Effects of Reward Shaping on Curriculum Learning in Goal Conditioned Tasks

Mihai Anca¹, Matthew Studley², Mark Hansen², Jonathan D. Thomas¹ and Dabal Pedamonti¹

Abstract—Real-time control for robotics is a popular research area in the reinforcement learning (RL) community. Through the use of techniques such as reward shaping, researchers have managed to train online agents across a multitude of domains. Despite these advances, solving goal-oriented tasks still require complex architectural changes or heavy constraints to be placed on the problem. To address this issue, recent works have explored how curriculum learning can be used to separate a complex task into sequential sub-goals, hence enabling the learning of a problem that may otherwise be too difficult to learn from scratch. In this article, we present how curriculum learning, reward shaping, and a high number of efficiently parallelized environments can be coupled together to solve the problem of multiple cube stacking. Finally, we extend the best configuration identified on a higher complexity environment with differently shaped objects.

Index terms - Reinforcement Learning

I. INTRODUCTION

Long horizon manipulation tasks, such as object stacking, require planning over an extended period of time while taking into account various mandatory checkpoints. Creating agents that can solve this type of tasks using complex reward signals is a key challenge in deep reinforcement learning (deep RL). Recent advances have shown that RL applied to robotic arms can perform object stacking [1]–[4], however, the problem is usually constrained or the solution is architecturally complex.

Moreover, standard RL approaches do not scale well when the number of task objects is increased. We believe that a curriculum learning (CL) framework can help alleviate this issue by decomposing the task into sub-goals of incremental difficulty. This has been shown [5] to be effective in applications such as robotics and games.

We leverage the new simulation platform Isaac Gym [6], which allows thousands of environments to be asynchronously controlled by an RL agent. We found that there is a gap in the on-policy RL literature when dealing with heavily parallelized goal-oriented tasks, such as stacking. Our work, to the best of our knowledge, provides the first results in this territory.

In summary, the paper makes the following contributions:

1) We present various CL setups and couple them with reward shaping, enabling the stacking of three objects using standard proximal policy optimization (PPO) [7]. To the best of our knowledge no prior work successfully achieved this task using on-policy vanilla RL algorithms (i.e. PPO).

2) We analyse the elements that hinder learning and discuss the key components that are critical for task completion.

3) We show that in using an asynchronous setup such as Isaac Gym [6] to learn goal-oriented tasks, the learning horizon becomes an important hyperparameter, and we consider the effects of a curriculum for adjusting this value.

4) Finally, we provide the codebase (available post review) to the community to act as a baseline for future work in multiple cube stacking tasks.

II. BACKGROUND AND RELATED WORK

The “Pick and Place” task was popularised as part of a reinforcement learning (RL) baseline with the release of OpenAI robotics gym pack [8]. Both online and offline RL methods have been shown to work on the dense reward version of this environment. For example, in [1], the authors used a model-based vision approach and self-supervision to pick and place one cube on top of another.

However, the literature to date offers no simple solution to the problem of stacking at least three objects, which represents a key example of a long horizon task. In [2], the authors discretised the action space and managed to learn a RL policy that successfully stacks multiple objects. This method relies on prior knowledge and is only applicable to simple-to-grasp objects such as cubes. The recent work of [3] shows a potential solution when stacking differently shaped objects; however, the problem solved is heavily constrained and requires multiple training sessions.

This problem becomes increasingly difficult when posed in a sparse reward environment, although, in [4], the authors managed to use graph neural networks (GNN) coupled with attention layers to overcome this and stack up to six cubes in simulation. Even though the number of trainable parameters is similar to a fully connected network, the new architecture requires considerably more VRAM when computing the message parsing step of the GNN. Additionally, it fails to explain the underlying issue of why vanilla RL approaches are incapable of learning this task. We understand the importance of the results showed in [4], however, a practical comparison is outside the scope of this paper and we leave this test for future work.

Another popular example of stacking multiple cubes using sparse rewards is that presented in [5]. The authors have
made clever use of the critic network in an actor-critic architecture to label the importance of demonstrations. Coupled together with HER [10], the trained agent learns how to stack up to six cubes being provided with only 100 demonstrations.

The question is, therefore, whether RL is capable of solving this stacking problem with the use of complex reward shaping and CL, as opposed to highly constrained and/or architecturally intricate scenarios or with the help of demonstrations. Examples of a complex problem solved using reward shaping and RL include [11] and [12]. The advantage that this architectural simplicity brings is a reduction of pipeline complexity for reproducibility, which can aid with safety constraints that can be added to the system later on.

A. Reinforcement Learning

Throughout the paper, we will be considering the standard RL formalism with a fully observable environment. An environment consists of a set of states $S$, a set of actions $A$, a distribution of initial states $p(s_0)$, a reward function $r : S \times A \to \mathbb{R}$, transition probabilities $p(s_{t+1} | s_t, a_t)$, and a discount factor $\gamma \in [0, 1]$.

A policy $\pi$ is used as a mapping between a state and a distribution over actions. Every episode starts by sampling an initial state $s_0$. At every timestep $t$ the agent produces an action based on the current state: $a_t \sim \pi(\cdot|s_t)$. Each action is then rewarded using the function $r_t = r(s_t, a_t)$ and a new state $s_{t+1}$ is sampled from the distribution $p(\cdot|s_t, a_t)$. The discounted sum of future rewards, also referred to as the return, is defined as $R_t = \sum_{i=t}^{T} \gamma^{i-t}r_i$, where $T$ is the horizon that the agent optimises over. The RL objective is to maximize the expected return $E_{s_0}[R_0 | s_0]$, where the expectation is taken over the initial state distribution, policy, and environment transitions accordingly to the dynamics specified above.

The problem studied in this paper falls under the umbrella of goal-conditioned RL [13]. This means that the state is augmented with a goal, $g$. Those goals are individually sampled from a separate distribution $p(g)$ at the beginning of each episode. The agent’s goal therefore becomes: to maximize the expected augmented return $E_{s_0,g}[R_0 | s_0, g]$.

The action-value function $Q(s_t, a_t) = E[R_t | s_t, a_t]$ and state-value function $V(s_t) = E[R_t | s_t]$ are used to derive the optimal policy $\pi$. The advantage $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$ represents whether the action $a_t$ is better than the average action the policy can take at the current state $s_t$.

In order to find this policy $\pi$, we utilise proximal policy optimization (PPO) [7]. This algorithm optimizes a stochastic policy while also approximating a value function. At the end of each episode, the advantage is calculated using the generalized advantage estimator (GAE) [14], and an update is computed towards minimising the following loss:

$$L(\theta) = E \left[ \min(r_t(\theta)A^{GAE}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A^{GAE}) \right]$$

Where $r_t(\theta)$ is the probability ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and $\epsilon$ is a hyperparameter.

B. Curriculum Learning

CL is a training procedure that increases performance and/or training speed on a high complexity task by decomposing it into a sequence of lower difficulty sub-goals. Due to an increased sample efficiency, the agent amasses knowledge faster and is able to generalize towards the complex task, reducing the exploration required if it were to train uninitialized.

The first application of CL in robotics dates back to 1994 [15], where a parameter was used to change the difficulty of the environment in which the agent operates. Other influential work includes [16], where increasingly difficult sub-problems were identified and trained on. More recently, [17] have demonstrated how CL can be combined with intrinsic motivation to overcome certain robotics tasks.

For a more comprehensive overview of curriculum learning please refer to [5].

C. Reward Shaping

Goal-oriented tasks are naturally described using sparse reward functions where the agent receives a positive reward only when achieving the task, and zero otherwise. Unfortunately, this makes learning of such tasks difficult due to the amount of guided exploration required to reach the goal even once. Most commonly [18], the process of reward shaping is used to overcome this issue. Reward shaping refers to modifying the original sparse reward function by incorporating domain knowledge in the form of additional reward.

Regrettably, there are downsides to using a shaped reward, as observed by [19]. Also explained in [20], without sufficient care, the changes brought to the reward function can mislead the agent towards an unexpected optimal policy. We show an example of this in our work in Section IV.

III. METHODS

A. Environment

The environment used (Figure 1) in our experiments consists of a Franka Panda (7 DOF) robotic arm, a table and 3 boxes. The goal is to stack the objects on top of each other. The goal location and initial objects positions are sampled randomly every episode within the confines of the table. The initial joint positions are obtained by setting them to a hard-coded home position to which a small perturbation (8%) is added: $s_0 = s_{\text{home}} + 0.08 * L * U[-1, 1]$, where $L$ represents the joint ranges. A total of 9 actions are used, which include joint positions and gripper control. Information about the observation space can be found in the Appendix.

B. Implementation details

We implemented the environment described above in Isaac Gym [6], an end-to-end high performance robotics simulation platform. Throughout our experiments we used the RL Games [21] (a highly-optimized GPU RL framework) implementation of Proximal Policy Optimization algorithm [7]. Our main experiments were repeated over 3 seeds. Table IV contains more details regarding the architecture used.
Fig. 1. Our setup using a Franka Panda arm with 3 cubes in simulation using Isaac Gym [6]. The first image shows an example of reward hacking where the agent is using a single cube to collect all three sub-goal one-time bonus rewards. The second image shows an instance where the end-effector orientation is reversed such that the negative “touching table” reward is not triggered. The third image shows the robotic arm pushing aside the first correctly placed box, therefore postponing the end of the episode. The final image shows the higher complexity setup with the T, L and I objects.

C. Types of curriculum

In our experiments, we explored two curriculum setups which are described in this section. The switch between stages of a curriculum takes place automatically whenever the agent has reached an accuracy of over 90%, striking a balance between time and optimality.

The first curriculum modifies the number of active objects. At each stage, an extra object is added to the table and its observations are enabled.

The second curriculum setup consists of adjusting the episode and horizon window length, such that at each stage, these values are updated based on required time for solving the task with the active number of objects. This form of curriculum requires the first curriculum type to also be active.

IV. REWARD FUNCTION

Training agents in simulation provides faster iteration times and overcomes many safety concerns. We can demonstrate that through reward shaping we significantly speed up the training of our agents even further, but this comes with certain risks. In general, the goal is to eventually transfer the learned behaviour to a real-world robot, however, our agents sometimes exploit the reward functions in unintended ways. This is known as reward hacking [19].

A. Method

Similar to [11] and [12], the objective of the environment is encoded using a shaped reward function: \( r_t = \sum_{i \in S} \lambda_i R_i \).

The original function is rewarding the agent when all three cubes are stacked. Additionally, we added different signals to steer the behaviour towards the desired performance. Initially, we ignored the original reward by only providing subgoal rewards, however, that led to various reward hacking behaviours that are discussed next. A dense reward (Eq. 1) signifies a value provided after each action taken, while a sparse signal (Eq. 2) indicates a conditioned bonus reward dependent on the state achieved post taking the previous action.

\[
\begin{align*}
 r_{\text{dense}}(s, \text{obj}) &= \sum_{i \in X, Y, Z} (x_i^s - x_i^{\text{obj}})^2 \\
 r_{\text{sparse}} &= 1 \text{ iff } s_g - \epsilon \leq s_t \leq s_g + \epsilon \\
 &\text{otherwise } r_{\text{sparse}} = 0
\end{align*}
\]

We introduced three additional signals for keeping the agent’s choice of action as safe and efficient as possible. These include punishing wasteful actions (Eq. 3), punishing touching the table (Eq. 4) and rewarding orientations that are close to the initial joint configuration (Eq. 5). The latter helps overcome the second example of reward hacking highlighted in Figure 1.

\[
\begin{align*}
 r_{\text{action}} &= \sum_{i} a_i^2 + (\theta_i - \theta_0)^2 \\
 r_{\text{table}} &= 1 \text{ iff } x_{ee} \leq \Theta \\
 &\text{otherwise } r_{\text{table}} = 0 \\
 r_{\text{orientation}} &= \text{quat mult}(\omega_D, \text{quat conjugate}(\omega_{\text{end effector}}))
\end{align*}
\]

, where \( \theta \) represents a joint position.

\[
\begin{align*}
 r_{\text{table}} &= 1 \text{ iff } x_{ee} \leq \Theta \\
 &\text{otherwise } r_{\text{table}} = 0
\end{align*}
\]

, where \( \Theta \) represents a height threshold and \( x_{ee} \) represents the position of the end-effector.

\[
\begin{align*}
 r_{\text{orientation}} &= \text{quat mult}(\omega_D, \text{quat conjugate}(\omega_{\text{end effector}}))
\end{align*}
\]

, where \( \text{quat conjugate} \) and \( \text{quat mult} \) represent the quaternion conjugate and multiplication functions, and \( \omega \) is the orientation represented in quaternions.

| Signal                  | \( \lambda \) | Type   |
|-------------------------|---------------|--------|
| Goal achieved           | 150           | \( r_{\text{sparse}} \) |
| Box on target           | 1             | \( r_{\text{sparse}} \) |
| One time bonus per sub-goal | 150         | \( r_{\text{sparse}} \) |
| Box to goal distance    | -5            | \( r_{\text{dense}} \) |
| End-effector to box distance | -5         | \( r_{\text{dense}} \) |
| Action magnitude penalty| -0.01         | \( r_{\text{action}} \) |
| Touching table          | -5            | \( r_{\text{table}} \) |
| Orientation error       | -0.1          | \( r_{\text{orientation}} \) |

Our methods are compared against an absolute reward baseline, where the dense reward is calculated as shown in Eq 6 where \( \Theta \) represents the set of all objects in the environment.
The agent learns that it can collect all bonus rewards using a single cube and moving to all positions in the target stack. The final such example we came across was when training the agent when the end-effector touches the table. However, the agent did find a reward hack by holding the end-effector upside down when moving the box (Figure 1). To overcome this, an extra signal was added that punishes the agent for modifying the orientation of the end-effector too much.

Another example (Figure 1) of reward hacking we came across was when the agent was moving the first correctly placed box aside using the second box, such as not to end the episode early. More details about this and the solution are discussed in Section V.

The safety constraint that we integrated as part of the reward was the “touching table” signal. This punishes the agent when the end-effector touches the table. However, the order of boxes in the stack does not matter and the agent gets a one-time bonus for each sub-goal achieved.

Lastly, we used the staggered reward (Eq 1) without any curriculum (i.e.: all objects must be stacked) as a secondary baseline, similar to the one described in [11]. In this case, the reward is calculated with respect to one box at a time. In other words, the goal is conditioned on the success of the agent.

**B. Results**

One safety constraint that we integrated as part of the reward was the “touching table” signal. This punishes the agent when the end-effector touches the table. However, the agent did find a reward hack by holding the end-effector upside down when moving the box (Figure 1). To overcome this, an extra signal was added that punishes the agent for modifying the orientation of the end-effector too much.

Another example (Figure 1) of reward hacking we came across was when the agent was moving the first correctly placed box aside using the second box, such as not to end the episode early. More details about this and the solution are discussed in Section V.

The final example we came across was when training the agent when the “Any box on target” reward. In this scenario, the order of boxes in the stack does not matter and the agent gets a one-time bonus for each sub-goal achieved. The agent learnt that it can collect all bonus rewards using a single cube and moving to all positions in the target stack one at a time (Figure 1).

Some of the reward hacking scenarios discussed were found while adjusting the $\lambda$ parameter for “Goal achieved/One time bonus per sub-goal” and “Box to goal distance/End-effector to box distance”. These results can be observed in Figure 2.

**V. INPUT REMAPPING**

Our initial reward function was based on an established order for the boxes, similar to [4]. For example, in each episode, the same red box would have to be placed at the bottom of the stack. During the first stage of the curriculum, when only one box is enabled, the state of the other boxes are replaced with 0s in the agent’s observation. The agent learns the sequence of actions that leads to the goal only with respect to the first box. When the second object is activated through the second stage of the curriculum, the learned policy fails to even move the first object to the bottom of the stack anymore. This is because the problem has shifted into the domain of continual learning [22].

**A. Related Methods**

This problem has been studied on a different set of environments and confirmed in [23]. Similarly, in [4], the authors achieved high accuracy of stacking up to 6 cubes by using a curriculum and an architecture of GNNs with attention. GNNs provides input permutation invariance, therefore overcoming the problem of input remapping. Our work differs through architectural simplicity and the insights discovered that enabled the learning to take place.

Alternative solutions include the use of elastic weights consolidation [24]. This method identifies the weights that are essential to previously learned tasks and reduces further changes, such that the behaviour is preserved. Finally, the remapping issue observed can also be overcome by using an auto-encoder [25], which translates the states into a latent representation that is invariant to the permutation of inputs. Both of these solutions would require architectural changes so they are left for future study.

We believe input remapping is one of the reasons why vanilla RL methods were not capable of solving this problem of stacking multiple objects before. Without addressing this issue first, the curriculum of sub-tasks brings almost no benefits to the training since the trajectory of rewarding actions has to be learned from scratch each time a new object is introduced.

**B. Results**

We identified a solution that requires no architectural changes, however, all objects must be present from the start. The issues discussed above have to do with remapping of inputs. This is overcome by crafting the reward function in a way that teaches the agent to interact with all the boxes even in the first stage of curriculum. To achieve this, every time the episode is reset, a permutation of boxes is chosen randomly, such that a different stacking order is required for achieving the goal. Subsequently, the 0 padding mentioned earlier is no longer required. As a result, all input weights associated with objects are trained equally. See Figure 3 and Table II for a comparison of our methods. We use episode length as a metric as it easily allows us to compare performance of our algorithms. A lower number of steps represents an agent that can solve the task more quickly. The curriculum run that achieved the best results was trained for approximately 21.5 billion steps, which is equivalent to 20 hours on our workstation (see Appendix for hardware details).

**VI. HORIZON LENGTH**

PPO [7] is an on-policy online RL method, meaning that updates take place after each episode using the gathered experiences. The algorithm was designed with parallelism in mind, where each agent collects the same amount of
environments. The baseline (Absolute Reward) fails to solve the first stage, while the other two take significantly longer to reach similar accuracy.

### A. Method

Isaac Gym enables us to efficiently parallelize the experience gathering across tens of thousands of environments on a single workstation. One downside is that the process is no longer synchronous. In fact, when dealing with single environments, the end of the episode is clearly marked, however, in Isaac Gym, the environments are asynchronously reset. Each environment finishes once the agent has reached the given goal or when a certain amount of timesteps have passed.

These resets are taking place because we are using an infinite-horizon setting with early stopping. We chose this setup because it significantly speeds up learning due to focusing the exploration around the starting position of the agent. If the time horizon is infinite, during the first stage of training, the agent spends a lot of time in non-rewarding states, such as outside of table area, or it even gets stuck in un-recoverable states.

In each parallel agent has access to its own simulator where experiences are gathered asynchronously from the learner and resets can happen in sync with data acquisition. Isaac Gym reaches a high number of parallel environments by creating them all using the same simulator. As a consequence, the training takes place after the number of steps dictated by the horizon length have been gathered across all environments.

### B. Results

Using a curriculum of tasks means that the optimal value of horizon length $T^*$ shifts with each change of stage. Our first experiment looks at updating horizon $T$ after each curriculum transition. In other words, we create a separate curriculum, such that the following equation is satisfied for each sub-task: $T ≈ T^*$ and $T ≥ T^*$.

### Table II

| Name                | Accuracy | Final Ep. Length | Steps | Wall Time |
|---------------------|----------|------------------|-------|-----------|
| Curriculum          | 90.6%    | 78.94            | 24.5B | 20h       |
| Staggered Reward    | 82.2%    | 92.34            | 32.5B | 31h       |
| Absolute Reward     | 0%       | 150              | 49B   | 72h       |
| Horizon Curriculum  | 87.6%    | 74.07            | 44.6B | 45h       |

Fig. 3. Results of various curriculum methods against the baselines. The only method reaching 90% in second stage and triggering the automatic curriculum update at the end of the run is the (blue) Curriculum method. The baseline (Absolute Reward) fails to solve the first stage, while the other two take significantly longer to reach similar accuracy.

Fig. 4. The horizon window (blue) and episode length (dotted black) match during the first part of the training as the agent is incapable of solving the problem. As the agent gathers more experience, the average episode length diminishes, leading to the horizon window covering more than one episode at a time. This leads to various reward hacking scenarios.

Fig. 5. Left: 10x10 maze environment with discrete action space using sparse rewards provided when the agent reaches the goal (red square). Right: Training steps required to learn a policy capable of solving the maze environment over a range of values for horizon length. Providing more time in each episode seems to have diminishing returns for the overall training time.

If the episode and horizon lengths are equal and the agent cannot achieve the goal (i.e. at the beginning of training), the asynchronous experience gathering is equivalent with the synchronous method. However, when the agent starts achieving the goal with higher accuracy, the environment resets are out of sync, therefore the horizon window can span multiple episodes (see Figure 4).

We trained an agent in a toy environment (Figure 5) to determine the best approach for choosing the horizon length. The results show that there is an optimal value and any further increases lead to a less than optimal training time. On the other hand, if the value is too small, the agent cannot find a policy to solve the task. Our hypothesis is that if the horizon length is less than the required steps to solve the problem, the trajectories gathered do not encapsulate the solution, hence making the problem impossible to learn.

Ideally, we would want to select the optimal horizon length for all tasks. Unfortunately, this value is hard to determine without solving the task beforehand, especially for long-horizon tasks requiring multiple checkpoints to be reached, such as cube stacking. In order to reduce the time required per iteration, the results discussed next were derived from testing only on the first two stages of curriculum. This means that the problem consists of successfully moving and stacking two cubes.
In this setup, the agent finds a reward hack during the second phase of the curriculum. A policy is found that can generate the most reward in a window by placing the first box, but not ending the episode with the second box. Instead, it displaces the first box using the second, refusing to finish the episode early such that it circumvents the early negative reward of a fresh episode (Figure 4).

In order to incentivise the agent to finish the episode, we increased the bonus reward for achieving any sub-goal, including the overall environment goal. This, however, requires a balance (see results in Figure 2), since, if this number is set too high, the agent finds the quickest path towards claiming the intermediate reward by throwing the boxes towards the target location.

The solution discussed above takes considerably longer to train when compared to the baseline of the staggered reward (Figure 3). Our hypothesis is that an increase in episode and horizon length leads to a drastic change in expected reward for each state. The returns are approximated using the GAE method and the value of a state is estimated based on the amount of reward that can be gathered until the end of the episode starting from that state. Since the agent can now explore more states, due to the increase in maximum timesteps, the expected reward also increases in magnitude. The agent cannot progress on learning the requirements for the new sub-goal until the value function is updated based on the new horizon length. This additional time required is what we believe is slowing down training.

An alternative setup we consider is when \( T \) is set to a close to optimum value for the last stage even from the beginning. This means the early phases of curriculum will take longer to train, however, as shown in Figure 3 it speeds up training in the long run. Table III encapsulates the ablation study done to compare different setups for episode and horizon length over 3 billion steps.

### TABLE III

| Max Ep. Length | Horizon Length | Accuracy | Average Ep. Length |
|----------------|----------------|----------|---------------------|
| 100            | 100            | 90%      | 58                  |
| 50-100         | 50-100         | 67%      | 62                  |
| 100            | 10             | 6%       | 100                 |
| 100            | 50-100         | 71%      | 64                  |

VII. INCREASING COMPLEXITY

Additionally, we tested the active objects curriculum on a more challenging environment (Figure 4). In this setting, the cubes are replaced with three pieces from the set of five tetrominoes (L, T and I). The increase in complexity comes from the need to learn new different grasping methods associated with each object. To account for this, we increased the network capacity by one layer. In our experiments, the items just needed to be stacked in a specified position, although, more complexity can be introduced in later experiments where a certain orientation is required.

VIII. DISCUSSION

A. Method and Results

We designed the objects to have the same volume as the previously used cubes, however this led to smaller grasping areas. The agent failed to learn a good policy within the same time frame, therefore we added a new reward signal (Eq. 7) to assist in guiding the grasping. We found a \( \lambda_{\text{grasp}} = 2 \) to be most effective.

\[
r_{\text{grasp}} = \begin{cases} 1 & \text{iff } |x^{ee} - x^{obj}| \leq \epsilon \\ 0 & \text{otherwise} \end{cases}
\]  

(7)

In a separate test, we increased the complexity further by providing a random initial orientation to each object. However, to enable learning, we also concatenated the end-effector’s rotation to the observations. Results (Figure 6) show that curriculum learning combined with reward shaping is enabling the learning even in this complex scenario.

VIII. DISCUSSION

Limitations and future work The trained agents are currently unable to cope with situations where the tower of stacked objects is knocked down. Instead of trying to reposition the previous objects, the agent continues as though nothing happened. We leave this investigation for future work.

In the higher complexity environment, the agent has to stack 3 different shapes of objects. During further work, we would like to experiment with an increased number of shapes/objects, and test the effect of an extra signal that rewards the agent based on how high the stack gets.

Conclusion In this work, we investigated the problem of applying curriculum learning to a goal oriented environment such as the stacking of boxes. To accomplish this, we trained an RL agent using a shaped reward function and coupled it together with multiple variations of curricula architectures. We found that a simple architecture combined with a curriculum setting can achieve the desired behaviour, whereas vanilla RL fails to complete the task. We provided solutions for a number of reward hacking scenarios identified and showcased results obtained in a higher complexity environment. Together, these findings establish a strong foundation for future research on CL applied to goal-oriented tasks.
REFERENCES

[1] R. Jeong, Y. Aytar, D. Khosid, Y. Zhou, J. Kay, T. Lampe, K. Bousmalis, and F. Nori, “Self-supervised sim-to-real adaptation for visual robotic manipulation,” in *2020 IEEE international Conference on Robotics and Automation*, IEEE, 2020, pp. 2718–2724.

[2] A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager, “‘good robot!’: Efficient reinforcement learning for multi-step visual tasks with sim to real transfer,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6724–6731, 2020.

[3] A. X. Lee, C. M. Devin, Y. Zhou, T. Lampe, K. Bousmalis, J. T. Springenberg, A. Byravan, A. Abdolmaleki, N. Gileadi, D. Khosid, C. Fantacci, J. E. Chen, A. Raju, R. Jeong, M. Neunert, A. Laurens, S. Saliceti, F. Casarini, M. Riedmiller, r. hadsell raia, and F. Nori, “Beyond pick-and-place: Tackling robotic stacking of diverse shapes,” in *Proceedings of the 5th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 164, Aug. 2022, pp. 1089–1131.

[4] R. Li, A. Jabri, T. Darrell, and P. Agrawal, “Towards practical multi-object manipulation using relational reinforcement learning,” in *2020 IEEE International Conference on Robotics and Automation*, IEEE, 2020, pp. 4051–4058.

[5] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, “Curriculum learning for reinforcement learning domains: A framework and survey,” *arXiv preprint arXiv:2003.04960*, 2020.

[6] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, “Isaac gym: High performance gpu-based physics simulation for robot learning,” *arXiv preprint arXiv:2108.10470*, 2021.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

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

[9] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Overcoming exploration in reinforcement learning with demonstrations,” in *2018 IEEE international Conference on Robotics and Automation*, IEEE, 2018, pp. 6292–6299.

[10] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” *Advances in neural information processing systems*, vol. 30, 2017.

[11] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, “Autonomous navigation of stratospheric balloons using reinforcement learning,” *Nature*, vol. 588, no. 7836, pp. 77–82, 2020.

[12] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, “Learning to utilize shaping rewards: A new approach of reward shaping,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 15931–15941, 2020.

[13] L. P. Kaelbling, “Learning to achieve goals,” in *IJCAI*, Citeseer, vol. 2, 1993, pp. 1094–8.

[14] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *arXiv preprint arXiv:1506.02438*, 2015.

[15] T. D. Sanger, “Neural network learning control of robot manipulators using gradually increasing task difficulty,” *IEEE transactions on Robotics and Automation*, vol. 10, no. 3, pp. 323–333, 1994.

[16] J. Schmidhuber, “Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem,” *Frontiers in psychology*, vol. 4, p. 313, 2013.

[17] S. Forestier, R. Portelas, Y. Mollard, and P.-Y. Oudeyer, “Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning,” To be published at Journal of Machine Learning Research 2022, Nov. 2021.

[18] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *Icml*, vol. 99, 1999, pp. 278–287.

[19] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, “Concrete problems in ai safety,” *arXiv preprint arXiv:1606.06565*, 2016.

[20] M. Grzes, “Reward shaping in episodic reinforcement learning,” 2017.

[21] D. Makoviuchik and V. Makoviychuk, *RL games*, 2022. [Online]. Available: https://github.com/Denys88/rl_games

[22] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Diaz-Rodriguez, “Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges,” *Information fusion*, vol. 58, pp. 52–68, 2020.

[23] Y. Tang and D. Ha, “The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[24] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al., “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[25] A. Raffin, A. Hill, R. Traore, T. Lesort, N. Diaz-Rodriguez, and D. Filliat, “Decoupling feature extraction from policy learning: Assessing benefits of state
representation learning in goal based robotics,” *arXiv preprint arXiv:1901.08651*, 2019.

[26] A. Stooke and P. Abbeel, “Accelerated methods for deep reinforcement learning,” *arXiv preprint arXiv:1803.02811*, 2018.

APPENDIX

**TABLE IV**

| OBSERVATION SPACE | Observation Size Info                  |
|-------------------|----------------------------------------|
| Joint positions   | 9 Scaled between -1 and 1              |
| End-effector position | 6 Position, velocity and orientation |
| Goal position     | 3                                       |
| Total             | 19                                      |

**TABLE V**

| HYPERPARAMETERS | Value |
|-----------------|-------|
| Algorithm       | PPO   |
| Separate Actor/Critic | False |
| MLP size        | [256, 256, 256] |
| Fixed σ         | False |
| Activations     | ELU   |
| Learning rate (η)| 5e-4 |
| Discount factor γ| 0.99 |
| GAE τ           | 0.95  |
| KL threshold    | 0.008 |
| Nr environments| 0/32 - 16/34 |
| Minibatch size  | 51268 |
| Horizon Ep length| 150  |

Hardware used: 2x Nvidia GeForce RTX2080ti