BAM! Base Abstracted Modeling with Universal Notice Network: Fast Skill Transfer Between Mobile Manipulators

Mehdi Mounsif[1], Sébastien Lengagne[1], Benoit Thuilot[1] and Lounis Adouane[2]

***Abstract*— Following recent trends, it appears that robot presence within human day-to-day lives is likely to grow and become ubiquitous. As many actors are engaged in this automation effort, it is plausible that the various cultural backgrounds of these actors will result in a broad range of different robots that will nevertheless need to perform similar tasks. Due to the excessively large number of experiences samples needed to successfully train a learning-based control policy, it would be remarkably useful to be able to efficiently transfer the skills acquired by a given agent to other, structurally distinct, robots. Accordingly, the BAM (Base-Abstracted Modeling) methodology proposed in this paper is a fast transfer learning approach that relies on a clear segmentation between the task model, that is a learned policy for solving a specific task and the learned robot control policy. The evaluation on two manipulation tasks using twelve different configurations of mobile manipulators demonstrates the strong potential of this approach as the segmentation results for more robust policies than *naive* methods and that an efficient transfer can be done in a fraction of the initial training time.**

I. Introduction

The rise of human societies was supported by various specific human traits, such as the human brain size or the opposable thumb. A key feature of this expansion was however our ability to share and transfer knowledge to our peers. Indeed, seen from a global point of view, shared insights allowed important population clusters to coordinate their actions and achieve goals that are beyond the reach of a single individual, while, on a more local scale, it enables unexperimented individuals to quickly learn from domain experts, thus fastening the generation process of capable elements of the society.

Reproducing this process of knowledge transfer within a population of robots is a highly desirable goal, due to resources needed to train a learning-based agent. For instance, in the context of an industrial chain, as shown in Figure 1, where several robots are liable to perform a similar assignment, it would be interesting not to have to train each robot separately, but instead, create a task knowledge module that can be shared between different instances, even if their kinematic structure (i.e: number of DoF, segment lengths) is different, thus reducing the time for recovering an operational process, should one robot be damaged for instance.

The BAM (Base-Abstracted Modeling) method proposed in this work focuses precisely on this transfer issue as it introduces a methodology to transfer knowledge between two differently structured entities. Specifically, it relies on a staged training process that creates a robot-agnostic, independent task model called the UNN (Universal Notice Network) that can then be seamlessly paired with a robot-specific controller (called the base modules). **This enables the latter robot to solve a task by using the knowledge created by another agent**, with minimal fine-tuning. In this setting, the BAM method is an environment modeling strategy that was designed to tackle potential issues that may arise when the UNN learns how to solve the task while using a poorly-conditioned robot module.

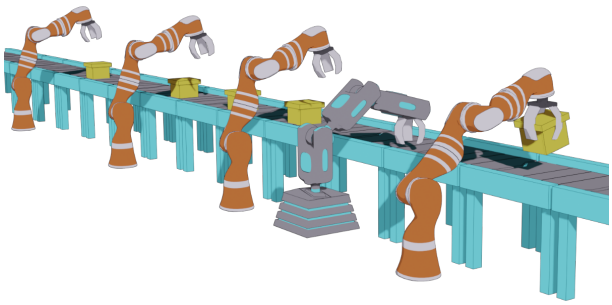


Fig. 1: Industrial integration of the Universal Notice Network approach

The UNN/BAM method capabilities are demonstrated in the context of mobile manipulators. Using three mobile bases (omni-directional, bicycle and differential), four robotic arm configurations and two environments (**Pick'n Place** task and the **Stacking** task), the experiments, set within a RL (Reinforcement Learning) framework, detailed below show that the concept of task knowledge advantages are twofold. Firstly, it is beneficial to the initial learning process, as it eases the reward function design and prevents the RL agent to fall within local minima. Next, it is particularly suitable for transferring skills between differently shaped entities, outperforming *naive* transfer both in final performances and training time perspective and even enabling zero-shot transfer in a particular case.

II. Related Works

Perform complex motions to complete a task is still an open issue. While there exists optimization methods based on complete models [1], these are usually computationally intensive and may not satisfy real-time constraints. To tackle this issue, the community has also been leveraging predictive methods that are completely suitable for real-time control. However, this execution efficiency is usually achieved by

[1]Université Clermont Auvergne, CNRS, SIGMA Clermont, Institut Pascal, F-63000 Clermont-Ferrand, France. mehdi.mounsif@uca.fr

[2]Université de Technologie de Compiègne, CNRS, Heudiasyc, F-60200 Compiègne, France

sacrificing the model complexity and thus considering simplified versions that may not have enough degrees of freedom to express the full problem spectrum [2]. These methods present the strong and undeniable advantage of allowing direct transfer of skills between different agents. However, while it is possible to conceive hybrid controllers that can mitigate each approach's weaknesses, they nevertheless rely on an analytical model for both the task and the robot that can be hard if not impossible to define. As a result, it is common to integrate various hypotheses and assumptions that highly impact the precision of the model.

Data-driven approaches, in particular RL (Reinforcement Learning) have intensely been discussed in recent works as these methods are model-free [3] and are able to generate control policies for highly complex and non-linear tasks [4], [5]. Nevertheleess, training these performant policies require such an important computational training budget that the benefit of task knowledge transfer become obvious. However, as opposed to usual practices in CV (Computer Vision) and NLP (Natural Language Processing) [6], [7], RL architecture (being shallow and without dimensional bottleneck) are not designed for knowledge transfer to other entities. This usually results in an entangled knowledge representation. As a matter of fact, the RL paradigm focuses rather on training a single couple agent/task. In some cases, the author investigate whether the agent's previously learned representations can be repurposed by modifying slightly the task [8], [9]. In this view, to broaden an agent's usability, some works propose entropy-based loss functions [10] to increase the agent's curiosity and exploration as well as meta-Learning methods [11]. Although these works do enhance the relative reusability of an agent, they are still directly aiming at a single agent/single task frame while there exist numerous real-world cases that would rather benefit from transferring knowledge from one entity to a distinct one. This issue lacks representation in literature and to the best of our knowledge, there is so far no work that focuses on transferring task knowledge between differently shaped entities.

III. Universal Notice Network

The UNN (Universal Notice Network) presented in [12] is a framework that implements the idea of knowledge segmentation between the agent and the task. Using 2D robots, the authors of this work show that it is possible to pair a policy learned using RL approaches with analytical controllers to achieve and transfer simple tasks. The main UNN principle is derived from the idea that a task can be achievable by any agent, by following the right set of instruction, provided the agent is able to move adequately. For this last constraint to be satisfied, the agent is pre-trained on a primitive task that enables it to acquire basic motor skills. This process is shown in Figure 2. Once trained, the UNN module can be transferred to a structurally different agent from the one used to create it and consequently enable this new agent to perform the task without prior interaction with it. In Figure 3, stage I corresponds to the initial choice of task and agent. The computation pipeline is created in stage II by pairing the trained task UNN and the base module of the performing agent (this agent is different from the one that initially constructed the UNN). This makes possible the achievement for this task by the new agent as shown in stage III. The next paragraph describes the computational pipeline set up, while the rest of this section provides details on the modules creation.

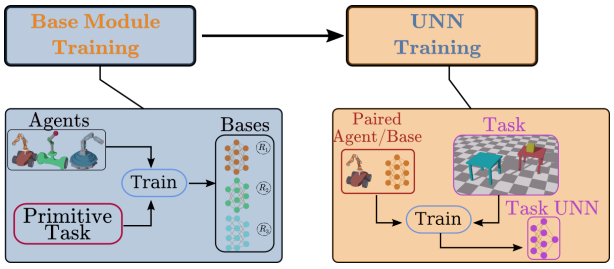


Fig. 2: The original UNN Pipeline staged training first creates a primitive agent controller (in blue) for a chosen agent configuration. The UNN for the task relies on the generated primitive motor skills to learn a successful policy (in red).

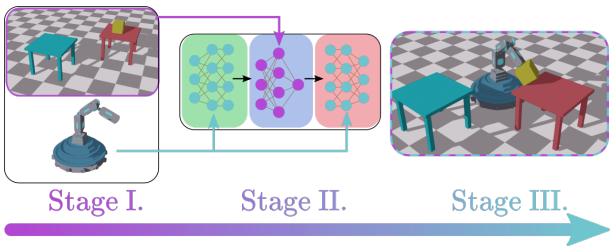


Fig. 3: The successive steps for deploying the transferred UNN to a new configuration

A. The UNN pipeline

In practice, the UNN pipeline is a model composed of three sub-modules, $m_i^r, m_u^{\mathcal{T}}, m_o^r$, as shown in Figure 4. The first and the last parts of this model m_i^r, m_o^r, respectively the input (green) and the output (red) modules, are specific to the robot r, while the center model $m_u^{\mathcal{T}}$, the UNN (blue), is designed to be robot-agnostic and specific to the task $\mathcal{T}$. Specifically, the state vector that is observed at each timestep, can be split into two parts $s^{i,r}, s^{\mathcal{T}}$, respectively holding data intrinsic to the considered robot and task-related information, independent from the agent. The agent specific state $s^{i,r}$ is used by its input module to compute a state agent representation:

$$s^{i',r} = m_i^r(s^{i,r}) \tag{1}$$

The UNN (i.e.: the task module), conditioned by the task observation vector $s^{\mathcal{T}}$, uses the processed agent representation vector to compute a vector that can be seen as a high-level instruction:

$$o^{out} = m_u^{\mathcal{T}}(s^{\mathcal{T}}, s^{i',r}) \tag{2}$$

Finally, the effective robot action is then recovered by concatenating the initial intrinsic vector $s^{i,r}$ with o^{out} and feeding it to the output base, that is:

$$a^r = m_o^r(o^{out}, s^{i,r}) \tag{3}$$

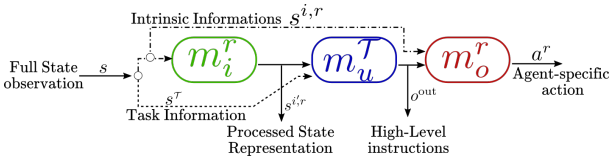


Fig. 4: The UNN Pipeline is composed of three stages: the input and output bases (in green and red) are specific to the robot, while the UNN (blue) is task-related

In the cases discussed below, the UNN and the ouput module are neural networks, but in practice, there is no theoretical restriction on the computational model used, as long as the vector returned by the modules has the expected dimensionality. From a functional point of view however, to ensure that the UNN is compatible with transfer, it is necessary to design the vector $s^{i,r}$ that is passed between the UNN and the output module in a way that ensures the Markov property is satisfied: it must encompass enough information for selecting the next action without needing to consider additional previous steps. The same logic is applied to the vector passed between the input module and the UNN.

B. Base Abstracted Modeling

Ultimately, the goal is, for any UNN/agent couple, to find the module functions $m_i^r, m^{\mathcal{T}}, m_o^r$, that produces the movement $\mathcal{M}$ generated by actions a^r provided by the pipeline for solving a task τ and that must ensure the set of constraints g_τ to perform the task and the set of constraints g_r to ensure the physical limits of the robot, such as:

$$\begin{aligned} &\text{find} && m_i^r, m^{\mathcal{T}}, m_o^r \\ &\text{such as} && g_\tau(\mathcal{M}) \leq 0 \\ & && g_r(\mathcal{M}) \leq 0 \\ &\text{With:} && \mathcal{M} = f(a_r) = f(m_o^r(m_u^{\mathcal{T}}(s^{\mathcal{T}}, m_i^r(s^{i,r})), s^{i,r})) \end{aligned} \tag{4}$$

However, due to the RL tendency to fall into local minima, the UNN can discover a successful strategy for solving the task that depends on its body configuration (for instance, blocking an object between two articulations). In these cases, Equation 4 is longer respected. While this is not an issue for the current agent, it is detrimental for the efficiency of a future transfer to an agent with a different structure. To ensure that the UNN constraints are not entangled with the agent's, the Base Abstracted Modeling (BAM) is introduced. This approach relies on a environment modification that assimilates the robot to its effectors, thus creating a virtual robot, as shown in Figure 5. On the formulation side, we set the bases models m_i^r, m_o^r as identity functions and provide directly the UNN output to the command. These two modifications release most of the constraints that would have been distilled by the pre-trained base and prevent the UNN from learning a configuration-specific strategy, resulting in the creation of a UNN closer to a model-agnostic setting.

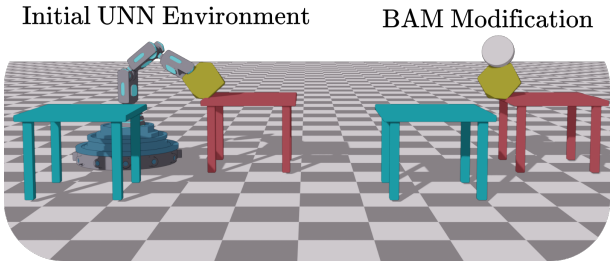


Fig. 5: The BAM version of the environment allows the UNN to be bias-free

C. Learning techniques

In the experiments presented below, the UNN is optimized *via* a combinaison of reinforcement learning methods. The main technique used is PPO (Proximal Policy Optimization [3]), a policy gradient algorithm that maximizes the expected sum of rewards:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{\inf} \gamma^t \times r_t] \tag{5}$$

where r_t is the reward obtained at time t by the stochastic policy π_θ parameterized by a vector of parameters θ, γ is a discount factor and $\mathbb{E}$ represents the averaging operator. PPO also relies on an actor-critic with trust region to improve the stability and robustness of the gradients estimation.

While RL excels in reactive environments, it is usually less straightforward to design reward functions for sequential environments like those presented in Section IV-B. Indeed, in these cases, tuning the reward function in order to balance between a negative reward that may induce divergence in the agent's behaviour and a positive reward that may lead the agent to accumulate rewards without a clear line of action is very subtle and requires a considerable amount of tweaking. Thus, in order to improve the learning performances, PPO is coupled with two additional techniques:

- Pre-training with a Behavior Cloning based [13] approach
- Inverse Reinforcement Learning [14]

These two techniques, that seamlessly integrate with UNN/BAM approach, leverage a small dataset of expert demonstrations (in our case, a human operator) and provide a supplementary signal for guiding the agent within the environments.

IV. Experimental Setup

A. Robots

We consider three mobile bases configurations (Omnidirectional, bicycle and differential), as well as four different manipulator types (KUKA, BLUE, Generic 2, Generic 3). Which, in total, leaves us with 12 possible configurations, three of them are displayed in Figure 6. The bicycle mobile base is non-holonomic and relies on two values: thrust and

steering, for moving. The differential base is also non-holonomic and uses two torque values to compute the resulting direction. Finally, the omnidirectional base is the simplest as it is holonomic and the two values determine linearly its next position. Concerning the manipulators, both the KUKA and BLUE robots have 5 DoF. The Generic arms (rightmost configuration in Figure 6) have 2 DoF per articulation, which leads to a total of 4 and 6 DoF for the agents considered through our experiments. All serial manipulators are controlled using joints target position.

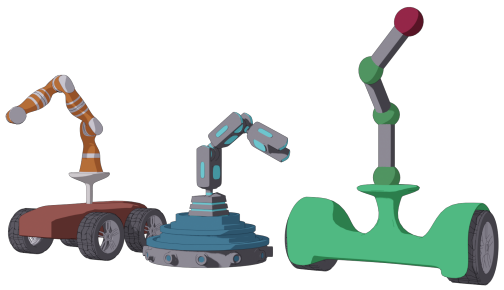


Fig. 6: A subset of the possible agent configurations. From left to right: Bicycle KUKA, Omni-directional BLUE, Differential G3. Parts can be exchanged between agents.

B. Environments

This section presents the two environments, implemented in the ML-Agents framework (introduced in [15]), that were created in order to demonstrate the UNN and the BAM advantages. An additional environment, the **Reaching** environment is used to train the base modules and is presented below. For each of these environments, we explain the general goal, define the task-related states and detail the extrinsic reward signal used to guide the policy in the environment.

In these environments, the UNN expects a vector $s^{i'}$ from its input base and always outputs the same type of information o^{out}. In practice, we have:

- The processed intrinsic information $s^{i'} \in \mathbb{R}^6$ consist of the end effector linear position and velocity, computed by an analytical approach. Specifically, the input base relies on a forward kinematic approach to compute the processed intrinsic information.
- The UNN outputs $o^{out} \in \mathbb{R}^3 \times \mathbb{B}$, composed of desired effector linear velocity and a Boolean for sucker action.

Reaching Task: This is the primitive base module training environment. Consequently, the state and actions of this specific task are agent-related. This task involves touching a given point in space with the agent's effector. The target point is regularly moved in order to force the agent to use its mobile base to get in range. This task features the following MDP:

- State: $s_t \in \mathbb{R}^{16+n}$: mobile base pose $\in \mathbb{R}^3$, arm pose $\in \mathbb{R}^n$ (n joints positions), vector to target $\in \mathbb{R}^3$, LIDAR sensors $\in \mathbb{R}^{10}$
- Action: $a_t \in \mathbb{R}^{2+n}$: mobile base movement vector $\in \mathbb{R}^2$ (specific to each mobile base configuration) and target joints speed $\in \mathbb{R}^n$.
- Reward: $r_t = -\alpha \times d_{E,T} + \sum_i \beta_i \times c_i$ where $d_{E,T}$ is the distance between the effector and the target, α is a normalizing constant and $\beta_i \times c_i$ is a weighted constraint on the arm to prevent it from falling into undesired local minima

For training the UNN and evaluating the BAM approach, we created two custom environments.

Pick'n Place Task: the agent is expected to pick up an object from one table and drop it to a given point in space. In this case, the task-related state is $s_{pap} \in \mathbb{R}^6 \times \mathbb{B}$, composed of the current object position, the drop target position and a boolean for signaling whether the cube is held or not. The agent relies on the following reward function:

$$r_p = \alpha(d_0 + d_1)^{-1} \tag{6}$$

where r_p, the reward for the **Pick'n Place** environment, depends on α a positive scaling factor, d_0 the distance from the object to the effector and d_1 the distance from the object to the drop target position.

Stacking Task: the goal is to stack a number of cubes. The task-related state is conceptually similar to the Pick'n Place, that is $s_{\text{stacking}} \in \mathbb{R}^{3*c} \times \mathbb{B}^4$ where c is the number of cubes to stack, in our case $c = 4$. We also design a reward that penalizes the agent according to the summed distance between each of the cube and their ideal position. That is:

$$r_s = [\sum_{i=1}^{4} \alpha_i(d_{0,i} + d_{1,i})]^{-1} \tag{7}$$

where r_s, the **Stacking** task reward, depends on α_i a positive scaling factor that is adjusted according to the agent progression in the task, $d_{0,i}$ the distance from the cube i to the effector and $d_{1,i}$ the distance from the cube i to its target position.

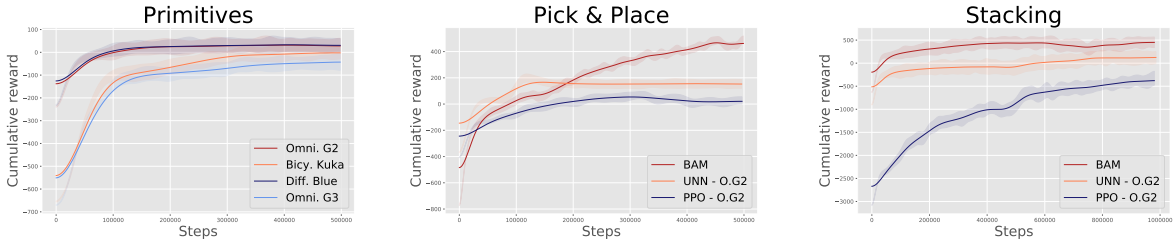
V. Results

In this section, the results of our method on both training and testing setup, over all environments are presented. In particular, we compare our architecture, the BAM modeling, with several baselines, namely the *naive* PPO approach and the UNN segmentation. A video of our results is available at `https://bit.ly/324Sxnz`.

A. Training

During training, we monitor the agent's evolution through the mean cumulative reward per episode. This metrics is common in RL and serves as a measure of how well the agent behaves within its environment.

Curves in Figure 7a are specific to our architecture and measure the output base training performances with respect to the reaching task. Figure 7b and Figure 7c respectively display these results for the whole architecture (bases +



(a) Primitive training on 4 predefined configurations (b) Comparative Pick'n Place training performances (c) Comparative Stacking training performances

Fig. 7: Learning performances over various environments

UNN) for the **Pick'n Place** and **Stacking** environments using an Omnidirectional Generic 2 robot configuration. Due to the intermediate model outputs, constrained through the bases, the UNN is able to learn the task more efficiently than a *naive* agent. As a matter of fact, we can see that both UNN-based techniques, in red and orange, respectively representing the UNN and the BAM approaches, clearly outperform the *naive* PPO approach, in blue. We note that the BAM (red) and UNN (orange) curves both reach a higher final cumulative reward than the *naive* PPO, while also being faster, indicating a better performance at solving the task overall.

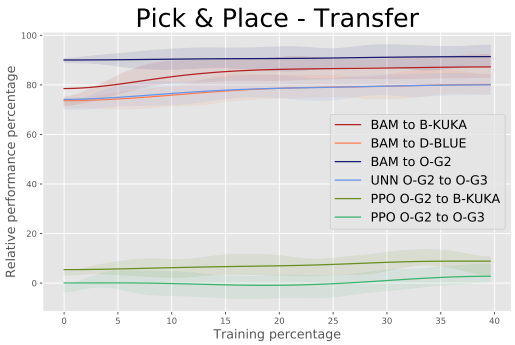
B. Transfer

UNN-based approaches aim at making the skills transfer between agents of diverse constitutions possible. Hence, after training, we pass pre-trained models on a given configuration to another one with a different structure and compare their performances. In an ideal case, pairing the UNN would not require any fine-tuning. However, in practice, most cases require additional training iterations for the UNN to complete the adoption by another agent configuration. We display some examples of this process in Figure 8a and Figure 8b, respectively for **Pick'n Place** and **Stacking** environments. These curves show for each environment how fast the performance can be recovered for a given configuration. Specifically, the y-axis measures the relative mean reward of the current agent projected in the lowest-to-highest interval from all initial agents. The x-axis also indicates how much of the initial training time was necessary to reach said performance. For the **Pick'n Place** environment, see Figure 8a, it appears that transfer within the *naive* framework is not efficient and even detrimental to the agent. The policy weights when transferred to another agent do not provide a decent initialization point and, through our experiments, these agents systematically failed at the task. In the contrary, using the UNN-based techniques allows to jump-start the new agent configuration with an initial level of performance that exceeds 70% of the previous agent and allows to recover almost the full performance, evaluated by the mean cumulative reward per episode, in a fraction of the previous training time, mostly depending on the base conditioning. For the **Stacking** environment, similar observations emerge from the Figure 8b where, again, the UNN-based methodology allows agents with different body configuration to access to the task knowledge, thus enabling quicker performances recovering.

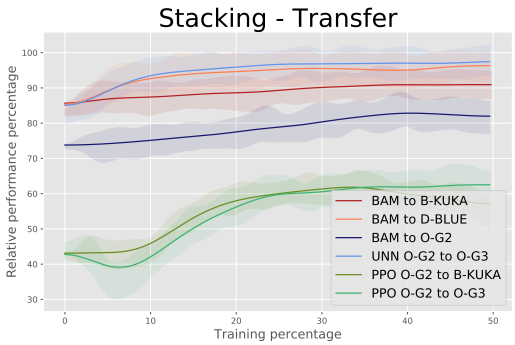
While mean reward is a very common metric in RL, we also consider test time performances by averaging over 100 runs the number of steps necessary to complete the task for various agents in both environments. In this setting, the best agents are the fastest and need the lowest number of steps to complete the task. Table I details results of this evaluation for **Pick'n Place** environment. In this case, we observe very contrasted results. No *naive* PPO agent was able to succeed in the task after being transferred, while the UNN-based transfer was particularly beneficial. Indeed, in these configurations, the performance was mostly conserved, with differences being related to the output base quality and movement capabilities (that is, it takes more time for the bicycle mobile base to reach a specific point than it does for the omni-directional base.) This is further confirmed by results presented in Table II, concerning the **Stacking** task, where test performances are globally similar for all UNN-based agents with a transferred task-model. While these results are not sufficient to unequivocally determine an optimal configuration, they do however show that in both environments, the UNN and the BAM method yield an increase in performance and allow transfer between entities. This is due to the fact that, in this modelling approach, the UNN can focus on the task at hand, without having to learn simultaneously how to correlate external perceptions to low-level orders.

TABLE I: Transfer: Pick'n Place

Archi.	Config.	Agent	Training %	Perf.
PPO	Initial	Omni. Generic 2	100%	1355.4
	Transferred	Omni. Generic 3	40%	Fail
		Bicy. KUKA	40%	Fail
		Diff. BLUE	40%	Fail
UNN	Initial	Omni. Generic 2	100%	848.8
	Transferred	Omni. Generic 3	20%	784.8
		Bicy. KUKA	20%	801.7
		Diff. BLUE	20%	443.6
BAM	Initial	-	100%	412.3
	Transferred	Omni. Generic 2	20%	751.6
		Bicy. KUKA	20%	707.9
		Diff. BLUE	20%	747.7



(a) Pick'n Place transfer fine-tuning performances



(b) Stacking transfer fine-tuning

Fig. 8: Fine-tuning performances using various architectures

TABLE II: Transfer: Stacking

Archi.	Config.	Agent	Training %	Perf.
PPO	Initial	Omni. Generic 2	100%	4571.9
	Transferred	Omni. Generic 3	40%	Fail
		Bicy. KUKA	40%	Fail
UNN	Initial	Omni. Generic 2	100%	3547.6
	Transferred	Omni. Generic 3	20%	4004.1
		Bicy. KUKA	20%	3966.1
		Diff. BLUE	20%	3847.9
BAM	Initial	-	100%	2879.3
	Transferred	Omni. Generic 2	20%	3540.4
		Bicy. KUKA	20%	3391.8
		Diff. BLUE	20%	3312.7

C. Discussion

Through our experiments, it clearly appears that the UNN output base's quality is paramount for establishing acceptable performances. Indeed, the overall UNN performances, both for learning and transfer are tightly tied to the stability and robustness of the bases. More specifically, RL-trained bases do not generalize very well to a distribution of states that does not match the initial training distribution. As a result, it is difficult to predict the agent's behavior for remotely visited states. To prevent this and increase the base reliability, it is important to ensure that the base training visits the state-space regions that will be required to perform the task. This additional precaution implies that the primitive task should be carefully crafted and leads to wonder whether this whole setup is necessary. The visual observation of a *naive* agent provides some clues to address this concern. As a matter of fact, the *naive* version relies solely on the reward to guide the agent's conduct, and, even though these agents are likely to find a way to perform the task, no control or constraint are included, which might result in unwanted body configurations. While it is theoretically possible to include behavioral terms in the reward function, it is difficult to foresee the effects of such constraints and, in practice, this accumulation of weighted reward terms provides a noisy signal to the agent, drowning the task reward signal with the penalties for unwanted or encouraged behavior, hence usually minoring the performance. Using the UNN architecture provides a clear improvement from this point of view because the base's training can be more thoroughly controlled, thanks to the carefully crafted primitive task, without being distracted by the task reward, thus resulting in a more reliable agent.

This analysis was the starting point for the BAM introduction that lessens the difficulty for learning the tasks for the UNN, placing instead the burden on the interface UNN-base. Through this segmentation and a meticulously designed primitive task, it is possible to create more efficient policies that outperform the *naive* PPO agent on a variety of environments.

VI. Conclusion

The BAM approach presented in this work is a fast transfer learning method that dissociates the agent's control logic from the task it is supposed to accomplish. This technique also introduced an environment modeling modification that tackles issues related to RL-policy learning bias, making the UNN method more scalable and robust. We assessed the viability of this technique, using two custom 3D dynamic environments and several agent configurations, by comparing it with a current state-of-the-art algorithm and demonstrating that the task model segmentation implies strong benefits both during the learning phase, as the performing agent is able to learn faster and reach higher rewards by not being distracted by low-level parameters, and also during transfer. Specifically, the transfer, which is likely to fail using *naive* methods is not only possible using the BAM method but quite fast, even enabling zero-shot transfer in some cases. Future works will investigate the creation of a structural distance metric to evaluate how different two robots can be to still benefit from the transfer and whether it is possible to project the agent state representation within a shared latent space to avoid manually crafted representations.

Acknowledgment

This work has been sponsored by the French government research program Investissements d'Avenir through the RobotEx Equipment of Excellence (ANR-10-EQPX-44) and

the IMobs3 Laboratory of Excellence (ANR-10-LABX-16-01), by the European Union through the program of Regional competitiveness and employment 2007-2013 (ERDF - Auvergne Region), by the Auvergne region and the French Institute for Advanced Mechanics.

REFERENCES

[1] S. Lengagne, J. Vaillant, A. Kheddar, and E. Yoshida, "Generation of whole-body optimal dynamic multi-contact motions," International Journal of Robotics Research, p. 17, Apr 2013.

[2] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in Fast Motions in Biomechanics and Robotics, ser. Lecture Notes in Control and Information Sciences, M. Diehl and K. Mombaur, Eds. Springer Berlin / Heidelberg, 2006, vol. 340, pp. 65–93.

[3] 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

[4] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," CoRR, vol. abs/1808.00177, 2018. [Online]. Available: http://arxiv.org/abs/1808.00177

[5] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," 2019.

[6] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin, "Distilling task-specific knowledge from BERT into simple neural networks," CoRR, vol. abs/1903.12136, 2019. [Online]. Available: http://arxiv.org/abs/1903.12136

[7] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," CoRR, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[8] B. Trapit, P. Jakub, S. Szymon, S. Ilya, and M. Igor, "Emergent complexity via multi-agent competition," arXiv - OpenAI Technical Report, 2017.

[9] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent Tool Use From Multi-Agent Autocurricula," arXiv e-prints, p. arXiv:1909.07528, Sep 2019.

[10] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," CoRR, vol. abs/1802.06070, 2018. [Online]. Available: http://arxiv.org/abs/1802.06070

[11] L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson, "CAML: Fast Context Adaptation via Meta-Learning," ArXiv e-prints, Oct. 2018.

[12] M. Mounsif, S. Lengagne, B. Thuilot, and L. Adouane, "Universal notice network: Transferable knowledge among agents," in IEEE - International Conference on Control, Decision and Information Technologies (IEEE-CoDIT 2019), 2019.

[13] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Integrating behavior cloning and reinforcement learning for improved performance in sparse reward environments," 2019.

[14] J. Ho and S. Ermon, "Generative adversarial imitation learning," CoRR, vol. abs/1606.03476, 2016. [Online]. Available: http://arxiv.org/abs/1606.03476

[15] A. Juliani et al., "Unity: A general platform for intelligent agents," CoRR, vol. abs/1809.02627, 2018.