Sample-Efficient Reinforcement Learning for Pose Regulation of a Mobile Robot

Walter Brescia
Politecnico di Bari
Bari, Italy
walter.brescia@poliba.it

Luca De Cicco
Politecnico di Bari
Bari, Italy
luca.decicco@poliba.it

Saverio Mascolo
Politecnico di Bari
Bari, Italy
saverio.mascolo@poliba.it

Abstract—**Reinforcement Learning (RL) has gained interest in the control and automation communities thanks to its encouraging results in many challenging control problems without requiring a model of the system and of the environment. Yet, it is well-known that employing such a learning-based approach in real scenarios may be problematic, as a prohibitive amount of data might be required to converge to an optimal control policy. In this work, we equip a popular RL algorithm with two tools to improve exploration effectiveness and sample efficiency: the *Episodic Noise*, that helps useful subsets of actions emerge already in the first few training episodes, and the *Difficulty Manager*, that generates goals proportioned to the current agent's capabilities. We demonstrate the effectiveness of such proposed tools on a pose regulation task of a *four wheel steering four wheel driving* robot, suitable for a wide range of industrial scenarios. The resulting agent learns effective sets of actions in just a few hundreds training epochs, reaching satisfactory performance during tests.**

Index Terms—**Deep Reinforcement Learning; Mobile Robots; TD3; Sample Efficiency.**

I. Introduction and Background

Learning-based model-free approaches are rapidly gaining interest in the field of robotics due to their ability of controlling robots without requiring the knowledge of their model, while fully leveraging their maneuverability and dexterity. Such approaches allow to design robots that learn from the environment and are able to adapt to successfully carry out new tasks. Reinforcement Learning (RL) is an unsupervised machine learning approach that seeks to produce an optimal *policy*, with respect to the goal it has to accomplish, by *training* an agent over the collected experience, typically obtained in a trial and error process.

In the literature, the RL algorithms commonly employed in control applications with continuous controlled variables are the *Proximal Policy Optimization* (PPO) [1] and the *Deep Deterministic Policy Gradient* (DDPG) [2]. Another promising technique recently proposed is the *Self Supervised Reinforcement Learning* (SSRL) [3], which leverages an imitation learning approach with a self-supervised replay buffer.

The size of the collected experience required for convergence to an optimal policy is related to as the *sample efficiency* of a RL algorithm. In industrial scenarios, collecting experience, i.e., interacting with the environment, might be costly, especially in the initial phases of the training where the safety of the surroundings and equipment involved might not be granted. Hence, designing a sample-efficient RL algorithm is a key issue to decrease both costs and chances of damage.

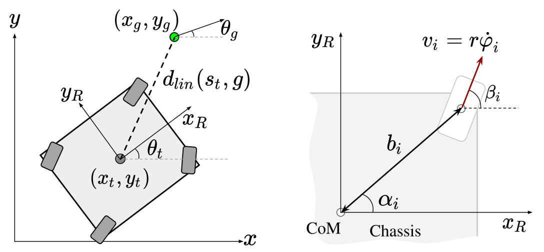
This work has been partially supported through the AGREED project (ARS01_00254) funded by MIUR. Walter Brescia holds a PhD grant funded by FSE REACT-EU.

An interesting approach recently proposed in the literature to tackle such an issue is *Curriculum Learning* (CL) [4]. In a nutshell, the main finding of [4] is that the key to improve RL agents' sample efficiency and generalization capabilities is the fashion through which examples (or task goals) are proposed to the agent. We also take inspiration from [5] and [6], in which a different take on the same idea is proposed.

In this paper, we embrace the idea of CL and propose a methodology for efficiently training a RL agent to equip with pose regulation functionalities a four steerable and driving wheels mobile robot. Notice that the trained policy needs to drive concurrently a total of 8 motors. In the literature, classical control techniques have been proposed to control this type of platforms. Some approaches impose added constraints on the wheels [7], [8], others leverage a different type of steering, thus simplifying the kinematics of the robot and limiting its maneuverability [9]. Instead, in this work we are interested in training an RL agent without imposing any constraints on the actions, thus allowing the agent to freely impose independent actions on the 8 motors. To the best of our knowledge, this is the first attempt of controlling such a mobile robot with a RL agent with an end-to-end approach, without introducing constraints or limitations in the action space.

To this end, we propose two tools to control such a mobile robot, namely (i) a *Difficulty Manager (DM)* and (ii) an *Episodic Noise*, specifically designed to improve the sample efficiency, exploration effectiveness, and robustness of a RL algorithm.

Rather than partitioning the space into zones to present goals with increased difficulty as proposed in [10], the proposed DM samples goals from a negatively skewed distribution whose average is shifted further only when an improvement in the agent's performance is measured. The resulting agent efficiently learns a subset of effective actions already in the first few epochs, exhibiting improved exploration, and next improves performance by refining the learned actions. The trained agent can fully exploit the capabilities of the mobile robot, without introducing limitations and/or constraints on the actions as typically performed in the literature.



(a) Notation (b) Chassis

Fig. 1: The considered robot

II. Problem Statement

A. The mobile robot

The Wheeled Mobile Robot (WMR) chosen as the subject of this work is equipped with four steerable and driving standard wheels, all independent of each other in both direction and speed, also known as *4 Wheel Steering 4 Wheel Driving* (4WS4WD). Such a robot is over-actuated, yet it is subject to non-holonomic constraints due to the use of standard wheels. For this reason, it is also referred to as non-holonomic omnidirectional [11] or pseudo-omnidirectional [12] robot.

Fig. 1a shows the coordinate system used in the notation, whereas Fig. 1b highlights the relevant notation in terms of actuation variables and geometric parameters referred to the i-th wheel ($i = 1, \ldots, 4$). The pose of the robot is identified by the triple (x, y, θ), i.e. the coordinates of the Center of Mass (CoM) of the robot and its orientation with respect to the world reference frame. The dynamics of the robot are influenced by the following parameters: (i) M, the mass of the robot; (ii) r, the radius of the wheels; (iii) b_i, the length of the segment connecting the CoM and the i-th wheel's steering axis; (iv) α_i, the angle between the x_r-axis and the segment connecting the CoM and the i-th wheel's steering axis; (v) β_i, the wheel's steering angle (a control input); (vi) $\dot{\varphi}_i$, the wheel's angular velocity (a control input).

The high number of controllable wheels and their independence from each other make this robot tailored for a wide variety of tasks, starting from autonomous driving up to fault tolerant control. In fact, the particular locomotion system of this robot allows to obtain the same motion vector through a wide range of actuation solutions, which makes the task of controlling this robot non-trivial.

The goal of this work is to design a robust RL agent able to regulate the pose of the robot in a plane free of obstacles. Hence, the agent must learn how to drive the 8 motors that govern the motion of the robot to move it towards the desired pose.

B. Notation

We consider a RL framework in which an agent interacts with a fully observed environment E in discrete timesteps. At each timestep t, the agent observes a state $s_t \in \mathcal{S}$, with $\mathcal{S}$ being the *Observation Space*, produces an action a_t according to its policy $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$, with $P(\mathcal{A})$ being the probability distribution over the *Action Space* $\mathcal{A}$, and receives a reward $r_t = R(s_t, a_t)$. The task is modeled as a *Markov Decision Problem* (MDP), which is composed of: (i) a transition function $T(s_t, a_t, s_{t+1})$, that represents the probability $p(s_{t+1}|s_t, a_t)$ of visiting the state s_{t+1}, given the state-action pair (s_t, a_t); (ii) continuous observation and action spaces (consistent with classical control problems) $s \in \mathcal{S}$ with $\mathcal{S} \subseteq \mathbb{R}^N$, $a \in \mathcal{A}$ with $\mathcal{A} \subseteq \mathbb{R}^M$; (iii) an initial state distribution $p(s_1)$, and (iv) a reward function $R(s_t, a_t)$. The return from a state s_t is defined as the sum of discounted future rewards $R_t = \sum_{i=0}^{\mathcal{T}} r_i \gamma^i$ with $\gamma \in [0, 1)$ being a discount factor that prioritises short-term (or long-term) rewards and $\mathcal{T}$ the total number of steps. The main goal is to learn an optimal policy that maximizes the expected return. Hence, a parametric form of the policy π_θ is leveraged, in order to update the parameters θ towards the optimal solution. The experience collection is organized in *episodes*, in which the agent explores the environment and stores the collected experience into a *Replay Buffer*.

C. Definition of goal g

For each episode a new goal $g = (x_g, y_g, \theta_g) \in \mathcal{G} \subset \mathbb{R}^2 \times SO(2)$ is generated, where x_g and y_g are the Cartesian coordinates with respect to the global reference frame, and θ_g is the desired heading angle (see Fig. 1a).

Following the notation introduced in [13], we employ a reward function $r(s_t, g)$ parametrized on g. Each goal g is a state belonging to $\mathcal{S}_g \subset \mathcal{S}$, where $\mathcal{S}_g$ is defined as the set of states satisfying the goal g. Equipped with this notation, the agent is said to reach the goal whenever the current state s_t gets in $\mathcal{S}_g$. In the following, we formally define $\mathcal{S}_g$.

Let $d_{lin}(s_t, g) = \sqrt{(x_t - x_g)^2 + (y_t - y_g)^2}$ be the linear Euclidean distance between the current state s_t and the goal g; similarly, $d_{ang}(s_t, g) = |\theta_t - \theta_g|$ is the absolute value of the heading error. A graphical representation of the introduced notation is provided in Fig. 1a. We define $\mathcal{S}_g$ to be the set of states satisfying both the conditions: (i) $d_{lin}(s_t, g) < \epsilon_{lin}$, (ii) $d_{ang}(s_t, g) < \epsilon_{ang}$, with ϵ_{lin} and ϵ_{ang} being respectively the linear and angular thresholds accepted (tolerances).

To have a compact definition, we can write equivalently $\mathcal{S}_g = \{s_t \in \mathcal{S} : \|f(d(s_t, g))\|_\infty < \epsilon\}$ with $d(\cdot)$ being a function returning a vector of two components collecting the linear and angular distances, $f(\cdot)$ being a function that normalizes the linear and angular distances with respect to their tolerances and ϵ being a threshold on the distance defined over the triple (x_t, y_t, θ_t) with respect to g. Under this notation, in simple terms, the agent reaches the goal g if the state s_t enters the set $\mathcal{S}_g$, i.e. the robot tracks the desired pose with a given tolerance on the Euclidean and angular errors.

D. Observation and action spaces

The observation space has been designed to be composed of: the linear (e_l) and angular (e_a) errors with respect to the goal coordinates, normalized to their respective initial errors; the ideal velocities in x (V_x^{CoM}) and y (V_y^{CoM}) axes of the CoM of the robot, coherent with the maximal velocity each wheel can impose; each wheel's velocity error with respect to

V_x^{CoM} and V_y^{CoM}, defined as $e_{V_x^{CoM}}$ and $e_{V_y^{CoM}}$ respectively; the ideal angular velocity ($\dot{\omega}_t^i$) and the actual velocity ($\dot{\omega}_t$). Thus, the dimension of the observation space is 14.

The action space is composed of the linear velocities and the steering angles of each wheel. The steering angle is represented through its sine and cosine components, following [14]: the representation of a connected set of rotations in a circumference through its angles in the range $[-\pi, \pi)$ introduces a discontinuity. In fact, [14] argues that such discontinuities are typically harder to learn for a neural network. We found this issue to hold by testing both configurations. Indeed, representing the rotations directly in radians resulted in an agent incapable of controlling the robot. Also, a representation of an angle through its cosine and sine components allows one to formulate an error on the said angle in the form of a norm.

E. Reward

An extensive phase of *reward shaping* has been conducted and converged to define the reward r_t as the sum of the following components: (i) the linear distance from the goal coordinates $d_{lin}(s_t, g)$; (ii) the angular distance from the goal coordinate $d_{ang}(s_t, g)$; (iii) the sum of the quadratic errors of the wheels' velocities in the x and y directions $e_t^{V_x y}$; (iv) the error of the angular velocity applied with respect to the ideal one $e_t^{\dot{\omega}^i}$. Hence, the reward function r_t becomes parametric with respect to the current state s_t, the epoch's goal g and the action a_t:

$$r_t(s_t, g, a_t) = -d_{lin}(s_t, g) - d_{ang}(s_t, g) - e_t^{V_x y} - e_t^{\dot{\omega}^i} \quad (1)$$

We consider *ideal* the configuration of a unicycle robot. Introducing such an indication of how to move helps to increase the efficiency of exploration, especially in the initial phases.

III. Methodology

In this work, we propose two tools that aim at improving the exploration, sample efficiency, robustness, and the overall performance of the RL agent. We introduce the *Difficulty Manager (DM)*, as a tool to feed the agent with goals always proportioned to its current performance, and the *Episodic Noise*, as a tool to increase the efficiency of the exploration of the action space.

A. Difficulty Manager (DM)

The key idea for the *DM* (Fig. 2) is to create a progression in the difficulty of the goals presented to the agent. The goals g are sampled from a distribution with an average that is increased accordingly to the skills of the current agent. The tail of said distribution will also include all the *simpler* goals, with the intent of preventing the agent from over-fitting on the current distribution.

Hence, in the early phases of training, when the agent is hardly able of moving at all, goals will be generated near the initial pose and, as the agent learns to move efficiently, goals placed further away will be presented to allow the agent improve its performance.

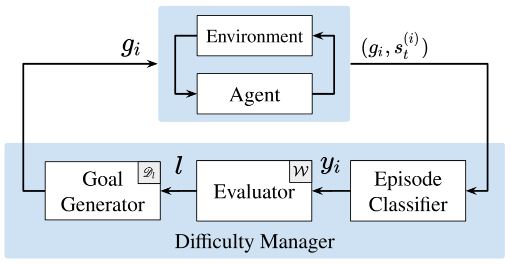


Fig. 2: Difficulty Manager (DM)

The proposed system is based on three main components as shown in Fig. 2.

Each training episode i produces a couple goal-terminal state, i.e. (g_i, s_T^i). The *Episode Classifier* reads this couple and produces a label y_i associated with the current episode to classify its performance. The label y_i is sent to the *Evaluator*, which temporarily stores the label in a queue to observe the evolution of training, and potentially triggers an *evaluation* phase to quantify, by considering the success rate over a set of uniformly sampled goals, the robustness and reliability of the agent; if the model satisfies a certain condition (see below), the current level l is increased and sent to the *Goal Generator*. This component holds the goal distribution $\mathscr{D}_l$ related to the current *difficulty level* $l \in \{1, \ldots, L\}$, with L being the maximum difficulty level; for each new training episode it samples a new goal g_i which is sent in input to the new episode.

Episode Classifier. This block maps each episode i to a label y_i that classifies the agent's performance. In particular, the episode can be classified as a *success* ($y_i =$ S, the agent has reduced the distances under the threshold), a *failure* ($y_i =$ F, distances have been reduced but not under the threshold), or a *serious failure* ($y_i =$ SF, the final distances are greater than the initial ones). The classification is performed according to the following conditions on the terminal state s_T^i associated with the i-th goal g_i:

$$y_i = \begin{cases} \text{S} & \|f(d(s_t^i, g_i))\|_\infty \leq \epsilon \\ \text{F} & \epsilon < \|f(d(s_t^i, g_i))\|_\infty < (1-\epsilon) \\ \text{SF} & \|f(d(s_t^i, g_i))\|_\infty \geq (1-\epsilon) \end{cases} \quad (2)$$

Goal Generator. This component implements a negatively skewed distribution $\mathscr{D}_l$, with an average D_l, associated with the current level l. For each episode, it samples a new goal from the current distribution and sends it in input to the new episode.

Evaluator. This block holds a window of W episodes' labels that allows to observe the leaning evolution. To the purpose, the labels of the latest epochs are stored in a FIFO queue $\mathcal{W}$. We define $\mathcal{W}^{\text{SF}} = \{y \in \mathcal{W} : y = \text{SF}\}$ and $\mathcal{W}^{\text{F}} = \{y \in \mathcal{W} : y = \text{F}\}$ the set containing only the labels of episodes in $\mathcal{W}$ classified as *serious failures* or *failures* respectively.

Then, the agent accesses an evaluation phase if and only if the following *evaluation criteria* are met: (i) no serious failures have been observed in the current window (i.e., $|\mathcal{W}^{\text{SF}}| = 0$)

and (ii) the ratio of episodes rated as *failed* ($y_i = \mathrm{F}$) is lower than a given threshold η: $|\mathcal{W}^{\mathrm{F}}|/|\mathcal{W}| < \eta$.

During the evaluation, the model is tested on a set of different goals generated uniformly in the range of distances reached so far. The objective is to save a robust model (π_θ^l, π_Q^l), associated with the current level l, that will be carefully evaluated later. The robustness is evaluated considering the following conditions: (i) the successful tests must exceed a given threshold T (i.e., the number of failures must be below the complementary threshold $1 - \mathrm{T}$); (ii) no serious failures must be registered. When an evaluation phase concludes positively, the model is saved, the level is increased by one ($l \leftarrow l + 1$) and the window is emptied.

Clearly, the first models to be saved only saw the initial distributions and, as a consequence, will hardly be capable of generalising to goals placed at higher distances, however, it is interesting to analyze the different behaviors that each saved model exhibits. More insights are presented in Section IV.

B. Episodic Noise

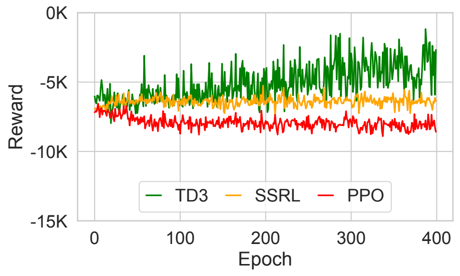
The second tool we provide concerns the noise applied over the actions. In this work, an "*episodic noise*" has been used to improve the exploration of the action space, in contrast to the typical noise identified by a zero mean Gaussian (or uniform) distribution. One of the main reasons behind this choice is that completely random actions (especially in the initial phases) may cause the agent to barely move from the initial position, due to the peculiar dynamics of the robot. Instead, for each episode, we sample a value ρ from a Gaussian distribution, clipped in the range $(-1, 1)$. Such value is then used at each step of the current episode as the mean of a new distribution with standard deviation σ (a hyperparameter), from which the actual noise is sampled. This generates a noise that, inside the episode, keeps around the set average (most likely different from 0), allowing the appearance of more useful and continuous behaviors than the ones obtained from noise with zero mean. As the agent learns, this signal is reduced in order to make more space for exploitation.

IV. Results

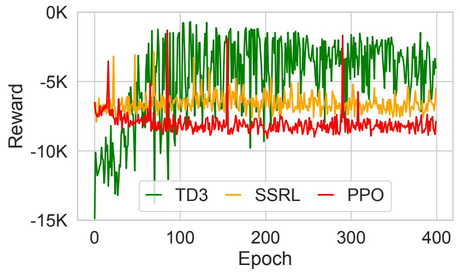
In this section, we compare three state-of-the-art RL algorithms (PPO, SSRL, and TD3), and then focus on an ablation study of the proposed tools on the best-performing algorithm. The results shown in the following have been obtained by implementing the robot and environment described in Section II in "*Gym Ignition*" [15]. Such a simulation environment has a light physics engine and benefits from the interfaces of "*OpenAI Gym*" [16]. This simplifies the process of creating new environments and tasks.

Fig. 3 shows the results of the three agents with (Fig. 3b) and without (Fig. 3a) our proposed tools[1].

Even though the proposed tools allow all the considered algorithms to improve performance in the first stages, neither



(a) Baseline Agents



(b) Agents with Ep.N and DM

Fig. 3: RL Algorithms Comparison

PPO nor SSRL manage to solve the task over 600k interaction timesteps. We attribute these negative results to two factors. Being PPO an on-policy RL algorithm, during training it cannot make use of meaningful episodes (the ones during which the robot moves properly, which are rarer when the agent is still learning) as much as off-policy algorithms do, since the latter are equipped with a replay buffer. Regarding SSRL, we argue that the main limitations might be ascribed to the update technique and the buffer employed: in the initial phases the agent will most likely experience trajectories associated with poor performance and on this experience it employs imitation learning that leads toward suboptimal updates, falling in a vicious cycle that impairs significant improvements. Note that, in Fig. 3 both PPO and SSRL equipped with the proposed tools show some improvements (several peaks can be observed) but they fall on the same pattern soon after epoch ~ 100. Even if our tools slightly improve (at least in the beginning) the performance of all the considered agents, they are not meant to solve issues related to the algorithms themselves, but rather to improve their sample efficiency.

From Fig. 3a, we observed that TD3 baseline is able to improve performance but its performance is not robust. However, by leveraging the proposed tools, TD3 reaches good performance in less than 100 episodes (less than 150k interaction timesteps) and stabilises over time, even on more complex goals (Fig. 3b).

For this reason, in the remaining part of this paper, we focus our study on the TD3. In particular, we consider four agents trained with the TD3 algorithm: (i) *Baseline* (BL), which is the vanilla TD3; (ii) *Baseline with DM* (BL+DM): which is the vanilla version equipped only with the DM tool; (iii) *Baseline with Episodic Noise* (BL+Ep.N), which is the vanilla version

[1]Note: these results are obtained on several hyperparameter configurations. For brevity and due to lack of space here we report the results from one configuration only.

	Hyperparameters	Value
Agent	LR Actor	0.0001
	LR Critic	0.0001
	γ	0.95
	τ	0.1
	σ	0.3
	Noise Decay	0.99
	Actor Update Rate	125
	Batch Size	32
Memory	Type	PER
	Mem. Capacity	50000
	α	0.5
	β	0.4
DM	W	10
	η	0.1
	Init. Dist. (m)	0.6
	Step Dist. (m)	0.4
	Init. Ang. Dist. (rad)	0.75
	Step Ang. Dist. (rad)	0.6
	Max Goal Dist. (m)	4.0
	T	0.85

TABLE I: Hyperparameters

equipped only with the Episodic Noise tool; (iv) *Baseline with DM and Episodic Noise* (BL+DM+Ep.N, also referred to as "Ours" for brevity purposes), which is the vanilla version equipped with both the tools[2].

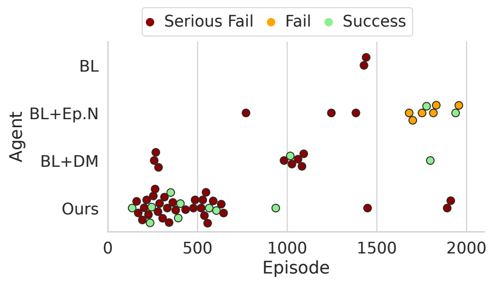
Table I reports the key hyperparameters of the algorithm and the proposed tools. It is worth mentioning that both the actor and critic approximators are composed of two hidden layers, 512 and 256 respectively. All tests consist of reaching the same 200 goal coordinates for each agent, generated uniformly in a range of 6 meters for the Cartesian coordinates and in the whole range of $[-\pi, \pi)$ for the angular goal. During training, which lasts at most 2000 epochs, all agents will experience goals distant at most 4 meters away, hence, we not only evaluate the performance of the model on known distances, but also on greater ones. This lets one assess the generalization capabilities of the model considered.

Note that, in all tests conducted, the BL agent has never met our *evaluation criteria*. However, for a comprehensive comparison of all the agents, the BL agent's models presented have been saved every 500 epochs.

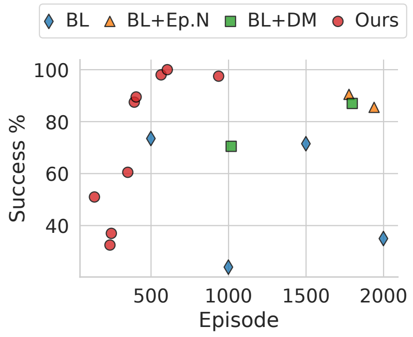
A. Overall Performance

In order to assess the impact of each component, we evaluate the performance of each agent during training. Fig. 4a reports every access to the evaluation phase and the corresponding episode at which such evaluation occurred. Each evaluation's result is reported and represented by a $\circ$, differentiating between successful evaluations (green $\circ$), failing evaluations (orange $\circ$) and evaluations interrupted by a serious fail (red $\circ$). Notice that evaluations are grouped by agent (the y-axis of Fig. 4a). As an overview of the performance of each considered agent, Fig. 4b shows the success rates obtained during testing (expressed in percentage) for each model that succeeded an evaluation in the sense defined in Section III-A.

[2]https://c3lab.poliba.it/SERL provides videos showing the behavior of the trained agents as they progress through their training phases



(a) Evaluations per Agent

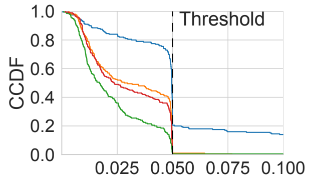


(b) Tests' Success Rate

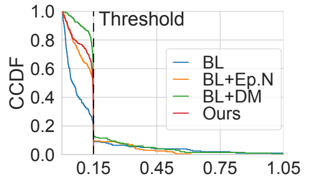
Fig. 4: Overall Performance

As one can observe from both Fig. 4a and Fig. 4b, the baseline (BL) does not really show a progression: the success rates of the sampled models are modest and quite variable, indicating poor robustness of the trained policy. Moreover, BL shows no progression as training is carried out. The BL+Ep.N obtains good performance (85-90% success rate) only after $\sim$1750 episodes of training, producing two "*good models*", i.e. models that pass the evaluation phase. The BL+DM passes only two "levels", experiencing goals at a maximum average of $\sim$1.5 meters. The first model (obtained quicker than BL and BL+Ep.N) shows good generalisation capabilities, especially considering the (simpler) goals experienced. The second model shows slightly increased performance. Finally, our agent ("Ours") shows a particularly interesting trend: the first good model to pass the evaluation is obtained very quickly (epoch 135) and is followed by a condensed sequence of evaluations (successful in many cases), highlighting how the tools help the algorithm steadily improve performance towards convergence. The peak in performance is registered at epoch 605 with a success rate of 100%. In summary, the proposed agent ("Ours") improves the sample efficiency compared to the other agents that lack the two tools.

We argue that providing goals far from the origin during the early training stages impairs the efficiency of the training process. This is due to a vicious circle triggered by three main factors: (i) in the early stages the agent cannot produce significant movements; (ii) hence, the observations in the replay buffer will not be characterized by a wide diversity (since the agent keeps close to the origin), thus possibly producing updates that lead to local sub-optimal policies and (iii) as a further consequence, the rewards associated to these



(a) CCDF by Normalized Distance



(b) CCDF by Heading Error [rad]

Fig. 5: CCDFs on linear and angular distance

state-action pairs will be quite similar to each other, creating a valley in the cost function. In order to escape this local minimum, the agent should pseudo-randomly produce at least a barely good trajectory (a situation that is more likely to occur when the sampled goal is close enough to the origin). Providing closer goals increases diversity in the buffer, due to a reduced magnitude of the errors, allowing the agent to repel ineffective actions and select much more frequently those actions associated with higher rewards.

B. Linear and Angular Distances Coverage

Next, we investigate the robustness of the best models obtained for each agent. Fig. 5a and Fig. 5b represent the complementary cumulative distribution functions (CCDFs) of, respectively, normalized linear distance and normalized angular distance. Both figures show a dashed vertical line which represents the thresholds for linear and angular distances. Only those models whose CCDFs are below both the thresholds are considered to have reached good performance, see Section II-C.

Observing Fig. 5a, it may seem that the BL+DM agent (in green) achieves the best performance; however, a closer look reveals that it does not grant the satisfaction of the success condition, since the curve slips beyond the threshold. On the contrary, the agent equipped with both the proposed tools (in red) grants complete coverage, never trespassing the threshold.

Fig. 5b clearly shows that only the agent equipped with both the tools (in red) is capable of respecting the thresholds in all the cases. The second best performing agent is the baseline equipped with the episodic noise, which grants on $\sim 90\%$ the complete coverage of the angular distance.

C. Maneuverability Exploitation

Another interesting result regards the ability of the agent of fully exploiting the robot's maneuverability: we found that agents equipped with the Episodic Noise learn to balance the minimization of linear and angular distances throughout the epoch, while agents without such tool learn to minimize either linear or angular distance first and only later they minimize (or try to) the other metric. Due to the lack of space, we cannot include figures showing the trajectories followed by each agent during the tests and point the interested reader to videos reported at https://c3lab.poliba.it/SERL.

V. Conclusions

In this work, we studied the control of an over-actuated mobile platform for a pose regulation task. Unlike the classical approach, which usually limits the maneuverability of the mobile platform, we followed a learning-based, model-free approach, leveraging a RL algorithm. We studied the performance of some of the most popular RL algorithms and a more recent one and then focused on the TD3, which showed the most promising initial results even though it never met our requirements. Hence, we introduced two new tools: the Episodic Noise and the Difficulty Manager (DM). The combination of these two tools allows improving the performance of the baseline algorithm, outperforming it and reaching 100% of success rate during tests, including goals never experienced.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. of ICLR*, 2015.

[3] D. Zha, K.-H. Lai, K. Zhou, and X. Hu, "Simplifying deep reinforcement learning via self-supervision," *arXiv preprint arXiv:2106.05526*, 2021.

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[5] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. of NIPS*, 2015.

[6] S. Sharma and B. Ravindran, "Online multi-task learning using active sampling," in *Proc. of ICRL*, 2017.

[7] H. Yang, V. Cocquempot, and B. Jiang, "Optimal fault-tolerant path-tracking control for 4WS4WD electric vehicles," *IEEE Transactions on intelligent transportation systems*, vol. 11, no. 1, pp. 237–243, 2009.

[8] X. Zhang, Y. Xie, L. Jiang, G. Li, J. Meng, and Y. Huang, "Fault-tolerant dynamic control of a four-wheel redundantly-actuated mobile robot," *IEEE Access*, vol. 7, pp. 157 909–157 921, 2019.

[9] J. Liao, Z. Chen, and B. Yao, "Performance-oriented coordinated adaptive robust control for four-wheel independently driven skid steer mobile robot," *IEEE Access*, vol. 5, pp. 19 048–19 057, 2017.

[10] S. Krishnan, B. Boroujerdian, W. Fu, A. Faust, and V. J. Reddi, "Air learning: a deep reinforcement learning gym for autonomous aerial robot visual navigation," *Machine Learning*, vol. 110, no. 9, pp. 2501–2540, 2021.

[11] J.-B. Song and K.-S. Byun, "Steering control algorithm for efficient drive of a mobile robot with steerable omni-directional wheels," *Journal of mechanical science and technology*, vol. 23, no. 10, p. 2747, 2009.

[12] C. P. Connette, C. Parlitz, M. Hagele, and A. Verl, "Singularity avoidance for over-actuated, pseudo-omnidirectional, wheeled mobile robots," in *Proc. of IEEE ICRA*. IEEE, 2009, pp. 4124–4130.

[13] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *Proc. of ICML*. PMLR, 2018, pp. 1515–1528.

[14] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5745–5753.

[15] D. Ferigo, S. Traversaro, G. Metta, and D. Pucci, "Gym-ignition: Reproducible robotic simulations for reinforcement learning," in *2020 IEEE/SICE International Symposium on System Integration (SII)*, 2020, pp. 885–890. [Online]. Available: https://github.com/robotology/gym-ignition

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.