Deep Reinforcement Learning for Task Planning of Virtual Characters

Caio Souza[1] and Luiz Velho[1]

IMPA - Instituto de Matemática Pura e Aplicada
lvelho@impa.br

Abstract. Intelligent agents are a long-standing area of interest in Artificial Intelligence. This paper uses deep reinforcement learning to develop an intermediate time scale controller for a virtual character. Decoupling the learning of *motion synthesis* controller (fine time scale) of the *task planning* controller (intermediate time scale) speeds up the learning. It also allows more control over the motion synthesis controller's animations and freedom in its choices. Finally, we compare and analyze controllers trained with various modeling, including visual sensing, for our fetch environment.

Keywords: virtual characters, intelligent agent, task planning, deep reinforcement learning

1 Introduction

Achieving a general intelligent agent is a long goal of AI. These agents have a broad range of applications, from automation to personal assistants and have been taking great advantage of recent advances of Deep Learning techniques (Bengio et al, 2007; LeCun et al, 2015). Current "state-of-the-art" agents are more focused on low-level controllers and working on specific narrow tasks or take advantage of task-specific context or protocols (i.e., Naderi et al (2017); Lee et al (2018b); Hong et al (2019); Merel et al (2020)).

Figure 1 presents a hierarchical abstraction of a broad agent. First, the low-level comprises the *motion synthesis*, which directly controls the agent actuators; next, the mid-level controls *task planning*: a series of actions directed by a goal; lastly, the high-level is in charge of *task selection* controlling the active task and goal. This type of abstraction bridges the concept of time-scale objectives in an explicit hierarchy.

Recently, there are extensive works in the literature covering motion synthesis (Geijtenbeek et al, 2013; Liu et al, 2016; Peng et al, 2016; Holden et al, 2016, 2017; Liu and Hodgins, 2018; Peng et al, 2018a; Yu et al, 2018; Zhang et al, 2018; Jiang et al, 2019; Luo et al, 2020; Peng et al, 2018b; Lee et al, 2014, 2018a; Xu et al, 2019), but works covering the mid and high levels are less numerous and usually are bound to specific low-level controllers or other specificities of complex fields such as robotics.

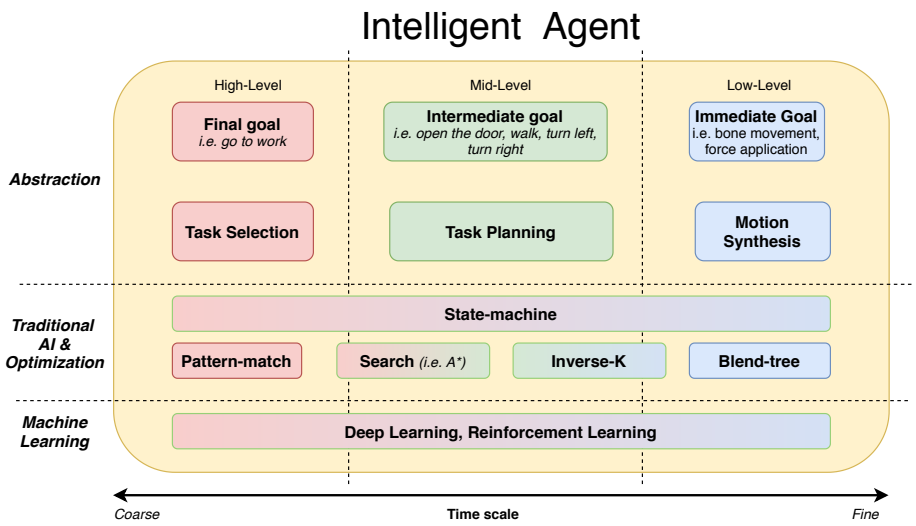

Fig. 1. Hierarchy levels of an intelligent agent and examples of solutions for each level.

Here we investigate how to develop these task planning behaviors for virtual characters using deep reinforcement learning. The virtual character chosen is a dog named *DogBot*. For simplicity, it uses a traditional parametric low-level controller composed of state-machines, blend-trees, and standard animations, a mid-level controller comprising of the learned behavior and finally, the high-level is also a state-machine for trigger-based tasks. We evaluate various controllers for a fetching behavior using hand-crafted real-valued and third-person visual observations, continuous and discrete actions, and other environmental design choices.

Our modeling appeals to interactive storytelling applications where there is a guided narrative, but the non-playable characters need to adapt to dynamic environments. These applications impose some constraints, for example, the choice for the motion synthesis module, which usually requires naturalness more than physical accuracy for animations. Nevertheless, the explicit hierarchy and a guided narrative can also bring advantages. The first allows us to focus solely on the mid-level development and re-use the agent and environment for training with any motion synthesis module; the second will enable us to induce specific protocols (or expected actions) based on the narrative context simplifying the task selection. Lastly, deep reinforcement learning removes the computational burden of traditional planning methods based on optimization & search.

2 Related work

Developing intelligent agents tackling high to low-level controls (inclusive using vision-based sensing) are not new. Terzopoulos et al (1997) produced artificial fishes handling the motion synthesis controller with dimension reduction techniques and task planning with optimization, combining different sequences of low-level controllers. While it achieved good results, it required craftsmanship for each part of the system and had low scalability for today's standards. Lin (1992) investigated reinforcement learning usage with artificial neural networks for planning but was uncertain if it could scale for more difficult tasks. More recently, deep learning developments and its successful application with reinforcement learning achieving super-human performance on Atari games (Mnih et al, 2013, 2015) addressed the scalability issue.

Since the success with Atari games, Reinforcement Learning has been gaining momentum for motion synthesis varying from locomotion (Peng et al, 2016, 2017; Yu et al, 2018; Lee et al, 2019; Jiang et al, 2019; Luo et al, 2020; Merel et al, 2020; Ling et al, 2020); locomotion with balance (Liu and Hodgins, 2017); basketball dribbling (Liu and Hodgins, 2018); body skill mimics such as cartwheel and backflips (Peng et al, 2018a,b); drone flying (Xu et al, 2019) and others.

A few works use visual input sensing. Mnih et al (2013) uses a 2D view of Atari games for planning directed by a high-score goal. Levine et al (2016) uses an end-to-end scheme for training a 7-DoF robotic arm. While the input comes from a 3D environment, the camera position is fixed and resembles a view-from the top, being close to global sensing. Nakada et al (2018) uses a complex foveated 3D vision and accurate biomechanical physical motion synthesis for head-neck and limbs, but lacks full-body motion. Given its complexity, kinematic approaches addressing low-level controllers (i.e., (Zhang et al, 2018; Starke et al, 2020)) may be more suited for real-time applications expected to run on commodity hardware. Similar to ours, Merel et al (2018, 2020) uses a first-person camera that translates to learning from a partially observable environment and needs additional sensing inputs for the task and proprioception, given its physics-based environment. Our approach relies on a third-person camera that stills partially observable and accounts for the kinematic character proprioception without the need for additional or hand-crafted inputs of the agent's state.

Solving task-planning is relevant for many fields; classical solutions for planning involve search or optimization through state space, which can become intractable for high-dimensional or high-dynamic environments. In this regard, heuristics and online approaches have mainly been researched (and surveyed: heuristics for robotics Mac et al (2016), online Ross et al (2008)) trying to circumvent the shortcomings as mentioned earlier. Various techniques were developed in computer graphics, most of them tightly coupled to their motion synthesis modules. A few examples are Levine et al (2011) using an $A*$ variation for dynamic path planning, Agrawal and van de Panne (2016) applying optimization for foot-step planning, Naderi et al (2017) planning climbing routes also through optimization, Lee et al (2019) using deep reinforcement learning for trajectory mimics.

Interestingly, the most recent work of Ling et al (2020) uses DRL to control locomotion tasks such as path following and maze runner. They call their approach a "model-then-control", resembling our decoupled modules for motion synthesis and task planning, and uses a simple vision sensing with 16 rays for the maze runner task. We believe that decoupling each module allows for a better understanding and solution of each hierarchy level. For the specific case of task planning, focusing on the modeling of the learning environment work as a *learning methodology*, which can be applied to different motion synthesis modules, and hence can be a more general strategy.

3 Overview

Our agent implementation uses the *Unity3D* game engine and its machine learning framework *ML-Agents*(Juliani et al, 2018). A diagram of the agent modeling is shown in figure 2. The agent is split into three levels (Figure 2 in yellow); first, the task selection communicating with the environment, waiting for specific conditions or triggers to select the appropriate behavior. Next, the task planner uses deep reinforcement learning; it takes various sensing of the environment and output actions such as turn left or right, synthesized into animations by the lower-level controller.

The realization of these schematics is our dog agent, later evaluated on a fetch game scene using different learned controllers. In the next section, we start with a brief background of Markov Decision Processes followed by detailed modeling of our agent and environment properties.

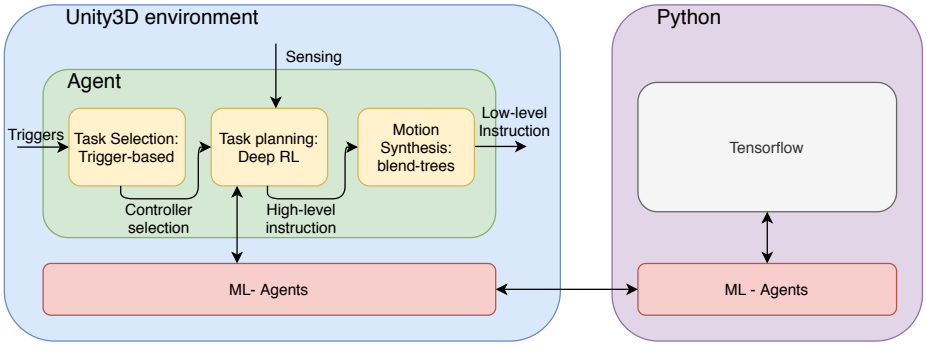

Fig. 2. Diagram of our agent modeling and the connections between each part. Note that the python connection is only used while training.

3.1 Background

Markov Decision Processes are the mathematical framework that idealizes sequential decision process. It is composed of a tuple (S, A, T, R), where S is the

set of states; A is the set of actions; $T : S \times A \times S \to [0, 1]$ is a probability distribution modeling the agent-environment dynamics with $T(s, a, s') = P(S_{t+1} = s' \mid S_t = s, A_t = a)$ and $R : S \times A \times S \to \mathbb{R}$ a function signaling the reward received when taking the action $a \in A$ at state $s \in S$ leading to the next state $s' \in S$.

Learning how to behave or a *policy* is the goal of MDP's. Deep reinforcement learning represents such policies as a parametric model, more specifically a neural network $\pi_\theta : S \times A \to [0, 1]$, which outputs the probability of choosing action a at state s. Here θ is the parameters, which can be learned by maximizing the expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)]$$

where τ is a trajectory $(s_0, a_0, r_0, \cdots, s_n, a_n, r_n, s_{n+1})$ and $R(\tau) = \sum_{i=0}^{n} \gamma^i r_i$ is the total discounted reward of an experienced trajectory with discount factor $\gamma \in (0, 1]$. Methods using this formulation are know as policy gradient optimization.

The MDP abstraction fits very well with reinforcement learning as it covers a broad spectrum of problems. States and actions can be represented in various forms. The time steps t are not required to be equally spaced, and the intrinsic dynamics T and R are not needed to be known for learning, but trajectories or realizations of the process. Nevertheless, it is worth noting that the agent and environment design choices directly impact these dynamics, influencing the time complexity of the learning process and policy final's quality.

4 Environment and Agent Modeling

4.1 Environment

The environment is a Unity scene where the character can act and interact with objects. It consists of a square gray plane 110 × 110m with a white border of 1m diameter, which visually delimits the area of interest.Figure 3 shows the top view of the environment in the Unity editor.

The collectible in the agent's view is a pink box of $1m$ side. Notice it is the collectible agent view, but a player in the scene can view any other complex geometry or rendering. It allows one to consistently reduce the scene complexity for the agent, which reduces the computational complexity of observations by reducing the visual observation's size, an inherent advantage of virtual environments.

Environment Observability : an essential conceptual distinction of environments is whether they are completely or partially observable. Although we describe as an environment property, observability is directly associated with the agent's perception of its world. This differentiation of how much information the agent collects and how it perceives its surroundings is a significant learning performance component. Partially Observable Markov Decision Processes (POMPD)

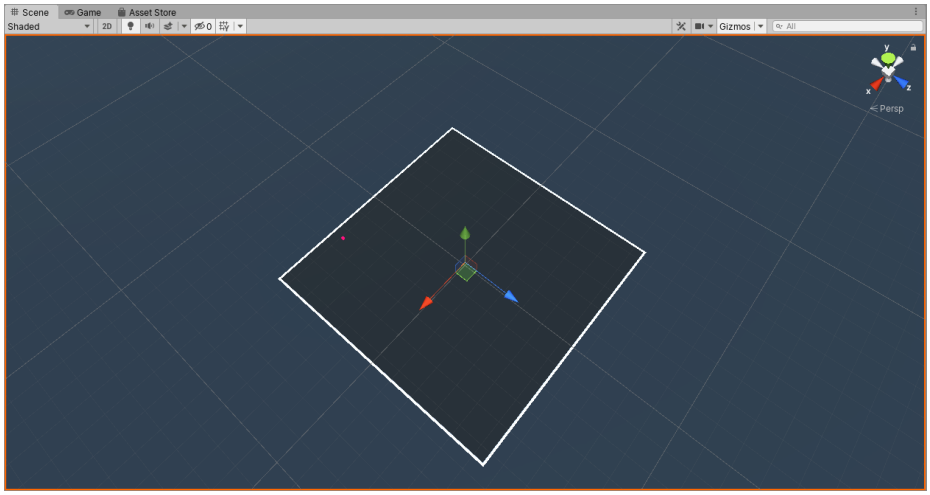

Fig. 3. Top View from the environment inside the Unity Editor.

are usually harder to solve, so it must be accounted for when developing the agent's observations.

4.2 Agent

The agent is an entity abstraction, which is itself a *behavior policy*. Nevertheless, we can imagine it as an entity with sensors gathering observations and actuators interacting where the policy link observations to actions. Our agent comprises all hierarchy levels, but the policy or controller learned is solely the task planning module. Notable features in an agent are: what it observes, how both the observations and actions are encoded, and how the associated rewards are assigned.

Agent Observation : an observation is any sensing made from the agent itself or the environment state encoded by numbers that serve as input to the behavior policy. Here, we employ two types of observations:

– Vector observation: composed of hand-crafted features:
  - Normalized direction to target: $d_{\text{target}} = (x, y, z)$, $\|d_{\text{target}}\|_2 = 1$
  - Normalized distance to border: $d_{\text{border}} = (x, y)$, $\|d_{\text{border}}\|_\infty < 1 \rightarrow$ inside, $\|d_{\text{border}}\|_\infty \geq 1 \rightarrow$ outside
  - Linear velocity: $v_{\text{linear}} = (x, y, z)\text{m/s}$
  - Angular velocity: $v_{\text{angular}} = (x, y, z)\text{rad/s}$
  - Normalized agent forward direction: $d_{\text{forward}} = (x, y, z)$, $\|d_{\text{forward}}\|_2 = 1$
  - Normalized agent up direction: $d_{\text{up}} = (x, y, z)$, $\|d_{\text{up}}\|_2 = 1$
  - Agent local position (agent's position referent to the center of the environment): $p_{\text{local}} = (x, y, z)$

– Visual observation: 3rd-person-like camera aligned with agent forward direction down-sampled from the original rendered agent's view (Figure 4):
  • 2D image: matrix $I_{84\times84\times3}(r, g, b)$

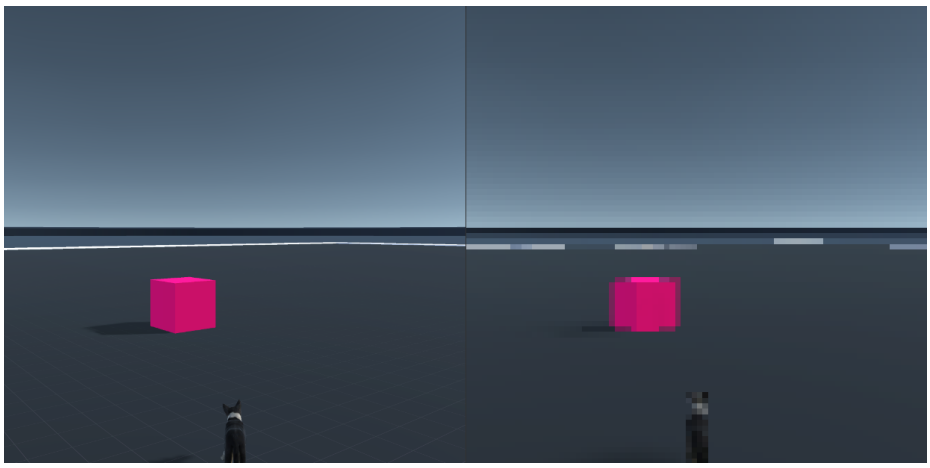

Fig. 4. Example of 3rd-person camera used for the agent. (Left)Agent's third-person camera. (Right)Downsampled visual observation.

Observation Constraint : we differentiate observations into two categories: Self-contained - Depends only on the agent state and sensors; Environment-dynamics dependent - Depends on the environment underlying mechanics. From those criteria, the 2D image is a self-contained agent sense. Concurrently, the presented vector observation needs access to the underlying environment dynamics to be fed to the agent (i.e., target position). The latter can limit the agent's usage in other unknown environments; however, the 3rd-person 2D image observations make the environment only partially observable.

Agent Actions : the DogBot's actions are defined by the motion synthesis module which is a character composed of various animations and a controller (state-machine and blend-tree) which receives four parameters controlling the X-Axis velocity, the Y-Axis rotation speed, and booleans jump/crouch. Their encodings for continuous and discrete action space are:

– Continuous action space:
  • Forward and backward movement: $\in [-1, 1]$
  • Steering left and right: $\in [-1, 1]$
  • Jump: $j \in [-1, 1]$, *true* if $j > j_0$
  • Crouch: $c \in [-1, 1]$, *true* if $c > c_0$

where c_0 and j_0 are threshold parameters intrinsic to the agent controller.

- Discrete action space:
  - Forward and backward movement: {backward, none, walk, trot, run}
  - Steering: {left, none, right}
  - Jump: {true, false}
  - Crouch: {true, false}

Agent's task : is a Fetch game - reach the collectible and bring it back. Its activation is trigger-based (i.e., throwing a stick) controlled by the task selection state-machine.

4.3 Reward

The entire universe of reinforcement learning bases itself on encouraging the best behavior through rewards (much like teaching a trick to a pet); in other words, rewarding the right actions accordingly. Developing a good reward signal is the key to learn an acceptable policy. Yet, most of the time, it is not easy to qualify a given action and state pair individually, but only the outcome of a sequence of actions and states. In theory, even for the cases where only the final state is rewarded, in the limit after many (infinite) experiences, it would be possible to learn an optimal behavior policy. AS computational power and time are finite resources, engineering good reward systems are crucial. Here, our agent experiments with a sparse reward - $+1.0$ is given when the agent reach its goal or final state, and per action reward - $r = +0.01(v_{\text{linear}} \cdot d_{\text{target}})$ which is positive when the agent moves toward its goal. For both cases, leaving the training area leads to a negative reward of -1.0 ending the episode. Also, a small negative reward -0.0005 is given at each time step t_i. It is close to $-\frac{1}{\#\text{ steps}}$, so the accumulated penalty will not saturate the total reward signal.

Note that the per action reward needs a broader knowledge of the environment, depending on both the agent and the environment's underlying state. Conversely, the sparse is assigned using only the agent's sensing but brings in the credit assignment problem. This notion is appealing because it resembles the agent's self-contained observation property, directly impacting the learning performance. In this case, even for the environment-dynamics dependent reward, it does not prevent the agent from being used in a new unknown environment after the learning process finishes, but for complex tasks may not be computable at all.

Next, we investigate the effects of the modeling choices and other variables, which are also part of the reward system, over the learned policies.

5 Experiments

For our experiments, we train various controllers from the combinations of environment and agent variations. For comparison purposes, we include another environment developed by Unity3D "Puppo, the Corgi" Unity3D (2018), which jointly learns the task planning and the low-level motion synthesis controller based on a physically simulated environment using joint torque and force.

5.1 Training

The tools used for training were Unity Editor 2019.3 and its machine learning framework ML-Agents (Juliani et al, 2018) in its version v0.13.1 together with Tensorflow 1.14.0. All experiments use the Proximal Policy Optimization (Schulman et al, 2017) algorithm and train for 10^7 steps. Experiments were run only once, given that they are computationally demanding. While the results are expected to be similar between different runs, but some variability should be taken into account when analyzing the results.

The training of the "Unity Puppo, the Corgi" uses their original configuration, besides the total number of steps. The environment is the same provided by (Unity3D, 2018), ported to the newer version of ML-Agents(v0.13.1) used here. The only modification to the original environment was its training area size to match DogBot's area size.

Specific details of all parameters of our low-level controller, hyper-parameters, and network layout used for training are on appendix A and B.

5.2 Results

The table 1 and 2 contains the result of various trained controllers evaluated over 200 episodes (10^6 steps) on the test scene containing n collectibles which randomly re-spawn when collected. The evaluation metric is the *Score*, (the number of objects collected overall episodes) and *Reset*, (the number of times the agent was reset due to leaving the training area).

Table 1. Results for the various models on the standardized test scene. In order, the columns are: Experiment number(Exp), observation type(Obs. Type), action type(Act. Type), number of collectibles in the training environment(Train Env.), Reward Type(Reward Type), number of collectibles in the test environment (Test Env.), Score(Score) and Reset(Reset).

Exp	Obs. Type	Act. Type	Train Env.	Reward Type	Test Env.	Score	Reset
1	Vector	Discrete	1, box	Per action	1, box	1027	61
2	Vector	Continuous	1, box	Per action	1, box	2324	82
3	Vector	Discrete	1, box	Sparse	1, box	4	670
4	Vector	Continuous	1, box	Sparse	1, box	0	0
5	Visual	Discrete	1, box	Per action	1, box	1370	7
6	Visual	Continuous	1, box	Per action	1, box	2883	0
7	Visual	Continuous	24, boxes	Sparse	1, boxes	12	0
8	Visual	Continuous	24, boxes	Sparse	24, boxes	7119	3
9	Visual	Continuous	1, box	Per action	24, boxes	6163	0

Figure 5 presents the comparison of Puppo and DogBot training convergence. This result is not directly comparable with the following figure 6 because, for comparison purposes, we use the same modeling from Puppo, which differs from

Table 2. Results for the down-scaled environment on the standardized test scene. In order, the columns are: Experiment number(Exp), observation type(Obs. Type), action type(Act. Type), number of collectibles in the training environment(Train Env.), Reward Type(Reward Type), number of collectibles in the test environment (Test Env.), Score(Score) and Reset(Reset).

Exp	Obs. Type	Act. Type	Train Env.	Reward Type	Test Env.	Score	Reset
10	Visual	Continuous	1, box	Per action	1, box	4606	599
11	Visual	Continuous	24, boxes	Sparse	1, box	1502	241

ours in various ways. As expected, our approach to learning the task planning alone demands fewer steps. Our agent is also faster at completing the task; still, we cannot directly infer how efficient their policies are, given the differences in their low-level controller (i.e., maximum speed and minimum turning radius, etc.).

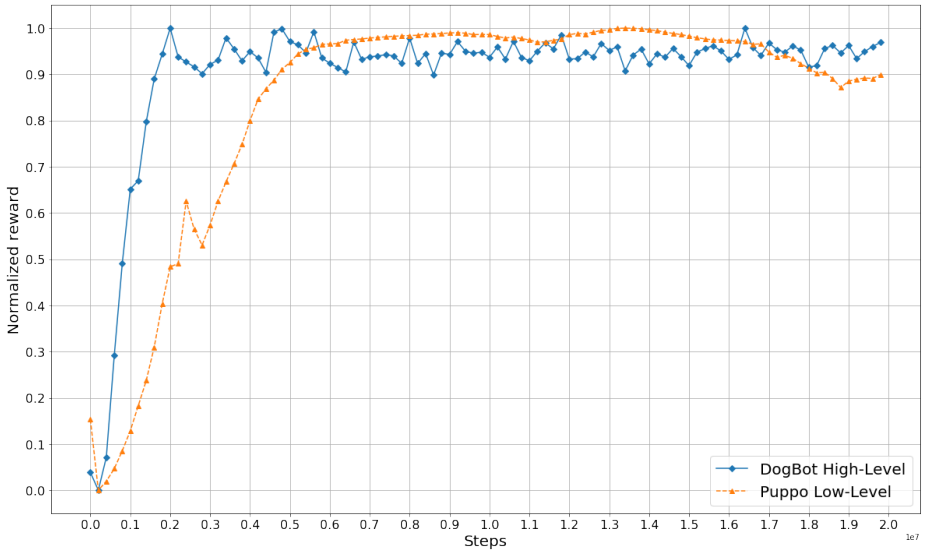

Fig. 5. Comparison of the training convergence of the "Puppo, The Corgi" which uses a low-level control approach (joint torque and force) versus the DogBot higher-level abstraction (left, right, etc.).

Figure 6 shows the progression of training from various configurations. The total reward is normalized to highlight the convergence of each controller. Interestingly, the controllers trained with sparse rewards vary from not learning anything (with a single collectible) to the faster showing progress in the learning (with multiple collectibles).

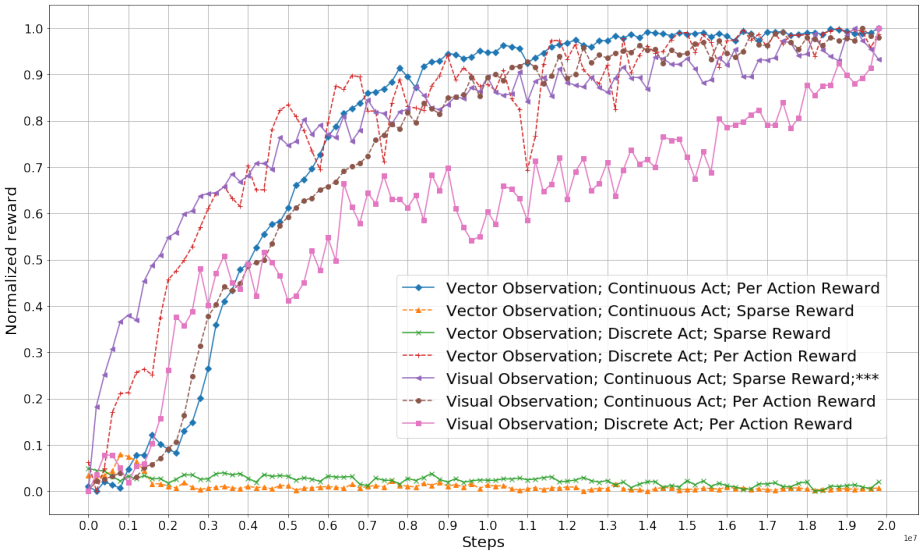

Fig. 6. Normalized reward progression during the training process. The naming scheme is the following: "*observation type*"; "*action type*";"*reward type*". The entry marked with ***** stands for the environment with multiple collectibles, while all others were using a single collectible.

5.3 Discussion

This paper evaluated various mid-level controllers for task planning, which is a crucial role for intelligent agents. While the fetching task is simple, it is a good test-bed for accessing the impact of modeling choices in the agent's final performance. The insight built from a simple case helps build more complex behaviors as the learning environment is de facto, where the developer has more control and understanding of its choices.

Our modeling coverage still far from extensive; purposefully, we left out network layout and hyper-parameters selection. While we agree these play a fundamental role in learning systems, they are much more experimental and sometimes hard to explain. In contrast, the concepts treated in our analysis for the fetch scene generalize more comfortably for other applications. In the following paragraphs, we individually analyze various aspects of modeling choices and their impact on the experiments.

Motion synthesis and task planning learning : jointly training for low and mid-level control gives the opportunity of end-to-end learning (such as the Puppo), but in the present case, it converges slower than learning the task planning alone. Another downside is that it requires more effort for developing both the character and the learning agent, while the final animation result is not predictable for aspects such as naturalness.

Action branches and time scale : one challenge when training with various action branches at the same time is the "curse of dimensionality" and the risk of not learning a proper policy. In our case, when starting from a random policy, the agent would be stuck alternating between going forward, backward, left, and right. A simple solution was to introduce a bias for the forward movement, but a more promising solution for this problem is bootstrapping the learning from examples. This approach showed to be effective in various works (i.e., Vinyals et al (2019); Ling et al (2020); Merel et al (2020)) and also has specific techniques for reinforcement learning, for example, Generative Adversarial Imitation Learning (Ho and Ermon, 2016). The time scale in which the agent can make decisions directly influences the learning; high frequency makes the agent more responsive but is harder for learning, starting from a random policy as the agent may not commit to a decision long enough to receive a reward.

Observation type : visual observations were as good or better than hand-crafted vector observations using the same reward system. Although they require more computational power, passing raw inputs applies to situations where hand-crafted observations are not viable. The only downside of raw inputs is the increased computational cost when training.

Action space : continuous action space achieved better results than the discrete action space, yet with slower convergence. We believe this result is directly related to the more precise control allowed by continuous actions, which requires more exploration explaining the slower convergence.

Reward : using a per action reward system produced good outcomes in all cases, while sparse reward only worked in the scene with multiple objects. It is apparent that despite the agent's drawback of possibly exploiting the reward instead of learning the real objective, a per action approach is more reliable. Another crucial point is that the per action reward developed an acceptable exploration policy when the collectible was not visible. In these cases, the agent ran to the opposite side of the training area, while the agent trained with sparse reward, usually was stuck running in circles until it was close enough to notice the collectible. Having access to the underlying environment can help develop the reward system, but caution and extra work may be needed to achieve an appropriate per action reward. On the other hand, the sparse reward made the learning process entirely dependent on the environment's difficulty. A convenient approach in these cases could be using a curriculum (Bengio et al, 2009), but requiring more human resources in the development process.

Multiplicity of collectibles and area size : having many objects in the scene completely changed the result of using sparse rewards, from unable to learn anything to achieve the best result in the test environment with multiple collectibles. This example shows how the environment modeling is crucial to learning; simple changes can make the initial random policy find rewards much often and learn faster. Interestingly, the agent trained with multiple collectibles underperformed

with a single collectible test. From visual inspection, it did not develop an acceptable exploration policy and could not collect faraway objects. In contrast, in a reduced environment, the same agent performed better. Notwithstanding, while both agents could complete the task with a reduced area, they were not as effective as their original training area and would often leave the marked area. Indeed, training with such variation would lead to a better generalizing agent.

Conclusion & Future Work

We presented a hierarchical approach to developing intelligent virtual agents focusing on applying deep reinforcement learning to the intermediate time-scale level: task planning. Through a simple study case of a fetch behavior, we learned various controllers using different modelings and could build insight from each specific choice. This knowledge helps to make the base for future (more complex) works, from which we comment in the next paragraph.

A not extensive list of future directions is: Integrating more behaviors into our agent's repertoire is unquestionably the next step; an exciting possibility for a dog agent are tricks such as hoop. Joining together these behaviors can lead to a fascinating virtual character for interactive narratives. Next, treating collisions and navigation in complex scenarios is a must for its applicability. Finally, more far fetched goals can be using more complex motion synthesis modules, integrating another sensing (i.e., audio) and memory to the agent. The memory of past events and actions is fundamental to specific long tasks; for example, a maze runner would be easier solvable if the agent keeps track of the visited spaces.

Bibliography

Agrawal S, van de Panne M (2016) Task-based locomotion. ACM Transactions on Graphics (TOG) 35(4):1–11

Bengio Y, LeCun Y, et al (2007) Scaling learning algorithms towards ai. Large-scale kernel machines 34(5):1–41

Bengio Y, Louradour J, Collobert R, Weston J (2009) Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning, pp 41–48

Geijtenbeek T, Van De Panne M, Van Der Stappen AF (2013) Flexible muscle-based locomotion for bipedal creatures. ACM Transactions on Graphics (TOG) 32(6):1–11

Ho J, Ermon S (2016) Generative adversarial imitation learning. In: Advances in neural information processing systems, pp 4565–4573

Holden D, Saito J, Komura T (2016) A deep learning framework for character motion synthesis and editing. ACM Transactions on Graphics (TOG) 35(4):1–11

Holden D, Komura T, Saito J (2017) Phase-functioned neural networks for character control. ACM Transactions on Graphics (TOG) 36(4):1–13

Hong S, Han D, Cho K, Shin JS, Noh J (2019) Physics-based full-body soccer motion control for dribbling and shooting. ACM Transactions on Graphics (TOG) 38(4):1–12

Jiang Y, Van Wouwe T, De Groote F, Liu CK (2019) Synthesis of biologically realistic human motion using joint torque actuation. ACM Transactions on Graphics (TOG) 38(4):1–12

Juliani A, Berges VP, Vckay E, Gao Y, Henry H, Mattar M, Lange D (2018) Unity: A general platform for intelligent agents. arXiv preprint arXiv:180902627

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. nature 521(7553):436–444

Lee K, Lee S, Lee J (2018a) Interactive character animation by learning multi-objective control. ACM Transactions on Graphics (TOG) 37(6):1–10

Lee S, Yu R, Park J, Aanjaneya M, Sifakis E, Lee J (2018b) Dexterous manipulation and control with volumetric muscles. ACM Transactions on Graphics (TOG) 37(4):1–13

Lee S, Park M, Lee K, Lee J (2019) Scalable muscle-actuated human simulation and control. ACM Transactions on Graphics (TOG) 38(4):1–13

Lee Y, Park MS, Kwon T, Lee J (2014) Locomotion control for many-muscle humanoids. ACM Transactions on Graphics (TOG) 33(6):1–11

Levine S, Lee Y, Koltun V, Popović Z (2011) Space-time planning with parameterized locomotion controllers. ACM Transactions on Graphics (TOG) 30(3):1–11

Levine S, Finn C, Darrell T, Abbeel P (2016) End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research 17(1):1334–1373

Lin LJ (1992) Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning 8(3-4):293–321

Ling HY, Zinno F, Cheng G, Van De Panne M (2020) Character controllers using motion vaes. ACM Transactions on Graphics (TOG) 39(4):40–1

Liu L, Hodgins J (2017) Learning to schedule control fragments for physics-based characters using deep q-learning. ACM Transactions on Graphics (TOG) 36(3):1–14

Liu L, Hodgins J (2018) Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. ACM Transactions on Graphics (TOG) 37(4):1–14

Liu L, Panne MVD, Yin K (2016) Guided learning of control graphs for physics-based characters. ACM Transactions on Graphics (TOG) 35(3):1–14

Luo YS, Soeseno JH, Chen TPC, Chen WC (2020) Carl: Controllable agent with reinforcement learning for quadruped locomotion. arXiv preprint arXiv:200503288

Mac TT, Copot C, Tran DT, De Keyser R (2016) Heuristic approaches in robot path planning: A survey. Robotics and Autonomous Systems 86:13–28

Merel J, Ahuja A, Pham V, Tunyasuvunakool S, Liu S, Tirumala D, Heess N, Wayne G (2018) Hierarchical visuomotor control of humanoids. arXiv preprint arXiv:181109656

Merel J, Tunyasuvunakool S, Ahuja A, Tassa Y, Hasenclever L, Pham V, Erez T, Wayne G, Heess N (2020) Catch & carry: reusable neural controllers for vision-guided whole-body tasks. ACM Transactions on Graphics (TOG) 39(4):39–1

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al (2015) Human-level control through deep reinforcement learning. nature 518(7540):529–533

Naderi K, Rajamäki J, Hämäläinen P (2017) Discovering and synthesizing humanoid climbing movements. ACM Transactions on Graphics (TOG) 36(4):1–11

Nakada M, Zhou T, Chen H, Weiss T, Terzopoulos D (2018) Deep learning of biomimetic sensorimotor control for biomechanical human animation. ACM Transactions on Graphics (TOG) 37(4):1–15

Peng XB, Berseth G, Van de Panne M (2016) Terrain-adaptive locomotion skills using deep reinforcement learning. ACM Transactions on Graphics (TOG) 35(4):1–12

Peng XB, Berseth G, Yin K, Van De Panne M (2017) Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Transactions on Graphics (TOG) 36(4):1–13

Peng XB, Abbeel P, Levine S, van de Panne M (2018a) Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. ACM Transactions on Graphics (TOG) 37(4):1–14

Peng XB, Kanazawa A, Malik J, Abbeel P, Levine S (2018b) Sfv: Reinforcement learning of physical skills from videos. ACM Transactions on Graphics (TOG) 37(6):1–14

Ross S, Pineau J, Paquet S, Chaib-Draa B (2008) Online planning algorithms for pomdps. Journal of Artificial Intelligence Research 32:663–704

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:170706347

Starke S, Zhao Y, Komura T, Zaman K (2020) Local motion phases for learning multi-contact character movements. ACM Transactions on Graphics (TOG) 39(4):54–1

Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press

Terzopoulos D, Rabie T, Grzeszczuk R (1997) Perception and learning in artificial animals. In: Proc. of the 5th Int. Workshop on Artificial Life: Synthesis and Simulation of Living Systems (ALIFE-96), pp 346–353

Unity3D (2018) Puppo, the corgi: Cuteness overload with the unity ml-agents toolkit. URL https://blogs.unity3d.com/pt/2018/10/02/puppo-the-corgi-cuteness-overload-with-the-unity-ml-agents-toolkit/

Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, Choi DH, Powell R, Ewalds T, Georgiev P, et al (2019) Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature 575(7782):350–354

Xu J, Du T, Foshey M, Li B, Zhu B, Schulz A, Matusik W (2019) Learning to fly: computational controller design for hybrid uavs with reinforcement learning. ACM Transactions on Graphics (TOG) 38(4):1–12

Yu W, Turk G, Liu CK (2018) Learning symmetric and low-energy locomotion. ACM Transactions on Graphics (TOG) 37(4):1–12

Zhang H, Starke S, Komura T, Saito J (2018) Mode-adaptive neural networks for quadruped motion control. ACM Transactions on Graphics (TOG) 37(4):1–11

A Motion Synthesis Module

The motion synthesis module is a Unity character controller responsible for receiving the task steps (i.e., left, right, jump, etc.) and translating to animations; it is the agent's actuators. Its parameters control the agent's velocity, turn speed, gravity, and other effects of the character's physical properties. They are intrinsic to the agent's actuators as they are not visible to the task planning module. In the table 3, there is a detailed list of the values used for each parameter.

B Training Configuration

In the table 4, the training parameters are shown. The network architecture consists of the encoders and the decoders. The encoder for the vector observation consists of two dense layers fully connected with Swish activation function; for the visual observation the encoder is similar to Mnih et al (2015) with three convolutional layers of (# filters, kernel size, stride): (32, 8×8, 4), (64, 4×4, 2), (64, 3×3, 1) and final dense layer with 512 units and leaky ReLU activation function instead of ReLU for of its all layers. Figure 7 presents a diagram of the

Table 3. DogBot's character controller parameters.

Name	Value	Description
Moving turn speed	45.0 deg/s	Turn speed when not stationary
Stationary turn speed	30.0 deg/s	Turn speed when stationary
Jump power	5.0 m/s	Vertical velocity applied when jumping
Forward Velocity	9.0 m/s	Maximum forward velocity
Backward Velocity	2.0 m/s	Maximum backward velocity
Gravity multiplier	1.0	Multiplier for gravity simulation
Anim speed multiplier	1.0	Multiplier for animation time scale

visual encoder. The encoders' output is concatenated and fed to the decoders: the policy head and the value head. The first is responsible for choosing the agent actions, and its size varies according to its possibilities; the latter is liable for the state value estimate.

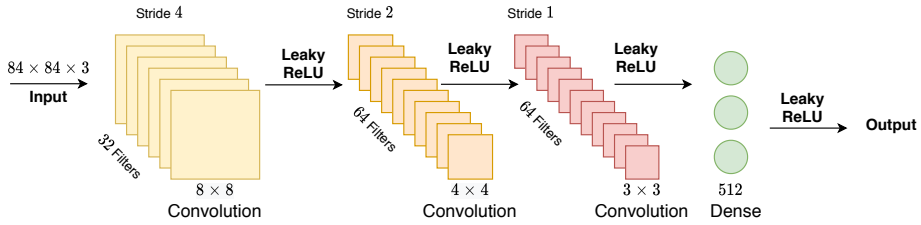

Fig. 7. Diagram of the network architecture used for the visual input encoder "nature_cnn".

Table 4. Training parameters.

Name	Value	Description
batch size (Continuous)	4096	Number of samples used for each optimization step for continuous action space.
batch size (Discrete)	256	Number of samples used for each optimization step for discrete action space.
buffer size	40960	Number of samples collected for each policy update.
hidden units	512	Number of neurons per hidden layer.
num layers	2	Number of hidden layers used for the model.
learning rate	3.0×10^{-4}	Initial learning rate for training.
max steps	2×10^{7}	Number of total simulation steps (actions) taken for training.
num epochs	5	Number of times each collected observation is used for training.
time horizon	1000	Horizon for learning, it represents how far in time steps one action can influence a past reward.
gamma	0.995	Discount factor, it represents how much of a n-future reward (R_n) is assigned to a present action in the form $\gamma^n R_n$.
Curiosity strength	0.1	Strength of the curiosity intrinsic reward signal.
Curiosity gamma	0.99	Discount factor for the curiosity reward.
Visual encoding type	nature_cnn	Type of architecture used for the convolutional layers for the visual observation encoding.
Visual input size	$84 \times 84 \times 3$	Size of the visual input image.
Max episode length	5000	Maximum number of steps until an episode ends.
Action repeat	2	Number of frames an action is repeated.
Decision frequency	30Hz	Frequency which the agent take decisions.