Continuous Control Actions Learning and Adaptation for Robotic Manipulation through Reinforcement Learning

Article *in* Autonomous Robots · March 2022

DOI: 10.1007/s10514-022-10034-z

CITATIONS
10

READS
1,336

4 authors, including:

Asad Ali Shahid
Dalle Molle Institute for Artificial Intelligence
12 PUBLICATIONS **70** CITATIONS
SEE PROFILE

Dario Piga
176 PUBLICATIONS **2,160** CITATIONS
SEE PROFILE

Loris Roveda
Dalle Molle Institute for Artificial Intelligence
95 PUBLICATIONS **1,029** CITATIONS
SEE PROFILE

Noname manuscript No.
(will be inserted by the editor)

Continuous Control Actions Learning and Adaptation for Robotic Manipulation through Reinforcement Learning

Asad Ali Shahid[1,2] · **Dario Piga**[1] · **Francesco Braghin**[2] · **Loris Roveda**[1,*]

Received: date / Accepted: date

Abstract This paper presents a learning-based method that uses simulation data to learn an object manipulation task using two model-free reinforcement learning (RL) algorithms. The learning performance is compared across on-policy and off-policy algorithms: Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). In order to accelerate the learning process, the fine-tuning procedure is proposed that demonstrates the continuous adaptation of on-policy RL to new environments, allowing the learned policy to adapt and execute the (partially) modified task. A dense reward function is designed for the task to enable an efficient learning of the agent. A grasping task involving a Franka Emika Panda manipulator is considered as the reference task to be learned. The learned control policy is demonstrated to be generalizable across multiple object geometries and initial robot/parts configurations. The approach is finally tested on a real Franka Emika Panda robot, showing the possibility to transfer the learned behavior from simulation. Experimental results show 100% of successful grasping tasks, making the proposed approach applicable to real applications.

Keywords Reinforcement Learning · Continuous Control · Robotic Grasping · Policy Optimization

The work has been developed within the project ASSASSINN, funded from H2020 CleanSky 2 under grant agreement n. 886977.

Asad Ali Shahid
E-mail: asadali.shahid@idsia.ch

Dario Piga
E-mail: dario.piga@idsia.ch

Francesco Braghin
E-mail: francesco.braghin@polimi.it

Loris Roveda
E-mail: loris.roveda@idsia.ch
[1] Istituto Dalle Molle di studi sull'Intelligenza Artificiale (IDSIA), Scuola Universitaria Professionale della Svizzera Italiana (SUPSI), Università della Svizzera italiana (USI), via Cantonale 2C- 6928, Manno, Switzerland

[2] Politecnico di Milano, Department of Mechanical Engineering, Milano, Italy

1 Introduction

1.1 Context

Robots are being increasingly used in different applications, where the operating scene is not anymore fixed, pre-defined and well-known, and such need is demanded by all application areas, such as autonomous driving, industrial robots, assistive robots Seif and Hu (2016); Tsarouchi et al. (2016); Kearney et al. (2018). Focusing on the industrial field, the new paradigm of Industry 4.0 Lasi et al. (2014) requires robots to achieve smart behaviors in dynamic and flexible environments. In fact, production plants are going to be transformed into digital work-places, interconnecting machine systems, humans, products, etc., to re-configure the production on the basis of the current needs. This new paradigm creates the necessity for the robot manipulator to self-adapt its behavior for the assigned ever-changing tasks. It is, therefore, not anymore possible to pre-program all the possible scenarios. Thus, intelligence has to be embedded into the robotic system, to sense and analyze the working-scene and to take decisions for the correct task execution, facing safety, performance and production issues. Artificial intelligence and machine learning approaches find, therefore, a huge space in the robotics domain to achieve such ambitious goals Rajan and Saffiotti (2017); Van Roy et al. (2020). The main contribution of the paper is to propose an efficient learning

approach for the robot to learn a target task in a simulation environment, and transferring such knowledge to the real robot for the real task execution. In this way, the robot is able to safely learn and execute a specific application, being able to generalize and rapidly adapt its behavior to new tasks. In the following, the state of the art related to the proposed context, together with the objectives of the current study are detailed.

1.2 Related works

In literature, there are mainly two learning approaches for robots to accomplish tasks: i) Learning from Demonstration (LfD) or Imitation Learning, and ii) Reinforcement Learning (RL). The main distinction between these approaches lies around the fact that whether or not human demonstrations in any form are exploited for learning behaviors. LfD aims to learn tasks based on demonstrated trajectories, whereas RL discovers the optimal behavior for a task through trial and error, employing a reward function that encourages desired behavior. Another approach involves the combination of both approaches to learn a task from human provided demonstrations and improve the behavior with RL. In Silver et al. (2016), an algorithm was able to master the game of Go by first learning the competitive Go policy from an expert's demonstrations, and then improving that policy through reinforcement learning. Considering i), LfD methods have been used to learn control commands for a robot from raw sensory signals Rahmatizadeh et al. (2018), extracting a reward function from demonstrations Boularias et al. (2011), also known as inverse reinforcement learning. Considering ii), RL methods have been successfully applied in a wide variety of contexts, ranging from playing games Mnih et al. (2015), natural language processing Cui et al. (2020), to robotic locomotion Heess et al. (2017). In robotics, RL has enabled robots to learn tasks, such as playing table tennis Mülling et al. (2013), flipping a pancake Kormushev et al. (2010), aerobatic helicopter maneuvers Abbeel et al. (2007), and general manipulation skills Kalakrishnan et al. (2011). Earlier RL methods used low dimensional representation techniques, such as movement primitives, to solve control problems Kormushev et al. (2010); Pastor et al. (2011). Recently, RL has been used by exploiting the function approximation power of deep neural networks Lillicrap et al. (2015); Schulman et al. (2015b); Duan et al. (2016). Such techniques, named as deep reinforcement learning, leverage the capabilities of neural networks in learning representations from high dimensional input data, thus making it possible to learn control in end-to-end manner.

Recent works have seen emerging use of deep reinforcement learning techniques for robot manipulation Gu et al. (2017); Rajeswaran et al. (2017). Notably, guided policy search, a model-based approach, is used to train *visuomotor* (coordination between vision and control) policies directly from raw images Levine et al. (2016). While this method achieves impressive results on real world manipulation tasks, significant human involvement is required for data collection process Levine et al. (2015). Model-based RL approaches generally learn dynamic models from data that generate trajectories to subsequently aid policy learning Deisenroth et al. (2013). They aim to learn either a smooth global model Deisenroth and Rasmussen (2011) or local time varying linear dynamics models Levine and Abbeel (2014); Levine et al. (2015). In both cases, these methods struggle to learn policies in tasks that have inherently discontinuous dynamics and rewards as demonstrated by Chebotar et al. (2017). Another idea is to use large-scale data collection to learn control in a self-supervised manner Levine et al. (2018), or through RL Kalashnikov et al. (2018). However, such kind of data collection is not economically feasible, requiring multiple robots and months to collect. Some other recent works exploit simulation data to learn RL-based visuomotor policies Quillen et al. (2018); Martín-Martín et al. (2019); Gu et al. (2017). Due to unrealistic rendering capabilities of simulation engines, visuomotor policies learned in simulation demand additional sim-to-real techniques Tobin et al. (2017) to perform in real world settings. However, state space without a vision component serves as sufficient representation of common industrial tasks where part locations are generally pre-defined. Therefore, RL policies trained on kinematic state information alleviate the problem of transfer to the real world in such conditions, since the simulated environment can roughly match the real robot kinematics and part positions.

Besides RL, other successful approaches for grasping involve modular pipelines and open-loop control of a planned grasp, *e.g.*, first detecting grasping candidates Lenz et al. (2015) to predict the best grasping contact points and then using motion planning to reach the location ten Pas et al. (2017); Mahler et al. (2017). These approaches, however require precise calibration between sensing and control Morrison et al. (2018). Some other techniques focus on closed-loop grasp execution through visual servoing models trained using synthetic data in simulation Viereck et al. (2017). However, these approaches constrain the state and/or action space to make the learning problem tractable resulting in policies that learn simple strategies and fail to generalize to significant variations of the environment without extensive retraining. Another idea to speed-up

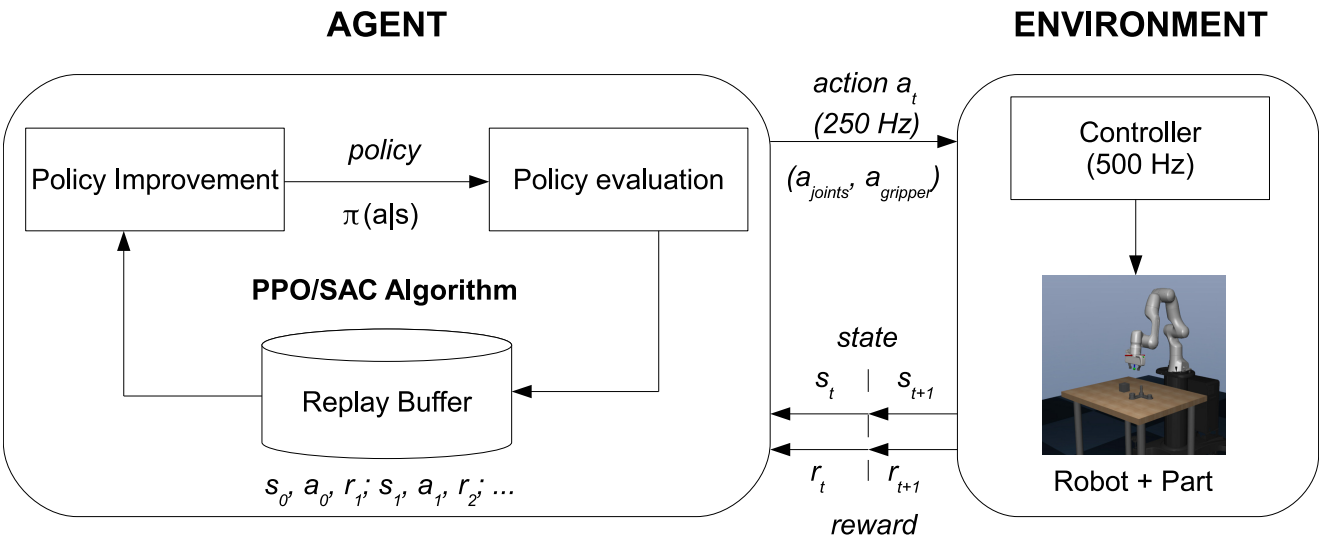


Fig. 1: The proposed learning schema exploiting PPO/SAC algorithm.

the learning process and handle novel configurations is to include human provided demonstrations Song et al. (2020) but this requires specialized hardware to collect demonstrations. In contrast, RL provides a general framework for dynamic closed loop control with long horizon reasoning during the task execution. This allows RL methods to autonomously acquire generalizable strategies for completing the task and adapt to different environment conditions in very few iterations.

The main challenge with the current RL methods is that typically each task is learned in isolation from scratch resulting in a poor sample efficiency, while, the common approach in other machine learning domains is to fine-tune the model previously trained on a similar task resulting in a much better sample efficiency Kornblith et al. (2019). Adapting policies in robotics remains largely unexplored area. Some recent works have studied the problem of adapting RL policies in robotics focusing on off-policy settings Smith et al. (2021); Julian et al. (2020). Unlike these works, this paper aims to address the problem of adapting policies in on-policy RL setting based purely on simulated data.

1.3 Paper contribution

In this paper, model-free RL algorithms are used that leverage simulation data and proprioceptive state information to learn robot manipulation tasks. An on-policy RL algorithm —Proximal Policy Optimization (PPO)— is elected first to train the robot controller. The task is then also learned using an off-policy RL algorithm —Soft Actor-Critic (SAC)— to compare the learning performance. In both cases, the control actions (*i.e.*, joint command signals and gripper command signal) are learned in continuous space on the basis of a designed reward function. To improve the sample efficiency in (partially) new task settings, a fine-tuning procedure is proposed to initialize the target task policy with the learned base policy, thus, reducing the required number of episodes, *i.e.*, speeding up the training. In particular, the base policy is first pre-trained by taking into account the success of the task. The base policy is then used to initialize the new task's policy that includes additional performance objectives. The main objectives of the proposed reward function are: success in grasping and lifting of the target object, robot's redundancy management to avoid reaching the joint limits, avoiding collisions with the obstacles in the scene and smoothing control actions for an implementation of the learned controller on a real robot. The learned behavior is then transferred to the real robot, to execute the target task.

In summary, the main contributions of this work are:

- a fine-tuning method for the learned skill adaptation to (partially) modified task settings;
- simultaneous learning of a source task along with the sequence of secondary sub-tasks;
- demonstrating that a robot with non-constrained action space can learn a target task in simulation and transfer to reality making the proposed approach more general w.r.t. other works that constrain the robot's action space.

The proposed approach is evaluated on a grasping task involving a Franka Emika Panda manipulator. The

task requires the robot to reach the part (nominal part: cube of 6 cm length), grasp it, and lift it off the contact surface. The proposed approach is demonstrated to be generalizable across multiple object geometries and initial robot/parts configurations. Despite the training performed only on a single cube with no prior knowledge, the learning is able to generalize across different geometric shapes and sizes, minor changes in the position of the object to be manipulated, and across multiple initial configurations of the robot. The proposed approach is finally tested on a real Franka Emika Panda robot, showing the possibility to transfer the learned behavior from simulation (zero-shot transfer). Experimental results show 100% of successful grasping tasks, making the proposed approach applicable to real applications. The developed software and a video for the evaluation of the proposed approach (in which both learning in simulation and knowledge transfer to the real robot are shown) are available at the GitHub repository Shahid (2020).

2 Problem formulation

The primary objective of this contribution is twofold: 1) to autonomously learn the robot control actions for new tasks execution without requiring any real data, and 2) to improve the training efficiency in modified task conditions by re-using the learned policy. The acquired knowledge can then be transferred to the real robot, to execute the target task and to quickly adapt the learned behavior when operating conditions change. To address the first objective, large data is generated in simulation, learning to map observed states directly to commanded joint velocities. As real-world robotics problems are best represented as continuous state and action spaces, the learning of the control actions is performed in continuous spaces. Two model-free reinforcement learning algorithms are employed and learning performance is compared. On-policy algorithm, Proximal Policy Optimization (PPO), and off-policy algorithm, Soft Actor-Critic (SAC), are proposed to train the robot controller. Both these algorithms aim to learn the stochastic policies. To address the second objective, the learned base task is modified by including additional performance objectives and the target task policy is initialized with the pre-trained policy during the fine-tuning phase. The reward function is designed to guide the learning, taking into account the task success/failure, performance, and safety issues. The block-diagram describing the proposed approaches is shown in Figure 1, and highlighting the implemented PPO and SAC algorithms and their connection with the simulation environment for learning purposes.

3 Methodology

3.1 Preliminaries

In this paper, a RL framework is considered, where an agent interacts with the environment in discrete time steps. The RL problem is modeled as discrete time continuous Markov decision process (MDP) Sutton and Barto (2018), consisting of $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is continuous state space, $\mathcal{A}$ is continuous action space, $\mathcal{P}$ is environment transition dynamics defining the state transition probabilities for a given action $p(s_{t+1}|s_t, a_t)$, $\mathcal{R}$ is a reward function $\mathcal{R}(s_t, a_t) = r_t \in \mathbf{R}$, $\gamma \in [0, 1]$ is a discount factor determining the importance of future rewards relative to immediate reward. A policy π is a stochastic map from states to distribution over actions $\mathcal{S} \rightarrow \mathcal{A}$. The objective of an agent is to learn a policy $\pi(a_t|s_t)$ that maximizes expected returns $\mathbf{E}[\sum_{t=0}^{H} \gamma r_t]$, where H is the target horizon. In deep reinforcement learning, the policy is represented by a non-linear function approximator, a neural network parameterized by θ. The main goal then reduces to optimizing θ, such that the optimal behavior is achieved.

3.2 Task description

The aim of this paper is to implement the approach proposed in Figure 1 to autonomously learn a grasping task. The task consists of a robot interacting with a cube of nominal size 6 cm placed on a table. The goal of the robot is to successfully position its grip site around the cube, grasp it, and then finally lift it off the contact surface. In order to evaluate the proposed approach on a grasping task learning, Franka Emika Panda, a 7-DoF torque-controlled robot, is used as a robotic platform.

3.3 Baseline reward shaping

In order to guide policy learning and provide frequent feedback to the agent on appropriate behaviors, the proposed approach exploits the dense reward function. The task is split into two main phases of reaching and lifting, with distinct reward for each phase. The robot learns both of these phases simultaneously (*i.e.*, in a single iteration of a simulated task).

Considering the reaching phase, the reward given at time step t is composed of three contributions:

$$r_{t,reach} = w_d r_d + w_v r_v + w_g r_g, \tag{1}$$

where r_d is the distance reward computed using relative position of gripper site and object (the gripper site is

the ref. frame located between the fingers of the gripper to define the gripping area), r_v is the velocity reward computed using end-effector velocity vector, and r_g is gripper open reward depending upon the action of gripper. In particular, r_v and r_g encourage the robot end-effector to approach cube with small velocity and open gripper. All the three contributions r_d, r_v, r_g are weighted with respective weights $w_d = 0.6$, $w_v = 0.3$, $w_g = 0.1$ and defined as:

$$
\begin{aligned}
r_d &= 1 - tanh(\mid \mathbf{p}_{grip} - \mathbf{p}_{cube} \mid), \\
r_v &= \begin{cases} 1 - tanh\left(\frac{\sum_{i=1}^{n} \mid \mathbf{v}_{ee}(i) \mid}{n}\right) & if \;\; r_d < 8 \text{ cm} \\ 0 & otherwise, \end{cases} \\
r_g &= \begin{cases} \mid a_{grip} \mid & if \;\; a_{grip} < 0 \\ 0 & otherwise, \end{cases}
\end{aligned} \tag{2}
$$

where $\mathbf{p}_{grip}$ and $\mathbf{p}_{cube}$ denote the position vectors of gripper site and cube in world reference frame each containing 3 elements specifying the spatial positions, $\mathbf{v}_{ee}$ is the end effector velocity vector containing both linear and angular components with n equal to 6 (*i.e.*, the number of Cartesian degrees of freedom), i selects the i^{th} Cartesian degree of freedom, and a_{grip} specifies the action of gripper with negative values indicating that gripper is opening.

When the robot's grip site is in close proximity to the cube, the lifting phase is initiated. The distance threshold for switching is set to 2.5 cm. This is because nominal cube measures as 6 cm and the robot should be able to grasp the cube when it's grip site is within cube's boundary. Considering the lifting phase, the reward is computed considering five contributions:

$$
r_{t,lift} = r_d + r_v + w_g r_g + r_c + r_s, \tag{3}
$$

where r_d and r_v are the same as in (1). r_g is reversed now to provide reward for closing the gripper and it is weighted by $w_g = 0.1$. In addition, r_c is given if both fingers of the gripper are in contact with cube. r_s rewards for successful completion of the task. Success is determined by examining if the gripper holds the cube above certain height. r_g, r_c, r_s in lifting case are defined as:

$$
\begin{aligned}
r_g &= \begin{cases} \mid a_{grip} \mid & if \;\; a_{grip} > 0 \\ 0 & otherwise, \end{cases} \\
r_c &= \begin{cases} 0.5 & if \;\; fingers \;\; in \;\; contact \\ 0 & otherwise, \end{cases} \\
r_s &= \begin{cases} 2 & if \;\; z_{cube} > 0.1 + z_{cube_{initial}} \\ 0 & otherwise. \end{cases}
\end{aligned} \tag{4}
$$

The final reward r_t is then computed as the accumulated sum of $r_{t,reach}$ in (1) and $r_{t,lift}$ in (3).

3.4 Embedded performance specifications for the learned task adaptation

Because the RL methods are data intensive, the policy is first trained taking into account only the success of the task. In the second phase, the pre-trained grasping policy is further fine-tuned considering additional performance objectives in a modified environment. For fine-tuning, the target task policy is initialized with the parameters of the base policy. In order to test the adaptation capability of control policy to new modified task conditions, the following performance specifications are successively included into the reward r_t in re-training conditions:

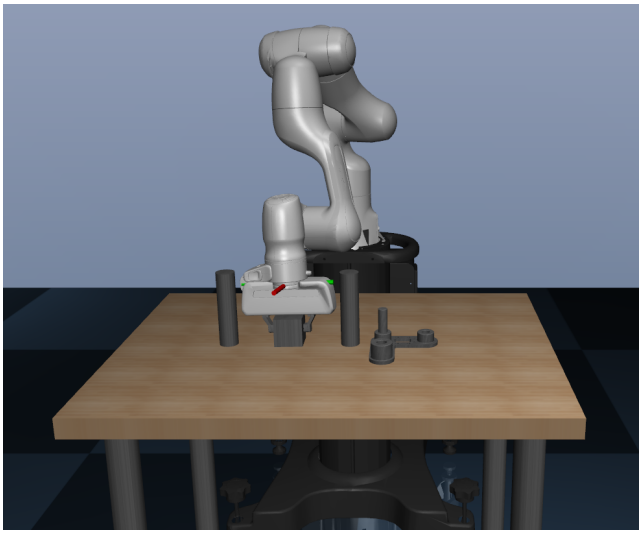


Fig. 2: Franka Emika Panda Robot grasping the cube while avoiding collision with the obstacles in the modified scene.

– robot redundancy management: a penalty p_{red} is added to the reward function r_t in the case that joint positions come close to the joint limits. In such a way, the robot needs to manage its redundancy in order to avoid (if possible) reaching the joint limits. p_{red} is defined as:

$$p_{red} = \sum_{i=1}^{n_j} \mathbf{p}_{red}(i)$$
$$\mathbf{p}_{red}(i) = \begin{cases} -1 & if \mid \mathbf{e}_q(i) \mid < tol \mid \mathbf{q}_{lim}(i) \mid \\ 0 & otherwise, \end{cases} \quad (5)$$

where $\mathbf{e}_q(i) = \mathbf{q}_{lim}(i) - \mathbf{q}(i)$, $n_j = 7$ is the number of robot joints, $\mathbf{q}_{lim}$ is the joint limits vector, i selects the i^{th} joint, and tol specifies the tolerance and it is set to 15%;

– smoothing the control actions: a penalty p_{sm} is added to the reward function r_t in the case that joint accelerations exceed a specified threshold. In such a way, the learned controller is capable to achieve smooth control signals avoiding jerky behaviors. p_{sm} is defined as:

$$p_{sm} = -tanh\left(\frac{\sum_{i=1}^{n_j} \mid \ddot{\mathbf{q}}(i) \mid}{n_j}\right), \quad (6)$$

where $\ddot{\mathbf{q}}$ is the joint acceleration vector and i selects the i^{th} joint;

– avoiding collisions with obstacles in the operating scene: a penalty p_{oa} is added to the reward function r_t in the case that the robot collides with obstacles. In such a way, the learned behavior is capable to avoid collisions in the operating scene while reaching its target. p_{oa} is defined as:

$$p_{oa} = \begin{cases} -5 & if\ collision\ occurs \\ 0 & otherwise. \end{cases} \quad (7)$$

In this case, the contact collision is checked between the robot, the cube and the cylindrical objects. If the robot comes in contact with the cylinders, the penalty is given. The agent is also penalized if the robot lifts the cube and then touches the cylinder with the cube. In this way, the agent needs to learn the appropriate actions to grasp and lift the cube while avoiding the contact with the cylinders simultaneously. This performance objective is more difficult because of the narrow space between the two cylinders, leaving just enough room for the hand to position itself and lift the cube without coming in contact with the cylinders. The modified scene is shown in Figure 2.

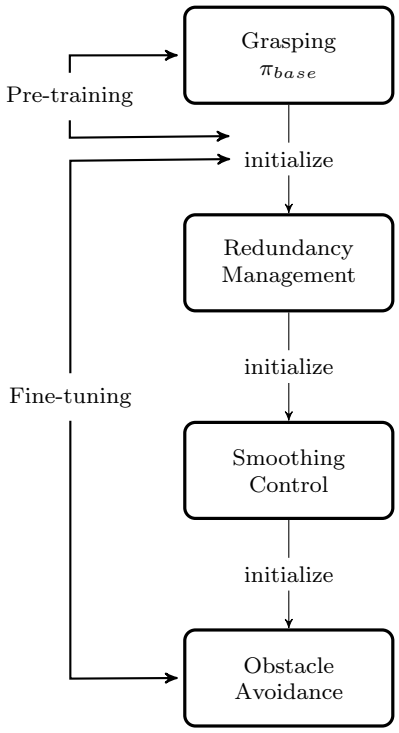


Fig. 3: Schematic of Adaptation framework. The base policy π_{base} is pre-trained on a grasping task and then fine-tuned on sequence of modified tasks by successively introducing new performance objectives

It is important to note that the adaptation process is performed in a continual-learning manner, as it is shown in Figure 3. First, the pre-trained base grasping policy is fine-tuned to satisfy the redundancy management objective, then this adapted policy is fine-tuned on smoothing control actions objective not the previous base policy. Finally, the newly adapted policy is further fine-tuned on collision avoidance objective. The final policy successfully completes the target task while satisfying all 3 performance objectives.

4 Implementation settings

4.1 Learning environment

The proposed learning environment is developed based on Fan et al. (2018), exploiting MuJoCo physics engine Todorov et al. (2012) to simulate the physical system. MuJoCo enables fast and accurate simulation with contacts. The simulation environment is shown in Figure 4.

4.1.1 States and Actions

Both the state space and the action space used in the proposed evaluation are continuous. State of the system consists of two main input modalities: robot proprioception information and object information. The proprioceptive data contains joint positions, velocities, gripper joint position and end-effector pose and velocity, thus making a 36-dimensional vector. Object information is 10-dimensional and includes cube's pose and a relative position vector of cube and grip site. The state of the environment is:

$$S = \left[S_{prop}\ S_{obj}\right] \in \mathbf{R}^{46}$$

$$S_{prop} = \left[\mathbf{q}\ \dot{\mathbf{q}}\ \mathbf{q_{grip}}\ \mathbf{p_{ee}}\ q_{ee}\ \mathbf{v_{ee}}\ \omega_{\mathbf{ee}}\right] \in \mathbf{R}^{36} \tag{8}$$

$$S_{obj} = \left[\mathbf{p_{part}}\ q_{part}\ \mathbf{p_{rel}}\right] \in \mathbf{R}^{10}$$

Actions are 8-dimensional and correspond to reference joint velocities and gripper position. The robot is considered to be equipped with torque control compensating for the manipulator dynamics Siciliano and Villani (2000):

$$\boldsymbol{\tau} = \mathbf{B}(\mathbf{q})\boldsymbol{\tau}_{learn} + \boldsymbol{\tau}_f(\dot{\mathbf{q}}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}), \tag{9}$$

where $\mathbf{B}(\mathbf{q})$ is the robot inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the robot Coriolis vector, $\mathbf{g}(\mathbf{q})$ is the robot gravitational vector, $\boldsymbol{\tau}_f(\dot{\mathbf{q}})$ is the robot joint friction vector, $\mathbf{q}$ is the current joint position vector, and $\boldsymbol{\tau}$ is the control torque vector. $\boldsymbol{\tau}_{learn}$ is the learned task-related control torque vector mapping the reference joint velocities to joint torques for the target task execution:

$$\boldsymbol{\tau}_{learn} = \mathbf{k}_v(\dot{\mathbf{q}}^d - \dot{\mathbf{q}}), \tag{10}$$

where $\dot{\mathbf{q}}^d$ and $\dot{\mathbf{q}}$ are the reference (*i.e.*, the objective of the learning) and current joint velocities respectively, and

$$\mathbf{k}_v = diag(8, 7, 6, 4, 2, 0.5, 0.1)$$

is a diagonal matrix of fixed proportional gains for joint trajectory-tracking purposes. Such control gains have to be tuned to allow the robot to track the reference joint velocities $\dot{\mathbf{q}}^d$. The here defined values have been selected on the basis of Martín-Martín et al. (2019), resulting in accurate trajectory tracking performance. If needed, such control gains can be either learned or experimentally tuned to improve trajectory tracking performance Roveda et al. (2020).

4.1.2 Control parameters

An important parameter that affects the trade-off between computational time and accuracy of the simulated task in MuJoCo is the simulation time step. The default value of 2ms is used, ensuring a good trade-off between simulation accuracy and stability. The controller of the Franka Robot operates at 500 Hz. Different values of the policy frequency (the rate at which policy outputs actions) have been tested (500 Hz, 250 Hz, 100 Hz), resulting in the best setting of 250 Hz. Since the robot controller generates commands at higher frequency than the policy, a linear interpolation is performed between each successive policy action.

5 Evaluation

In this Section, the PPO algorithm for the training of the target task is evaluated. Results related to training performance considering the baseline reward, to fine-tuning with an additional performance specifications, and to knowledge transfer from simulation to real application are given.

5.1 Training results - baseline reward function

The proposed approach is applied to the grasping and lifting task of the cube above a certain height, considering the baseline reward function. The policy is trained for a total of 10 million time steps where each episode lasts for 600 steps, giving an agent approximately 2.5 seconds to accomplish the task. At the beginning of each episode, initial configuration of the robot and the cube is reset to the fixed position. The simulation has

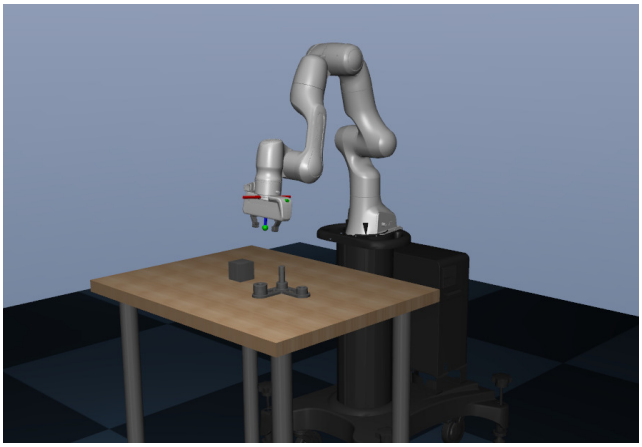


Fig. 4: Franka Emika Panda Robot in MuJoCo simulation environment.

Fig. 5: PPO training results. The plot depicts the progression of mean accumulated reward for episodes.

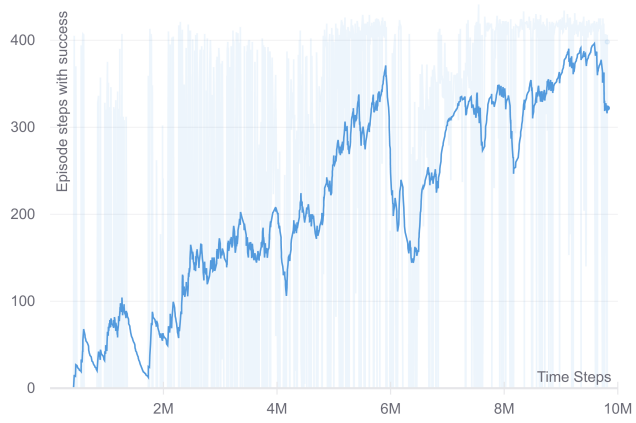


Fig. 6: PPO training results. The plot depicts the number of episode steps with success.

been roughly sped-up 2000 x faster than real-time in order to train faster (requiring approximately 4 and a half hours of wall-clock time). The results of training are shown in Figure 5 and Figure 6 (depicting the evolution of the mean accumulated reward and the evolution of the mean episode steps with success, respectively during the training), suggesting that the agent has learned to perform the task successfully after 5 million steps.

The generalisation capabilities and robustness of the learned neural network policy have been tested, considering the nominal cube and different geometries to be grasped:

- nominal cube with size 6 cm;
- smaller cube with size 4 cm;
- cylinder with size 3 cm radius and 3 cm height;
- screw-driver.

10 test trials have been run and results are summarised w.r.t. successful completion of the task in Table 1. The proposed model shows the success rate of 100% in all four cases. In the last two tests, the policy's ability to grasp new shapes is assessed by replacing the cube with a cylinder and a screw-driver placed nearly at the cube's original position.

Table 2 shows the results in which the original position of the robot and/or of the cube have been changed. In the first case, a small random noise is added to the initial robot joint positions at the beginning of each episode. The robot can still grasp and lift the cube in most cases, while 2 failures have been registered due to the impossibility to position the gripper around the cube. For the second case, the nominal position of the cube has been modified by ± 8 cm in order to test the policy's robustness to variation in part's position. This test is required to verify the robustness of grasping policies in the real manipulation tasks, where the position of the part is identified by a vision system (exploiting object localization algorithms) with some uncertainty in the measurement (and in the computation related to the used algorithms). Generally the vision systems have measurement uncertainty in the range of few millimeters Roveda et al. (2021). Therefore, the considered variability in position of the target part makes the proposed approach robust in real conditions. For this specific case, 20 trials are performed, 10 for each direction. The task is completed successfully in most of the test runs. In failures situations, the robot either grasped the cube at the edges, resulting in an unstable grasping, or collided its gripper with the cube, not being able to grasp it successfully.

In Table 3, the policy adaptation results on the modified task conditions are presented. In the first two tests, the nominal position of the cube has been modified by 10 cm and 20 cm respectively and the learned task's policy is evaluated on its adaptation capability. Retraining the learned policy for an additional 30 minutes, the resulting performance achieves 100% success. The last test considers the dynamic environment where the cube is moving with some velocity (to simulate the real industrial settings where the robot might be required to grasp objects placed on a moving conveyor belt). In this dynamic setting, the static grasping policy is not able to grasp the moving cube. However, when this

Table 1: Evaluation trials for fixed positions.

Test	Object	Success rate
1.	Nominal cube	10/10
2.	Smaller cube	10/10
3.	Cylinder	10/10
4.	Screw-driver	10/10

pre-trained policy is fine-tuned for an additional hour, the robot learns to adapt its performance and lift the moving cube with 100% success, indicating the quick adaptation of learned behavior to perform in modified task settings.

5.2 Performance specifications-based fine-tuning

In this section, the baseline reward function is modified with additional objectives (*i.e.*, considering all the three specified objectives in Section 3.4) in order to test the potential of the proposed fine-tuning approach in improving the previously learned behavior with less data requirements, and to test additional scenarios with higher difficulty level of the task. The fine-tuning procedure involves initializing the parameters of the target task networks (policy network and value network) with the pre-trained baseline networks. For all three specifications, the state space, the action space and the hypermarameters remain unchanged and only additional performance objectives are included successively.

5.2.1 Redundancy management specification

Under the baseline policy, the robot took actions that resulted in one or more joints reaching the limits. In order to encourage the robot to not take actions near joint limits, redundancy management objective in (5) is included in the reward function and the task is trained for half a million steps (roughly corresponding to 30 minutes of training). Figure 7 shows the performance comparison of an adapted policy with the baseline policy for 10 test trials. The results are reported as percentage of episode steps where the robot configuration exceeds the set threshold of joint limits. As hypothesized for the case of joint limits penalty, the robot joint configurations satisfy the set tolerance of 15% in all episode steps, confirming the quick behavior adaptation.

Table 2: Evaluation trials for varied positions.

Variable	Amount	Success rate
Robot position	max 2%	80 %
Cube position	±8 cm	70 %

5.2.2 Smoothing control action specification

The goal of smoothing performance objective is to limit the change in acceleration of the robot's joints in order to minimize the jerky movements. To achieve this, the agent is penalized for taking actions that generate excessive joint's accelerations with the penalty defined in (6), and the results are compared against the baseline model. The velocities of all joints are plotted against time for the baseline case and the case with acceleration penalty in Figure 9. It is evident that the behavior is improved with the addition of acceleration penalty, yielding a smooth change in velocities throughout the episode, particularly for joints 1 and 3. The commanded joint torques for both cases are plotted in Figure 8. The difference is more evident for the torque commands sent to the robot joints with much smoother torque commands for the penalized case.

5.2.3 Obstacles avoidance specification

Considering the collision avoidance objective, the working scene is modified by adding other objects in the working space of the robot. Specifically, two cylindrical objects of 2 cm radius and 8 cm height are placed on the left and right side of the cube to be grasped. In the modified scene, the baseline policy successfully lifts the cube, but at the expense of colliding with cylinders. The aim of this performance specification is to significantly penalize the robot actions that result in collisions with

Table 3: Evaluation trials for policy adaptation.

Variable	Amount	Re-train time	Success rate	
			Base policy	Adapted policy
Cube position	+10 cm	30 mints	20 %	100 %
Cube position	+20 cm	30 mints	0 %	100 %
Cube velocity	100 mm/sec	60 mints	0 %	100 %

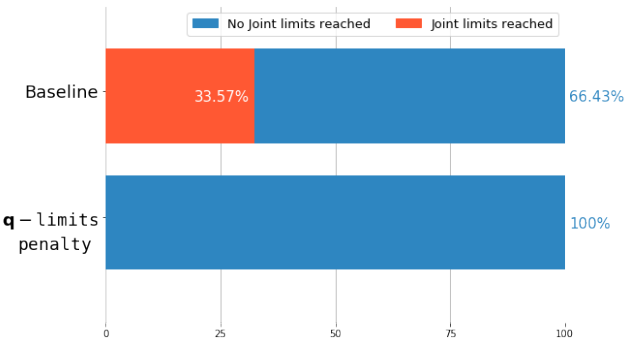


Fig. 7: Adaptation results - Percentage of episode steps with joint limits reached for baseline and **q**-limits penalty scenarios.

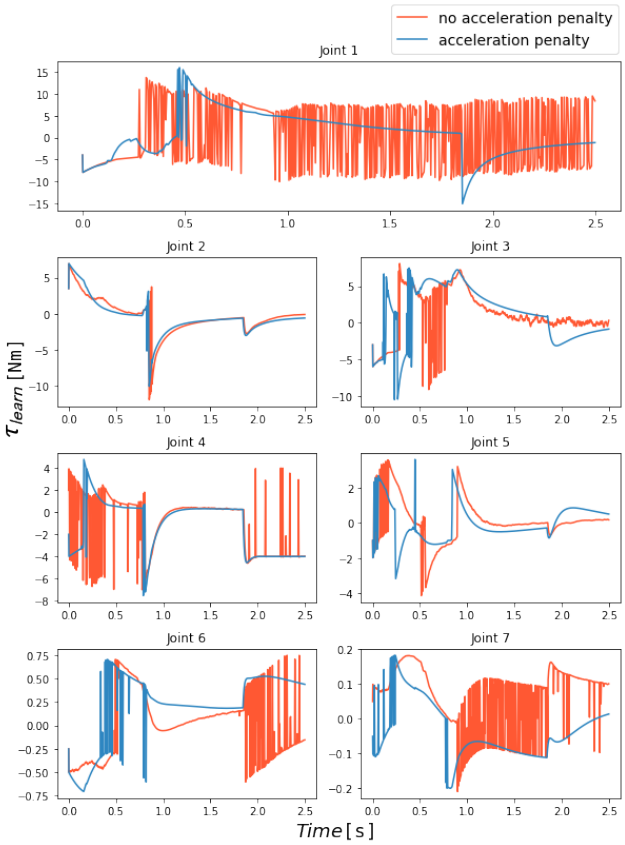


Fig. 8: Adaptation results - Commanded joint torques for baseline and acceleration penalty scenarios.

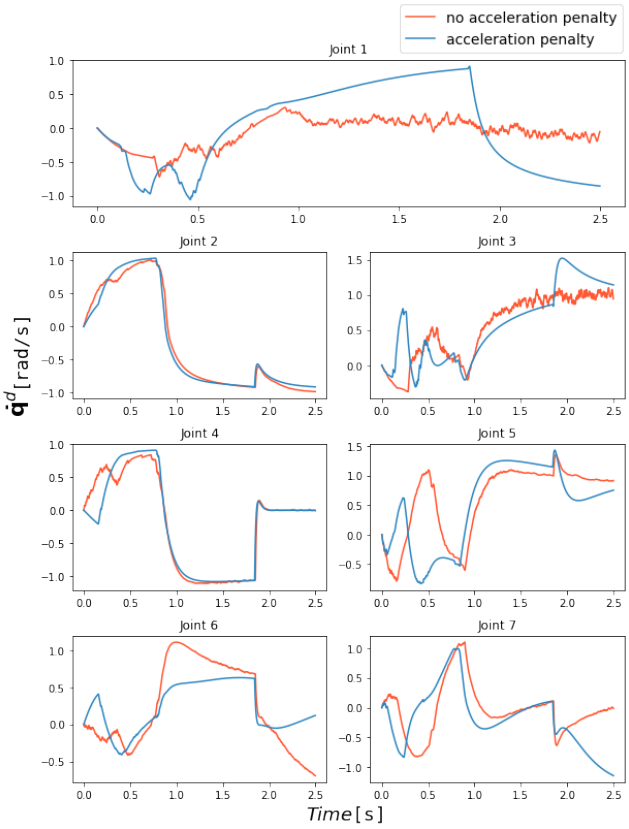


Fig. 9: Adaptation results - Joint velocities for baseline and acceleration penalty scenarios.

the cylinders. By adding the objective defined in (7) into the reward function and training for half a million steps, the resulting behavior has been adapted and the robot successfully navigates the cylindrical obstacles to lift the cube.

5.3 One-shot learning

In the last experiment, the base task has been trained from scratch by including all 3 performance specifications specified in the Section 3.4 in order to evaluate the performance of one-shot learning. The policy is trained for a total of 11.5 million steps (equivalent duration as the combined training of base task and adaptation process). The final learned policy, however, failed to accomplish the complete task while satisfying all the performance objectives. After training for 11.5 million steps, the robot only learned to reach the cube and command smooth joint velocities proving the difficulty of learning in the presence of multiple objectives.

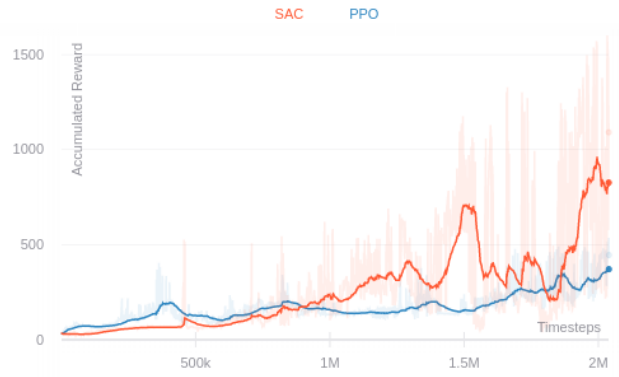


Fig. 10: Training comparison - Mean accumulated reward for first 2M time steps.

5.4 Experimental results

The learned behaviors (from the training including the performance specifications into the reward function) have been transferred to the real Franka Emika Panda robot in order to evaluate the effectiveness of the proposed approach. Two scenarios have been considered: i) no obstacles in the operating scene, ii) obstacles in the

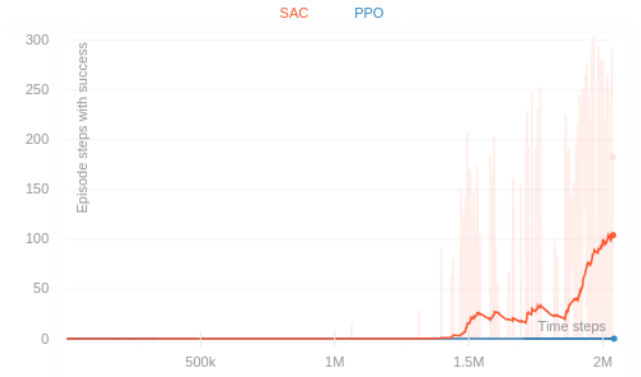


Fig. 11: Training comparison - Number of episode steps with success for first 2M time steps.

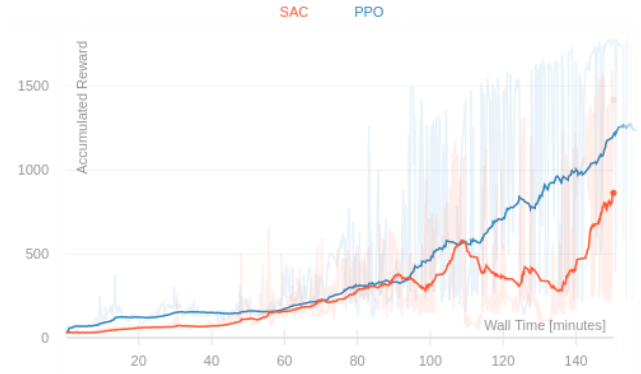


Fig. 12: Training comparison - Mean accumulated reward vs. wall time.

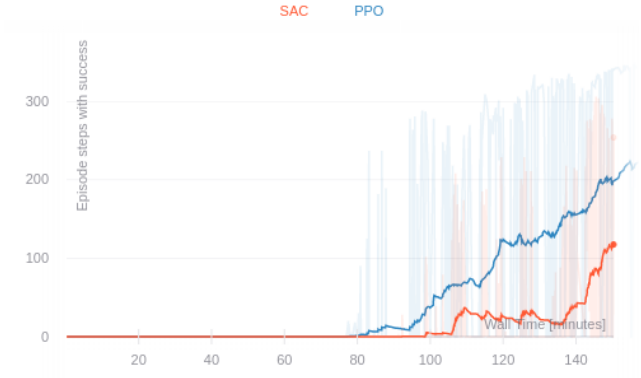


Fig. 13: Training comparison - Number of episode steps with success against wall time.

operating scene (same scenario as in obstacles avoidance performance specifications in Section 5.2.3). Each scenario has been tested 10 times. The learned reference joint velocities $\dot{\mathbf{q}}^d$ have been re-sampled from 250 Hz to 1000 Hz (*i.e.*, to the robot control frequency). Exploiting the performed learning, the real Franka Emika Panda robot has been able to complete the target tasks with a success rate of 100%, highlighting the feasibility of the proposed approach to be transferred from simulation to real applications. Videos showing the results achieved by the proposed PPO-based approach is also available at the GitHub repository Shahid (2020). The repository is updated on the basis of the work-in-progress software development, new achieved results and experiments.

6 The comparison of PPO and SAC

In order to compare the performance of the on-policy based PPO algorithm with the performance of an off-policy algorithm, the baseline scenario is further trained using the SAC algorithm described in Section A.1.2. An off-policy algorithm could improve the sample efficiency during the training, speeding up (*i.e.*, reducing the required episodes) the learning process. In fact, since SAC is an off-policy algorithm that makes use of past transitions data stored in a replay buffer, it should learn the task faster in terms of required episodes. In order to validate this hypothesis, training performance achieved by the SAC algorithm is compared against the one achieved by the PPO algorithm for the analyzed grasping task. The mean reward and number of episode steps with success is plotted for the first 2M time steps for the two algorithms and comparison is shown in Figure 10 and in Figure 11. Confirming the hypothesis, the SAC algorithm learns to accumulate much higher average reward than PPO, and also learns to achieve the task success in just 2M time steps (about 3300 episodes), whereas PPO shows no success in terms of task completion for the same number of episodes. However, analyzing the mean reward and number of episode steps with success plotted against wall time in Figure 12 and in Figure 13, the results are quite opposite for both algorithms. Because SAC has to process large amount of past data stored in a replay buffer, it needs higher computation power to perform each update iteration of networks. Considering computation power, SAC is less efficient than PPO and learns to accumulate lower average reward than PPO (800 vs. 1200) in the same time as evident from Figure 12. Off-policy updates of SAC accumulate lower reward on each update iteration than the on-policy updates of PPO as shown in Figure 14 (for the reward) and in Figure 15 (for the number of episodes steps with success). Although each update of networks in PPO is performed after collecting 3 episodes vs. 1 episode for SAC, the reward and the number of episode success steps in each of the update for SAC is lower than the ones of PPO. Thus, when trained on the same hardware configuration for roughly 2 and a half hours of wall-clock time, the accumulated mean

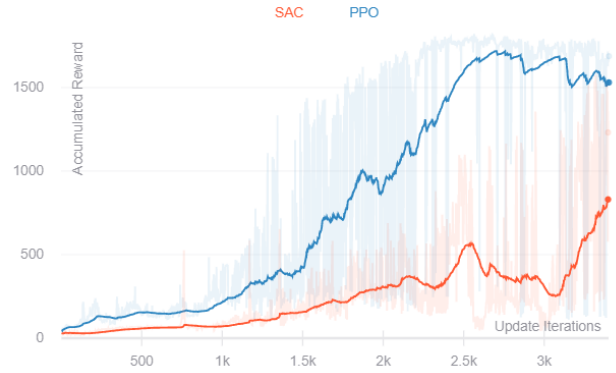


Fig. 14: Training comparison - Mean accumulated reward against update iterations of networks.

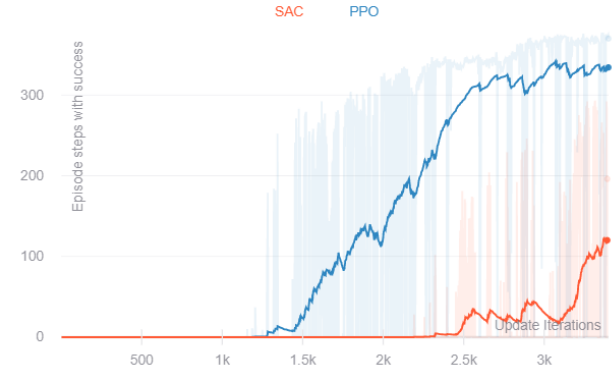


Fig. 15: Training comparison - Number of episode steps with success against update iterations of networks.

reward and the success achieved by SAC are lower than the one achieved by PPO. Moreover, the SAC training curve shows more significant non-monotonic behavior with high variance in results. The resulting behavior of two algorithms can be explained considering the general nature of on-policy algorithms in requiring more experience to explore the environment than off-policy algorithms. In the analyzed task, the work space of the robot is unconstrained, demanding appropriate exploration to learn the target task. Therefore, the trade-off between on-policy and off-policy algorithms is not so straightforward, but instead is dependant on the specific situation. Off-policy methods like SAC might be preferred if the new experience is costly to obtain and computation power is not an issue. On the other hand, on-policy methods like PPO could achieve better results in situations with less-costly data collection, *e.g.*, in simulation. On-policy methods, due to the exploratory nature of learned ultimate policy, can also adapt the learned behavior quite quickly as demonstrated in Section 5.2

7 Discussion and current/future works

In this paper, an intelligent task learning has been formulated in the form of a RL problem, demonstrating the possibility of learning low-level continuous control actions purely from gathered experience in a simulated environment. Results show that it is possible to train continuous control actions based only on the kinematic state information, and in a reasonable amount of interaction time. It has been shown that the on-policy fine-tuning procedure substantially improves sample efficiency and allows the learned policy to adapt and execute partially modified task. The performance specifications have been embedded into the learning in order to manage the redundancy of the robot, to smooth the learned control actions, and to avoid obstacles in the operating scene. Achieved results highlight that the learned policy performs well to new situations, and it can also adapt its learning (retraining the robot behavior on the basis of previous training) to significant variations of the environment with slight amount of additional training, exploiting the previously learned network models. In contrast, one-shot learning, struggled to achieve the same performance and learn the complete task. The rigorous comparison of one-shot learning with fine-tuning based adaptation should, however, be treated in the future. An on-policy algorithm (PPO) and an off-policy algorithm (SAC) have been compared in terms of learning performance, highlighting the faster learning process by exploiting the SAC algorithm in terms of required episodes, while being slower in terms of wall time (considering the same adopted hardware). The learned task has been successfully transferred to a real robot, making it able to execute the target grasping task. Although, the fine-tuning process has shown to improve sample efficiency by allowing the policy to adapt in less iterations, there exist a gap to apply such methods for direct learning in real world without the use of simulators. Some of the techniques for addressing the sample efficiency issue could be to use fine-tuning on model-based methods for first learning the dynamics model and then adapting that model through some online interaction on a different performance objective. Model-based methods are generally more efficient than model-free counterparts. Another idea could be to consider Meta-RL setting to learn variety of tasks in order to efficiently learn new tasks.

In the future work, more dynamic environments will be considered. In fact, having the robot able to behave in the presence of moving obstacles/targets while avoiding dynamic obstacles is a challenging application. In addition, safety rules for the robot motion-adaptation in the presence of humans will be consid-

ered. Another interesting direction is to extend the approach to modular configurations and use multi-agent framework to learn the task in decentralised manner as done in Shahid et al. (2021), in which, however, the coordination between the agents is still a challenging problem that needs to be further investigated. Furthermore, the ability to adapt to more novel environments can be considered, including the usage of vision systems to model/update the operating environment in the training environment.

References

Abbeel P, Coates A, Quigley M, Ng AY (2007) An application of reinforcement learning to aerobatic helicopter flight. In: Advances in neural information processing systems, pp 1–8

Achiam J (2018) Spinning up in deep reinforcement learning. URL `https://spinningup.openai.com/en/latest/algorithms/sac.html`

Boularias A, Kober J, Peters J (2011) Relative entropy inverse reinforcement learning. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp 182–189

Chebotar Y, Kalakrishnan M, Yahya A, Li A, Schaal S, Levine S (2017) Path integral guided policy search. In: 2017 IEEE international conference on robotics and automation (ICRA), IEEE, pp 3381–3388

Cui F, Cui Q, Song Y (2020) A survey on learning-based approaches for modeling and classification of human-machine dialog systems. IEEE Transactions on Neural Networks and Learning Systems

Deisenroth M, Rasmussen CE (2011) Pilco: A model-based and data-efficient approach to policy search. In: Proceedings of the 28th International Conference on machine learning (ICML-11), pp 465–472

Deisenroth MP, Neumann G, Peters J (2013) A survey on policy search for robotics. now publishers

Duan Y, Chen X, Houthooft R, Schulman J, Abbeel P (2016) Benchmarking deep reinforcement learning for continuous control. In: International Conference on Machine Learning, pp 1329–1338

Fan L, Zhu Y, Zhu J, Liu Z, Zeng O, Gupta A, Creus-Costa J, Savarese S, Fei-Fei L (2018) Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In: Conference on Robot Learning, pp 767–782

Gu S, Holly E, Lillicrap T, Levine S (2017) Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE international conference on robotics and automation (ICRA), IEEE, pp 3389–3396

Haarnoja T, Zhou A, Abbeel P, Levine S (2018a) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:180101290

Haarnoja T, Zhou A, Hartikainen K, Tucker G, Ha S, Tan J, Kumar V, Zhu H, Gupta A, Abbeel P, et al. (2018b) Soft actor-critic algorithms and applications. arXiv preprint arXiv:181205905

Heess N, TB D, Sriram S, Lemmon J, Merel J, Wayne G, Tassa Y, Erez T, Wang Z, Eslami S, et al. (2017) Emergence of locomotion behaviours in rich environments. arXiv preprint arXiv:170702286

Julian R, Swanson B, Sukhatme GS, Levine S, Finn C, Hausman K (2020) Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. arXiv preprint arXiv:200410190

Kalakrishnan M, Righetti L, Pastor P, Schaal S (2011) Learning force control policies for compliant manipulation. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, pp 4639–4644

Kalashnikov D, Irpan A, Pastor P, Ibarz J, Herzog A, Jang E, Quillen D, Holly E, Kalakrishnan M, Vanhoucke V, et al. (2018) Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. arXiv preprint arXiv:180610293

Kearney KT, Presenza D, Saccà F, Wright P (2018) Key challenges for developing a socially assistive robotic (sar) solution for the health sector. In: 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), IEEE, pp 1–7

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980

Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms. In: Advances in neural information processing systems, pp 1008–1014

Kormushev P, Calinon S, Caldwell DG (2010) Robot motor skill coordination with em-based reinforcement learning. In: 2010 IEEE/RSJ international conference on intelligent robots and systems, IEEE, pp 3232–3237

Kornblith S, Shlens J, Le QV (2019) Do better imagenet models transfer better? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 2661–2671

Lasi H, Fettke P, Kemper HG, Feld T, Hoffmann M (2014) Industry 4.0. Business & information systems engineering 6(4):239–242

Lazaric A, Restelli M, Bonarini A (2008) Reinforcement learning in continuous action spaces through sequential monte carlo methods. In: Advances in neural information processing systems, pp 833–840

Lee Y, Yang J, Lim JJ (2019) Learning to coordinate manipulation skills via skill behavior diversification. In: International Conference on Learning Representations

Lenz I, Lee H, Saxena A (2015) Deep learning for detecting robotic grasps. The International Journal of Robotics Research 34(4-5):705–724

Levine S, Abbeel P (2014) Learning neural network policies with guided policy search under unknown dynamics. In: Advances in Neural Information Processing Systems, pp 1071–1079

Levine S, Wagener N, Abbeel P (2015) Learning contact-rich manipulation skills with guided policy search. Proceedings - IEEE International Conference on Robotics and Automation 2015, DOI 10.1109/ICRA.2015.7138994

Levine S, Finn C, Darrell T, Abbeel P (2016) End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research 17(1):1334–1373

Levine S, Pastor P, Krizhevsky A, Ibarz J, Quillen D (2018) Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. The International Journal of Robotics Research 37(4-5):421–436

Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971

Mahler J, Matl M, Liu X, Li A, Gealy D, Goldberg K (2017) Dex-net 3.0: Computing robust robot vacuum suction grasp targets in point clouds using a new analytic model and deep learning. arXiv preprint arXiv:170906670

Martín-Martín R, Lee MA, Gardner R, Savarese S, Bohg J, Garg A (2019) Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. arXiv preprint arXiv:190608880

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533

Morrison D, Corke P, Leitner J (2018) Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. arXiv preprint arXiv:180405172

Mülling K, Kober J, Kroemer O, Peters J (2013) Learning to select and generalize striking movements in robot table tennis. The International Journal of Robotics Research 32(3):263–279

ten Pas A, Gualtieri M, Saenko K, Platt R (2017) Grasp pose detection in point clouds. The International Journal of Robotics Research 36(13-14):1455–1473

Pastor P, Kalakrishnan M, Chitta S, Theodorou E, Schaal S (2011) Skill learning and task outcome prediction for manipulation. In: 2011 IEEE International Conference on Robotics and Automation, IEEE, pp 3828–3834

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, et al. (2019) Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32:8026–8037

Peters J, Schaal S (2008) Natural actor-critic. Neurocomputing 71(7-9):1180–1190

Quillen D, Jang E, Nachum O, Finn C, Ibarz J, Levine S (2018) Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 6284–6291

Rahmatizadeh R, Abolghasemi P, Bölöni L, Levine S (2018) Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 3758–3765

Rajan K, Saffiotti A (2017) Towards a science of integrated ai and robotics

Rajeswaran A, Kumar V, Gupta A, Vezzani G, Schulman J, Todorov E, Levine S (2017) Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. arXiv preprint arXiv:170910087

Roveda L, Forgione M, Piga D (2020) Robot control parameters auto-tuning in trajectory tracking applications. Control Engineering Practice 101:104488

Roveda L, Maroni M, Mazzuchelli L, Praolini L, Bucca G, Piga D (2021) Enhancing object detection performance through sensor pose definition with bayesian optimization. In: 2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), IEEE, pp 699–703

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

Schulman J, Moritz P, Levine S, Jordan M, Abbeel P (2015b) High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:150602438

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:170706347

Seif HG, Hu X (2016) Autonomous driving in the icity—hd maps as a key challenge of the automotive

industry. Engineering 2(2):159–162
Shahid AA (2020) Github repository: Intelligent-task-learning. URL `https://github.com/Asad-Shahid/Intelligent-Task-Learning`
Shahid AA, Sesin JSV, Pecioski D, Braghin F, Piga D, Roveda L (2021) Decentralized multi-agent control of a manipulator in continuous task learning. Applied Sciences 11(21):10227
Siciliano B, Villani L (2000) Robot Force Control, 1st edn. Kluwer Academic Publishers, Norwell, MA, USA
Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al. (2016) Mastering the game of go with deep neural networks and tree search. nature 529(7587):484
Smith L, Kew JC, Peng XB, Ha S, Tan J, Levine S (2021) Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. arXiv preprint arXiv:211005457
Song S, Zeng A, Lee J, Funkhouser T (2020) Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. IEEE Robotics and Automation Letters 5(3):4978–4985
Sutton RS (1988) Learning to predict by the methods of temporal differences. Machine learning 3(1):9–44
Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press
Tobin J, Fong R, Ray A, Schneider J, Zaremba W, Abbeel P (2017) Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp 23–30
Todorov E, Erez T, Tassa Y (2012) Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, pp 5026–5033
Tsarouchi P, Makris S, Chryssolouris G (2016) Human–robot interaction review and challenges on task planning and programming. International Journal of Computer Integrated Manufacturing 29(8):916–931
Van Roy V, Vertesy D, Damioli G (2020) Ai and robotics innovation. Handbook of Labor, Human Resources and Population Economics pp 1–35
Viereck U, Pas A, Saenko K, Platt R (2017) Learning a visuomotor controller for real world robotic grasping using simulated depth images. In: Conference on Robot Learning, PMLR, pp 291–300
Wei Q, Wang L, Liu Y, Polycarpou MM (2020) Optimal elevator group control via deep asynchronous actor-critic learning. IEEE Transactions on Neural Networks and Learning Systems
Yang Z, Merrick K, Jin L, Abbass HA (2018) Hierarchical deep reinforcement learning for continuous action control. IEEE transactions on neural networks and learning systems 29(11):5174–5184

A Method Implementation Details

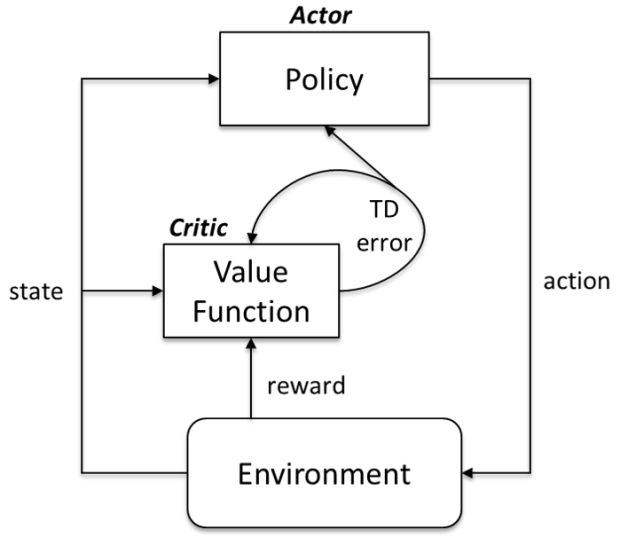
RL methods fall in to 3 main categories: value-based, policy search and actor-critic Konda and Tsitsiklis (2000). Value-based methods learn a value function that implicitly derives a policy. Policy search methods, instead, use parameterized policy and directly search for optimal policy parameters. The key idea of actor-critic approaches is to learn both the value function and the policy. The role of critic is, on the basis of the learned value function, to critique the actions taken by an actor, which then updates it's policy parameters based on critic's feedback Lazaric et al. (2008). Actor-critic methods have been used in control tasks for learning deterministic policies Yang et al. (2018) and stochastic policies Peters and Schaal (2008); Wei et al. (2020). In this paper, the proposed approach follows the actor-critic style and learns both the parameters of stochastic policy and value function approximator.

RL algorithms can further be classified into *on-policy* and *off-policy* methods, based on how they generate and use experience data. On-policy methods evaluate the same policy that is used to select actions, whereas, off-policy methods learn the target policy that is different from the behavioral policy used to generate experience. Because on-policy methods use the most updated policy to take decisions, they have better online performance Sutton and Barto (2018). The on-policy agent learns not the optimal policy but instead the best exploratory policy, *i.e.*, the policy that still explores. Off-policy methods, on the other hand, make efficient use of data and are not restricted to learning from data generated by execution of a specific policy, instead, data can be generated from any arbitrary policy. In off-policy algorithms, the temporal difference (TD) error Sutton (1988) is computed using the policy that may act more greedily. The general scheme of on-policy and off-policy approaches for actor-critic algorithms is shown in Figure 16.

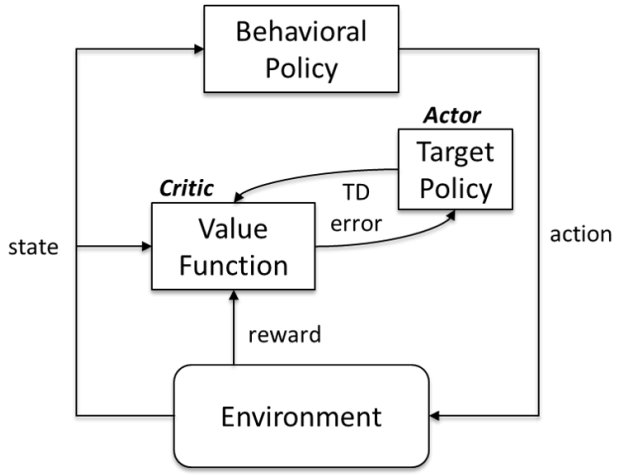
A.1 Algorithms

A.1.1 Proximal Policy Optimization (PPO)

In this paper, a model-free RL approach has been used as it eliminates the need for accurate dynamics model, which is often difficult to learn for discontinuous problems. Primarily, the policy is trained with Proximal Policy Optimization (PPO) Schulman et al. (2017). PPO is one of the most successful on-policy RL algorithms. It requires training two neural networks to simultaneously optimize stochastic policy and approximation of value function. The policy network maps states to Gaussian distribution over actions, while the value network estimates the discounted sum of future rewards expected in a given state. The PPO algorithm is an approximate version of trust-region policy optimization (TRPO) Schulman et al. (2015a) with first order gradients. The optimization objective in PPO is performed on the surrogate loss function



(a) On-policy RL



(b) Off-policy RL

Fig. 16: Two different classes of RL methods. (a) On-policy. (b) Off-policy.

L_{PPO}:

$$\begin{aligned} L_{PPO} &= \mathbb{E}\left[\min\left(arg_1, arg_2\right)\right], \\ arg_1 &= \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t^{GAE}, \\ arg_2 &= clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{GAE}, \end{aligned} \tag{11}$$

where $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is an importance sampling term that denotes the probability ratio between the given action under current policy π_θ and the same action under old behavioral policy $\pi_{\theta_{old}}$ that was used to generate trajectory. $\hat{A}_t^{GAE}$is the advantage estimate computed using generalized advantage estimation (GAE) Schulman et al. (2015b), and ϵ is a clipping hyperparameter. With importance sampling, the PPO algorithm tries to improve the sample efficiency of on-policy learning by performing multiple steps of optimization using the same trajectory. A small entropy bonus $S[\pi_\theta](s_t)$, weighted by a coefficient c_2, is further added to the PPO loss in order to ensure sufficient exploration of the environment. The pseudo-code of selected PPO algorithm is shown in 1.

A.1.2 Soft Actor-Critic (SAC)

Soft Actor-Critic Haarnoja et al. (2018b,a) is a model-free RL algorithm based on maximum entropy RL framework. SAC follows an off-policy way to optimize a stochastic policy. The optimization objective in SAC considers an expected return and the entropy to encourage exploration. This objective is equivalent to adding a bonus reward at each time step proportional to the entropy of the policy at that step Achiam (2018), *i.e.*, the agent is incentivized to successfully complete the task while also acting as randomly as possible. The SAC optimization objective is described as:

$$\mathbb{E}_{(s_t,a_t)\sim\pi}\left[\sum_t \gamma\mathcal{R}(s_t,a_t) + \alpha\mathcal{H}(\pi(.|s_t))\right] \tag{12}$$

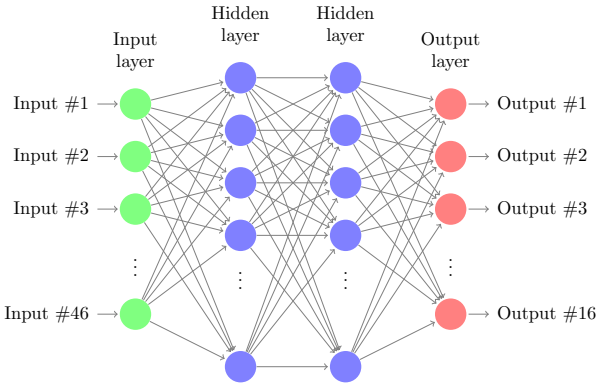
where $\mathcal{H}(P) = \mathbb{E}_{x\sim P}[-\log P(x)]$ denotes the entropy of a random variable x computed from its distribution $P(x)$ and α is a temperature parameter. In this paper, the automatic adjustment of α is used to match a target entropy set to $-dim(\mathcal{A})$. Since SAC is an off-policy algorithm, it stores the past experience data in to a replay buffer and reuses it for sample-efficient training.

Algorithm 1 Proximal Policy Optimization (PPO) (adapted from Schulman et al. (2017)).

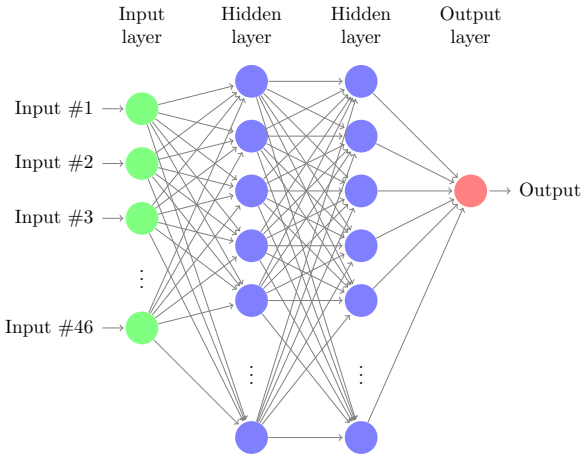
```
Randomly initialize policy parameters θ
Initialize replay buffer B
for iteration = 1, ..., N do
    for episode = 1, ..., M do
        Generate a rollout following policy π_θ until H timesteps
        Store transitions(s_t, a_t, r_t) in B
        Compute advantages Â_t^GAE with GAE
    end for
    for epochs = 1, ..., K do
        Sample mini-batch of N_b transitions from B
        Optimize surrogate loss L_PPO w.r.t. θ
        θ_old ← θ
    end for
    clear B
end for
```

A.2 Training Details

Both the policy and value networks are encoded by multi-layer perceptron (MLP). The policy network is a 3-layer MLP with hidden layer size of 64 and Rectified Linear Unit (ReLU) non-linearity. The output layer of a policy network produces the mean and the standard deviation for each action dimension. After sampling each action from the Gaussian distribution, *tanh* activation function is applied to enforce action bounds in range of $[-1, 1]$. The value network is represented as a 3-layer MLP that outputs a scalar value specifying the



(a) Policy Network



(b) Value Network

Fig. 17: Neural Networks architecture. (a) the policy network with considered 46-dim input (states), 64-dim hidden layers and 16-dim output, and (b) the value network with considered 46-dim input (states), 128-dim hidden layers and 1-dim output.

corresponding value of a state. The value network uses a hidden layer size of 128 units with ReLU non-linearity. All inputs fed to the policy and value network are normalized with running estimates of mean and variance. The library employed for training the agents is Pytorch Paszke et al. (2019). More detailed training parameters for both algorithms are given in Table 4 with some hyperparameters settings taken from Lee et al. (2019).

Table 4: Hyperparameters used for PPO and SAC.

Hyperparameter	Value
Hardware configuration	1 NVIDIA GPU + 12 CPU cores
Discount factor γ	0.99
Generalized Advantages Estimation λ	0.95
PPO clipping parameter ϵ	0.2
Optimizer	Adam Kingma and Ba (2014)
Actor's learning rate	3×10^{-4}
Critic's learning rate	3×10^{-4}
Mini-batch size	512
Number of epochs	60
Value loss weight	0.5
Entropy loss coefficient c_2	1×10^{-4}
Replay buffer size PPO (episodes n.)	3
Replay buffer size SAC (episodes n.)	1×10^{5}
SAC reward scale	1
SAC entropy target	-8