Delay Aware Universal Notice Network: Real world multi-robot transfer learning

Samuel Beaussant, Sebastien Lengagne, Benoit Thuilot, Olivier Stasse

Delay Aware Universal Notice Network: Real world multi-robot transfer learning

Samuel Beaussant, Sébastien Lengagne, Benoit Thuilot
Université Clermont Auvergne, CNRS, Clermont Auvergne INP,
Institut Pascal, F-63000 Clermont-Ferrand, France,
firstname.surname@uca.fr

Olivier Stasse
LAAS-CNRS, Université de Toulouse, CNRS
Toulouse, France,
olivier.stasse@laas.fr

***Abstract*—General purpose simulators provide cheap training data to learn complex robotic skills. However, the transition from simulation to reality is often very challenging for the agent. One major issue is the delay on the physical robot that may deteriorate the performance of the deployed agent. Furthermore, once a successfully trained learning-based control policy is available, re-purposing the knowledge acquired by the agent to enable a structurally distinct agent to perform the same task is hazardous if done naively. In this work, we address the above issues with a single method, the DA-UNN (Delay Aware Universal Notice Network), which decomposes the knowledge into robot-specific and task-specific modules for fast transfer. Our framework deals with delays immanent to physical systems in order to improve sim2real transfer. We evaluate the efficiency of our approach using simulated and actual robots on a dynamic manipulation task where delay management is crucial.**

***Index Terms*—Transfer learning, Sim2real, Reinforcement learning, Robotic task, delay**

I. Introduction

The Reinforcement Learning (RL) field has been successfully applied to a wide range of problems in the past years, demonstrating both its versatility and efficiency. From reaching human-level control on Atari games [1] to beating world class champions at complex games [2], [3], the RL field is full of promises. The robotic domain also benefited from the tremendous progresses made in RL as shown in recent work for quadruped robots [4] even succeeding in very intricate manipulation tasks such as solving a rubik's cube with a shadow hand [5]. However, despite great achievements, RL suffers from very low sample efficiency, which means that a large amount of interactions with the environment is needed to obtain a high-performance policy. One way to mitigate this issue is through the use of transfer learning which alleviates the burden of training models from scratch by re-purposing knowledge acquired on another domain.

Nevertheless, in the context of RL, the neural network architectures are too shallow and do not encourage knowledge segmentation as it is often the case in large Computer Vision models [6]. As a consequence, if no particular precaution is taken, the unconstrained backpropagation procedure may result in an entangled knowledge representation. In this setting, it is difficult to determine which part of the network is relative to the task or the robot, making a partial or total transfer of the policy network hazardous if done naively and with very low chances of success.

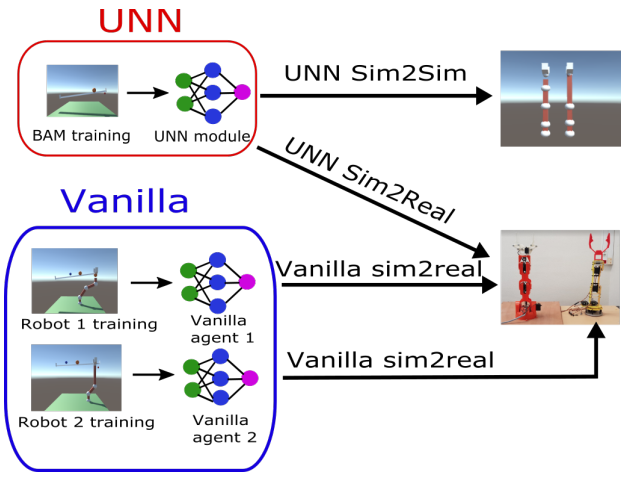


Fig. 1: Transfers considered. UNN module is trained with the BAM robot and then transferred to all robots. Vanilla agents are trained directly on the simulated robots and then transferred on the corresponding physical robots.

The UNN (Universal Notice Network) framework introduced in [7] tackles this issue by implementing the idea of knowledge segmentation between the agent and the task. More precisely, the purpose of the UNN framework is to enable multi-task and multi-robot transfer by creating a reusable and robot-agnostic module of skills. Previous work [7] already demonstrated the efficiency of UNN transfer in simulation on a wide variety of tasks and robots morphology, exhibiting zero-shot performance in some cases. By applying the same transfer method on real world robots, we wish to find out if results obtained in simulation generalize well to the real world.

However, simulation to real world transfer (also called sim2real in the literature) is still an open problem. In general, simulations are imperfect and difficult to calibrate. The resulting modeling discrepancies cause a reality gap, which makes the transfer of RL policies from simulation to the real-world non-trivial. Sim2real is especially appealing as it

offers an efficient alternative to expansive real world data collections for learning complex robotic skills by training in a general-purpose simulator. However, most of the time, methods focus solely on domain adaptation between the real world and the simulation [8], [9]. They tend to ignore troublesome hardware specific issues such as control latency induced by medium of data transmission, computation delay, sensor sampling rates (etc.) unmodeled in the simulator. Consequently, the policy obtained by training in simulation could be drastically disturbed once transferred in the real world if the task requires short reaction time.

In this paper, we consider the time delay associated with the physical system as another model's input by including it in the observed state. At training time, we randomize the value of the delay and show that the agent is able to adapt to multiple delays on a dynamic and delay-sensitive manipulation task. Our contributions are as follows:

1) We present and evaluate a delay-aware method to deal with the immanent delay on real hardware, thus furthering the adaptation capabilities of the UNN.
2) We evaluate the benefits of the UNN multi-robot transfer method over a vanilla transfer on real world robots. A pool containing four differently shaped real and virtual robots will solve a dynamic manipulation task they have not been trained on, by using the knowledge created by another agent as depicted in Figure 1.

II. Related Work

Transfer learning in RL has been recognized as an important direction towards building more sophisticated agents. For instance, multi-task learning aims at improving robots versatility via methods such as meta-learning [10], [11]. Another interesting approach is to hierarchically decompose complex problems into tractable, simpler and reusable modules of skills through the use of concept networks [12].

Transfer between morphologically distinct robots on the other hand, is currently less studied in the scientific literature. A method proposed in [13] encapsulates and leverages skills learned by a task expert by using GAN's discriminators as support for the knowledge transfer. Work by Gupta and Devin [14] presents a method to learn an invariant feature space for transferring skills between different robots. Other work by the same authors [15] uses a modular approach by training policy modules that are decomposed over robots and tasks. By doing so, they address both multi-robot and multi-task transfer by re-combining these modules to form a policy, with minimal additional re-training. Multiples combinations of robot-task pairs are trained together to create a latent space common to all modules. The UNN method used in this work, while very similar, differs from the prior method by explicitly defining the state shared between the task-specific and robot-specific modules. This choice suppresses the need of relying on a shared latent space between robot and task which may be prone to over-fitting if the number of available modules is too low. Furthermore, in the UNN framework, the task module can be trained separately and only once, thus creating a truly robot-agnostic module while saving the time and trouble of having to train multiple possible pairs of the training set.

Several methods has been proposed to deal with delays in robotic. In [16], the authors proposed a neural network based method to address the control delay issue. A predictor, approximated by a neural network, must infer the current state of a fast moving robot by observing a vector of stacked outdated states. To account for the imperfect actuators of the real hardware, authors of [5] introduced action delays with a probability of 0.5 at the beginning of every simulation training episode of a neural network with a LSTM layer. By doing so, they force the memory enhanced neural network to adapt to action delays. A cornerstone paper in RL with delay [17] learns a model of the undelayed Markov Decision Process (MDP) to simulate the most likely state in which the agent currently is, given the last observed (delayed) state and the k last actions taken since, k being the delay in timestep. This allows the agent to take decisions based on the expected current state rather than an outdated state, effectively undoing the harmful effect of delays. They also introduced the concept of Constant Delayed MDP used in section IV-A. Ramstedt and Pal [18] studied real-time RL by taking into account the computation time needed to select an action (supposed inferior to one time step). Their proposed algorithm, Real-Time Actor-Critic, additionally takes as input the action from the previous step in order to compensate the one-step action delay. However, previous methods are not suited for transfer between system with different delays. The agent must learn again from scratch whenever the system delay changes contrarily to our proposed method.

III. Universal Notice Network

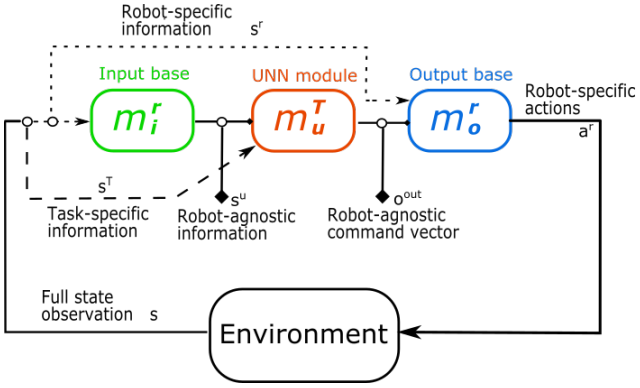


Fig. 2: Schematic representation of the UNN [7]

.

A. The UNN pipeline

Instead of learning a single policy that will have to handle both robot control and task resolution, the UNN method [7] relies on an explicit decomposition between task-specific and robot-specific knowledge as depicted in Figure 2. In this modular approach, a model of the task, the *UNN* (Figure 2) is created in such a way that any robot, regardless of its morphology, number of articulations or actuators can

efficiently benefit from it. This is similar to creating a notice containing a set of high level instructions, that any kind of robot could follow to solve a given task. It is primary to ensure that the robot has the mobility and capacities required to comply with the UNN instructions and accomplish the task. Defining R the set of feasible actions the robot can produce and U the set of actions required by the UNN to perform a task, we assume in this paper that the robot can perform the required actions, as sum up in:

$$U \subseteq R \tag{1}$$

Once this assumption is verified, two conditions are essential:

- First, the UNN module must be robot-agnostic to enable multi-robot transfer. This implies that robot-specific observations have to be translated into a feature space shared by the considered robots before being fed to the UNN module.
- Secondly, it is necessary to design a controller that will map the UNN commands from the shared feature space into low-level, robot-specific actions that the robot can execute.

These two requirements are handled by two additional modules inherent to the robot morphology called the *bases*. They are paired with the UNN and serve as an interface between the robot and the task module. More formally, the three modules form a pipeline composed of the input base m_i^r and the output base m_o^r, specific to the robot, which handle respectively the first and the second conditions, and the UNN m_u^T robot-agnostic and specific to the task only. In this setting, the state vector that is provided by the environment at each timestep can be split into two parts s^r, s^T, respectively holding data intrinsic to the considered robot and task-related information, independent from the agent. The input base m_i^r receives s^r to compute:

$$s^U = m_i^r(s^r) \tag{2}$$

which can be considered as robot-agnostic. The input base m_i^r is thus responsible for mapping the robot space to the shared feature space where the UNN operates. The next processing stage is the UNN module (i.e. the task module), conditioned by the task related observation s^T and the processed agent representation vector s^U. It then computes:

$$o^{out} = m_u^T(s^T, s^U) \tag{3}$$

where o^{out} is the command vector in the shared feature space. Finally, o^{out} is then re-mapped to the robot space by the output base with the following transformation:

$$a^r = m_o^r(o^{out}, s^r) \tag{4}$$

which yields a^r the effective action taken by the robot. In other words, the UNN module focuses solely on solving the task at hand, ignoring low level considerations such as the robot's DoF and shape, handled by the bases. This approach makes it possible to create a reusable module of skills that can be transferred to structurally different robots as long as their bases are available. It is then possible to build a library of UNN modules and robot's modules, draw any subset of interest from it and combine a UNN/Bases pair into a novel fully functional policy.

B. Modules training

In practice, each of the three sub-modules m_i^r, m_u^T, m_o^r can be either learned or obtained via analytical methods.

1) Bases modules: In the case where bases are obtained using neural networks, they can be trained on a suitable primitive task to acquire basic motor skills. Another alternative is to collect a dataset of trajectories of the robot and fit a regression model with supervised learning techniques. A last alternative, used in this work, consists to use analytical models for the robots bases.

2) UNN module: The UNN module (or task module) can be trained with or without the bases modules. In the first case, the UNN is coupled with a robot and its associated bases. The UNN interacts with the environment through the bases and its error on the task is back propagated through the network. In this case, we affect only the UNN module weights. However, the UNN module may then take advantage of the robot hardware structure to achieve the task (for instance, blocking an object between two articulations). As a consequence, the UNN may favor certain body configurations which may be detrimental for transfer. This issue is solved by using the Base Abstracted Modeling (BAM) method [19]. It assimilates the robot to its effector by setting m_i^r and m_o^r to identity mappings, thus making no assumption on the robot's constitution and preventing any bias related to the bases. This is equivalent to considering a purely virtual and free-flying robot. Using BAM enables faster convergence of the policy and a more defined knowledge segmentation, which in turns improves UNN transfer.

IV. Delay Aware Universal Notice Network

A. Constant Delayed Markov Decision Process

The standard UNN proved its efficiency and versatility on a broad panel of tasks in simulation. However, these results were obtained with perfect robots (e.g no offset and no delay) acting in a standard Markov Decision Process (MDP). Traditionally it is assumed in RL that at every timestep, the environment pauses while the agent receives the current observation, in order to derive an action that will be executed without delay. Of course, things do not behave this way in the real world. All agent observations and actions are delayed by an amount depending on the hardware used for the task. Therefore, an agent trained in simulation without exposition to delays will perform worse or even fail in the real world if no precaution is taken.

This brings up the need to adopt a different decision process modeling to solve tasks in the presence of delay. As we consider the delay to be constant, we found the Constant Delay MDP formulation introduced in [17] to be well suited. A CDMDP defines the delay d as the number of timesteps between an agent occupying a state and receiving its feedback

from the environment, where $d \in \mathbb{R}^+$. The delay is assumed to be part of the environment. A known result in CDMPD is that observation delay and action delay are equivalent from the agent's point of view [20]. Hence, we treated the total delay as being entirely caused by observation delay (see Figure 3).

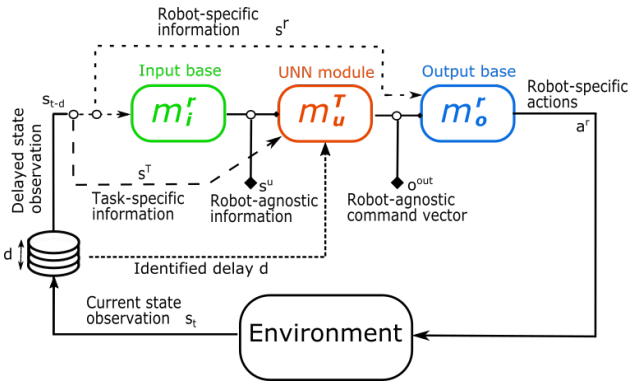


Fig. 3: Schematic representation of the delay aware UNN. Observations are queued into a pile of length d and each timestep, the observation at the top is fed to the agent (first in, first out).

B. Delay Aware UNN

A CDMDP can be transformed into a regular MDP by enlarging the state space with a history of the d last actions taken since the last observation. This transformation allows theoretically to derive an optimal policy for the CDMDP considered [21]. However, this approach does not allow direct transfer between systems with different delays as the input dimension depends on d. In this work we address the delay issue by augmenting the state space of the UNN module with the estimated delay of the system and by training the agent on a corresponding delayed environment as depicted in Figure 3. A key feature of the UNN is its ability to adapt to any robot regardless of its morphology. To keep this idea of "universality", the delay was randomized during training to ensure that the UNN can adapt to a wide range of delay. By giving it access to the immanent delay, we enable the UNN to act accordingly and to develop predictive capabilities. Thus, we add d to the task specific observations.

During training, the delay is sampled regularly from a discrete uniform distribution as $\mathcal{U}(d_{min}, d_{max})$ where d_{min} and d_{max} are respectively the minimum and the maximum delay considered for the environments. Since there is no assumption about the systems, we assumed a uniform distribution of the delay. But any knowledge could be used to deduce a better delay distribution. When deployed, the identified delay of the system is fed to the UNN, so it can act accordingly. While our approach can only yield sub-optimal CDMDP policies due to the incomplete state space considered, we believe that it represents an interesting trade-off between optimality and flexibility. This very simple method can improve drastically the performance of an agent on a delayed MDP as presented in section VI, given that the delay has been accurately determined and is suited for transfer on systems with different delays.

V. Experimental Setup

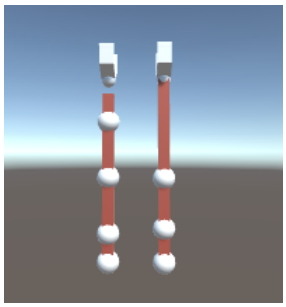
A. System Architecture and Robots

In this section, we briefly present the different robots adopted throughout these experiments. We tested our method on both physical and simulated robots to demonstrate its efficiency and versatility. The physical robots used were a serial arm braccio robot with 5 DoF and a 4 DoF serial arm (see Figure 4b). These DIY robots are cheap and usually hard to work with, given their low reliability. Still, we manage to use them efficiently in our experiments. We also considered their simulated counterparts (see Figure 4a).

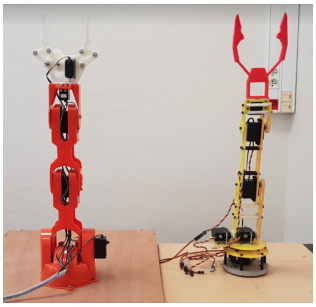
To sum things up, 5 different kinds of robots were used:

- **BAM**: the virtual BAM robot with the identity bases.
- **Robot 1**: the virtual braccio robot.
- **Robot 2**: the virtual 4 DoF robot.
- **Robot 3**: the physical braccio robot.
- **Robot 4**: the physical 4 DoF robot.

Significant offset was present in the robots joints, making each movement inaccurate. In this regard, the offsets first needed to be identified, in order to use analytical models efficiently on both physical robots.



(a) Robot 1 (left) and 2 (right).



(b) Robot 3 (left) and 4 (right)

Fig. 4: Robots considered for the experiments

A fixed webcam was used to obtain the required pose estimations with OpenCV. The control frequency was 10 Hz, which means the agent was observing the environment state and acting every 0.1 second. The nominal delay was in average 300 ms on the physical systems. We identified the delay by measuring the time between a command send to the robot and the observation by the agent that the robot moved.

On the simulation side, agent's training was performed in simulation using the Unity physic simulator with the ML-agent package introduced in [22], a set of convenient tools for RL with a complete and reliable implementation of several RL algorithms. The PPO algorithm [23] was used to create the neural network policies, as it provides a monotonous performance improvement while being perfectly adapted to continuous action spaces. On-policy algorithm are also known to deal better with delays. We trained four kinds of agents:

- **Delay Aware UNN Agent:** The BAM virtual robot is trained in simulation with exposition to randomized delays to create the UNN.
- **Delay Aware Vanilla Agent:** The agent is trained from scratch directly on the simulated robot, with exposition to randomized delays.
- Finally, we also considered their **delay unaware** counterparts, trained without exposition to delays, in order to display the benefits of our delay management approach.

These agents will be used for the transfers detailed in section VI-B (see Figure 1).

B. Task description

We display our method benefits on a 2D manipulation task (planar task), where a robot needs to keep a ball at a desired position on a gutter. In this regard, only 3DoF were required for the physical robots (base rotation and wrist roll unused). To further increase the gap between both robots, the Robot 4 was used as a 2 DoF robot (wrist pitch unused). The gutter is fixed at one end and held at the other end by the robot's effector which therefore decides of its orientation and, as a consequence, of the position of the ball (see Figure 5). This task can be formalized with the following MDP:

State: $s_t \in \mathbb{R}^{4+1}$: the ball position and velocity on the gutter, the effector height, the desired ball position and the system delay d for the delay aware agents.

Action: $a_t \in \mathbb{R}^n$ is the target joints position (n being the number of considered joints). However, the vanilla agent was not making any progress with a full access to the action space. Indeed, to balance the ball on the gutter, it is first needed to hold it properly. These desired body configurations are just a fraction of the full state space and it is very unlikely to discover them without any prior knowledge of the task. To ease the vanilla agent learning process, its action space was constrained to output joints offsets values w.r.t a reference joints position which maintained the gutter in an equilibrium position.

Reward:

$$r_t = \begin{cases} r - \beta|\theta_e| & \text{if } \ d_{des,b} < \delta \\ -\alpha d_{des,b} - \beta|\theta_e| & \text{else} \end{cases} \tag{5}$$

where r is a small positive reward, δ is the positive reward area and $d_{des,b}$ is the distance between the ball and the desired ball position. $|\theta_e|$ is the angle between the effector pose and the vertical plane, α and β a weighting constant. This penalty ensures that the effector is in the right orientation to hold the gutter properly for the vanilla agent. The effector orientation constraint for the UNN is handled by the output base, which means that β is set to zero when training the UNN.

C. Delay Aware UNN creation

Creating the UNN module means training an agent to balance the gutter and keep the ball at the desired position. As the BAM method showed better transfer results [19], we decided to use it to create the UNN module. In this setting the robot is assimilated to its effector and the UNN output

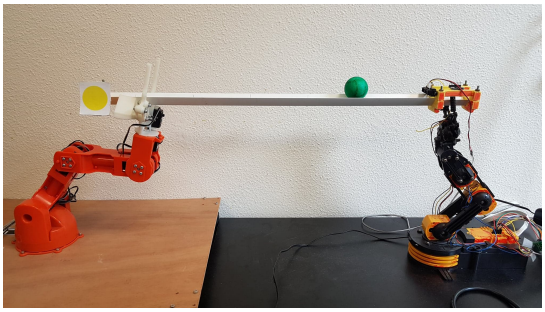


Fig. 5: Physical experiments setup. Left robot will perform the task, while the right robot is used only to hold one end of the gutter.

$o^{out} \in \mathbb{R}$ is a single value indicating at what height below or above the horizontal reference position of the gutter the effector should be. To take actions, the agent observes s^T, specific task information, as well as the effector height given by $m_i^r(s^r)$. For this task, we chose the intermediate state s^u and o^{out} shared between the UNN module and the bases to be the effector position. The UNN module was receiving an extra input $d \sim \mathcal{U}(0.1, 1)$ representing the current delay of the system during training. A delay range between 0 and 1 is recommended as it corresponds to a normalized input. In our case, it also corresponds to our actual delay in second, with 0.1s being the smallest delay possible for our control frequency. The delay was created on the simulator by stacking the observations in a FIFO buffer, before feeding them to the UNN module.

VI. Results

In this section, we present our results both on training and transferring on the chosen manipulation task. In particular, we compare the UNN agents with the vanilla agents with and without delay awareness. For further experiments, delay was added artificially to the real system with the same FIFO method seen in section V-C. All the hyper-parameters used during training are available in appendix A. A description of the robot morphology is presented in B. Code can be found at github.com/sabeaussan/DelayAwareUNN. Videos showing our results are available here.

A. Training

During the training, the desired ball position and system delay (for delay aware agents) were regularly changed to improve the adaptive capabilities and re-usability of the UNN. More precisely, a new delay d was sampled from $\mathcal{U}(0.1, 1)$ every 15 episodes. The desired ball position given to the model, varying between 20% and 80% of the gutter length, was also sampled from a uniform distribution $\mathcal{U}(0.2, 0.8)$ every 1000 training steps. Both the BAM agents and the Vanilla agents were trained for 4 millions steps. Figure 6 shows the cumulative reward obtained per episode. Only the term $d_{des,b}$ (distance between the ball and the desired position) common to both reward functions was considered for the comparison, as it reflects the agent overall progression

on the task. As shown in Figure 6, the BAM agents in both settings converge slightly faster than their vanilla counterparts. The BAM agents focus solely on the task, leaving robot specific considerations to their bases. This decomposition of the learning problem similar to hierarchical RL eases the learning process. It is also worth noting that introducing varying delay during training reduces the convergence speed, as the task becomes more challenging. However, in the UNN framework, this training overhead is outweighed by the increased reusability of the UNN module.

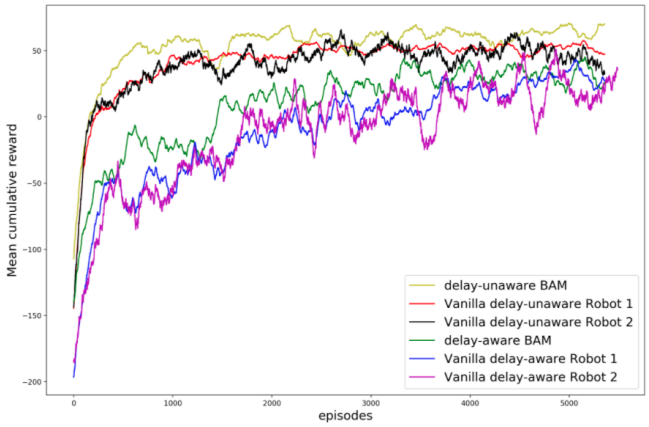


Fig. 6: Training curves. All the agents were trained for 400000 steps.

B. Transfer

There is two kinds of transfer to consider: simulation to real robot transfer and robot to robot transfer. The UNN framework mitigates the sim2real transfer problem by considering the real robot and the simulated one as two different robots, each one with its own bases, thus partially addressing the sim2real transfer as a robot to robot transfer. In this section we evaluate two methods of transfer

- **UNN transfer:** Once trained to convergence with the BAM robot, the UNN module is transferred to each robot of the set.
- **Vanilla transfer:** The vanilla agents trained on the simulated robot are directly transferred to their physical counterpart. This will serve as a baseline to study the UNN benefits for sim2real transfer.

The performance metric used was the integral of the absolute value of the error between the ball position and the desired ball position over time:

$$\int_t d_{des,b}\, dt \tag{6}$$

Where $d_{des,b}$ is the distance between the ball and the desired ball position. This metric has the advantage of taking into account both settling time and the steady state error (the closer to 0, the better). For a fair comparison, each experiment has been conducted with the same settings (same initial ball position and desired ball position). Performances displayed in Tables I and II were averaged over 50 episodes.

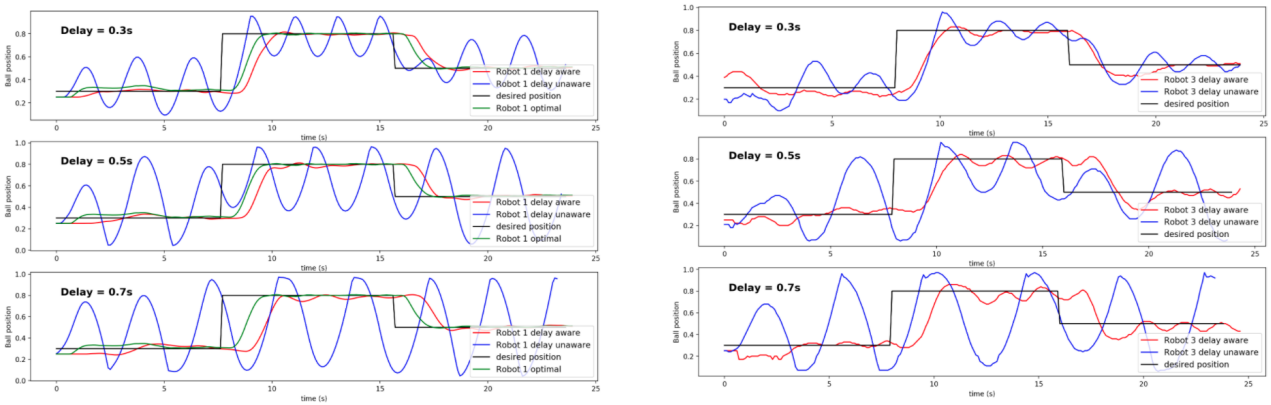
Robots/Delays	0.3	0.5	0.7
BAM	3.12 / 15.57	3.94 / 22.57	4.91 / 25.28
UNN Robot 1	3.26 / 11.84	3.96 / 20.12	4.98 / 23.48
UNN Robot 2	3.43 / 12.21	4.19 / 18.47	5.10 / 21.19

TABLE I: Sim2sim transfer. Performances obtained for the UNN transfer on the simulated robots. Results are displayed with delay aware method on the left / delay unaware method on the right.

1) Influence of delay: In this part, we evaluate the first contribution of this work: our delay management method, on both simulated and physical robots. Three delays were considered for the experiments: 300 ms (corresponding to the delay on the physical system), 500 ms and 700 ms. Figure 7 shows ball trajectories for the three delays considered, obtained by the UNN agents on robot 1 (virtual braccio robot) and 3 (physical braccio robot). On the simulation side, we added an optimal trajectory obtained with the delay unaware UNN agent acting on an undelayed environments to serve as a reference (see Figure 7a). The same agent was then exposed to the delays considered to study how quick performance deteriorates for unaware agents as the delay increases. As shown, agents not exposed to delays during training completely failed and systematically overshot when trying to get the ball at the required position in delayed environment. Figure 7b emphasizes the inability of the delay unaware agents to cope with the physical system immanent delay (300 ms) as the ball starts oscillating. Moreover, as the delay increases, the delay unaware agents tend to become unstable. As for delay aware agents, in the simulation, they still manage to follow closely the optimal trajectory.

Table I shows the performances obtained in sim2sim transfer with the UNN agents on both delay aware and unaware settings. It is shown that delay aware agents perform from 3.5 to 5.72 times better than their unaware counterparts. It is also clear from looking at Figure 7b and Table II, which shows the average performance after sim2real transfer, that dealing with delay in simulation greatly improves the results of the UNN agents once deployed on the physical robots. Vanilla agents also benefited from this delay management method, as shown in Table IIb, demonstrating the versatility of the proposed method.

2) sim2sim transfer: In this paragraph, we discuss the results obtained when transferring the delay aware UNN module from the BAM robot to robots 1 and 2 in simulation. We also compare the performance obtained against delay aware vanilla agents which learned the task from scratch on robots 1 and 2. As shown in Tables I and IIb, UNN-based approachs slightly outperform the policy of the vanilla agents for the robots and delays considered. We want to emphasize that the UNN module has been trained only once and on only one robot, the BAM robot, but still performs better than the vanilla agents specifically trained on robots 1



(a) UNN Robot 1

(b) UNN Robot 3

Fig. 7: Ball trajectory with 0.3, 0.8 and 0.5 as desired ball position.

Robots/Delays	**0.3**	**0.5**	**0.7**
BAM	3.12 / 15.57	3.94 / 22.57	4.91 / 25.28
UNN Robot 3	4.78 / 9.88	5.05 / 22.32	8.02 / 24.43
UNN Robot 4	5.86 / 15.72	7.45 / 22.42	8.76 / 24.41

(a) UNN transfer: BAM → robot 3 and BAM → robot 4

Robots/Delays	**0.3**	**0.5**	**0.7**
Vanilla Robot 1	3.32 / 17.33	4.22 / 26.17	5.56 / 31.48
Vanilla Robot 3	5.43 / 19.65	5.97 / 28.22	9.33 / 33.43
Vanilla Robot 2	3.78 / 18.63	4.81 / 26.45	6.16 / 32.48
Vanilla Robot 4	7.58 / 21.13	9.55 / 27.05	10.62 / 33.82

(b) Vanilla transfer: robot 1 → robot 3 and robot 2 → robot 4

TABLE II: Sim2real transfer. Performances obtained for the vanilla transfer and the UNN transfer on the physical robots. Results are displayed as delay aware method on the left / delay unaware method on the right.

and 2. These results demonstrate the appealing re-usability and effectiveness of the UNN module. In some cases, the delay aware UNN agents achieve zero-shot performances (e.g robot 1 on delay 0.5). In the worst case, the transfer efficiency is 90.9% (3.12/3.43), 100% being the performance obtained by the UNN module on the BAM robot. In average, the transfer efficiency is 97.7 % for robot 1 and 93.7% on robot 2. Ideally, the UNN module paired with any of the robots would yield similar performance as with the BAM robot if equation (1) was respected. However, in some cases the body configurations required to comply with the UNN commands are not precisely achievable by the robot. For instance, the desired effector position need some of the joints to rotate beyond their limits. This also explain why the UNN transfer is less efficient on the 2 DoF robot, as it is less expressive and has a harder time following UNN commands.

3) sim2real transfer: In this paragraph, we study the UNN methodology as a sim2real transfer tool. More specifically, we compare the performance obtained after transfer on the physical robots for Vanilla agents and UNN agents. In this case, both the UNN module obtained on the BAM robot and the vanilla agents obtained on the simulated robots, were transferred to the physical robots. As shown in Figure 7, the UNN agent on robot 3 still manages to put the ball at the desired positions without too much overshooting. Table II shows the results obtained. As usual, the agents trained in simulation and transferred to the real world show lower performance than their virtual counterparts due to the reality gap. However, they still manage to obtain decent performances. One notable result is that the delay unaware agents transferred to the physical robot obtain very poor performance unlike their delay-aware counterpart. Once again, UNN based agents outperform vanilla agents. Moreover, the UNN based transfer reaches up to 78% (100% corresponds to the BAM performance) in the best case, while the vanilla transfer reaches 70.6% (100% corresponds to the vanilla agent on robot 1). In average, the UNN sim2real transfer efficiency is 68% on robot 3 and 54% on robot 4, against 63% on robot 3 and 52.7% for robot 4 for the vanilla sim2real transfer. As mentioned earlier, this slight sim2real improvement can be attributed to the robot-agnostic nature of the UNN module. Indeed, even if the vanilla agents were trained in simulation with a virtual copy, it remains an inaccurate model of the physical robot. The UNN on the other hand ignores those discrepancies by considering the physical robot and the virtual one as two different robots, each with their own bases.

C. Discussion and perspectives

From the previous results, it appears clearly that the delay management method used considerably improves the performances when working with delayed environment, as its often the case on the real world. Moreover, the UNN approach not only achieves very efficient transfer between robots in simulation, but slightly improves sim2real transfer over vanilla transfer. However, the zero-shot sim2real transfer efficiency is nowhere near what was obtained for the sim2sim transfers but further training could be done on the physical robots to achieve better performance. As aforementionned, the UNN mitigates the sim2real transfer by considering the physical system as just another robot that can be interfaced

with the UNN module. Nevertheless, the UNN module which was trained in simulation can still overfit on its environment. As a result, the instructions given can be unsuitable if it is placed in a new domain with a slightly different state distribution, e.g the real world. Fortunately, the UNN approach can be combined with state-of-the art sim2real methods such as automatic domain randomization [5] to improve sim2real transfer.

VII. Conclusion

In this work, we studied the benefits of the UNN transfer for a sim2real application. More specifically, we addressed the delay management problem that occurs when working with a physical system by making the UNN "aware" of the latency of the system it is working with. By doing so, we extended the versatility of the UNN method and the range of compatible systems. We demonstrated this method efficiency by solving a dynamic manipulation task where delay management is paramount and showed that transfer across systems with heterogeneous delays and structurally distinct robots is possible. However, the UNN approach only is not sufficient for efficient sim2real transfer, but could be enhanced with other sim2real methods. This work empirically demonstrated the feasibility of our approach on a low dimensional task. Future work will investigate the efficiency of our delay-management method on a higher dimensional task.

References

[1] V. Mnih et al. Playing Atari with Deep Reinforcement Learning. In NIPS Deep Learning Workshop, 2013.

[2] D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419):1140–1144, December 2018.

[3] OpenAI et al. Dota 2 with Large Scale Deep Reinforcement Learning. arXiv:1912.06680, 2019.

[4] V. Tsounis et al. DeepGait: Planning and Control of Quadrupedal Gaits using Deep Reinforcement Learning. RA-L, 2020.

[5] OpenAI et al. Solving Rubik's Cube with a Robot Hand. arXiv:1910.07113, 2019.

[6] J. Donahue et al. Decaf: A deep convolutional activation feature for generic visual recognition. In Proceedings of the 31st International Conference on Machine Learning, volume 32 of Proceedings of Machine Learning Research, pages 647–655, 2014.

[7] M. Mounsif et al. Universal Notice Network: Transferable Knowledge Among Agents. In 6th International Conference on Control, Decision and Information Technologies (CoDIT), pages 563–568, 2019.

[8] K. Bousmalis et al. Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In IEEE International Conference on Robotics and Automation (ICRA), pages 4243–4250, Brisbane, QLD, May 2018.

[9] K. Arndt et al. Meta Reinforcement Learning for Sim-to-real Domain Adaptation. In 2020 IEEE International Conference on Robotics and Automation (ICRA), pages 2725–2731, Paris, France, May 2020.

[10] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 1126–1135, 2017.

[11] Y. Duan et al. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. arXiv:1611.02779, November 2016. arXiv: 1611.02779 version: 2.

[12] A. Gudimella et al. Deep Reinforcement Learning for Dexterous Manipulation with Concept Networks. ICRA, 2020.

[13] M. Mounsif et al. CoachGAN: Fast Adversarial Transfer Learning between Differently Shaped Entities:. In Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics, pages 89–96, 2020.

[14] A. Gupta et al. Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning. ICLR, 2017.

[15] C. Devin et al. Learning modular neural network policies for multi-task and multi-robot transfer. In IEEE International Conference on Robotics and Automation (ICRA), pages 2169–2176, Singapore, 2017.

[16] S. Behnke et al. Predicting away robot control latency. In RoboCup 2003: Robot Soccer World Cup VII, pages 712–719, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[17] T.J. Walsh et al. Learning and planning in environments with delayed feedback. Autonomous Agents and Multi-Agent Systems, 18(1):83–105, February 2009.

[18] S. Ramstedt and C. Pal. Real-Time Reinforcement Learning. NeurIPS, 2019.

[19] M. Mounsif et. al. Bam ! base abstracted modeling with universal notice network : Fast skill transfer between mobile manipulators. In 7th 2020 International Conference on Control, Decision and Information Technologies (IEEE-CoDIT), July 2019.

[20] K.V. Katsikopoulos and S.E. Engelbrecht. Markov decision processes with delays and asynchronous cost collection. IEEE Transactions on Automatic Control, 48(4):568–574, April 2003.

[21] Dimitri P. Bertsekas. Dynamic programming and optimal control. Vol.1. Number 1 in Athena scientific optimization and computation series. Athena Scientific Publ, Belmont, Mass, 2. ed edition, 2000. OCLC: 833754683.

[22] A. Juliani et al. Unity: A General Platform for Intelligent Agents. arXiv:1809.02627, May 2020. arXiv: 1809.02627.

[23] J. Schulman et al. Proximal Policy Optimization Algorithms. arXiv:1707.06347, August 2017. arXiv: 1707.06347 version: 2.

[24] P. Ramachandran et al. Searching for Activation Functions. arXiv:1710.05941, October 2017. arXiv: 1710.05941.

[25] J. Schulman et al. High-Dimensional Continuous Control Using Generalized Advantage Estimation. Proceedings of the International Conference on Learning Representations (ICLR), 2016.

Appendix

A. Neural network parameters

The neural networks used for the UNN module were feedforward networks with 2 hidden layers of 128 units each. The activation function used was the Swish function [24]. For the PPO algorithm, the discount factor γ is 0.99, the clipping hyperparameter ϵ is 0.2, the learning rate was 3.10^{-4} with a linear decay. The batch size was 1024, the buffer size was 12000 and the number of epoch was 5. The Generalized Advantage Estimation [25] method was used to compute the advantage function with $\lambda = 0.95$. For the reward function, $r = 0.3$, $\alpha = 0.4$ and $\beta = 0.03$

B. Robots morphology

The robots's servomotors were MG995. See below for robots link lengths :

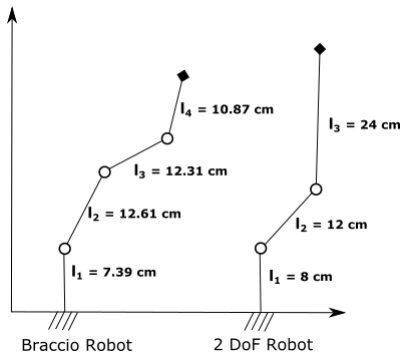


Fig. 8: Robots links lengths.