International Journal of Machine Learning and Cybernetics (2023) 14:3003–3023
https://doi.org/10.1007/s13042-023-01815-8

ORIGINAL ARTICLE

Learning positioning policies for mobile manipulation operations with deep reinforcement learning

Ander Iriondo[1,2] · Elena Lazkano[2] · Ander Ansuategi[1] · Andoni Rivera[1] · Iker Lluvia[1] · Carlos Tubío[1]

Received: 24 February 2022 / Accepted: 28 February 2023 / Published online: 17 March 2023

Abstract

This work focuses on the operation of picking an object on a table with a mobile manipulator. We use deep reinforcement learning (DRL) to learn a positioning policy for the robot's base by considering the reachability constraints of the arm. This work extends our first proof-of-concept with the ultimate goal of validating the method on a real robot. Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm is used to model the base controller, and is optimised using the feedback from the MoveIt! based arm planner. The idea is to encourage the base controller to position itself in areas where the arm reaches the object. Following a simulation-to-reality approach, first we create a realistic simulation of the robotic environment in Unity, and integrate it in Robot Operating System (ROS). The drivers for both the base and the arm are also implemented. The DRL-based agent is trained in simulation and, both the robot and target poses are randomised to make the learnt base controller robust to uncertainties. We propose a task-specific setup for TD3, which includes state/action spaces, reward function and neural architectures. We compare the proposed method with the baseline work and show that the combination of TD3 and the proposed setup leads to a 11% higher success rate than with the baseline, with an overall success rate of 97%. Finally, the learnt agent is deployed and validated in the real robotic system where we obtain a promising success rate of 75%.

Keywords Mobile manipulation · Pick and place · Deep reinforcement learning · Sim-to-real transfer

1 Introduction

Pick and place are basic operations in robotics applications, whether in industrial setups (e.g., machine tending, assembly, or bin-picking) or in service robotics domain (e.g. agriculture or home). Although in some structured scenarios pick and place operations have shown a strong performance, this is not the case in less structured uncertain environments. The efficiency and robustness of the solutions are a bottleneck for industrial applications, and those have not reached the market yet.

The majority of robotics applications focus either on navigation aspects of mobile platforms (e.g. industrial transportation systems, guide robots), or the manipulation of goods with robotic arms (e.g., bin-picking applications). Nonetheless, few applications consider mobile manipulation itself combining both robotic tasks. Despite there are several commercial mobile manipulators in the market, there is a lack of real applications due to the complexity and uncertainty introduced by combining both, manipulation and navigation.

The present work focuses on the picking operation of a randomly placed object from a table with a mobile

✉ Ander Iriondo
ander.iriondo@tekniker.es

Elena Lazkano
e.lazkano@ehu.eus

Ander Ansuategi
ander.ansuategi@tekniker.es

Andoni Rivera
andoni.rivera@tekniker.es

Iker Lluvia
iker.lluvia@tekniker.es

Carlos Tubío
carlos.tubio@tekniker.es

[1] Department of Autonomous and Intelligent Systems, Tekniker - Basque Research and Technology Alliance (BRTA), Iñaki Goenaga, 5, 20600 Eibar, Gipuzkoa, Spain

[2] Robotics and Autonomous Systems group (RSAIT), Department of Computer Science and Artificial Intelligence, University of the Basque Country (UPV/EHU), Po Manuel Lardizabal,1, 20018 Donostia-San Sebastián, Gipuzkoa, Spain

manipulator. Due to the particular morphology of robotic arms, their scope is limited, and not all the positions of the base near the table enable a successful picking.

Traditionally, such mobile manipulation operations have been solved using analytical planning and control methods [1]. These methods require explicit programming of the skills which can be very costly and error-prone particularly in problems where decision making is complex. The performance of these models depends on how well the reality fits the assumptions made by the model. Due to the impossibility of predicting all the cases that may occur in dynamic and unstructured environments, these methods are generally impractical.

In many explicitly programmed mobile manipulation applications, the base navigates first to zones where the object is within reach (only considering distance), and then carry out the picking trial. However, other works such as the one proposed by Stulp et al. in [2] challenge this and raise the following questions: *"Does well-in-reach always imply that the target can really be reached, given the hardware and control software of the robot?, Can we have a least-commitment realisation of 'places' such that the robot can refine a 'place' as it learns more about the context (e.g. clutteredness) of the surroundings?, How can such a concept of 'place' take into account uncertainties about the robot's self-localisation and the estimated target position?"*. As the authors mention, explicit programming to account for these factors is tedious and impractical, and more flexible solutions are needed.

Alternatively, data-driven methods allow learning directly from real data [3]. Recently, the combination of deep learning (DL) and reinforcement learning (RL), known as deep reinforcement learning (DRL), has allowed to tackle complex decision-making problems that were previously unfeasible [4]. In fact, DRL has proven to be the state-of-the-art technology for learning complex robotic behaviours through the interaction with the environment and solely guided by a reward signal.

The proposed method was originally based on the idea proposed in [2]. In that work authors propose to learn a data-driven model of "place" (areas where the base of the mobile manipulator has to be placed to guarantee the reach of the arm to the object) by interacting with the environment and using machine learning (ML). In that way, they take into account the limits of the robot hardware as well as the uncertainty in the localisation of both the robot and the target. While ML-based methods are generally used for offline forecasting, DRL is generally used online in sequential decision-making problems [5]. In fact, DRL allows to autonomously learn complex control policies through trial and error and only guided by a reward signal. In the case of robotics, the most common use case is to use such algorithms to model agents capable of performing continuous control of robots.

In a first proof-of-concept, we extended the idea and learnt a positioning policy for the mobile manipulator's base using state-of-the-art DRL algorithms [6]. We proposed a novel method which was based on learning a DRL-based controller for the base taking into account the reachability constraints of the arm. To do so, we proposed to use feedback from a traditional arm trajectory planner to optimise the base controller. In fact, the learnt base controller was able to drive the mobile manipulator to areas with a high probability of picking success. Although we showed the feasibility of applying DRL to learn such a complex behaviour, the experimentation was carried out only in a simplistic simulated environment, which was far from reality.

Following the same idea, in this work we go one step further and propose a fully improved DRL-based learning framework for the mobile manipulation task, with the final goal of deploying the learnt model in the real robot. The contributions of this article are as follows:

1. Based on our previous work [6], we developed an improved control architecture that allows learning the task in simulation and an easy deployment in the real system. In fact, the proposed approach combines the Twin Delayed Deep Deterministic Policy Gradient (TD3) [7] DRL algorithm to control the base with traditional planning and control algorithms for the arm. To the best of our knowledge, this is the first time that TD3 has been applied to learn a mobile manipulation skill.
2. We propose a novel reward formulation, task-specific state/action space definition, and neural architecture selection for TD3 that lead to a robust and stable controller. The reward function carefully combines dynamic variables related to distance to target, speed/acceleration, collision checking and feedback from the arm planner to encourage the robot to navigate towards areas with a high probability of pick success.
3. We develop a realistic simulation of both the robot and the environment in Unity [8] from scratch. The simulation is integrated into the Robot Operating System (ROS) [9] following the OpenAI Gym standard [10]. The drivers needed to command both the base and the arm are also developed. Robot and target position randomisation functionalities are also offered through ROS.
4. A thorough evaluation is conducted to assess the proposed method and compare it with the baseline using standard metrics. Finally, the learnt agent is deployed and validated in the real system. The results obtained show the feasibility of the proposed approach.

The mobile manipulator used was previously developed by Tekniker and is depicted in Fig. 1.

The rest of the paper is organised as follows. Section 2 reviews the literature. Section 3 explains the details of the

Fig. 1 The robot, developed by Tekniker, combines a commercial Segway omnidirectional mobile platform and a KUKA iiwa arm

design and implementation of the DRL-based control architecture for the mobile manipulator. In Sect. 4 we benchmark our method with the baseline approach, using multiple DRL algorithms. Then, details about the deployment and validation in the real robotic system are presented in Sect. 5. Finally, in Sect. 6 some conclusions are drawn, and the next steps are discussed.

2 Literature review

Traditionally, well known planning and control methods have been widely used for scheduling mobile manipulation behaviours [11], for example using the ROS navigation stack [12] for navigation and MoveIt! [13] for manipulation. For instance, Dömel et al. [14] focus on fetch and carry operations in industrial environments. In that work, both the arm and the base are considered as independent systems and, although a reachability study is carried out, it is done offline and thus ignoring the environment's dynamics. In addition, the reachability study is done for the sole purpose of establishing the design of the application. Xu et al. [15] propose to use an inverse kinematics database to estimate the feasible positions of the base to solve a pick-and-place operation with objects stored in trays. However, the database of feasible locations is composed of discrete poses that are estimated offline, and consequently they also ignore the dynamics of the environment. In other methods such as [16, 17, 18] both the base and the arm are seen as one and also rely on traditional path planning and control methods to schedule the task. Nevertheless, the computational cost of such a high dimensional planning is very high and although they attempt to model uncertainty in dynamic environments, the system is limited to the expected casuistry when programming the behaviour. Instead of considering the entire robot as a single system, works such as [19, 20] treat the arm and base independently, but use the force feedback from the arm to generate velocity commands for the base.

Although traditional methods have led to promising mobile manipulation skills in some specific tasks, they require the explicit programming of hard-to-engineer behaviours and often fail in more complex tasks where the decision-making process is hard. In addition, such solutions are generally very inflexible and error-prone due to the impossibility of modelling all the uncertainty of dynamic industrial environments when those are programmed.

Alternatively, data-driven approaches address the main limitations of traditional methods and propose to learn robotic behaviours from real experience [3], thus alleviating the cost of modelling complex behaviours. For instance, in one of the first approaches Lin and Goldenberg [21] use DL to model the motion control of a mobile manipulator using real experience. This approach allows them to use deep neural networks to model the uncertainties of the environment, which leads to a more robust controller compared to traditional ones. Later, Konidaris et al. [22] propose to use RL to automatise the skill acquisition on a mobile manipulator. Unlike DL, RL allows to automatically obtain the experience needed to learn robotic skills through trial-and-error and allows to learn complex decision-making policies.

Other works such as [2] use ML to learn the positioning of a mobile manipulator. In that work Stulp et al. propose to learn a concept of "place" (areas where the base of the mobile manipulator has to be placed to guarantee the reach of the arm to the object) for mobile manipulation operations using ML. Specifically, the authors propose to learn a probabilistic ML model using experience obtained through trial and error. The learnt model allows them to predict offline poses of the base which allow the arm to reach the object. As the authors claim, the explicit modelling of the problem is no longer required since the learnt models are grounded in real experience.

Recently, the combination of DL and RL, also known as DRL, has made it possible to tackle complex decision-making problems that were previously unfeasible. It combines the ability of DL to model very high dimensional data with the ability of RL to model decision-making agents through trial and error. In fact, it has become the technology of choice for learning complex robotic behaviours using the

experience gained through interaction with the environment [23]. DRL has been successfully applied in a wide variety of areas such as robotics, computer vision and gaming [4]. Taking into account the difficulty of modelling complex decision-making robotic skills, DRL offers a promising way to take advantage of the experience gathered interacting with the environment to autonomously learn complex robotic behaviours.

In particular, the field of DRL applied to robotics has recently gained popularity due to the remarkable performance obtained in applications with high decision-making and control complexity. Applications range from manipulation [24, 25], to autonomous navigation [26] and locomotion [27, 28].

An example of the potential of DRL based methods applied to robotic manipulation can be seen in [29]. In that work Kalashnikov et al. propose a vision-based self-supervised DRL framework, called *Qt-opt*, to learn a hand-eye coordinated grasping policy. In fact, they learn a controller for a real robotic arm to pick both known and novel items, by only using an over-the-shoulder RGB camera. Recently, DRL has been applied to learn to manipulate nonrigid objects such as cloths. Specifically, the method proposed by Jangir et al. [30] is based on the Deep Deterministic Policy Gradient (DDPG) [31] DRL algorithm and is only applied in simulation. Recently, Kim et al. [32] used the TD3 DRL algorithm to solve the path planning problem with 2/3-DoF manipulators, and showed that TD3 can be used to plan smoother paths compared to traditional algorithms such as Probabilistic Roadmap Planning [33]. In that work the experimentation is carried out fully in simulation.

DRL has also been successfully applied to learn robot navigation policies. For instance the problem of mapless navigation is tackled in [34], where a velocity controller for the mobile base is learnt making use of an asynchronous variant of DDPG. In fact, the controller is trained in simulation using as input 10-dimensional laser sensor readings, besides to the relative position of the robot with respect to the target, and predicts the target linear and angular velocities for the mobile base. The method is finally assessed in the real robotic system. The problem of large-scale 3D navigation of unmanned aerial vehicles is also addressed in [35] using a recurrent variant of the DDPG algorithm, called RDPG. In that work, authors use RDPG to solve the 3D large scale navigation as a Partially Observable Markov Decision Process (POMDP), with partially observable and uncertain states. Their system is only assessed in simulation.

Concerning locomotion applications, TD3 has also been successfully used to learn the continuous control of biped and quadruped robots in simulation [36, 37]. Unlike the general approach of training DRL agents in simulation, Haarnoja et al. [27] showed that its possible to learn complex control skills with real-world experience. In fact, authors designed the Soft Actor Critic (SAC) sample efficient algorithm and applied it to learn locomotion skills in the real world.

Although much work was done in the fields of manipulation, navigation and locomotion, the application of DRL in mobile manipulation was a totally unexplored world until recently. Seeing the excellent results that DRL has given in the previously mentioned fields, in a first proof-of-concept we demonstrated that it is possible to apply DRL to learn mobile manipulation behaviours [6]. Based on the idea proposed in [2], we learnt a DRL-based positioning policy to drive the robot's base with speed commands to areas that guarantee the reach of the arm to the target object. Unlike the original work in which they learn a classifier that is used offline to predict target poses from the base, our goal was to learn an online control policy for the robot. This approach allows the dynamics of the environment to be taken into account, as decision-making is reactive to the state of the environment. The aim of this work was to demonstrate the feasibility of the approach in simulation.

Subsequently, works such as [38, 39] tried to learn how to jointly control both the base and the arm of a mobile manipulator with DRL. On the one hand, Kindle et al. [38] use the Proximal Policy Optimisation (PPO2) [40] algorithm to perform the whole-body control of a holonomic mobile manipulator to solve the task of picking up an object from a shelf. On the other hand, the approach proposed by Wang et al. [39] was similar to the previous one but applied to a non-holonomic robot. In this case, they also use PPO2 to learn the task of picking up an object on a table. Both applications follow the simulation-to-reality approach, so they perform the training in simulation and then directly transfer the learnt behaviour to reality. Nevertheless, in both applications they limit the degrees of freedom of the arm to simplify the control problem, which can greatly limit the robot's reach.

In this paper, we extend our first proof-of-concept with the aim of improving the method to finally validate it in a real environment. Contrary to some state-of-the-art approaches that try to learn a whole-body reactive controller, we propose to learn a positioning policy for the base taking into account arm's range constraints. To that end, we propose to combine a traditional path planner for the arm with a DRL-based reactive controller for the base. In this way, we avoid limiting the degrees of freedom of the robotic arm, allowing for maximum flexibility in picking. We build on the idea that traditional planning and control methods such as the ones offered in MoveIt! provide the precision needed to manipulate objects. In addition, using a DRL-based reactive controller for the base allows us to take into account the dynamics of the environment during positioning. In fact, we believe that the feedback from the arm planner can help

optimise the base controller to take into account arm's range constraints.

For the first time we use the state-of-the-art TD3 algorithm to model a mobile manipulation behaviour. Furthermore, we propose an improved setup composed of a reward function, a definition of state/action spaces, and a selection of neural architectures to learn a robust positioning policy for the base. Following a simulation-to-reality approach, we first train and validate the method in a realistic simulation of the real workshop. In the benchmark carried out in simulation between the baseline and our approach, we show both, that TD3 and the proposed setup outperform the algorithms and the setup proposed in the baseline work. Once the performance and reliability of the controller is assessed in simulation, the controller is deployed and validated in the real robot.

3 Methodological approach

The developed method focuses on the picking operation of a randomly placed item on a table, using a mobile manipulator. In particular, we concentrate on learning a reactive controller for the base of the mobile manipulator, which by sending velocity commands will drive the robot to areas where the reachability of the arm to the object is ensured. The key novelty of our method is that the velocity controller for the base is optimised considering the feedback of the arm's path planner. The rationale behind our approach is to encourage the base controller to position the robot's base in zones where the arm's planner will likely succeed planning a trajectory up to the target object. In fact, we propose to learn the base controller with real experience obtained through the interaction with the environment. By doing so (1) the robot learns its own physical limitations; (2) it takes into account the uncertainty in both the robot's location and the target object; and (3) the decision making is learnt automatically, thus avoiding the programming of such behaviour.

The developed framework, depicted in Fig. 2, is based on ROS [9] and all the modules of the system are implemented as ROS nodes. The control architecture has been designed with the possibility of being executed both in the simulated and the real environments. On the one hand, we call it agent to the set of software modules involved in the decision-making of the actions to be executed by the robot. On the other hand, we call it environment for all the modules that participate in the execution of the predicted actions, both in simulation and in reality.

As far as the agent is concerned, its main module is the mobile manipulator base controller which is modelled using the TD3 DRL algorithm. In our case, the aim is to train TD3 to be a reactive controller to drive the robot's base with velocity commands and considering the scope of the arm. More details about the agent are given in Sect. 3.2.

To model the environment, we follow the widely used OpenAI Gym interface that is used in most state-of-the-art

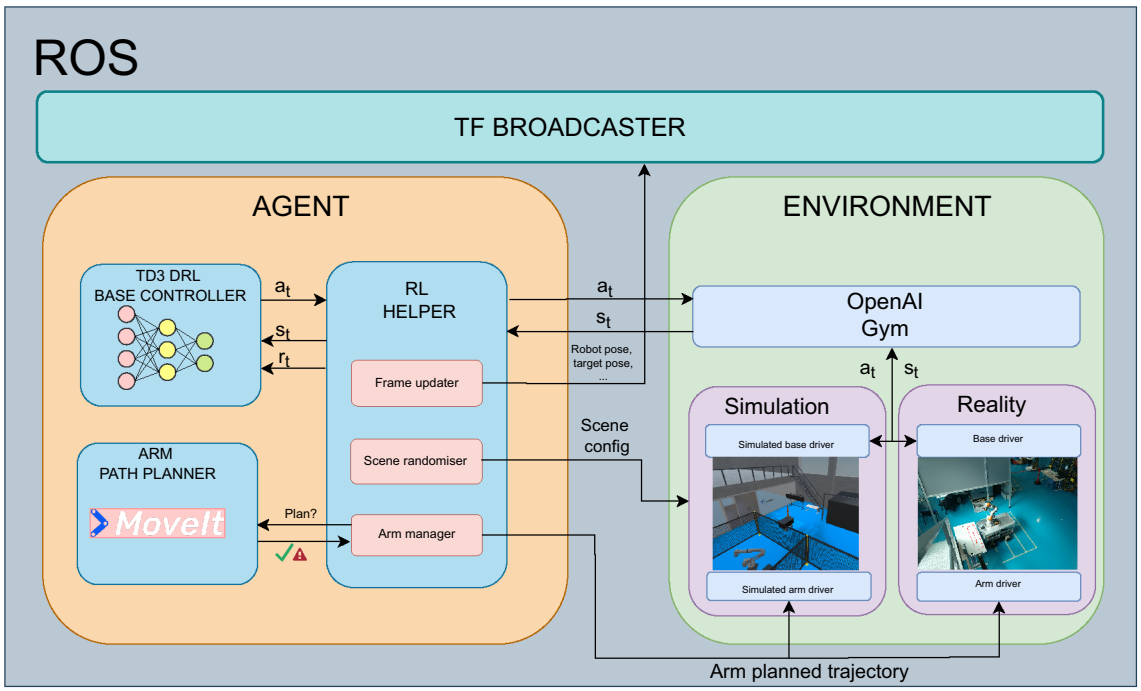


Fig. 2 Developed control framework. ROS nodes are represented with blue boxes. The main libraries used inside the ROS nodes are shown in red. Black arrows indicate ROS service calls

DRL algorithms [10]. OpenAI Gym is a library designed to develop and compare DRL algorithms by providing a standard API to communicate between learning algorithms and environments. Indeed, this communication interface has become a standard to design DRL algorithms and environments and most of the libraries follow it. The use of this API eases the simulation-to-reality transfer of the learnt controller and makes the environment being used transparent to the agent. The details of the implementation of the environment can be found in Sect. 3.1.

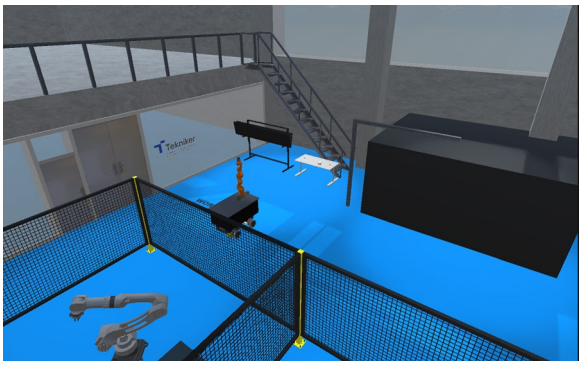


Fig. 3 Simulated environment in Unity

3.1 Environment

As previously mentioned, the environment has been developed following the OpenAI Gym interface, which facilitates the integration with the agent. Furthermore, it makes it transparent to the agent what type of environment the predicted actions are being executed in (simulated or reality), as both the input and output always have the same format.

The main objective of this work is to demonstrate the feasibility of the proposed method in a real environment. Nonetheless, due to the material and time costs involved in training DRL-based agents in a real environment, following the general approach, it is proposed to do so in a simulated environment. For this purpose, a simulation with a high degree of realism has been developed, which is detailed in Sect. 3.1.1. In addition to simulating the robot itself, we have also implemented the necessary drivers to control both the base and the arm of the mobile manipulator in the same way than the real ones.

In order to reduce the gap between simulation and reality and to learn an uncertainty-robust controller, the simulation offers services to randomise both the pose of the robot and the target object. The idea is to perform the training in the realistic simulation, introducing a certain degree of randomisation, so that the learnt controller can be used directly on the real robot.

3.1.1 Simulated environment in Unity

Unity is a real-time 3D development platform that consists of a rendering and physics engine, in addition to a graphical user interface [8]. Although this simulator has been used extensively for game development, it has also been used in many other areas such as the automotive, engineering, and film industries. Unity allows the development of environments rich in visual, physical, and task complexity, which is vital for robot-learning applications. It offers an easy integration and useful toolboxes to work with AI-based models.

The developed simulation has been integrated into the ROS environment using the ROS# library for the C# programming language [41] and following the OpenAI Gym interface. Figure 3 shows the main components of the simulated system, i.e. the mobile manipulator and the table integrated in a realistic environment. The robot is modelled following the URDF format, which is easily loaded to the simulated environment using ROS#.

In order to be able to control the mobile base with velocity commands, a low-level driver is developed taking the idea of the *gazebo planar move* plugin. This plugin implements a simple controller that receives as input a twist command and moves the simulated robot in the *xy* plane. The idea is to replicate the real driver of the base of the robot which allows it to be controlled by speed commands. Additionally, we introduce accelerations to make the simulation more realistic. The acceleration coefficient used has been experimentally measured in the real robot. In addition, the simulated base driver is in charge of publishing the localisation of the robot in the scene. It is widely known that real localisation systems are noisy, and therefore, the localisation is not perfect. However, the simulation tool used does not introduce localisation uncertainty. To somehow overcome this lack and make the system robust to imprecise localisation we add Gaussian noise to the robot pose given by the system at each time step. The modifications made in the localisation are detailed in Eq. 1.

$$\begin{aligned} x_r^w &= x_r^w + \mathcal{N}(0.0, 0.02) \\ y_r^w &= y_r^w + \mathcal{N}(0.0, 0.02) \\ \alpha_r^w &= \alpha_r^w + \mathcal{N}\left(0.0, \frac{\pi}{100}\right) \end{aligned} \tag{1}$$

To control the simulated 7-DoF arm, instead, we follow the idea of the *gazebo ros control* [42] to implement a simulated driver under the *ROS control* framework [43]. This driver acts as a bridge between Unity and the joint trajectory controller offered in MoveIt! and executes the trajectories given by the joint trajectory controller in simulation.

3.2 Agent

According to Sutton and Barto [44], a RL solution to a control problem is defined as a finite-horizon Markov Decision Process (MDP). At each discrete time-step t the agent observes the current state of the environment $s_t \in S$, takes an action $a_t \in A(s_t)$, receives a reward $r : S \times A \to \mathbb{R}$ and observes the new state of the environment s_{t+1}. At each episode of T time-steps, the environment and the agent are reset to their initial poses. The goal of the agent is to find a policy, deterministic $\pi_\theta(s)$ or stochastic $\pi_\theta(a|s)$, parameterised by θ under which the expected reward is maximised.

In our case, we implement the **DRL base controller** using the TD3 algorithm. At each time step $t \in T$ the algorithm takes as input the state of the environment s_t and predicts the optimal velocity command a_t that leads to the expected maximum reward at the end of the episode. In our case, the maximum reward is achieved when the robot positions itself in a zone where the target is within the reach of the arm.

Furthermore, we use two additional nodes that play a key role during learning. On the one hand, we have **Arm path planner**, which determines in each case whether the target is within the reach of the robot arm. As it is later explained in Sect. 3.2.1, the result of the arm planner is used to calculate the reward for the base controller. If the base controller has been able to drive the robot to an area where the arm is able to reach the object, it will receive the maximum possible reward. The idea is to encourage the base controller to drive the robot to areas with a high probability of grasping success. Arm's path planning is done using the MoveIt! library [13] which is, in fact, the default planning and control library for manipulation applications in ROS.

On the other hand, the **RL helper** node is in charge of orchestrating both the learning and the execution of the agent and acts as a bridge to communicate with the environment. The process runs as follows: In an episode of T time-steps, at time instant $t = 0$, it randomises both the pose of the robot and the target using the "scene randomiser" library. The goal is to introduce variability while training to make the controller robust to uncertainties. During the time instants $t = 1$ to $t = T - 1$, it receives the velocity commands a_t predicted by TD3 and executes them in the environment (simulated or real). As a consequence, it obtains the new state of the environment s_{t+1} and on that basis calculates the reward r_t. In addition, using the "frame updater" library, it obtains the state of the environment and spreads the transformations between coordinate systems (robot pose, target, etc.) in ROS using the TF broadcaster. The TF broadcaster node is built on top of the TF library of ROS [45], and allows to dynamically modify transformations between coordinate frames. Finally, at the last instant of time of the episode $t = T - 1$ the robot navigation is stopped. At this instant, the "arm manager" library previously developed and with a higher abstraction level than MoveIt!, calls the planner to see if the object is in the scope of the arm to reward or penalise the base controller accordingly.

3.2.1 TD3 as base controller

To model the base velocity controller, the TD3 algorithm has been chosen, which solves some major stability issues of its predecessor DDPG. Although DDPG has shown great performance learning some robotic skills, it usually tends to be brittle with respect to the tuning hyper-parameters. TD3 follows the actor-critic architecture and, similarly to its predecessor, learns a deterministic policy π_θ in an off-policy way, called actor. Both DDPG and TD3 are derived from the deterministic policy gradient theorem [46]. Based on the idea of a deep Q network (DQN) [47], TD3 also uses action-value functions Q_w to guide the learning process, also dubbed critic. Both the actor and the critic are parameterised functions and usually are implemented as non-linear function approximators. Similar to DDPG, TD3 is able to learn robust value functions due to two innovations: First, the networks are trained off-policy, getting experience samples from a replay buffer to eliminate temporal correlations. In addition, target networks, π' and Q' are used for both the actor and critic respectively, which are updated slower leading to consistent targets y during temporal difference (TD) learning [48].

Figure 4 depicts the network architecture of TD3. TD3 introduces three main novelties: First, it uses a second critic network to improve the stability of the learning process [7]. Thus, the actor is a parameterised policy $\pi_\theta(s) = a$, and critics are action-value functions, $Q_{w_1}(s, a)$ and $Q_{w_2}(s, a)$, which evaluate the quality of the execution of action a in state s. Second, the actor is trained slower than the critics. These less frequent policy updates result in a lower variance action-value estimate that leads to a more robust policy. And third, it adds noise to the action predicted by the target policy, to make it harder to the policy to exploit Q-function errors.

In short, TD3 is made up of one actor and two critic networks, plus the target network of each of them, resulting in 6 neural networks.

To train the networks, in each training iteration, a mini-batch of N transitions (s_t, a_t, r_t, s_{t+1}) is sampled from the replay buffer. On the on hand, to update the weights of the critics w the TD error is used [48], which can be seen in Eq. 2. On the other hand, the actor is updated using the deterministic policy gradient (DPG) theorem [46] which is shown in Eq. 3. However, as previously mentioned the actor is updated slower, usually once per two training steps.

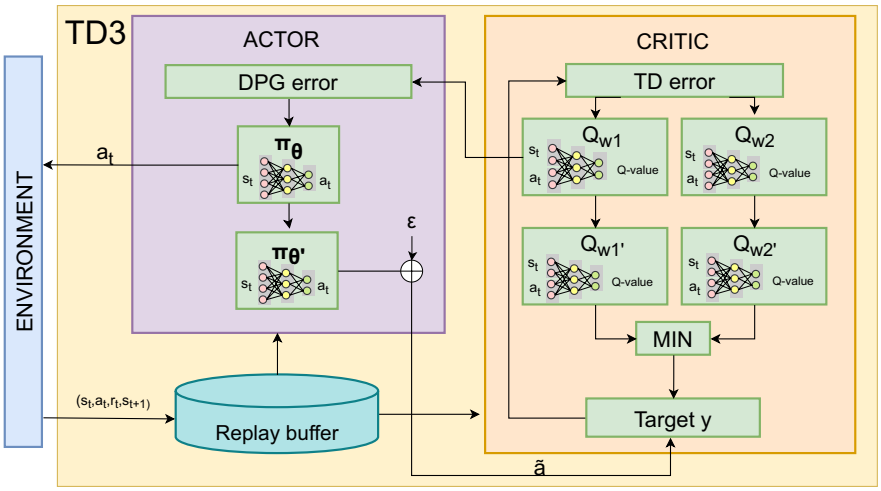


Fig. 4 Network architecture of TD3

$$\begin{aligned} \tilde{a} &\leftarrow \pi_{\theta'}(s) + \epsilon, \epsilon \approx clip(\mathcal{N}(0, \tilde{\sigma}), -c, c) \\ y &\leftarrow r + \gamma min_{i=1,2} Q'_{w'_i}(s_{t+1}, \tilde{a}) \\ w_i &\leftarrow min_{w_i} \frac{1}{N} \sum (y - Q_{w_i}(s, a))^2 \end{aligned} \tag{2}$$

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum \nabla_a Q_{w_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) \tag{3}$$

Concerning the target networks, those are a weighted copy of the original networks and are usually updated at the same frequency than the policy as shown in Eq. 4.

$$\begin{aligned} w'_i &\leftarrow \tau w_i + (1 + \tau) w'_i \\ \theta'_i &\leftarrow \tau \theta_i + (1 + \tau) \theta'_i \end{aligned} \tag{4}$$

For this research, the implementation of the *stable baselines* library is used [49], which already uses the OpenAI Gym interface.

3.2.2 State/action spaces and neural architectures

In the proposed setup, we model both the state and action as continuous spaces as described in Eqs. 5 and 6 respectively. The state observation $s \in \mathbb{R}^{13}$ is composed of the variables defined in Table 1. We modify the baseline state space by adding the observed robot's velocity. It must be noted that in the baseline setup, the predicted action in the previous time step a_{t-1} and the observed velocity v' are the same, but this does not happen in our setup as accelerations are considered. This modification lets the agent be aware of its current velocity in order to decide the velocity command for the next time-step.

The action $a \in \mathbb{R}^3$, instead, is composed of the target linear and angular velocities to be sent to the base of the holonomic robot.

$$s = \left[p_r^w, a_{t-1}, v', p_{target}^w, d_r^{target}, \frac{t}{T} \right] \in \mathbb{R}^{13} \tag{5}$$

$$a = [vx, vy, \omega z] \in \mathbb{R}^3 \tag{6}$$

Concerning the implementation of both the actor and the critics, both of them have been implemented as Multi Layer Perceptron (MLP) feed-forward networks [50].

On the one hand, the MLP of the policy $\pi(s) = a$ gets as input the state observation $s \in \mathbb{R}^{13}$ in the input layer and it is followed by two hidden layers, with 400 and 300 neurons respectively. Finally, the output layer predicts the action $a \in \mathbb{R}^3$. The *ReLU* activation function is used for the hidden layers, and *Tanh*, instead, for the output layer to bind the actions between –1 and 1. The variables that compose the state observation s are normalised before feeding the policy

Table 1 Variables that define the environment state

1. The pose of the base of the arm with respect to the world that is composed of the x and y coordinates and α rotation, where α is the rotation in z axis	$p_r^w = [x_r^w, y_r^w, \alpha_r^w]$
2. The predicted action in the previous time-step	$a_{t-1} = [vx_{t-1}, vy_{t-1}, \omega z_{t-1}]$
3. The observed linear and angular velocities	$v' = [vx', vy', \omega z']$
4. The position of the target with respect to the world	$p_{target}^w = [x_{target}^w, y_{target}^w]$
5. The distance between the base of the arm and the target	d_r^{target}
6. The normalised time-step in the episode	$\frac{t}{T}$

Table 2 Observation and action space limits

Observation space limits			
x_r^w	$[-2.0, 5.6]\ m$	x_{target}^w	$[4.1, 4.2]\ m$
y_r^w	$[-2.8, 2.8]\ m$	y_{target}^w	$[-0.7, 0.4]\ m$
α_r^w	$[-\frac{\pi}{2}, \frac{\pi}{2}]\ rad$	d_r^{target}	$[0 - 6.26]\ m$
$vx', vy', vx_{t-1}, vy_{t-1}$	$[-0.5, 0.5]\ m/s$	$\frac{t}{T}$	$[0 - 1]$
$\omega z', \omega z_{t-1}$	$[-0.5, 0.5]\ rad/s$		
Action space limits			
vx, vy	$[-0.5, 0.5]\ m/s$	ωz	$[-0.5, 0.5]\ rad/s$

MLP using the observation space limits shown in Table 2. The opposite happens with the predicted action that originally is in the [−1,1] range and is converted to the action space range (also shown in Table 2).

On the other hand, the critics $Q(s, a)$ get the $[s, a] \in \mathbb{R}^{16}$ feature vector as input. Then, the input layer is also followed by two hidden layers with 400 and 300 neurons respectively and, finally, the output layer predicts the $\mathbb{R}^1$ Q-value for the state-action pair. The *ReLU* activation function is used for the hidden layers, and for the output layer, however, no activation function is used. In this case, the inputs s and a are also normalised using their respective limits before feeding the critic MLPs.

3.2.3 Reward function

The design of the reward function is one of the most important steps during the modelling of a DRL-based agent, as it is the only signal that guides the learning process. As explained above in Sect. 3.2.1, the rewards are used to optimise the critic networks of TD3, the output of which is then used to optimise the actor. The most logical thing to do would be to reward the robot only when it achieves the goal, i.e. to position itself in such a way that the target object is within the arm's reach. Nevertheless, this usually does not work and it is necessary to reward the agent for achieving sub-goals in order to guide the learning process until an optimal one is achieved. This concept is known as reward shaping [51]. In our case, in addition to rewarding the agent when the main goal is met, we use other criteria such as distance to the goal or speed/acceleration to "shape" the reward function.

The designed reward function is detailed in Eq. 7 and its main components are the following:

- Distance reward, D: The closer the robot is from the target, the higher is the distance reward. When it gets closer than d_{thresh}, the distance reward becomes constant to avoid crashing with the table. This component is described in Eq. 8.
- Velocity penalty, V: The agent is penalised with a discount factor for moving with high velocities, particularly when the robot is closer than d_{thresh} from the goal (see Eq. 9). This discount factor is applied to D.
- Collision penalty, C: If there is a collision between the robot and any element in the environment, this variable takes the value 1. Otherwise, its value is 0.
- Acceleration penalty, W: The agent is penalised for predicting consecutive actions that would require high acceleration (Eq. 10). To this end, the L^2 norm of the difference between a_t and a_{t-1} is used.
- Grasp reward, G: The robot tries to plan a trajectory up to the target in the last time-step of the epoch $t = T - 1$. If planning succeeds, this variable takes the value 1 and, instead, takes the value –1. In the other time-steps of the episode its value is 0.

The weights w_d, w_c, w_w, w_g are used to determine the importance of each component of the reward function.

$$r = w_d \cdot D \cdot V \cdot (1 - C) - w_c \cdot C - w_w \cdot W + w_g \cdot G \tag{7}$$

$$D = \begin{cases} c_1, & \text{if } d_r^{target} < d_{thresh} \\ \frac{1}{d_r^{target}}, & \text{if } d_r^{target} \geq d_{thresh} \end{cases} \tag{8}$$

$$V = \begin{cases} (c_2 - min(\|v'\|_2, c_3))^{\beta}, & \text{if } d_r^{target} < d_{thresh} \\ (c_4 - max(\|v'\|_2, c_5))^{\beta}, & \text{if } d_r^{target} \geq d_{thresh} \end{cases} \\ \text{where, } \beta = \frac{c_6}{max(d_r^{target}, c_7)} \tag{9}$$

$$W = \|a_t - a_{t-1}\|_2 \tag{10}$$

Both the weights of each component and the constant values have been obtained experimentally and are shown in Table 3.

The most critical novelties introduced with respect to the reward function used in [6] are related with the distance reward and the velocity penalty.

On the one hand, in the baseline reward function the robot is rewarded for getting as close as possible to the target.

Table 3 Weights and constant values used for the reward function

Reward function weights and constant values			
w_d	1.0	w_c	1000.0
w_w	10.0	w_g	1000.0
c_1	100.0	c_2	1.0
c_3	1.0	c_4	2.0
c_5	0.5	c_6	2.0
c_7	0.65	d_{thresh}	0.8

However, this introduces a potential risk of collision with the table and, in addition, this could prevent the arm from successfully planning a trajectory to the target for being too close to it.

On the other hand, the baseline simulated low-level driver that is in charge of receiving velocity commands from the DRL agent and moving the robot does not consider accelerations. Because of this the robot is able to stop immediately from one instant of time to another, regardless of its speed. Nonetheless, this makes the baseline simulation to be far from reality and the learnt behaviour in simulation is hardly applicable in the real system. In fact, in the reward function proposed in the baseline setup, the robot is not penalised for moving fast near the goal, and this causes the robot to aggressively approach the target when accelerations are considered, which leads to collisions with the table.

The simulation developed in this work it does consider base accelerations and, thus, the aforementioned issues need to be solved. To that end, on the one hand, we modify the distance reward by giving a constant reward to the robot when the target is within the robot arm's reach. This helps to reduce the importance of the distance at this point and gives more importance the robot to reach to the last the time-step of the episode to perform the grasp. On the other hand, we introduce a velocity penalty to penalise the robot for moving fast near the goal. The absence of the velocity discount factor produces aggressive approximation to the table at high speed and thus, the probability of collision increases. The main goal of this discount factor is to give higher rewards for moving slowly near the target. Moreover, the smoother approximation to the table leads to a better exploration behaviour, and the robot becomes able to wait for the last time-step of the episode in a correct grasping place.

3.3 Training procedure

The training of the agent is done following an episodic scheme. Each training episode consists of $T = 512$ discrete time-steps, with a time-step duration of $T_s = 30ms$. The duration of each episode is around 15 s. At each discrete time-step of the episode $t \in T$, the agent observes the environment state $s_t \in S$ (see Eq. 5), predicts an action $a_t \in A(s_t)$ (see Eq. 6), receives a reward signal $r_t : S \times A \rightarrow \mathbb{R}$ (see Eq. 7) and observes the new state of the environment s_{t+1}. As detailed in Sect. 3.2.1, the transition (s_t, a_t, r_t, s_{t+1}) is finally stored in the replay buffer. At the beginning of each episode, both for training and validation, the initial poses of the robot and the target are randomly selected. The limits of the observation and action spaces used are constant during all the training episodes and are shown in Table 2. The detailed training algorithm of the agent can be seen in Algorithm 1 of Appendix A.

In this way, at each episode the robot has T time-steps to complete the positioning of the robot's base in a suitable zone for the arm to plan a trajectory up to the target. To that end, at each time-step the base controller observes the environment state, predicts the optimal velocity command, and after executing the action, it receives a reward signal. At time-step $t = T - 1$ the path planning trial is done with the arm and the robot is rewarded or penalised depending on whether the planning succeeds or not.

The localisation of the robot is published at $\approx 100Hz$ and the robot control is done at $\approx 33Hz$. The training is done using an Intel Core i7-8700 CPU (@ 3.20 GHz x 12) with a Nvidia Geforce GTX 1060 GPU and each training is executed during 4 M steps in simulation. Whenever the agent reaches a terminal state, the simulation is reset, and a new episode starts. Three terminal states are considered:

1. A collision occurs between the robot and any other element in the scene.
2. The localisation of the robot falls outside the limits defined in the observation space.
3. The robot reaches the last time-step of the episode $t = T - 1$.

The main hyper-parameters used to train the TD3 agent are shown in Table 4.

4 Benchmark in simulation

The main objective of the experimentation is to evaluate the method we propose in comparison with the baseline [6]. First, we want to demonstrate that the TD3 algorithm fits better to the problem at hand and allows learning a more robust behaviour compared to DDPG and PPO2. And second, we want to show that the setup (state/action spaces, the reward function and neural architectures) we propose in this work leads to better performing behaviour, regardless of the DRL algorithm used.

To do this, in addition to TD3, we use the DDPG and PPO2 algorithms proposed in the baseline work to model

Table 4 Training hyper-parameter for the TD3 algorithm

Learning rate (α)	$1e-3$	Gradient steps	100
Target network update rate (τ)	$5e-3$	Batch size (N)	128
Discount factor (γ)	0.99	Policy delay (d)	2
Replay buffer size	$1e6$	Action noise (ϵ)	$\mathcal{N}(0, 0.2)$
Learning starts	$1e4$	Target noise clip (c)	0.5
Training freq	100	Target policy noise (ϵ')	$\mathcal{N}(0, 0.2)$

the base controller. We train each algorithm with both the baseline setup and the one proposed in this paper in the realistic simulation detailed in Sect. 3.1.1. In this way, 6 training runs are carried out in total, one per DRL algorithm/setup combination, and as a result 6 trained agents are obtained.

During the benchmark, the performance and reliability of the agents are quantitatively measured using standard metrics. On the one hand, the comparison during training gives us an idea of the convergence of each algorithm with each setup, as well as the reliability shown during learning. On the other hand, the comparison during the evaluation (using trained agents) shows the real performance and reliability of the agents. The idea is, based on the latter, to select the best algorithm/setup combination and then deploy and validate it on the real robot.

The state/action spaces, reward function, and neural architectures used to train the TD3-based agent are reported in Sect. 3.2.1. In addition the hyperparameters used to tune TD3 are detailed in Sect. 3.3. The baseline setup and the hyperparameters used both for DDPG and PPO2 are those reported in [6].

The algorithm used to evaluate the agents during both training and evaluation can be found in Algorithm 2 of the Appendix A.

4.1 Training runs

While the agents are training, a 5-episode evaluation period is performed at all $1e4$ training steps to assess the quality of the agent learnt at that time. The training procedure followed is detailed in Sect. 3.3. In each evaluation episode, first both the initial poses of the robot and the target are randomly assigned, and then the agent is executed to carry out a picking trial. Taking into account these evaluation periods, the performance and reliability of each algorithm/setup combination is analysed. On the one hand, performance is used to see how fast each algorithm converges based on the chosen setup. On the other hand, the reliability of the model shows how stable the learning process is and what risk there is in the short and long term of a large drop in performance. The metrics used for this are defined in Sect. 4.1.1.

4.1.1 Metrics

To measure the performance of the algorithm/setup combinations while training the following metrics are used:

- Average accumulated rewards: In each evaluation episode the accumulated reward (i.e. the sum of the rewards during the T time-steps of the episode) is computed. Finally, the average accumulated reward is calculated among the 5 evaluation episodes of each evaluation period.
- Success rates: Per each evaluation period of 5 episodes, a success rate is computed which tells us how many times the learnt base controller has been able to drive the mobile manipulator's base to areas where the target is in the scope of the arm. For illustration purposes we show the average success rates considering 5 evaluation periods.

To quantitatively measure and compare the reliability of the algorithms during training, we use the metrics proposed in [52], that focus on measuring the risk and the dispersion of DRL algorithms:

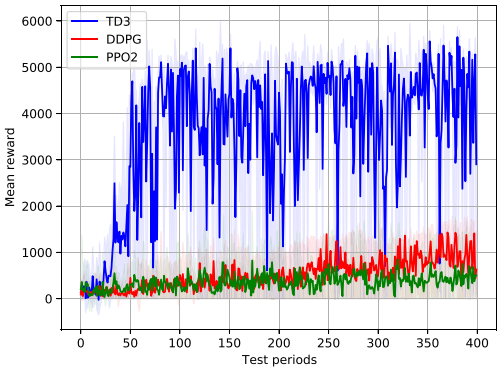
- Dispersion across Time (DT): Measures the dispersion of the average accumulated rewards during training. The dispersion is measured using the inter-quartile range (IQR) [53], in this case the distance between the $75th$ and $25th$ percentiles. This metric measures the short-term variability that corresponds to a noisy training.
- Short-term Risk across Time (SRT): The goal of this metric is to measure the most extreme reward drop from one evaluation step to the next. To that end, the conditional value at risk (CVaR) or the expected shortfall [54] of the differences in the rewards between successive evaluations is measured. CVaR measures the expected value below the α-quantile in the distribution formed by the reward differences. In our experiments we used the default value of $\alpha = 0.05$.
- Long-term Risk across Time (LRT): This metric measures long-term risk during each training run by measuring the most extreme drop in reward with respect to the peak, also called *drawdown* [55]. In this case, CVaR is applied to *drawdown*.

Because each setup has a different reward range, to compute the reliability metrics across the setups, those are converted into rankings as proposed in [52]. In fact, the rankings are first calculated per setup, and finally, the across-setup mean rankings are obtained.

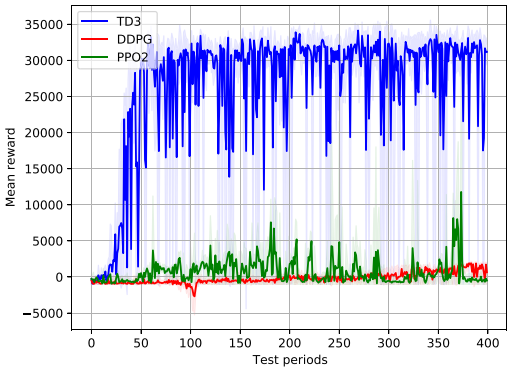
4.1.2 Results

The average accumulated rewards and the success rates obtained during training are depicted in Fig. 5, both for the baseline and the proposed setups.

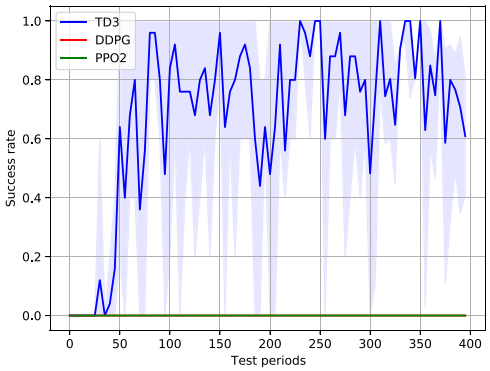
As can be observed in Fig. 5a and b, in both the baseline and the proposed setups, TD3 outperforms DDPG and PPO2 algorithms by far. The average accumulated rewards obtained by TD3 are much higher than the rest, indicating that from the beginning of learning, it has been able to learn a considerably more robust behaviour. Although looking at the rewards we can see the superior behaviour of TD3, it is not easy to know at which points the robot



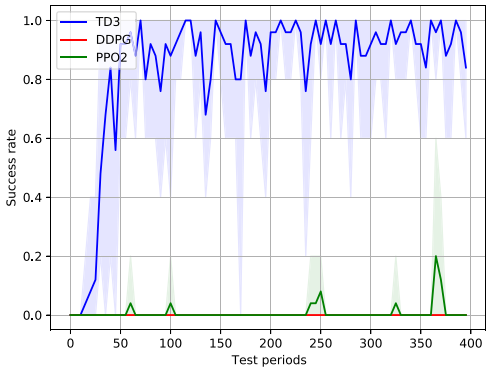
(a) Average accumulated rewards per each evaluation period, using the baseline setup.



(b) Average accumulated rewards per each evaluation period, using the proposed setup.



(c) Average success rates per 5 evaluation periods, using the baseline setup.



(d) Average success rates per 5 evaluation periods, using the proposed setup.

Fig. 5 Training average rewards and success rates with PPO2, DDPG and TD3 algorithms with the baseline and the proposed setups

has been able to successfully complete the task. This is due to reward shaping, as in addition to rewarding the robot for completing the task, it also rewards for a correct approach. Thus, Fig. 5c and d show the average success rates obtained during the training process. The success rates confirm the clearly superior performance of TD3 during training.

Looking at the rewards it is difficult to assess in which setup TD3 achieves better behaviour, as the reward scales are different. However, in Fig. 5c and d it can be clearly seen that with the proposed setup the success rates are higher and more stable during the learning process. Additionally, the combination of TD3 and the proposed setup leads to a faster convergence. While the baseline setup achieves a success rate higher than 80% around the 75th evaluation period, the proposed setup achieves it around the 40th evaluation period.

The dispersion and risk metrics rankings are shown in Fig. 6. The DT metric indicates that the algorithm with the highest dispersion across time is TD3, which can be clearly seen in the average accumulated rewards of Figs. 5a and 5b. Nonetheless, the SRT and LRT metrics indicate that it is the algorithm with the lowest short-term and long-term risk. Despite the less noisy average reward curves for DDPG and PPO2 shown in Fig. 5a and b, success rates in Fig. 5c and d indicate failure in solving the task. In contrast, the success rates for TD3 in the proposed setup suggest a more stable learning curve, with an average success rate above 80% for almost all of the training process.

4.2 Evaluation runs

Although the training performance and reliability metrics give us an intuition of how good each algorithm/setup pair

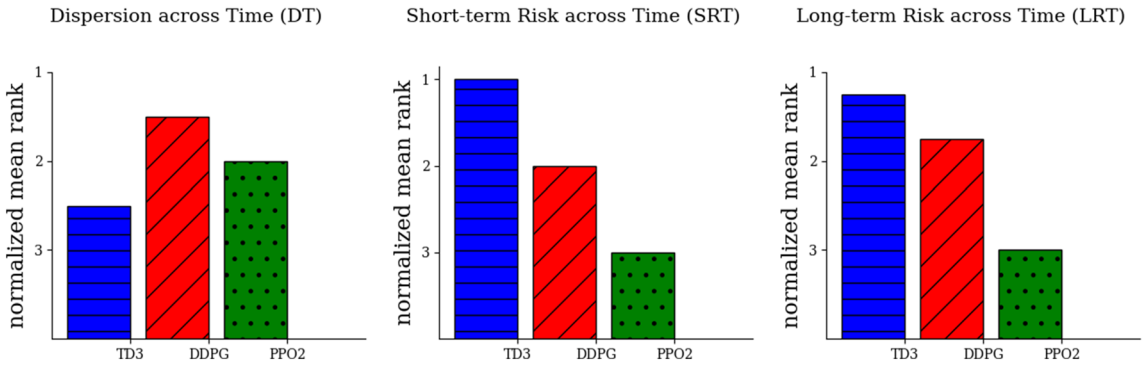


Fig. 6 Mean across-setup reliability rankings of TD3, DDPG and PPO2 algorithms considering the average reward training curves. Rank 1 always means the best reliability

work while the policy is being trained, the final evaluation must be done with fixed policies. Evaluation with fully trained models will give a clear view of the real performance and reliability of each agent. Therefore, once the training runs are completed, for each algorithm/setup pair, we select the model with the highest average accumulated reward for the evaluation. To evaluate them, we run 100 episodes of T time-steps with each fixed model. In each evaluation episode the initial pose of both the robot and the target box are randomly selected before executing the picking trial. Note that the randomly selected poses are common in all the tests, the results to be comparable. This random selection is done using the limits of the observation space (detailed in Table 2).

4.2.1 Metrics

During the evaluation of the trained models, the metrics used to measure the performance were slightly different comparing to those used during the training runs:

- Accumulated median rewards: Since a single evaluation period of 100 episodes is carried out, we just measure the median of accumulated rewards across all the runs.
- Success rate: This metric tells us how many times the learnt base controller has been able to position the mobile manipulator's base in areas where the target is in the scope of the arm. A single success rate is calculated considering the 100 trials.

Similarly as in the training runs, we measure the reliability of the algorithms during the evaluation using the following metrics, also proposed in [52]:

- Dispersion across Fixed-Policy Rollouts (DF): This metric measures the variability in performance when the same policy is run multiple times. This variability is measured using the IQR on the performance of the evaluation runs.
- Risk across Fixed-Policy Rollouts (RF): This metric measures the worst-case scenarios across the evaluation runs under a fixed policy. To that end, CVaR is applied to the performances in the evaluation runs.

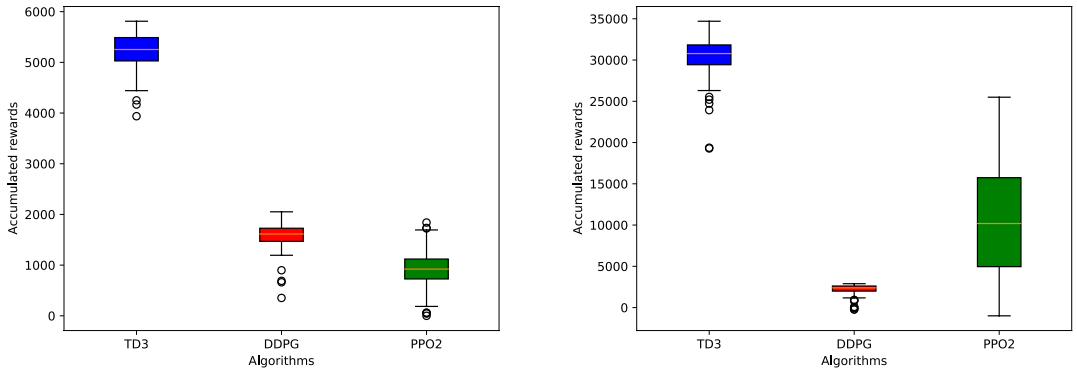
4.2.2 Results

As expected, the TD3 algorithm outperforms DDPG and PPO2 and obtains a higher accumulated median reward in both the baseline and the proposed setups (see Fig. 7). Similar to training, due to the reward shaping, only by looking at the accumulated rewards we cannot know with which of the two setups the TD3 algorithm performs better. Therefore, considering the 100 evaluation runs, we also measure the task success rates shown in Table 5. In contrast to training, where success rates are calculated throughout the learning process, in this case, the evaluation is performed on fixed models that have already been trained, so we obtain a single success rate per training.

The highest success rate is achieved by the combination of TD3 and the proposed setup, and surprisingly, DDPG fails to succeed in both cases. These results show how sensitive DRL-based agents are both to the setup and to the environment in which the agents have been trained.

The reliability metric rankings obtained from the evaluation are shown in Fig. 8. The DF metric indicates that the algorithm with the lowest variability in performance is DDPG. However, its median performance leaves much to be desired in both setups, and the success rates indicate failure in solving the task. Furthermore, according to the RF metric, the algorithm with the lowest risk is TD3, similar to the training runs.

On the one hand, considering the performance and reliability metrics it can be said that TD3 is the best algorithm out of the 3 candidates to learn a positioning policy for the



(a) Accumulated median rewards during the 100 evaluation periods with the baseline setup.

(b) Accumulated median rewards during the 100 evaluation periods with the proposed setup.

Fig. 7 Accumulated median rewards during the 100 evaluation periods with TD3, DDPG and PPO2

Table 5 Success rates for each algorithm/setup combination during the 100 evaluation periods

Success rates (%)	TD3	DDPG	PPO2
Baseline	86	0	0
Proposed	**97**	0	50

mobile manipulator's base that drives it to successful picking areas. On the other hand, the success rates show that the proposed setup enables TD3 to better learn the mobile manipulation task.

As the success rate indicates how many times the mobile manipulator has positioned itself in zones that enable a successful path planning for the arm up to the target, we also assess the quality of the trajectories generated by each controller. To that end, only focusing on the TD3-based agents, we measure the average distance travelled, considering the 100 evaluation runs. The travelled distance is measured taking into account the 3-DoF of the omnidirectional base during the positioning operation. Considering that both the set of initial and target poses are common across tests, a shorter average travelled distance indicates higher quality navigation. Since there are no obstacles between the initial pose and the table on which the target is located, the shortest path is always the best. As can be seen in Table 6, the combination of TD3 with the proposed setup leads to 8.3% shorter trajectories on average.

5 Validation in the real system

Details about the deployment and validation of the learnt controller in the real robotic environment are explained in this section.

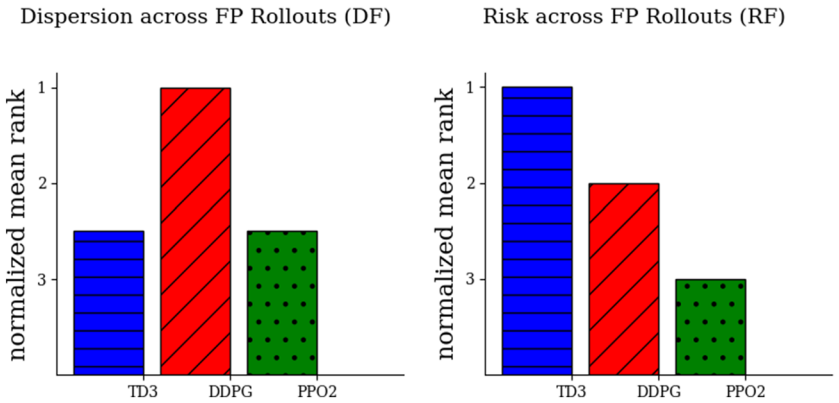


Fig. 8 Mean across-setup reliability rankings of TD3, DDPG and PPO2 algorithms considering the accumulated rewards during the 100 evaluation runs. Rank 1 always means the best reliability

Table 6 Median, mean and standard deviation of the distances travelled by the base of the mobile manipulator during the 100 evaluation periods with the TD3 algorithm

Travelled distance (m)	$\tilde{d}$	$\overline{d}$	σ
Baseline	5.787	6.084	1.543
Proposed	4.753	5.573	2.655

5.1 Considerations before the deployment

5.1.1 Localisation of the base

One of the most important issues is related to the localisation of the robot. In the real robotic system, the pose of the robot is provided by the odometry, which is known to contain a cumulative error. Thus, we use a 2D localisation system with the aim of correcting the robot's pose given by the odometry using a map of the environment.

To that end, the Adaptive Monte Carlo localisation (AMCL) algorithm is used as the 2D localisation algorithm for the robot [56]. This probabilistic algorithm represents the localisation of the robot on a map as a particle filter. In fact, it fuses multiple data sources such as the odometry or laser scans to estimate the position of the robot in the map. We created the map of the real workshop using the *Gmapping* [57] algorithm (Fig. 9). Nevertheless, one of the main limitations of AMCL is that the corrected odometry pose is estimated at a very low frequency ($3 - 7Hz$) due to the computational load of the sensor fusion process.

5.1.2 Arm's path planning to the target

As in the simulation, the path planning for the real mobile manipulator's arm is performed using MoveIt!. The drift between the real and the estimated robot's pose provided by AMCL, however, causes the estimation of the relative pose between the base of the arm and the target not to be accurate enough, and consequently, the planned path can be invalid for manipulating the item. Indeed, an on-board vision system would be needed to correct the localisation error and successfully pick the part.

The main objective of this work is to demonstrate that the DRL-based agent is able to position the mobile manipulator in areas where the target object is within the range of the arm. Therefore, in addition to using the base location given by AMCL to perform the arm planning, we also use the real location calculated manually for verification purposes. The aim of doing this is to remove the localisation error from the system to see if the object is really within range.

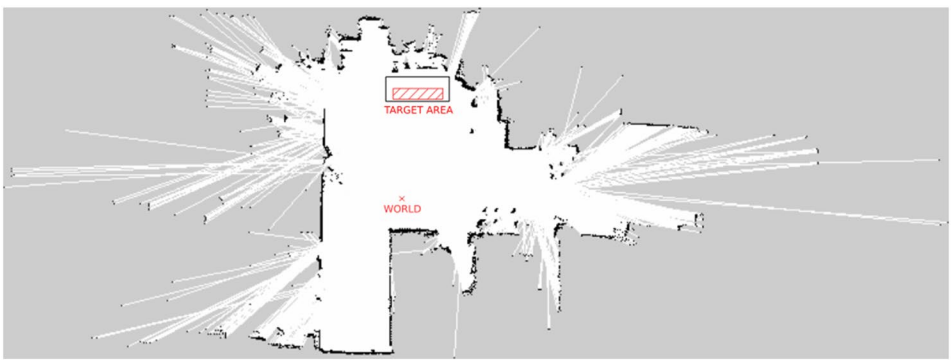
5.2 Validation procedure

We select the TD3-based agent which has been trained with the proposed setup to carry out the evaluation in the real robotic system (see Fig. 10). The evaluation consists of 20 trials where the robot's initial pose is randomly selected in each trial, and the position of the target, instead, is randomly selected once per 5 trials. The tests are carried out as follows:

1. The robot is set in a randomly selected initial pose and the DRL-based base controller is executed for T time-steps.
2. After T time-steps of navigation, once the robot has been positioned, at time-step $t = T - 1$ the arm tries to plan

Fig. 10 Real scenario

Fig. 9 Map of the real scenario



a trajectory up to the target. A successful plan indicates that the target object is within the reach of the robot. This first planning is done using the AMCL localisation. Finally, a success rate is computed considering the results of the plans.

3. The localisation uncertainty makes it impossible to use the robot's pose to deduce whether the target is reachable by the robot arm. Therefore, with the aim of removing the localisation error, the real localisation of the robot is manually measured. This lets us check whether the target is really reachable by the arm according to the arm's planner, and to use it as ground truth.

During each test, we take the following measurements:

- Difference between the observed and the measured real distance between the base of the arm and the target (Δd).
- Difference between the observed and real rotation on the z axis of the base of the arm with respect to the target ($\Delta\alpha$).
- Whether the base of the robot collides with the table when approaching the target (*coll.*).
- The attempt is considered successful if (1) there have been no collisions during the positioning and (2), the robot has been able to position itself in such a way that the arm is able to plan a trajectory to the object. The planning is done using both the localisation given by AMCL (*amcl_success*) and the real measured localisation (*real_success*).

5.3 Results

The measurements taken in the real validation environment are shown in Table 7. According to *amcl_success*, it can be seen that in 75% of the trials the robot is able to position itself in such a way that enables a successful planning of the arm up to the target, considering the localisation given by AMCL.

However, if we focus on the mean error between the estimated and manually measured robot-target distances and rotations at time-step $t = T - 1$ (Δd and $\Delta\alpha$), it can be seen that in average there is an error of $0.055m$ and $0.084rad$ respectively. At a glance, the average error magnitude seems important enough to cause unsuccessful manipulations. Note that the real execution of the manipulation plan is out of the scope at this phase because we want to see the performance of the base moving strategy itself. In fact, the main goal is to see if the DRL controller guarantees the reachability of the arm to the object.

Therefore, to eliminate the localisation error, in each test, a new plan is performed with the arm, considering the manually measured real localisation of the robot (*real_success*). In that case, the success rate is also 75%, which indicates that whenever the robot succeeds the planning using the AMCL localisation, it also succeeds with the real localisation. This indicates that in all cases the object is really within arm's reach. Consequently, we can conclude that although there is indeed an error in localisation, it is not large enough to cause the object to move out of the arm's reach.

Table 7 Measurements taken in the real environment

Trial	Δd (m)	$\Delta\alpha$ (rad)	*coll.*	*amcl_success*	*real_success*
1	0.051	0.088	False	True	True
2	0.092	0.060	True	False	False
3	0.021	0.091	False	True	True
4	0.048	0.072	True	False	False
5	0.187	0.178	True	False	False
6	0.018	0.082	False	True	True
7	0.047	0.065	False	True	True
8	0.003	0.125	False	True	True
9	0.032	0.260	False	True	True
10	0.083	0.059	False	True	True
11	0.034	0.061	False	True	True
12	0.056	0.057	False	True	True
13	0.037	0.047	False	True	True
14	0.019	0.040	False	True	True
15	0.050	0.066	True	False	False
16	0.040	0.081	False	True	True
17	0.088	0.062	False	True	True
18	0.052	0.090	False	True	True
19	0.087	0.097	True	False	False
20	0.053	0.078	False	True	True
Avg.	0.055 ± 0.039	0.084 ± 0.05	25%	75%	75%

The main decrease in performance is caused by small brushes with the table. The main causes for those brushes are the following:

Noisy localisation estimation: The localisation of the robot has a huge importance in the state representation of the base controller. Indeed, the reactive base controller makes its decision at each time-step based on the observation of the state. Although AMCL uses multiple data sources to correct the odometry, it still introduces an error in the localisation. This has a big effect, particularly when the robot navigates near the table. In addition, the low refresh rate of AMCL causes the localisation not to be updated in some iterations of the control loop and, thus, the consequence of each action of the robot is reflected with delay in the updated localisation. This delay in the localisation could cause the agent to make sub-optimal decisions at some critical time-steps of the positioning.

Learning to stop: Although the learnt controller successfully stops the robot when it is already positioned close to the target in simulation, this is not the case in reality. Indeed,

the controller learns to send near-0 velocity commands to stop the robot in simulation, but the real robot struggles to stop and oscillates. This happens due to the noisy localisation estimates, which make the controller believe that the robot is slightly moving when it is actually still. Therefore, the controller tries to correct this error sending opposite velocity commands and causes the robot to oscillate. This effect is aggravated by the sim-to-real gap. Although we add noise to the localisation to simulate this effect while training the agent, the robot fails to properly stop, and this oscillation near the goal sometimes causes the robot to brush the table.

6 Discussion and future work

In this work we propose an improved method to learn a mobile manipulation skill in simulation and show its feasibility in a real robotic system. Specifically, we learn a DRL-based reactive controller for the base which by sending velocity commands is able to position the mobile manipulator's base in zones that enable a successful picking. The rationale behind our approach is to encourage the base controller to position the robot's in zones where the arm's planner will likely succeed planning a trajectory up to the target object.

First, we develop a realistic simulation of the real environment in Unity, which also requires the development of simulated low-level drivers both for the mobile base and the robotic arm. Then, the agent that is in charge of controlling the robot's base is modelled based on the TD3 state-of-the-art DRL algorithm. We train it in simulation through the interaction with the environment and by introducing basic randomisation to make the learnt agent robust to uncertainties. The successful training of the controller requires a careful design of the reward function, as well as a correct definition of state/action spaces and neural architectures.

During the training process in simulation, we benchmark our method with the baseline approach. First we compare the training curves of TD3 with DDPG and PPO2 proposed in the baseline work, with both the proposed and the baseline setup, from the perspective of performance and reliability. Then, the same comparison is done at evaluation, running the trained models for 100 episodes. In both cases, TD3 showed to be the algorithm with the highest performance, and also the most reliable. The success rates indicate that the best performance is obtained by the combination of TD3 and the setup proposed in this work, with a success rate of 97%. Also the average travelled distances indicate that this combination leads to shorter trajectories in average.

Finally, the learnt base controller is deployed and validated in the real system as well. Even though the performance of the controller is remarkable in simulation, it worsens in the real system. In fact, the most problematic step of the execution of the task is when the robot's base needs to stop while positioning near the target. On the one hand, the simulation-to-reality gap and the error introduced by the AMCL cause an oscillation that prevents the robot's base from successfully stopping in the grasping zone. This oscillation is the main source of the brushes with the table and prevents the robot from staying still until the grasping trial is performed. On the other hand, although the manipulation itself is out of the scope of this work, the drift introduced by AMCL in the localisation is big enough to cause unsuccessful manipulations.

Therefore, to assess the real performance of the proposed system, we measure the real position of the robot and we use it as ground truth. By doing so, we intend to effectively measure whether the target object is really reachable for the robotic arm in spite of the localisation error. The performed experiments show that the proposed system successfully learns to position the robot's base in suitable picking areas with a success rate of 75%. Despite the localisation error, the results show that in all cases the object is really within arm's reach. This means that the use of a more accurate localisation system would allow the object to be successfully grasped.

In summary, we show that the proposed system successfully positions the mobile manipulator in zones that ensure the reachability of the arm to the target object. However, due to the error introduced by current localisation systems, the manipulator must have an on-board vision system to be able to accurately estimate the relative pose between the arm's end effector and the target before executing the grasp. In addition, the main future work lines will be focused on adding more sensing capabilities to the agent to increase the safety of the navigation, and on reducing the simulation-to-reality gap. On the one hand, the use of sensors such as lasers, cameras, etc. will let the agent sense the dynamic elements in the environment to safely navigate to the target. On the other hand, more advanced domain randomisation techniques will let us reduce the simulation-to-reality gap. Due to the difficulty in properly simulating the physical properties of both the robot and the environment, domain randomisation techniques suggest to randomise these physical properties in simulation, assuming that the real world properties are a particular case of the randomised variables.

Appendix A Agent training and evaluation algorithms

Algorithm 1 Training of TD3-based agent

Initialise critic networks Q_{w_1} and Q_{w_2}, and actor network π_θ with random parameters w_1, w_2, θ.
Initialise target networks $w_1' \leftarrow w_1$,$w_2' \leftarrow w_2$, $\theta' \leftarrow \theta$
Initialise replay buffer $\mathbb{B}$
$env \leftarrow gym.environment()$
$i \leftarrow 0$
while $i < train\_steps$ **do**
    randomiseScene()
    $s_t \leftarrow env.reset()$
    for $t = 0$ .. $T - 1$ **do**
        Select action with exploration noise $a_t \approx \pi_\theta(s_t) + \epsilon$, $\epsilon \approx \mathcal{N}(0, \sigma)$
        $s_{t+1}, r_t, done \leftarrow env.step(a_t)$
        if $t = T - 1$ **then**
            $plan? \leftarrow doPickingTrial()$
            if $plan?$ **then**
                $r_t \leftarrow r_t + G$
            end if
        end if
        Store transition tuple (s_t, a_t, r_t, s_{t+1}) in $\mathbb{B}$
        if done **then**
            break
        end if
    end for
    if i mod $train\_freq$ **then**
        Sample minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from $\mathbb{B}$
        $\tilde{a} \leftarrow \pi_{\theta'}(s) + \epsilon', \epsilon' \approx clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
        $y \leftarrow r + \gamma min_{i=1,2} Q'_{w_i'}(s_{t+1}, \tilde{a})$
        Update the critics $w_i \leftarrow argmin_{w_i} \frac{1}{N} \sum (y - Q_{w_i}(s, a))^2$
        if i mod d **then**
            Update θ by the deterministic policy gradient
            $\nabla_\theta J(\theta) = \frac{1}{N} \sum \nabla_a Q_{w_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$
            Update the target networks:
            $w_i' \leftarrow \tau w_i + (1 + \tau) w_i'$
            $\theta_i' \leftarrow \tau \theta_i + (1 + \tau) \theta_i'$
        end if
    end if
    $i \leftarrow i + t$
    $s_t \leftarrow s_{t+1}$
end while

Algorithm 2 Evaluation of TD3-based agent

Initialise actor network π_θ with trained parameters θ.
$accumulated\_rewards \leftarrow []$
$successes \leftarrow 0$
$env \leftarrow gym.environment()$
for $i = 0$ .. $eval\_episodes$ **do**
    $accumulated\_reward \leftarrow 0$
    randomiseScene()
    $s_t \leftarrow env.reset()$
    for $t = 0$ .. $T - 1$ **do**
        Select optimal action $a_t = \pi_\theta(s_t)$
        $s_{t+1}, r_t, done \leftarrow env.step(a_t)$
        if $t = T - 1$ **then**
            $plan? \leftarrow doPickingTrial()$
            if $plan?$ **then**
                $r_t \leftarrow r_t + G$
                $successes \leftarrow successes + 1$
            end if
        end if
        $accumulated\_reward \leftarrow accumulated\_reward + r_t$
        if done **then**
            break
        end if
    end for
    $accumulated\_rewards.append(accumulated\_reward)$
end for
$success\_rate \leftarrow \frac{successes}{eval\_episodes}$
$average\_accumulated\_reward \leftarrow average(accumulated\_rewards)$

Acknowledgements This publication has been funded by the Basque Government - Department of Economic Development, Sustainability and Environment - Aid program for collaborative research in strategic areas - ELKARTEK 2021 Program (File KK-2021/00033 TREBEZIA), and the project "5R- Red Cervera de Tecnologías robóticas en fabricación inteligente", contract number CER-20211007, under "Centros Tecnológicos de Excelencia Cervera" programme funded by "The Centre for the Development of Industrial Technology (CDTI)".

Data Availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Sandakalum T, Ang MH Jr (2022) Motion planning for mobile manipulators-a systematic review. Machines 10(2):97. https://doi.org/10.3390/machines10020097
2. Stulp F, Fedrizzi A, Mösenlechner L et al (2012) Learning and reasoning with action-related places for robust mobile manipulation. J Artif Intell Res 43:1–42. https://doi.org/10.1613/jair.3451
3. Kappler D, Pastor P, Kalakrishnan M, et al (2015) Data-driven online decision making for autonomous manipulation. In: Robotics: science and systems, https://doi.org/10.15607/RSS.2015.XI.044
4. Arulkumaran K, Deisenroth MP, Brundage M et al (2017) Deep reinforcement learning: a brief survey. IEEE Signal Process Mag 34(6):26–38. https://doi.org/10.1109/MSP.2017.2743240
5. Yang X, Xu Y, Kuang L et al (2021) An information fusion approach to intelligent traffic signal control using the joint methods of multiagent reinforcement learning and artificial intelligence of things. IEEE Trans Intell Transp Syst. https://doi.org/10.1109/TITS.2021.3105426
6. Iriondo A, Lazkano E, Susperregi L et al (2019) Pick and place operations in logistics using a mobile manipulator controlled

with deep reinforcement learning. Appl Sci 9(2):348. https://doi.org/10.3390/app9020348
7. Fujimoto S, Hoof H, Meger D (2018) Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning, PMLR, p 1587–1596, https://proceedings.mlr.press/v80/fujimoto18a.html
8. Juliani A, Berges VP, Teng E, et al (2018) Unity: a general platform for intelligent agents. arXiv preprint arXiv:1809.02627
9. Quigley M, Conley K, Gerkey B, et al (2009) Ros: an open-source robot operating system. In: ICRA workshop on open source software, Kobe, Japan, p 5, http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf
10. Brockman G, Cheung V, Pettersson L, et al (2016) Openai gym. arXiv preprint arXiv:1606.01540
11. Siciliano B, Khatib O (2016) Springer handbook of robotics. Springer, https://link.springer.com/content/pdf/10.1007%2F978-3-319-32552-1.pdf
12. Marder-Eppstein E, Berger E, Foote T, et al (2010) The office marathon: robust navigation in an indoor office environment. In: IEEE international conference on robotics and automation, IEEE, p 300–307, https://doi.org/10.1109/ROBOT.2010.5509725
13. Coleman D, Sucan I, Chitta S, et al (2014) Reducing the barrier to entry of complex robotic software: a moveit! case study. arXiv preprint arXiv:1404.3785https://doi.org/10.6092/JOSER_2014_05_01_p3
14. Dömel A, Kriegel S, Kaßecker M et al (2017) Toward fully autonomous mobile manipulation for industrial environments. Int J Adv Robot Syst 14(4):1729881417718588. https://doi.org/10.1177/1729881417718588
15. Xu J, Harada K, Wan W, et al (2020) Planning an efficient and robust base sequence for a mobile manipulator performing multiple pick-and-place tasks. In: IEEE International Conference on Robotics and Automation (ICRA), IEEE, p. 11018–11024, https://doi.org/10.1109/ICRA40945.2020.9196999
16. Padois V, Fourquet JY, Chiron P (2006) From robotic arms to mobile manipulation: On coordinated motion schemes. In: Intelligent Production Machines and Systems. Elsevier, p 572–577, https://hal.archives-ouvertes.fr/hal-00624374/file/2006ACTI1475.pdf
17. Tan J, Xi N, Wang Y (2003) Integrated task planning and control for mobile manipulators. Int J Robot Res 22(5):337–354. https://doi.org/10.1177/0278364903022005004
18. Berntorp K, Arzén KE, Robertsson A (2012) Mobile manipulation with a kinematically redundant manipulator for a pick-and-place scenario. In: Control Applications (CCA), 2012 IEEE International Conference on, IEEE, p 1596–1602, https://doi.org/10.1109/CCA.2012.6402361
19. Meeussen W, Wise M, Glaser S, et al (2010) Autonomous door opening and plugging in with a personal robot. In: Robotics and Automation (ICRA), IEEE International Conference on, IEEE, p 729–736, https://doi.org/10.1109/ROBOT.2010.5509556
20. Ibarguren A, Daelman P (2021) Path driven dual arm mobile comanipulation architecture for large part manipulation in industrial environments. Sensors 21(19):6620. https://doi.org/10.3390/s21196620
21. Lin S, Goldenberg AA (2001) Neural-network control of mobile manipulators. IEEE Trans Neural Netw 12(5):1121–1133. https://doi.org/10.1109/72.950141
22. Konidaris G, Kuindersma S, Grupen R, et al (2011) Autonomous skill acquisition on a mobile manipulator. In: Twenty-Fifth AAAI Conference on Artificial Intelligence, https://doi.org/10.1609/aaai.v25i1.7982
23. Ibarz J, Tan J, Finn C et al (2021) How to train your robot with deep reinforcement learning: lessons we have learned. Int J Robot Res 40(4–5):698–721. https://doi.org/10.1177/0278364920987859
24. Mohammed MQ, Chung KL, Chyi CS (2020) Review of deep reinforcement learning-based object grasping: techniques, open challenges and recommendations. IEEE Access. https://doi.org/10.1109/ACCESS.2020.3027923
25. Hansen J, Hogan F, Rivkin D, et al (2022) Visuotactile-rl: learning multimodal manipulation policies with deep reinforcement learning. In: 2022 International Conference on Robotics and Automation (ICRA), IEEE, p 8298–8304, https://doi.org/10.1109/ICRA46639.2022.9812019
26. Zhu K, Zhang T (2021) Deep reinforcement learning based mobile robot navigation: a review. Tsinghua Sci Technol 26(5):674–691. https://doi.org/10.26599/TST.2021.9010012
27. Haarnoja T, Ha S, Zhou A, et al (2018) Learning to walk via deep reinforcement learning. arXiv preprint arXiv:1812.11103arXiv:org/pdf/1812.11103pdf
28. Peng XB, Berseth G, Yin K et al (2017) Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Trans Graphics (TOG) 36(4):1–13. https://doi.org/10.1145/3072959.3073602
29. Kalashnikov D, Irpan A, Pastor P, et al (2018) Scalable deep reinforcement learning for vision-based robotic manipulation. In: Conference on Robot Learning, PMLR, p 651–673, https://proceedings.mlr.press/v87/kalashnikov18a.html
30. Jangir R, Alenyà G, Torras C (2020) Dynamic cloth manipulation with deep reinforcement learning. In: IEEE International Conference on Robotics and Automation (ICRA), IEEE, p 4630–4636, https://doi.org/10.1109/ICRA40945.2020.9196659
31. Lillicrap TP, Hunt JJ, Pritzel A, et al (2016) Continuous control with deep reinforcement learning. arXiv preprint arXiv: 1509.02971
32. Kim M, Han DK, Park JH et al (2020) Motion planning of robot manipulators for a smoother path using a twin delayed deep deterministic policy gradient with hindsight experience replay. Appl Sci 10(2):575. https://doi.org/10.3390/app10020575
33. Hsu D, Latombe JC, Kurniawati H (2006) On the probabilistic foundations of probabilistic roadmap planning. Int J Robot Res 25(7):627–643. https://doi.org/10.1177/0278364906067174
34. Tai L, Paolo G, Liu M (2017) Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, p 31–36, https://doi.org/10.1109/IROS.2017.8202134
35. Wang C, Wang J, Shen Y et al (2019) Autonomous navigation of UAVs in large-scale complex environments: a deep reinforcement learning approach. IEEE Trans Veh Technol 68(3):2124–2136. https://doi.org/10.1109/TVT.2018.2890773
36. Dankwa S, Zheng W (2019) Modeling a continuous locomotion behavior of an intelligent agent using deep reinforcement technique. In: IEEE 2nd International Conference on Computer and Communication Engineering Technology (CCET), p 172–175, https://doi.org/10.1109/CCET48361.2019.8989177
37. Khoi P, Giang N, Tan H (2021) Control and simulation of a 6-DOF biped robot based on twin delayed deep deterministic policy gradient algorithm. Indian J Sci Technol 14(30):2460–2471. https://doi.org/10.17485/IJST/v14i30.1030
38. Kindle J, Furrer F, Novkovic T, et al (2020) Whole-body control of a mobile manipulator using end-to-end reinforcement learning. arXiv preprint arXiv:2003.02637
39. Wang C, Zhang Q, Tian Q et al (2020) Learning mobile manipulation through deep reinforcement learning. Sensors 20(3):939. https://doi.org/10.3390/s20030939
40. Schulman J, Wolski F, Dhariwal P, et al (2017) Proximal policy optimization algorithms. p 1–12. 06347 arXiv preprint arXiv:1707.06347

41. Bischof M (2018) ROS-SHARP. https://github.com/siemens/ros-sharp, Accessed 16 Jan 2023
42. Qian W, Xia Z, Xiong J, et al (2014) Manipulation task simulation using ROS and gazebo. In: IEEE International Conference on Robotics and Biomimetics (ROBIO 2014), IEEE, p 2594–2598, https://doi.org/10.1109/ROBIO.2014.7090732
43. Chitta S, Marder-Eppstein E, Meeussen W, et al. (2017) rocontrol: a generic and simple control framework for ROS. The Journal of Open Source Software. DOIurlhttps://doi.org/10.21105/joss.00456
44. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction, vol 1. MIT press Cambridge, https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
45. Foote T (2013) tf: The transform library. In: Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on, Open-Source Software workshop, p 1–6, https://doi.org/10.1109/TePRA.2013.6556373
46. Silver D, Lever G, Heess N, et al (2014) Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on International Conference on Machine Learning, Vol. 32, ICML'14, p I-387-I-395, http://proceedings.mlr.press/v32/silver14.pdf
47. Mnih V, Kavukcuoglu K, Silver D et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://doi.org/10.1038/nature14236
48. Sutton RS (1988) Learning to predict by the methods of temporal differences. Mach Learn 3(1):9–44. https://doi.org/10.1007/BF00115009
49. Hill A, Raffin A, Ernestus M, et al (2018) Stable baselines. https://github.com/hill-a/stable-baselines
50. Murtagh F (1991) Multilayer perceptrons for classification and regression. Neurocomputing 2(5–6):183–197. https://doi.org/10.1016/0925-2312(91)90023-5
51. Ng AY, Harada D, Russell S (1999) Policy invariance under reward transformations: theory and application to reward shaping. In: ICML, p 278–287
52. Chan SC, Fishman S, Canny J, et al (2020) Measuring the reliability of reinforcement learning algorithms. In: International Conference on Learning Representations, Addis Ababa, Ethiopia, https://openreview.net/pdf?id=SJlpYJBKvH
53. Riaz M (2015) On enhanced interquartile range charting for process dispersion. Qual Reliab Eng Int 31(3):389–398. https://doi.org/10.1002/qre.1598
54. Acerbi C, Tasche D (2002) Expected shortfall: a natural coherent alternative to value at risk. Econ Notes 31(2):379–388. https://doi.org/10.1111/1468-0300.00091
55. Chekhlov A, Uryasev S, Zabarankin M (2005) Drawdown measure in portfolio optimization. Int J Theor Appl Financ 8(01):13–58. https://doi.org/10.1142/S0219024905002767
56. Fox D, Burgard W, Dellaert F, et al (1999) Monte carlo localization: Efficient position estimation for mobile robots. AAAI/IAAI (343-349):2–2. http://robots.stanford.edu/papers/fox.aaai99.pdf
57. Grisetti G, Stachniss C, Burgard W (2007) Improved techniques for grid mapping with Rao-Blackwellized particle filters. IEEE Trans Robot 23(1):34–46. https://doi.org/10.1109/TRO.2006.889486