.sciencemag.org/content/4/31/eaaw1924

[27] O. Khatib, *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*. New York, NY: Springer New York, 1990, pp. 396–404. [Online]. Available: https://doi.org/10.1007/978-1-4613-8997-2_29

[28] S. M. H. Rostami, A. K. S, J. Wang, and X. Liu, "Obstacle avoidance of mobile robots using modified artificial potential field algorithm," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, 2019.

[29] Y. bo Chen, G. chen Luo, Y. song Mei, J. qiao Yu, and X. long Su, "Uav path planning using artificial potential field method updated by optimal control theory," *International Journal of Systems Science*, vol. 47, no. 6, pp. 1407–1420, 2016. [Online]. Available: https://doi.org/10.1080/00207721.2014.929191

[30] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002. [Online]. Available: https://doi.org/10.1177/027836490202101200 2

[31] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 338–345.

[32] M. Saveriano and F. Hirt, "Human-aware motion reshaping using dynamical systems," *Pattern Recognition Letters*, vol. 99, pp. 96–104, nov 2017.

[33] A. De Luca and F. Flacco, "Integrated control for phri: Collision avoidance, detection, reaction and collaboration," in *2012 4th IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 2012, pp. 288–295.

[34] J. Mainprice and D. Berenson, "Human-robot collaborative manipulation planning using early prediction of human motion," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 299–306, 2013.

[35] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online pomdp planning for autonomous driving in a crowd," in *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA 2015)*, N. M. Amato, Ed. United States of America: Institute of Electrical and Electronics Engineers Inc., 2015, pp. 454–460. [Online]. Available: https://eprints.qut.edu.au/107521/

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," pp. 1–9, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[37] L. Timothy P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," *International Conference on Learning Representations*, vol. 40, no. 12, pp. 1059–1062, 2016.

[38] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 1343–1350, 2017.

[39] M. Everett, Y. F. Chen, and J. P. How, "Motion Planning among Dynamic, Decision-Making Agents with Deep Reinforcement Learning," *IEEE International Conference on Intelligent Robots and Systems*, no. iii, pp. 3052–3059, 2018.

[40] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," *Proceedings*

- *IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 6015–6022, 2019.

[41] M. El-Shamouty, X. Wu, S. Yang, M. Albus, and M. F. Huber, "Towards safe human-robot collaboration using deep reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4899–4905.

[42] B. Sangiovanni, A. Rendiniello, G. P. Incremona, A. Ferrara, and M. Piastra, "Deep Reinforcement Learning for Collision Avoidance of Robotic Manipulators," *2018 European Control Conference, ECC 2018*, no. December, pp. 2063–2068, 2018.

[43] C. J. C. H. Watkins and P. Dayan, "Q-learning," in *Machine Learning*, 1992, pp. 279–292.

[44] H. S. Koppula, A. Jain, and A. Saxena, *Anticipatory Planning for Human-Robot Teams*. Cham: Springer International Publishing, 2016, pp. 453–470. [Online]. Available: https://doi.org/10.1007/978-3-319-23778-7_30

[45] S. Pellegrinelli, H. Admoni, S. Javdani, and S. Srinivasa, "Human-robot shared workspace collaboration via hindsight optimization," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem, pp. 831–838, 2016.

[46] M. Cramer, K. Kellens, and E. Demeester, "Probabilistic decision model for adaptive task planning in human-robot collaborative assembly based on designer and operator intents," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7325–7332, 2021.

[47] R. Bellman, "On the theory of dynamic programming." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 38 8, pp. 716–9, 1952.

[48] R. S. Sutton, "Learning to predict by the methods of temporal differences," in *MACHINE LEARNING*. Kluwer Academic Publishers, 1988, pp. 9–44.

[49] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, England, 1994, vol. 37.

[50] R. G. A and N. M., "On-line q-learning using connection systems," 1994.

[51] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[52] H. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23. Curran Associates, Inc., 2010. [Online]. Available: https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf

[53] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[54] G. Tesauro, "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 03 1994. [Online]. Available: https://doi.org/10.1162/neco.1994.6.2.215

[55] L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, USA, 1992, uMI Order No. GAX93-22750.

[56] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015, cite arxiv:1509.06461Comment: AAAI 2016. [Online]. Available: http://arxiv.org/abs/1509.06461

[57] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, cite arxiv:1511.05952Comment: Published at ICLR 2016. [Online]. Available: http://arxiv.org/abs/1511.05952

[58] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1995–2003. [Online]. Available: https://proceedings.mlr.press/v48/wangf16.html

[59] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, p. 1057–1063.

[60] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, p. I–387–I–395.

[61] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[62] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1587–1596. [Online]. Available: https://proceedings.mlr.press/v80/fujimoto18a.html

[63] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[64] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," 2018. [Online]. Available: https://github.com/hill-a/stable-baselines

[65] H. V. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," 2015.

[66] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," no. 9, 2015. [Online]. Available: http://arxiv.org/abs/1511.06581

[67] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," pp. 1–21, 2016.

[68] P. Lasota, S. Nikolaidis, and J. Shah, "Developing an adaptive robotic assistant for close proximity human-robot collaboration in space," *AIAA Infotech at Aerospace (I at A) Conference*, pp. 1–8, 2013.

[69] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19.

[70] R. Elbasiony and W. Gomaa, "Humanoids skill learning based on real-time human motion imitation using Kinect," *Intelligent Service Robotics*, vol. 11, no. 2, pp. 149–169, 2018. [Online]. Available: https://doi.org/10.1007/s11370-018-0247-z

[71] OpenAI, "Openai five," https://blog.openai.com/openai-five/, 2018.

[72] R. Y. Tsai, R. K. Lenz *et al.*, "A new technique for fully autonomous and efficient 3 d robotics hand/eye calibration," *IEEE Transactions on robotics and automation*, vol. 5, no. 3, pp. 345–358, 1989.

[73] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011.

[74] A. Malaguti, M. Carraro, M. Guidolin, L. Tagliapietra, E. Menegatti, and S. Ghidoni, "Real-time Tracking-by-Detection of Human Motion in RGB-D Camera Networks," in *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2019.

[75] C. Tonola, M. Faroni, N. Pedrocchi, and M. Beschi, "Anytime informed path replanning and optimization for human-robot collaboration," in *Proceedings of the IEEE International Conference on Robot and Human Interactive Communication*, Vancouver (Canada), 2021.

[76] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846 – 894, 2011.

[77] M. Guidolin, E. Menegatti, M. Reggiani, and L. Tagliapietra, "A ROS driver for xsens wireless inertial measurement unit systems," in *2021 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2021, in press.

[78] S. L. Delp, F. C. Anderson, A. S. Arnold, P. Loan, A. Habib, C. T. John, E. Guendelman, and D. G. Thelen, "Opensim: Open-source software to create and analyze dynamic simulations of movement," *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 11, pp. 1940–1950, 2007.

[79] A. Seth, J. L. Hicks, T. K. Uchida, A. Habib, C. L. Dembia, J. J. Dunne, C. F. Ong, M. S. DeMers, A. Rajagopal, M. Millard, S. R. Hamner, E. M. Arnold, J. R. Yong, S. K. Lakshmikanth, M. A. Sherman, J. P. Ku, and S. L. Delp, "Opensim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement," *PLOS Computational Biology*, vol. 14, no. 7, pp. 1–20, 07 2018. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1006223

[80] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[81] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, pp. 2149–2154 vol.3.

[82] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[83] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Rönnau, and R. Dillmann, "Unified gpu voxel collision detection for mobile manipulation planning," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4154–4160, 2014.

[84] S. Gomez-Gonzalez, G. Neumann, B. Scholkopf, and J. Peters, "Adaptation and Robust Learning of Probabilistic Movement Primitives," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 366–379, 2017.

[85] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ByBAl2eAZ

[86] J. Mainprice, R. Hayne, and D. Berenson, "Predicting human reaching motion in collaborative tasks using inverse optimal control and iterative re-planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 885–892.