2018 European Control Conference (ECC)
June 12-15, 2018. Limassol, Cyprus

Deep Reinforcement Learning for Collision Avoidance of Robotic Manipulators

Bianca Sangiovanni[1], Angelo Rendiniello[1], Gian Paolo Incremona[2], Antonella Ferrara[1] and Marco Piastra[1]

Abstract— **In this paper a real-time collision avoidance approach using machine learning is presented for safe human-robot coexistence. More specifically, the collision avoidance problem is tackled with Deep Reinforcement Learning (DRL) techniques, applied to robot manipulators with a workspace invaded by unpredictable obstacles. Since the robotic systems are defined in the continuous space, a Normalized Advantage Function (NAF) model-free algorithm has been used. In order to assess the proposal, a robotic system, that is a COMAU-SMART3-S2 anthropomorphic robot manipulator, has been considered. The robotic system has been interfaced with external tools for evaluation, control, and automatic training. Simulations carried out on a virtual environment are finally reported to show the effectiveness of the proposed model-free deep reinforcement learning algorithm.**

I. Introduction and Motivation

The use of friendly-robots as multipurpose service assistants in the daily life and industry, giving rise to the so-called physical Human-Robot Interaction (pHRI), is nowadays a reality, due to the needs of performing complex or physically demanding tasks [1]. Since a close cooperation between humans and robots is required, in order to merge human motions and high performance of robots in terms of speed and precision, safety-oriented control strategies are mandatory in order to make a safe human-robots coexistence feasible. Hence, safety is the essential feature of pHRI, without which coexistence and collaboration cannot take place [2].

Particularly, collision avoidance is one of the most important challenges in pHRI. As discussed in [3], a typical real-time collision avoidance method consists of three parts: i) perception of the environment; ii) collision avoidance algorithms; and iii) robot control. A more detailed architecture of a collision avoidance method is reported in [4] where seven phases (pre-collision, detection, isolation, identification, classification, reaction and post-collision) in a collision avoidance pipeline and their outputs are reported.

Over past years, several planning and control approaches for obstacle avoidance have been proposed in the literature. Compliant mechanical design of manipulators [5] and collision detection/reaction strategies based on the use of exteroceptive sensors are among the possible solutions. Other approaches are based on the redundancy of the degrees of freedom of the robot, which becomes able to track the desired trajectory while avoiding obstacles [6]. Furthermore, many real-time planning algorithms are based on the so-called potential field approach introduced in [7] and further improved as in [8]. This method is based on the fact that a virtual repulsive field is associated to obstacles to avoid them, while an attractive field is associated to the target in order to reach the goal. Real-time adaptive motion planning algorithms have been also proposed relying on parametrized collision-free trajectories, the parameters of which are updated according to the environment changes discovered by the robot sensors [9]. In [10] robot and human are represented by a number of spheres and a collision-free trajectory is designed relying on the exploration of the possible end-effector movements in predefined directions. In [11] obstacles located in the robot workspace are approximated in a conservative way with circles in the configuration space, allowing to obtain an efficient trajectory planning and tracking. A virtual spring concept is proposed in [12] to generate collision-free motion trajectories also incorporating the prediction of human behavior. The majority of algorithms needs information about the environment and the obstacles. Basically, collision avoidance algorithms are based on a measure of the distances between the robot and the obstacles, received for instance directly from an image of the environment, by expanding the convex hull associated to the robot until the image associated to an obstacle is reached [13]. Recent works use instead depth data through visual feedback in order to compute the information needed for collision avoidance [3].

In this paper, the collision avoidance problem is tackled with Deep Reinforcement Learning (DRL) techniques, applied to robots in a virtual scenario involving a reaching task with an unpredictable obstacle invading the robot's workspace. This approach uses a general framework to accomplish specific tasks [14]. For systems with higher degrees of complexity, like robotic systems, model-free approaches based on Deep Neural Networks (DNN) for Q-function approximation have been proved to work [15]. In order to apply such methods to robotic systems defined in the continuous space, the Normalized Advantage Function (NAF) algorithm has been introduced in [16]. Differently from the majority of collision avoidance approaches, the proposed one is a model-free approach, thus considerably reducing the modelization and implementation phase for the designer. Moreover, it represents an easy to implement solution in field application, overcoming the problems related to the passage from the offline trajectory generation and the online update and execution. To the best of the authors' knowledge, the proposed approach is one of

[1]B. Sangiovanni, A. Rendiniello, A. Ferrara and M. Piastra are with Dipartimento di Ingegneria Industriale e dell'Informazione, University of Pavia, Via Ferrata 5, 27100, Pavia, Italy ({bianca.sangiovanni01, angelo.rendiniello01}@universitadipavia.it, {marco.piastra, antonella.ferrara}@unipv.it)

[2]G. P. Incremona is with Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133, Milano, Italy (gianpaolo.incremona@polimi.it)

the first examples of application of DRL techniques to robot manipulators for collision avoidance.

II. Basics on Reinforcement Learning

The essential concepts of Reinforcement Learning (RL) are perception, action and goal. The idea is that an *agent*, having learned from past experience, understands which *actions* will lead to maximize a numerical result (*reward*) in a given time horizon, for any given situation (*state*). In fact, the typical challenge of many RL problems is to maximize not only the best current reward, but also the best future ones, which can be obtained by influencing future *states* through actions. The agent must be able to perceive the state of the environment and take actions that will affect it, trying to reach one or more goals related to the accomplishment of a predefined task. Hence, RL takes into account two aspects: *exploitation* and *exploration*. The first aspect consists in exploiting what the agent already knows to obtain reward, the latter means exploring other actions in order to make better selections in the next steps. Looking for a trade-off between exploitation and exploration is one of the most challenging problems concerning RL. A strategy that favors the exploitation at the cost of exploration is called greedy.

A. Agent and Environment

At each time step t, the agent and environment can be modeled as a *state* $s_t \in \mathcal{S}$ with $\mathcal{S}$ being the state space, which contains all the relevant information about the system. Starting from a given state, the agent performs an action $a_t \in \mathcal{A}$ with $\mathcal{A}$ being the action space. Each action perturbs the environment and hence changes its state. Before advancing to the next time step $t+1$, the agent receives a *reward* $r \in \mathcal{R}$, with $\mathcal{R}$ being the reward space, and moves to the new state s_{t+1}, according to the *dynamics* indicated as $\rho(s_{t+1}\,|\,s_t, a_t)$. $\mathcal{S}$ and $\mathcal{A}$ can be either continuous or discrete sets, while the mapping from states to action is dictated by a policy π, that can be either deterministic or probabilistic. For a given control task, the learning process is divided into *episodes*, where the agent interacts with the environment for either a complete attempt to perform a goal task or a fixed number of time-steps before being reset. The whole training process includes a typically large number of such episodes, up to a predefined maximum.

B. Rewards and Policy

A *reward* r_t is a scalar feedback signal that indicates "how well" an agent has done at step t. The agent's goal is to maximize the (expected) cumulative reward it receives in the long run. In case of episodic tasks with finite horizon T, the expected cumulative reward R_t is defined as

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{1}$$

where the term $0 \leq \gamma \leq 1$ is the discount rate, used to prioritize earlier rewards over later ones. In RL, a Markov Decision Process (MDP), defined by the state and action sets and the transition probability matrix of the environment, is used. Given the current state s, action a and the next state s', the expected value of the next reward is defined as:

$$r(s,a,s') = \mathbb{E}\left[r_{i+1}|s_t, a_t, s_{t+1}\right] . \tag{2}$$

A *policy* $\pi(a|s)$ is the probability that the agent will perform action a while in state s. Depending on the task, the policy can be represented as a simple function or lookup table, or as a more complex deterministic or probabilistic function that requires extensive computation. The expected cumulative reward the agent can accumulate in the long run, starting from a certain state and following the policy π, is called *value function*.

C. Deep Reinforcement Learning

The MDP requires that the model, i.e., the transition probability, of the environment is completely observable. If this information is not available, which is usually the case of complex systems such as robotic systems, a different learning model is required. Q-learning is an off-policy algorithm that directly approximates the optimal action-value function $Q^\star$ independently of the policy being followed. While selecting the action, the agent can carry on exploring the environment with probability ε, by randomly choosing an action and ignoring its previous knowledge.

Q-learning can however become infeasible in case of continuous action problems with a large number of states. The problem could be overcome with the use of a parametric approximator of the action-value function, or Q-function. One way to build such approximator is a Deep Neural Network (DNN). A DNN is a parametric function that can model complex non-linear relationships. The term *deep* refers to the level of composition of the parameters and the use of multiple hidden layers between the input and output ones. Consider a parametrized action-value function $\tilde{Q}\left(s_t, a_t|\theta^Q\right)$ such that

$$\tilde{Q}\left(s,a|\theta^Q\right) = \tilde{A}\left(s,a|\theta^A\right) + \tilde{V}\left(s|\theta^V\right) \tag{3}$$

where $\tilde{A}$ and $\tilde{V}$ are approximators of advantage function $A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t)$ and the value function $V^\pi(s_t)$, respectively. The goal is to minimize a *loss function*

$$L(\theta^Q) = \mathbb{E}_{r,\rho^\beta,\beta}\left[\left(\tilde{Q}\left(s_t,a_t|\theta^Q\right) - y_t\right)^2\right] , \tag{4}$$

where the term y_t is the target

$$y_t = r(s_t,a_t) + \gamma\tilde{Q}\left(s_{t+1}, \mu(s_{t+1})|\theta^Q\right) , \tag{5}$$

θ^Q are parameters of the action-value functions, β is a stochastic behavior policy such that $a_t = \beta(s_t)$, and ρ^β is the state visitation frequency with policy β. The computation of the greedy policy $\mu(s_t,a_t) = \text{argmax}_a \tilde{Q}(s_t,a_t)$ is however, a major issue: the computational load of the maximization of a non-linear, complex function at each time step could be unfeasible and not suited for real-time processes such as robotics.

D. Normalized Advantage Function

Normalized Advantage Function (NAF), introduced in [16], is a method that allows one to enable Q-learning in continuous

action spaces with deep neural networks. The idea behind this method is to design the Q-function in a way that its maximum $\text{argmax}_a Q(s_t, a_t)$ can be easily computed during each update. More specifically, the algorithm uses a DNN to separately output the terms $\tilde{A}$ and $\tilde{V}$, where the advantage term is parametrized as a quadratic function,

$$\tilde{A}\left(s, a|\theta^A\right) = -\frac{1}{2}\left(a - \tilde{\mu}(s|\theta^\mu)\right)^T P(x|\theta^P)\left(a - \tilde{\mu}(s|\theta^\mu)\right) . \tag{6}$$

The term $P\left(s|\theta^P\right)$ is a state dependent, positive-definite square matrix. The Q-function is quadratic in the action and by computing the maximum, one has

$$\frac{\partial}{\partial a}\tilde{Q}(s, a|\theta^Q) = \left(\tilde{\mu}(s|\theta^\mu) - a\right)^T P(x|\theta^P) = 0 . \tag{7}$$

Therefore the action a that maximizes $Q(s,a)$ is always

$$a = \tilde{\mu}(s|\theta^\mu) . \tag{8}$$

The NAF algorithm is structured as in Algorithm 1 where TAR and PRED are two different networks, built in order to update the present target parameters θ^Q_{TAR} using the present predicted parameters θ^Q_{PRED} and the previous target parameters. The noise element $\mathcal{D}$ is a stochastic process added to the action to foster exploration, even though the algorithm uses a greedy policy. The Replay Buffer *RB* collects all the samples $i = \{s_t, a_t, r_t, s_{t+1}\}$ produced by the training process throughout the episodes. The Mini Batch *MB*, instead, contains a fixed number of samples randomly selected from *RB*; it can be normalized with unit variance and zero mean. Finally, the coefficients γ, η, and τ are the discount factor, the learning factor and the update factor, respectively.

Algorithm 1 NAF algorithm for continuous Q-learning

Randomly initialize $\tilde{Q}(s, a|\theta^Q_{PRED})$ $\quad \theta^Q := (\theta^\mu, \theta^P, \theta^V)$
Initialize the target network with $\theta^Q_{\text{TAR}} \leftarrow \theta^Q_{\text{PRED}}$
Initialize replay buffer $RB \leftarrow 0$
for each episode do:
  Initialize random process $\mathcal{D}$ for action exploration
  $s_0 \leftarrow Environment\,(reset)$
  for $t = 0$ to T do:
    $a_t \leftarrow \mu\left(s_t|\theta^\mu_{\text{PRED}}\right) + \mathcal{D}_t$
    $r_t \leftarrow r(s_t, a_t)$
    $s_{t+1} \leftarrow Environment\,(s_t, a_t)$
    $RB \leftarrow RB \cup \{(s_t, a_t, r_t, s_{t+1})\}$ store transition in RB
    Sample at random and normalize the mini batch MB
    for each sample $i = (s_i, a_i, r_i, s_{i+1})$ in MB
      $y_i = r_i + \gamma\tilde{V}\left(s_{i+1}|\theta^V_{TAR}\right)$
      Compute gradients
      $\frac{\partial}{\partial\theta^Q}\left(y_i - Q\left(s_i, a_i|\theta^Q_{\text{PRED}}\right)\right)^2$ (Loss function $L(\theta^Q)$)
      $\theta^Q_{\text{PRED}} \leftarrow \theta^Q_{\text{PRED}} - \eta\left(\overline{\frac{\partial}{\partial\theta^Q}L(\theta^Q)}\right)$
      $\theta^Q_{\text{TAR}} \leftarrow \tau\theta^Q_{\text{PRED}} + (1+\tau)\theta^Q_{\text{TAR}}$
    end for
  end for
end for

III. The Proposed Reinforcement Learning based Collision Avoidance Approach

In this section, the proposed collision avoidance method based on NAF algorithm is discussed in detail for the robotic case. To this end, one needs to define the setup for the RL framework, that consists in building the state space $\mathcal{S}$, the action space $\mathcal{A}$ and the reward function.

A. State Space

The observations relevant for the training process are the measured joint positions q, the joint velocities $\dot{q}$, the target point position p_{t}, the end-effector position p_{e}, the obstacle position p_{o} and its velocity $\dot{p}_{\text{o}}$. Note that typically robots are equipped with resolvers to only measure positions. However, if the quality of mechanical design is high, finite differentiation of the position signal is sufficient to obtain a good estimate of the velocity $\dot{q}$. If this is not the case, the estimate of the velocities $\dot{q}$ can be obtained for instance using observers, even of robust type, like sliding mode based differentiators [17] as successfully used in [18]. As for the end-effector position, it is assumed known and provided by cameras located in the workspace. Analogous considerations hold for the obstacle position, while its velocity is assumed to be correctly estimated. The state space $\mathcal{S}$ is hence defined as

$$\mathcal{S} = \{q,\, \dot{q},\, p_{\text{e}},\, p_{\text{t}},\, p_{\text{o}},\, \dot{p}_{\text{o}}\} . \tag{9}$$

B. Action Space

The action space $\mathcal{A}$ is defined as

$$\mathcal{A} = \{\dot{q}_{\text{tar}}\}, \tag{10}$$

where $\dot{q}_{\text{tar}}$ is the vector of the rotational velocity at each joint. Specifically, the velocity control of the joints is such that, at each step, the joint has to rotate at the target velocity using the maximum torque available.

C. Reward function

The reward function is a scalar function defined by the weighted sum of three terms: the distance between the end-effector and the target point, the magnitude of the actions, and the distance of the obstacle from the robot, i.e.,

$$r = c_1 R_{\text{T}} + c_2 R_{\text{A}} + c_3 R_{\text{O}} . \tag{11}$$

The distance R_{T} between the end-effector and the target point is computed using the Huber-Loss function,

$$R_{\text{T}} = L_\delta(d) = \begin{cases} \frac{1}{2}d^2 & \text{for } |d| < \delta \\ \delta\left(|d| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{12}$$

where d is the Euclidean distance between the tip and the target and δ is a parameter that determines the smoothness.

The magnitude of the actions R_{A} performed by the manipulator is computed as the square of the norm of the

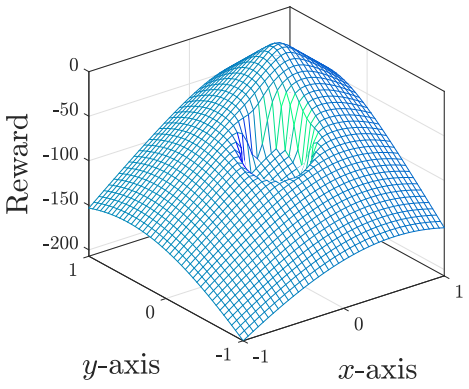

Fig. 1. Reward function on the planar section of the environment

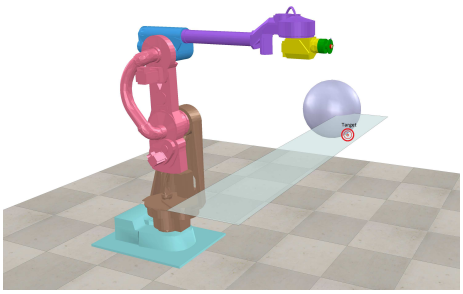

Fig. 2. Training scenario in V-REP with the virtual COMAU SMART3-S2, obstacle and target (red circle)

TABLE I
CHOICE OF PARAMETERS FOR THE EXPERIMENTS

Parameters	Value
Number of time steps	360
Time step	50 ms
c_1	1000
c_2	100
c_3	60
δ	0.1
p	8
d_{ref}	0.2
Discount factor γ	0.99
Update factor τ	0.001
Learning rate η	0.001
Noise type $\mathcal{D}$	*Ornstein–Uhlenbeck*
Noise decay factor	0.01
Noise scale	1

action vector a, as follows

$$R_A = -\|a\|^2 . \tag{13}$$

It gives a negative contribute to the reward, therefore smaller actions are encouraged. The distance between the robot and the obstacle R_{O} is computed as follows

$$R_{\text{O}} = \left(\frac{d_{\text{ref}}}{d_O + d_{\text{ref}}}\right)^p \tag{14}$$

where d_{ref} is a constant parameter so that $0 < R_{\text{O}} < 1$, d_{O} is the minimum distance from the obstacle as computed by V-REP and p is for the exponential decay of negative reward. The weights c_1, c_2, c_3 are necessary in order to tune the reward function depending on what is prioritized in the training process.

Figure 1 shows the behavior of the reward function computed on a planar cross section of the working environment of the robot. Such plane is parallel to the floor and at the same height of the target and the trajectory of the obstacle. The target point is placed in $(0.5, 0.5)$ and the obstacle is centered in $(0.1, 0)$. For the sake of simplicity, it is assumed that the robot is a point moving in the space and only the terms R_{T} and R_{O} of the reward function are considered. As it can be observed from Figure 1, the resulting function is concave with values less than zero, and it reaches its maximum when the tip of the robot is placed on the target point. Moreover, the value diminishes as the robot moves in proximity of the obstacle.

D. Hyperparameters

The hyperparameters are all the elements that have to be set up in order to adjust the learning environment. The exploration of the policy during the training process is defined by the following list of values:

- type of noise $\mathcal{D}$: the type of stochastic process added at each time step to the action, in order to keep exploring the environment; it can be either *Brownian* or *Ornstein–Uhlenbeck* noise;
- noise scale: scaling factor for noise exploration;
- noise decay factor: how quickly does the noise decay during the training episode.

Parameters related to the NAF learning algorithm are:

- update factor τ: soft target update at the end of each step;
- discount factor γ: how much the future rewards are valued with respect to the present one;
- learning rate η: how much the agent is able to acquire new knowledge.

Parameters related to the reward function are:

- c_1, c_2, c_3: weights of the three components, respectively the distance from the target, the action magnitude and the distance from the obstacle;
- δ: discriminating parameter for the Huber Loss;
- reference distance d_{ref}: default minimum distance between the obstacle and the body of the manipulator;
- exponential decay factor p: decay of the negative reward when the distance from the obstacle increases.

IV. CASE STUDY AND SIMULATION EVALUATION

In this section the results obtained by applying the proposed collision avoidance NAF algorithm will be illustrated for an industrial manipulator with 6 joints that has to reach a point in the space with its end-effector. While accomplishing its

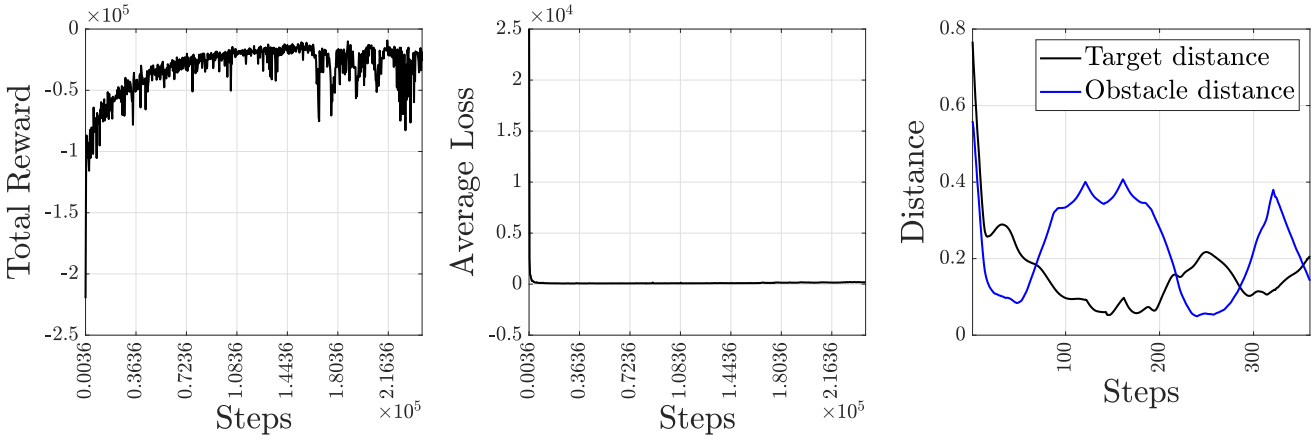

Fig. 3. Experiment results for fixed target, random moving obstacle

task, it must avoid collisions with obstacles that move in unpredictable ways. In order to carry on the evaluations, the physical environment for the training process was reproduced using the simulator V-REP. The simulator and the NAF learning algorithm, implemented using TensorFlow, are interfaced using V-REP's Remote API functions and Gym in Python. The used manipulator was the virtual replica of the COMAU SMART3-S2 anthropomorphic robot (see Figure 2). The target point is placed on a plane, parallel to the floor and in front of the robot, in order to emulate the production line the manipulator has to interact with. The obstacle (that in a real setting could be, for instance, a worker accidentally entering the robot's working space) is represented by a sphere moving on a linear path, placed between the target point and the manipulator.

The algorithm's performances were evaluated for different settings, in which the variable elements are the obstacle's movement and the position of the target point. All experiments use the same set of parameters, summarized in Table III-D. The settings considered for the experiments are:

(i) fixed target, moving obstacle: the position of the target point is the same for each episode, and the obstacle moves from one end to the other of the linear path, at constant velocity;
(ii) fixed target, randomly moving obstacle: the position of the target point is the same for each episode, and the obstacle moves randomly along the path; the sphere can change direction at any time or stop for an interval of time;
(iii) random target, moving obstacle: the position of the target point is randomly initialized at the beginning of each episode, and the obstacle moves back and forth in a deterministic way;
(iv) random target, random moving obstacle: the position of the target point is randomly initialized at the beginning of each episode and the obstacle moves randomly along the path.

For all the experiments, the total reward and the average loss function per episode have been traced. Moreover, the norm of the distance between the tip and the target and the norm of the distance from the obstacle have been retrieved at each step of the simulation to better show the behavior of the robot during one training episode. The results refer to the values obtained after 650 episodes of 360 steps each. All the training sessions have been carried out on a machine mounting a 8x intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz with a 32GB RAM and a NVIDIA Quadro k5000 GPU with 4GB DRAM. Each process took about 32 hours. As it is possible to observe in Figure 3, in all the experiments the cumulative reward converges to its maximum value: this means that the robot has learned a way to efficiently complete its task. This applies also to the trend of the average loss function, that presents little variations. Furthermore, as the distance from the obstacle diminishes, the robot adjusts its position with respect to the target, moving away further until the obstacle is again at a safe distance. The distance between the tip and the target point diminishes as the robot approaches the target and maintains its position until the sphere becomes dangerously close to the body of the robot. Then, the robot backs off from the target until the obstacle starts moving further away from the manipulator. The same behavior has been obtained in both the experiments with the deterministic moving obstacle and random moving obstacle.

A. *Transfer learning*

In machine learning, the term *transfer learning* refers to the application of previously gained knowledge to perform a different task. After extensive training sessions, the results of transfer learning on a process have been also evaluated, considering three different approaches:

(i) model transfer: reuse of the approximators parameters during the initialization;
(ii) experience transfer: reuse of the information contained in the Replay Buffer *RB*, i.e., the quadruplets $\{s_t, a_t, r_t, s_{t+1}\}$;
(iii) model and the experience transfer: both of the above.

The transfer has been made using knowledge acquired during the experiment with the random target and the deterministic moving obstacle for the training in case of a randomly moving obstacle. It can be observed that the learning process using

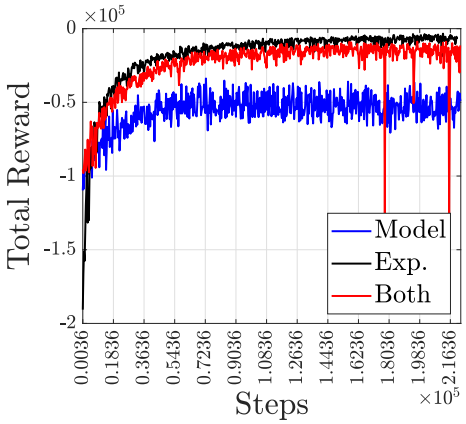

Fig. 4. Performance comparison between the three methods of transfer learning: model only (blue), experience only (black) and model and experience combined (red)

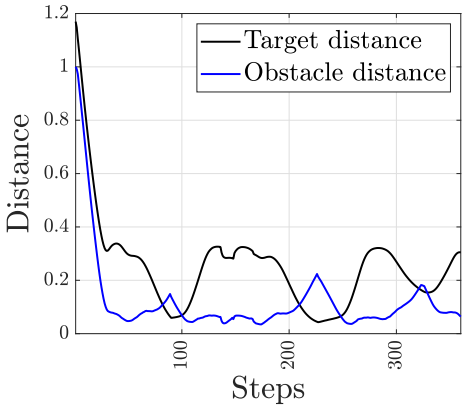

Fig. 5. Behavior of the manipulator with respect to the distance from the obstacle with experience transfer

the experience significantly improves its performances. On the other hand, the transfer of the model produces poor results. The use of both experience and model, instead, maintains a similar trend. Nevertheless, the reward function converges and the experiments give satisfactory results. An overall comparison between the performances of the three methods used for transfer learning is reported in Figure 4.

B. Robustness

In order to test the collision avoidance functionality, it has also been implemented a feature that allows one to manually change the direction of the obstacle with a simple graphic interface. Despite the attempts to collide with the robot, the knowledge has been such to avoid the obstacle. Figure 5 shows that attempts to get the obstacle close to the robot result in a collision avoidance. The results have been obtained by transferring the experience acquired in a training session with the deterministic moving obstacle.

V. Conclusions

In this paper, the applicability of deep reinforcement learning methods to problems of collision avoidance in robotics has been investigated. To this end, a Q-learning approach has been applied. It is a model-free learning approach, suitable for complex systems in the continuous framework, using a specific algorithm called Normalized Advantage Function, explicitly designed to simplify computation, making it practically applicable to this kind of problems.

References

[1] A. Bicchi, M. A. Peshkin, and J. E. Colgate, *Safety for Physical Human–Robot Interaction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1335–1348.

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

[3] F. Flacco, T. Krger, A. D. Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *IEEE International Conference on Robotics and Automation*, Saint Paul, MN, USA, May 2012, pp. 338–345.

[4] S. Haddadin, A. D. Luca, and A. Albu-Schaffer, "Robot collisions: A survey on detection, isolation, and identification," *IEEE Transactions on Robotics*, vol. PP, no. 99, pp. 1–21, 2017.

[5] S. Haddadin, A. Albu-Schaffer, A. D. Luca, and G. Hirzinger, "Collision detection and reaction: A contribution to safe physical human-robot interaction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, Sep. 2008, pp. 3356–3363.

[6] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 109–117, 1985.

[7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal Robotics Research*, vol. 5, no. 1, pp. 90–98, Apr. 1986.

[8] P. Ogren, L. Petersson, M. Egerstedt, and X. Hu, "Reactive mobile manipulation using dynamic trajectory tracking: design and implementation," in *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187)*, vol. 3, San Francisco, CA, USA, Dec. 2000, pp. 3001–3006.

[9] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[10] L. Balan and G. M. Bone, "Real-time 3d collision avoidance method for safe human and robot coexistence," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, PRC, Oct. 2006, pp. 276–282.

[11] L. M. Capisani, T. Facchinetti, A. Ferrara, and A. Martinelli, "Obstacle modelling oriented to safe motion planning and control for planar rigid robot manipulators," *Journal of Intelligent & Robotic Systems*, vol. 71, no. 2, pp. 159–178, Aug. 2013.

[12] S. Haddadin, H. Urbanek, S. Parusel, D. Burschka, J. Robmann, A. Albu-Schaffer, and G. Hirzinger, "Real-time reactive motion generation based on variable attractor dynamics and shaped velocities," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, Oct. 2010, pp. 3109–3116.

[13] S. Kuhn and D. Henrich, "Fast vision-based minimum distance determination between known and unkown objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, USA, Oct. 2007, pp. 2186–2191.

[14] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[15] S. Amarjyoti, "Deep reinforcement learning for robotic manipulation - the state of the art," *CoRR*, vol. abs/1701.08878, 2017.

[16] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *Proceedings of the 33rd International Conference on Machine Learning*, New York, NY, USA, Jun. 2016.

[17] A. Levant, "Robust exact differentiation via sliding mode technique," *Automatica*, vol. 34, no. 3, pp. 379–384, Mar. 1998.

[18] G. P. Incremona, A. Saccon, A. Ferrara, and H. Nijmeijer, "Trajectory tracking of mechanical systems with unilateral constraints: Experimental results of a recently introduced hybrid pd feedback controller," in *Proceeding of the 54th IEEE Conference on Decision and Control (CDC)*, Osaka, Japan, Dec. 2015, pp. 920–925.