Thinking, Fast and Slow: Application to Reinforcement Learning

Alexander Heilig
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
ucesi@student.kit.edu

Paul Jurisch
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
uskxa@student.kit.edu

Florian Kuhm
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
uwhbq@student.kit.edu

Karam Daaboul
Institute of Applied Informatics and Formal Description Methods (AIFB)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
mohammd.daaboul@kit.edu

***Abstract*—This paper introduces a novel approach for addressing the challenge of adapting Reinforcement Learning algorithms to changes in dynamics during long-term application. Drawing inspiration from Daniel Kahneman's "Thinking, Fast and Slow," our proposed approach equips the Reinforcement Learning agent with a fast and a slow "thinking" system. Specifically, we combine a Planner and a Soft Actor-Critic Policy based on a common Dynamics Model to improve performance during the adaptation phase without endangering long-term reward levels. We, therefore, provide a comprehensive explanation of our approach and subsequently present the results of various experiments. Although the initial approach does not achieve the expected performance, we demonstrate that relatively simple adjustments can significantly improve its efficacy, with the potential for further advancement in future research.**

***Index Terms*—model-based RL, SAC, CEM**

I. Introduction

Several approaches for Reinforcement Learning (RL) algorithms have demonstrated effectiveness in achieving good results in typical environments after sufficient training. However, the choice of approach can lead to significant differences in the improvement rate and asymptotic performance. For instance, in [1], the use of a Planner on OpenAI's Half-Cheetah environment achieves high rewards more quickly than the use of a Soft Actor-Critic (SAC) policy, while the latter promises better asymptotic performance and stability.

Assuming a model for a task with initial sufficient training time and data that is subsequently applied, one would naturally choose an approach with higher asymptotic performance. However, changes in dynamics may occur during long-term application due to changing external circumstances. Therefore, it is crucial to adapt the model as quickly as possible without fundamentally endangering the application's functionality while additionally maintaining optimal asymptotic performance. To address this challenge, we propose an approach inspired by Daniel Kahneman's "Thinking, Fast and Slow" [2], where the Reinforcement Learning agent is equipped with a fast and a slow thinking system. Specifically, we propose a combination of a Planning Algorithm (Thinking Slow) and a SAC policy (Thinking Fast) as illustrated in Figure 1. The combination of algorithms thereby works on a common Dynamics Model, while it aims to improve performance during the adaptation phase after dynamics changes without long-term performance losses.

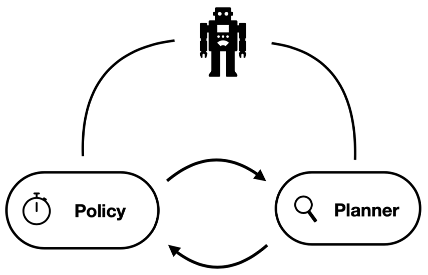


Fig. 1: Scheme for a novel agent that combines a Policy (Thinking Fast) and a Planner (Thinking Slow).

To this end, Section II introduces the notations and individual components of our approach, including the Dynamics Model, Soft Actor-Critic (SAC) Policy, and the Cross-Entropy Method (CEM). Section III subsequently provides a detailed explanation of the framework. In Section IV, we present the results of several experiments that help to clarify our basic hypothesis. Our initial approach faced justifiable problems, but with relatively simple adjustments, we achieved a significant improvement. We provide detailed explanations for the encountered issues in Section V.

While we cannot conclusively prove that our approach in its current form offers added value, we believe that a more sophisticated version of the approach could potentially serve this purpose. Our findings and future directions for improving the approach are discussed in the concluding section of this paper.

II. Preliminaries

In this section, we first introduce common notation and describe all components of the holistic RL framework that is explained in more detail in Section III.

A. Notation and Objective Function

Reinforcement learning (RL) is considered a subfield of Machine Learning that centers on how an agent learns to interact with an environment to optimize a cumulative reward signal. Thereby, the agent takes actions (a_t) based on the environment's state (s_t) and receives feedback in the form of rewards or penalties (r_t). Through continued interaction with the environment, the agent learns which actions are more likely to result in a high cumulative reward and adjusts its behavior accordingly. RL is a powerful approach that has been applied in a wide range of domains, including robotics, video games, and finance, and has shown remarkable success in complex tasks that are difficult for traditional methods to solve.

Mathematically speaking, the goal of maximizing returns over time by finding an optimal policy, denoted by π^*, can be formulated as follows

$$\pi^* = \arg\max_{\pi} E_{\tau \sim p_\pi} \left[\sum_{t=0}^{T} \gamma^t \cdot r(s_t, a_t) \right], \tag{1}$$

Moreover, RL can be divided into two main categories: model-based and model-free algorithms. Model-based algorithms learn a so-called Dynamics Model of the environment and then use this model to plan actions (see Subsection II-D). This approach is especially effective when the environment is well-understood and the model therefore accurately represents the real environment. However, it can be computationally expensive and can also suffer from errors if the learned Dynamics Model is not as accurate as thought. Model-free algorithms, on the other hand, learn a policy π from direct interaction with the environment. These algorithms do not require an additional model of the environment, making them more computationally efficient and easier to apply in complex environments (see Subsection II-C).

The individual components of the framework of chapter III will now be presented and explained in detail below. We will start by introducing the previously mentioned concept of a Dynamics Model.

B. Dynamics Model

Generally speaking, a Dynamics Model is a function that predicts how the environment will evolve in response to an agent's actions in RL. Specifically, given the current state (s_t) and an action (a_t), the Dynamics Model outputs the next state (s_{t+1}) and possibly the associated reward (r_t). Mathematically speaking it does so by approximating the (stochastic or deterministic) dynamics of the environment:

$$f(s_{t+1}|s_t, a_t) \approx p(s_{t+1}|s_t, a_t) \tag{2}$$

where p denotes the probabilities of the underlying distribution of the real environment and f represents the dynamic learned by the Dynamics Model.

Consequently, the Dynamics Model is the basic component of any model-based RL algorithm, such as CEM, which plans actions to maximize future rewards based on the model's predictions. While one could think of several ways to map a Dynamics Model, we use a neural network since it offers several advantages. First, the network can be trained efficiently using backpropagation to minimize the error between the predicted and observed next state and reward. This allows the model to be updated frequently and improves its accuracy over time. Secondly, neural networks have strong generalization abilities, enabling them to make comparability good predictions for the behavior of the environment even in previously unencountered states or actions. These properties are especially important, since a poorly designed Dynamics Model can lead to suboptimal actions chosen by the actual RL algorithm and degraded performance, especially for methods with very little inherent knowledge about the environment such as CEM.

To further improve the robustness of the model's predictions, we only predict the state changes from time t to $t+1$ and the associated reward. By predicting the changes instead of the absolute values, the model is more likely to capture the underlying dynamics of the environment, thus avoiding the accumulation of errors that can lead to significant inaccuracies. This approach has been shown to be effective in enhancing the accuracy of the Dynamics Model, especially when using CEM.

The following section will elaborate on which role the Dynamics Model plays with respect to our Fast Thinking System: SAC.

C. Thinking Fast: Soft Actor-Critic Algorithm (SAC)

Soft Actor-Critic (SAC) [3] is a Reinforcement Learning algorithm that aims to maximize expected cumulative reward in a model-free setting. SAC is designed to address the problem of exploration in Reinforcement Learning by encouraging the agent to take actions that are not only expected to lead to high rewards but also to maintain a diverse set of possible outcomes.

The specific objective function of SAC is given by:

$$J(\theta) = \mathbb{E}_{\pi_\theta} \Big[\sum_{t=0}^{\infty} \Big(r(s_t, a_t) + \alpha \mathcal{H}(\pi_\theta(\cdot|s_t)) \Big) \Big] \tag{3}$$

where θ represents the parameters of the policy. π_θ is the policy function that maps states to actions, r is the reward function, α is a temperature parameter that controls the trade-off between expected reward (exploitation) and entropy $\mathcal{H}$ of the policy (exploration). The expectation is taken with respect to the state-action distribution induced by the policy.

Evidently, the objective function consists of two main terms. The first term, $\sum_{t=0}^{\infty} r(s_t, a_t)$, can be described as the expected cumulative reward, which, as previously mentioned, is the standard objective in Reinforcement Learning. In contrast, the second term, $\sum_{t=0}^{\infty} \alpha \mathcal{H}(\pi_\theta(\cdot|s_t))$, represents the entropy of the policy's distribution. Additionally incorporating this term encourages exploration by penalizing policies with low entropy.

Furthermore, it is important to note that SAC is a model-free approach, meaning that it does not necessarily require a Dynamics Model yet directly interacts with the actual environment. However, in some cases, SAC can also be applied in a model-based setting. Specifically, a Dynamics Model simulating the environment is then used to generate the data for training the policy and value function networks, which can be beneficial as it typically improves sample efficiency. Due to this advantage and the need for a Dynamics Model when applying the Planning Algorithm (CEM), we will be using a model-based SAC throughout this paper. Next, we will turn our attention to the Slow Thinking System: CEM.

D. *Thinking Slow: Cross-entropy Method (CEM)*

The Cross-entropy Method (CEM) [4] is a planning algorithm that aims to find an optimal action sequence in a concrete model-based setting. Unlike model-free algorithms such as normal SAC, which usually optimize the weights of a neural network, thus trying to find good solutions for any occurring situation, CEM aims to maximize the accumulated reward over a specific sequence of actions over the planning horizon T. Hence, the objective function to be maximized by the procedure can be defined as:

$$J(a_0, \dots, a_T) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t \cdot r(s_t, a_t) \mid s_0\Big] \qquad (4)$$

where $a_0, \dots, a_T$ denotes the action sequence, s_0 represents the initial state, r the associated reward function, γ a decay factor, and T the planning horizon. More specifically, the maximization is carried out by repeatedly following a 3 step procedure:

1) Sampling of action sequences (trajectories) from a predefined distribution.
2) Evaluation of the sampled trajectories based on knowledge about the reward structure.
3) Update of the distribution based on the trajectories advantageousness.

The key elements of the procedure are further represented in the following Figure 2.

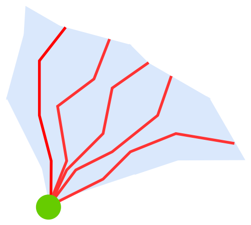


Fig. 2: Simple representation of exemplary trajectories (red) from an underlying distribution (blue) given a current state (green).

Thereby, the previously introduced Dynamics Model is used in two important ways during the procedure. First, it is used to evaluate the quality of an individual trajectory from a larger sampled set by predicting the future states and rewards that result from executing those sequences. This allows CEM to determine which trajectories are likely to result in the highest expected reward and therefore to iteratively make appropriate adjustments to the distribution the trajectories are sampled from.

Secondly, the Dynamics Model is used to simulate the environment during the optimization process. By simulating the environment, CEM can evaluate a large number of action sequences in parallel without the need for costly interactions with the real environment, which makes CEM a computationally efficient method for finding optimal action sequences.

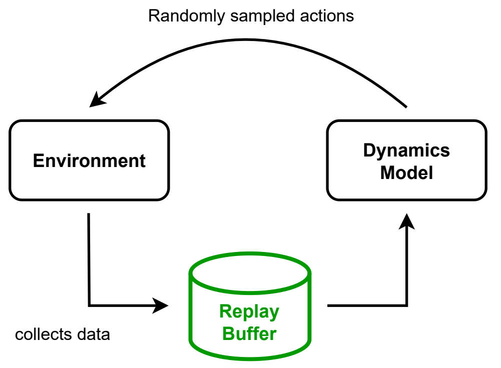
Now that we have introduced all the key components, the next Section III will briefly explain how exactly these components fit together to form the overall framework we propose.

III. Thinking Fast and Slow Framework

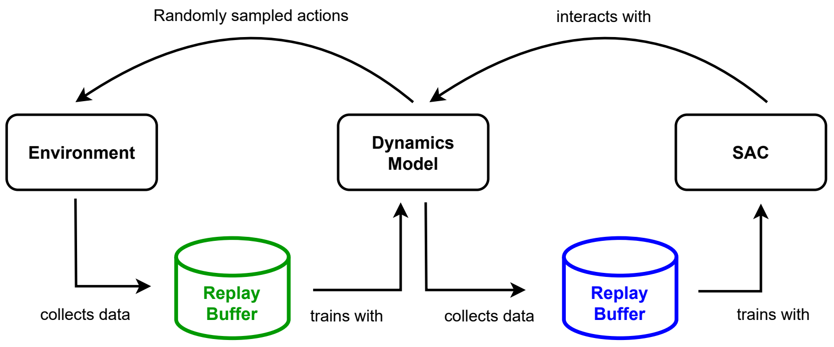
As previously described, we propose a novel framework that integrates the CEM-based Planner and the model-based SAC Policy in a mutually beneficial way, as depicted in Figure 3. Roughly speaking, we first train a Dynamics Model and an associated model-based SAC policy through interaction with the environment. If we subsequently detect a change in the environment dynamics, we switch to our Thinking Slow approach in form of the CEM-based Planner.

The exact procedure of the proposed framework can be described as follows: First, we randomly initialize a Dynamics Model and train it on a set of action state tuples (s_t, a_t) that are generated using randomly sampled actions and added to a Replay Buffer D_{real} containing real data. This training step ensures that the Dynamics Model can make fairly accurate predictions of the next state and reward before the SAC algorithm training begins. Subsequently, the model-based SAC algorithm interacts with the Dynamics Model and accesses a respective Replay Buffer D_{art} exclusively containing artificial data generated by the Dynamics Model. To ensure the stability of the algorithm, we used the same approach as in [5]. A starting state s_0 is sampled randomly from D_{real} and the Dynamics Model is initialized with this point. SAC then starts to collect a trajectory and explores the state action space in reach of the starting point. Subsequently, we use the latest version of the SAC policy and validate its performance by interacting with, yet not training on, the real environment. The resulting tuples (s_t, a_t, s_{t+1}, r_t) from the real environment are subsequently added to D_{real} and used to train the Dynamics Model. From here, we repeat the described procedure and start training SAC on the Dynamics Model again. This alternating training approach ensures that the Dynamics Model is trained to accurately capture the environment dynamics while also improving the performance of the SAC algorithm and is continued as long as the behavior of the environment remains unchanged.

If a change in environments dynamics is detected subsequently, the framework switches to the Thinking Slow approach, which aims to efficiently adapt the knowledge already collected to the changed dynamics. This is done by interacting with the real environment using actions suggested by the inherently unbiased CEM-based Planner which plans based on



(a) Firstly, the Dynamics Model is trained on real transitions sampled randomly from the environment.



(b) Secondly, model-based SAC and the Dynamics Model are trained iteratively.

Fig. 3: The framework trains a Dynamics Model on randomly generated action sequences before the model-based SAC algorithm interacts with the Dynamics Model, using virtual data generated by it, and subsequently training the Dynamics Model using action-state tuples generated by SAC.

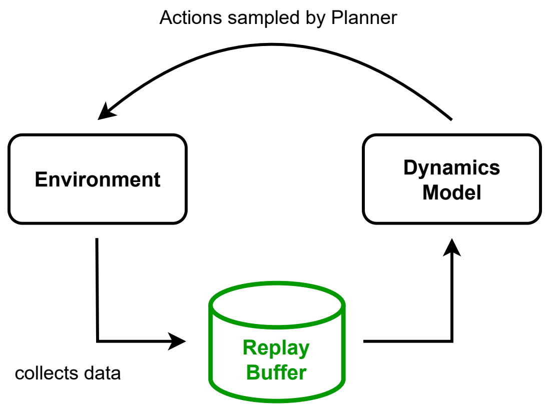


Fig. 4: When switching to System II, the Planner selects actions that are executed in the real environment, and these transitions are used to adapt the Dynamics Model to the new dynamics.

the common Dynamics Model. Each of the resulting transition tuples (s_t, a_t, s_{t+1}, r_t) is then added to the Replay Buffer D_{real}, which is in turn used to train the Dynamics Model. In this phase of the training, we additionally apply higher weights to the new samples in D_{real} generated by the planner, since these more accurately represent the new environment dynamics to be learned compared to the old samples. This is done in the hope of achieving a more efficient adaptation to the new dynamics compared to a procedure not involving the slow thinking system. However, one needs to bear in mind that this approach is primarily based on the assumption that the Thinking Slow system generates higher-quality samples in the short term and thus causes a faster adaptation of the Dynamics Model to the changed circumstances.

Once the Planner reaches a sufficient reward level, thus indicating that the underlying Dynamics Model accurately represents the new dynamics, the framework switches back to the fast-thinking system, involving SAC, described above. Doing so allows us to benefit again from SAC's asymptotic performance in the medium term while being able to ably the potentially more efficient slow-thinking system in phases of adoption. The procedure is briefly represented by the given Pseudo-Code 1 below.

To shed more light on the usefulness of the framework, we will present the procedure and results for some key experiments in the next section.

IV. Experiments

In the following sections, we will discuss our experiments in detail. To this end, we will first examine the behavior of the individual components SAC and the CEM-based Planner on the standard environment, and then test how the performance of both algorithms is affected by a substantial shift in the environment's dynamics.

For all experiments carried out, we used the code from the Github repository *mbrl*[1] as a basis for both the CEM-based Planner and the Dynamics Model. Specifically, the mentioned repo maps the Dynamics Model using a neural network with two hidden layers, each containing 200 neurons. To further incorporate SAC into this framework, we used code from the Github repository *rlkit*[2], which we modified to enable model-based training of SAC.

Furthermore, the Half-Cheetah environment from OpenAI's Gym Library with an action space of size $(1, 6)$ and an observation space of size $(1, 18)$ was used as the environment. Given this environment, the objective is to make the Half-Cheetah run as fast as possible, earning rewards for achieving higher speeds and receiving penalties for high absolute action values. While the default environment has a gravity of $9.81\frac{m}{s^2}$,

[1]Model-Based Reinforcement Learning Library of Facebook Research
[2]Reinforcement learning framework of Rail Berkeley

Algorithm 1: Thinking Fast and Slow Framework

```
Input: Real environment which can be interacted with
begin
    Initialize Dynamics Model $M$;
    Initialize Replay Buffer for artificial samples $D_{art}$;
    Initialize Replay Buffer for real samples $D_{real}$;
    Collect random tuples and add to $D_{real}$;
    Train $M$ on $D_{real}$;
    Set $change$ to False;
    while $running$ do
        while $not\ change$ do
            Create tuples by interaction with Dynamics
              Model $M$ and add to $D_{art}$;
            Train SAC on $D_{art}$;
            Create tuples with SAC in the real
              environment and add to $D_{real}$;
            Retrain $M$ on $D_{real}$;
            if $change\ in\ dynamics\ detected$ then
                Set $change$ to True;
            end
        end
        while $change$ do
            Sample actions with CEM and execute in
              real environment;
            Add resulting tuples to $D_{real}$;
            Retrain $M$ on $D_{real}$;
            if $sufficient\ performance\ reached$ then
                Set $change$ to False;
            end
        end
    end
end
```

it is possible to simulate a change in dynamics by adjusting the gravity value. We will start by showcasing the algorithm's behavior on the default gravity.

A. Behavior on Standard Gravity

To ensure comparability between both algorithms, the central measure evaluated is reward level based on the number of interactions with the real environment. Given the framework we propose, this number is equal to the number of elements in the Replay Buffer D_{real}. Regarding reward levels, we always consider the cumulative reward for a given trajectory of length 250.

Figure 5 compares the obtained reward levels for the CEM-based Planner and SAC. One thing to consider is that SAC is not trained directly, yet instead, the Dynamics Model is first pre-trained in a burn-in phase to ensure the stability of the subsequent training for SAC. After 7500 real samples we then iteratively begin training SAC following the training of the Dynamics Model based on the Replay Buffer D_{real} for 100 epochs with early stopping. SAC generates 10000 initial virtual data points using the Dynamics Model and adds them to the Replay Buffer D_{art}. Another 5000 data points are added to D_{art} during each of the training epochs. Subsequently, training is performed on this collected data. SAC only collects trajectories of length 10 when it interacts with the Dynamics Model such that the impact of noise due to imprecise model predictions is reduced (see [5], [6] for such a recommendation).

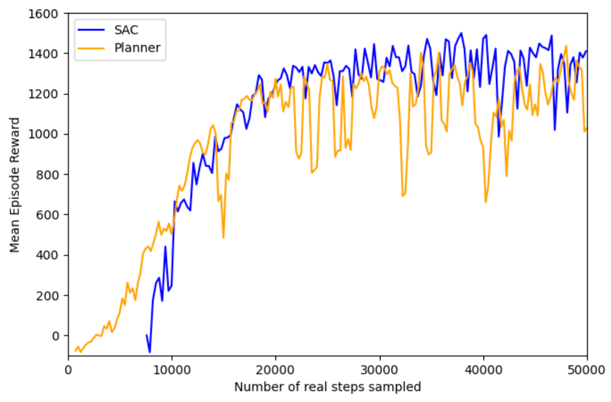


Fig. 5: The CEM-based Planner and SAC algorithms are compared in this Figure. While SAC's Dynamics Model is pre-trained before training starts, CEM trains the Dynamics Model directly on real data, improving the planner's accuracy over time in approximately the same way SAC improves.

In contrast, the CEM-based Planner starts training the Dynamics Model directly. Specifically, this is done by training the Dynamics Model based on a Replay Buffer of real data D'_{real}, with early stopping after each trajectory of length 250 for 100 epochs. It is easy to derive from the Figure 5 that the rewards of the applied planning algorithm steadily improve with the Dynamics Model becoming better at approximating the real environment.

Altogether Figure 5 clearly implies that both algorithms are able to to achieve sufficient performance based on the given environment. However, the asymptotic performance of SAC seems to be somewhat stronger and more stable, as suspected. The CEM-based Planner, on the other hand, seems to achieve better performance at the beginning of the training phase, which is partly due to the burn-in phase when using SAC. Overall, however, this part of the results is in line with the assumptions underlying the introduced framework. Next, we will present the performance of both algorithms when confronted with a change in the dynamics of the environment.

B. Adaption to Changes in Gravity

In this section, we present how SAC and the CEM-based Planner behave in case of a gravitational change. Figure 6 first presents the adaptation of SACs performance to four different individual changes in gravity after the training phase presented in Figure 5. Once more, a burn-in phase of 2500 real samples has been used to adapt the Dynamics Model after setting the new gravity, providing greater stability in the iterative training of SAC and the Dynamics Model that follows. The width of the confidence interval thereby clearly implies that an increased gravity leads to a fairly stable behavior of the agent. In contrast, a decrease in gravity causes the agent to make more fundamental mistakes, such as causing the cheetah to roll over and lie on its back, therefore naturally increasing the variance of the results obtained. Moreover, SAC quickly adapts to the new environment dynamics and approaches the initial return level fairly quickly if the change in gravity is comparably small (e.g. for blue and magenta scenario). Larger deviations, on the other hand, result in a less robust behavior of SAC and a slower re-training of the agent.

Figure 6 additionally presents the behavior of the Planner in response to changes in gravity. When compared to SAC, the Planner initially seems to require more samples to adjust to the novel circumstances. It is particularly noticeable that the Planner initially achieves a reward level close to 0, with potential reasons for this behaviour discussed in Section V. Despite this observation, it makes sense to equip the Planner with the Dynamcs Model obtained while working with SAC, since a completely new Dynamics Model leads to much slower performance improvement as can be derived from the right plot in Figure 6. In the following, we will discuss a concrete approach to improve the Planners initial performance.

C. SAC-adjusted CEM Planner

The results shown in the previous section show that the CEM-based Planner, working on the Dynamics Model trained with SAC, initially achieves comparably low reward levels and only reaches a performance level similar to SAC with an increasing number of samples. In the following, we therefore try to make an adjustment to the Planner to compensate for these performance disadvantages.

Specifically, we do so by capitalizing on even more a-priori information than just the Dynamics Model. Explicitly, the novel approach consists of determining the mean of the distribution from which the Planner samples its trajectories using the previously trained SAC policy. Even though this version of SAC still represents the old dynamics of the environment, it can still be considered a good basis for the sampling of the Planner that follows. Putting this SAC-adjusted Planner to test yields the performance presented in green in Figure 6. Compared to the previous plain CEM-based Planner, the adjusted approach yields a substantially higher initial reward level, while still allowing the Dynamics Model to gradually adapt to the new gravity. Overall, the performance achieved can be considered comparable to the one achieved by SAC, yet not significantly better.

D. Results of Combined Approach

The Combined Approach implements the framework from Section III and utilizes the SAC-adjusted Planner from Section IV-C. This Combined Agent shares the initial training phase with the Soft Actor-Critic (SAC) algorithm described in Section IV-A. In order to evaluate whether Thinking Slow in new environments is beneficial in terms of adaptation and robustness, we conducted the following experiment: First, to ensure convergence, the Combined Agent was trained for 125000 real sample steps under standard gravity. Afterwards, it was deployed on environments with different gravities, choosing actions according to the Planner for 10000 real sample steps. After 2500 steps, model-based SAC training commenced. At sampling step 135000, the SAC component was reemployed to select actions. For comparison, we use an Agent that has the identical model and SAC policy at the onset of the gravitational changes but uses and trains the model-based SAC policy throughout the experiment. Figure 7 shows the results. They are based on the aggregated runs of six independent Combined Agents and three independent SAC Agents, each evaluated ten times every 2500 steps. The uncertainty intervals display the two-sigma derivation from the mean.

In environments with gravitational forces exceeding the original training gravity (blue and magenta lines), the observations of the Sections IV-B and IV-C are reflected. The performance of SAC-adjusted Planner (first 10000 steps) is closely tied with that of the pure SAC Agent and proves equally robust (compare the uncertainty bands between the left and right image). Since SAC policy of the Combined Approach and the pure SAC Agent (last 10000 steps) perform similarly as well, Thinking Slow has no advantages in regard to adaptation in high gravities. However, in gravity-lower-than-training scenarios (cyan and orange), the Combined Approach shows an edge over the SAC Agent. Both Agents exhibit large variations in performance. However, while the sharp drop in performance of the SAC agent in the second half of the experiment with $g = 6.81\frac{m}{s^2}$ can be explained as a stochastic anomaly, the Combined Agent shows no severe drips in performance in its uncertainty interval despite its results are based on twice as many runs. This hints at a stronger robustness against catastrophic failures in learning. Profound are differences under the weakest gravity (orange). The SAC Agent performs consistently worse than the Planner and the SAC component of the Combined Approach. Here, Thinking Slow is beneficial.

We suspect that this is related to the close alignment of the Planner and SAC policy: The Planner examines the current assumptions of the policy, but deviates if they do not fit with the dynamics learned so far. Therefore, it allows the SAC policy to explore parts of the new environment without having to execute exploratory actions. Therefore, the reason why Thinking Slow is not useful in environments with large gravities might be that there are generally fewer sources of error in such environments rather than a conceptual flaw of the Combined Approach (see IV-B).

V. Problem Analysis

The findings presented in the preceding sections suggest that the combination of CEM and SAC in the simple manner discussed is inadequate for achieving significant performance improvement in most of the considered cases. However, a

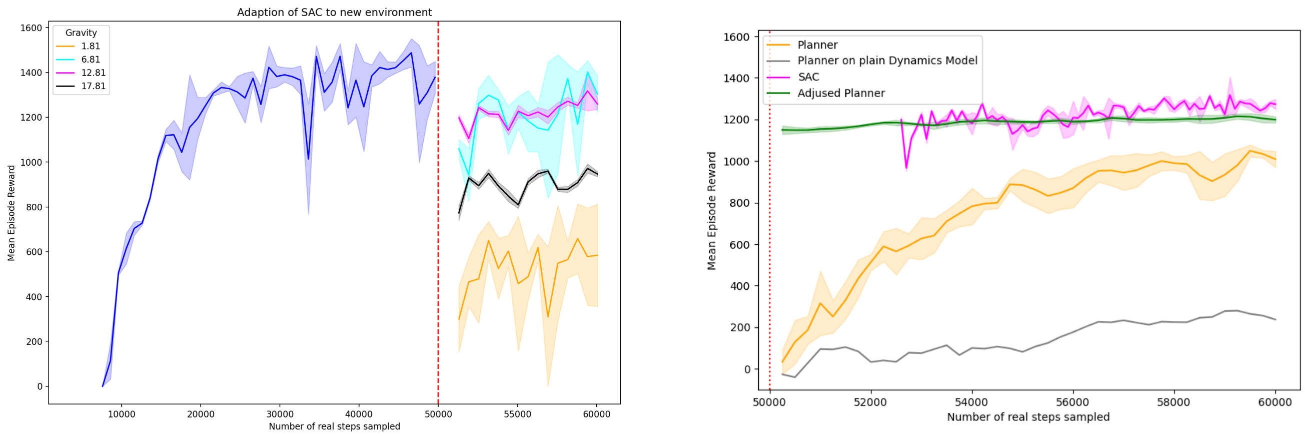


Fig. 6: The presented Figures illustrate the training process of the SAC algorithm as described in Section IV-A, where the Dynamics Model and SAC are retrained on 10,000 new real samples with varying gravitational conditions. The performance of SAC (magenta) is compared to that of the CEM Planner with reinitialized Dynamics Model (gray) and the Dynamics Model pre-trained by SAC (orange) in the right image. The SAC-adjusted Planner (green) exhibits performance similar to that of SAC.

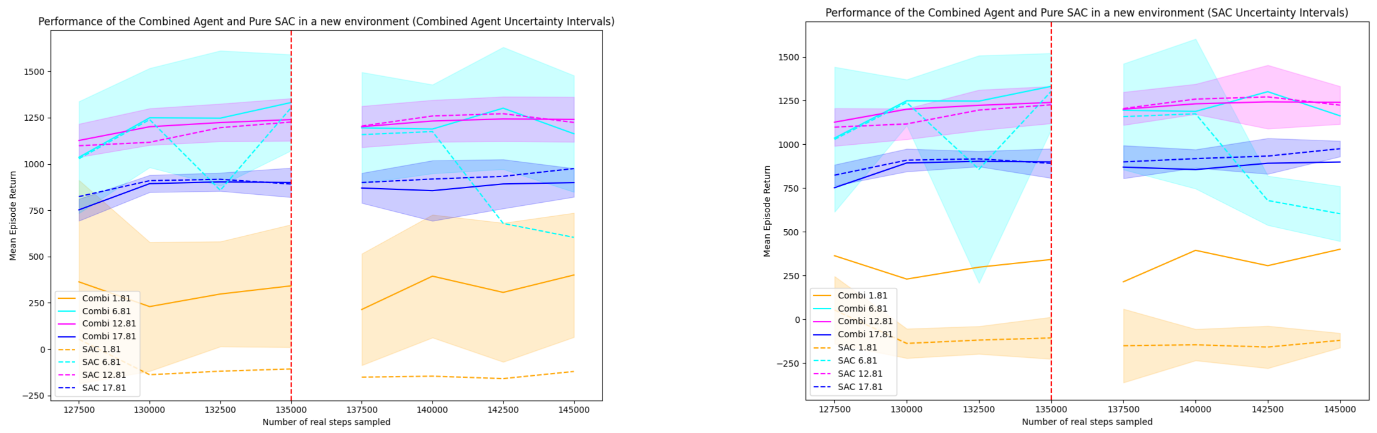


Fig. 7: Performance results of the Combined Agent (Section III) in comparison to the Soft Actor-Critic (SAC) algorithm (Section IV-A) in new environments with varying gravitational conditions. Solid lines indicate the performance of the Combined Agent, and dotted lines relate to SAC. Also shown are the uncertainty intervals of the Combined Agent (left) and the SAC Agent (right). While being identical until sample step 125000, both approaches branch off as soon as a new environment has been detected - and the Combined Agent starts to choose actions according to the SAC-adjusted Planner (Section IV-C). After sample step 135000, the Combined Agent switches back to its internally trained SAC component. The Combined Agent performers similar to SAC in environments with high gravity, but performs better than SAC in low gravitational environments.

deeper analysis uncovers possible explanations for this behavior, with the Dynamics Model playing a crucial role in this context. Specifically, the primary question to address is what conditions must hold for the Dynamics Model to facilitate seamless switching between SAC (Thinking Fast) and Planner (Thinking Slow). A prerequisite for this approach is that the Planner attains acceptable performance levels on the existing Dynamics Model despite expected losses resulting from the change in dynamics. However, as shown in Figure 6 and discussed in Section IV-B, the Planner initially struggles to achieve satisfactory performance with a relatively minor change in dynamics and only improves over time. Incorporating prior knowledge from SAC, as described in Section IV-C, significantly enhances the initial performance of the Planner. This observation suggests that the Dynamics Model provides a sound foundation, especially for SAC-typical actions. Conversely, the Dynamics Model appears to provide insufficient generalization for SAC-untypical actions. To test this hypothesis, we compared the accuracy of the predictions of the Dynamics Model for SAC actions versus random actions. While it is unsurprising that SAC actions generate more precise predictions with lower uncertainty, Figure 8 shows that random actions not only produce greater uncertainty but also generate systematic bias. As emphasized by [7], this issue is a natural challenge for all approaches with action distributions that vary significantly from that of of the model training policy, which in our case reinforces the need to rely on SAC in the thinking slow phase or select a SAC-like approach, such as the SAC-adjusted Planner that we deployed in this work.

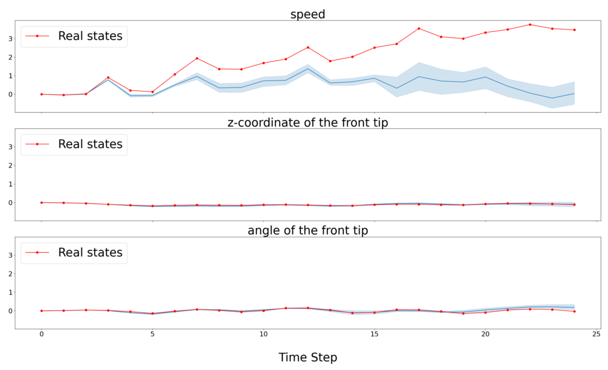


Fig. 8: Results of predicted (Dynamics Model) versus real observations of specific environmental aspects with random samples.

It has been noted in previous research [8] that utilizing a maximum entropy approach in RL algorithms results in stronger robustness of the agent to changes in dynamics, due to increased exploration in the action state space. This observation provides a possible explanation for the robustness demonstrated by the combined approach in response to changes, as compared to using the SAC-adjusted CEM-based Planner. In contrast, RL algorithms that do not explicitly account for policy entropy, such as TRPO [9] and PPO [10], would likely respond much more poorly to changes in dynamics, as does the model-based SAC approach.

VI. Conclusion

This work proposes the use of a combined RL agent consisting of a CEM-based Planning Algorithm (Thinking Slow) and a model-based SAC Policy (Thinking Fast) to improve performance in phases of adaptive learning caused by changes in the dynamics of the environment. To this end, we introduced preliminaries in Section II and presented the results of our experiments in Section IV.

Altogether, the results of the simple, initial approach we chose do not improve performance compared to the exclusive use of SAC. However, as shown in Section IV-C, a strong improvement in performance can be achieved by increasingly incorporating prior knowledge from the Thinking Fast (SAC) in the Thinking Slow (CEM) phase. Moreover, the performance of the Combined Approach in low gravitational environments observed in Section IV-D hints at a conceptual advantage in environments. Thus, we believe that the experiments conducted provide some valuable insights and indications for future research.

First of all, our study demonstrates that model-based SAC is more sample-efficient than its model-free counterpart. The ability of the Dynamics Model to generalize to unseen states enables SAC to improve its policy without necessarily requiring additional real samples.

Furthermore, our experiments fundamentally support the hypothesis that SAC can be considered fairly robust, at least for small changes in the dynamics of the environment. This finding supports the idea of integrating SAC more strongly during the Thinking Slow phase in future approaches, as we did in Section IV-C. Evaluating further adaptations and procedures via experiments in this context could help in achieving significant improvements compared to the pure use of SAC and can be considered an interesting approach for future research from our point of view.

Another significant finding, severely compromising the performance of the initial approach, is the fact that the Dynamics Model lacks generalization for the algorithms (CEM and SAC) used in our framework. The results from Section IV-B clearly indicate that when using a plain CEM approach with a Dynamics Model purely trained with actions from SAC leads to insufficient performance. While we have therefore decided to align the Planning Algorithm more strictly with SAC, one could also consider the approach of fostering an improvement of the Dynamics Model generalization capabilities. We thereby consider the question of how concrete approaches could look as another interesting direction for future research.

Hence, there are still plenty of open research questions that need to be addressed in order to enhance the proposed framework. In addition to the previously mentioned points, there is also a need to develop a sufficient approach for designing a mechanism for switching between the fast and slow thinking system. In this regard, various approaches for change detection have been proposed in the literature, including [11]–[13]. These can be explored to develop a suitable approach for the proposed framework.

References

[1] K. Chua, R. Calandra, R. McAllister, and S. Levine, “Deep reinforcement learning in a handful of trials using probabilistic dynamics models,” *Advances in neural information processing systems*, vol. 31, 2018.

[2] D. Kahneman, *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

[3] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, “Soft actor-critic algorithms and applications,” 2019.

[4] C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius, “Sample-efficient cross-entropy method for real-time planning,” 2020.

[5] M. Janner, J. Fu, M. Zhang, and S. Levine, “When to trust your model: Model-based policy optimization,” 2021.

[6] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning,” in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[7] J. B. Hamrick, A. L. Friesen, F. Behbahani, A. Guez, F. Viola, S. Witherspoon, T. Anthony, L. Buesing, P. Veličković, and T. Weber, “On the role of planning in model-based deep reinforcement learning,” *arXiv preprint arXiv:2011.04021*, 2020.

[8] B. Eysenbach and S. Levine, “Maximum entropy rl (provably) solves some robust rl problems,” *arXiv preprint arXiv:2103.06257*, 2021.

[9] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *arXiv preprint arXiv:1502.05477*, 2015.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," in *International Conference on Machine Learning*. PMLR, 2017, pp. 4658–4667.

[11] S. A. Sontakke, S. Iota, Z. Hu, A. Mehrjou, L. Itti, and B. Schölkopf, "Galilai: Out-of-task distribution detection using causal active experimentation for safe transfer rl," *arXiv preprint arXiv:2110.15489*, 2021.

[12] R. Luo, R. Sinha, A. Hindy, S. Zhao, S. Savarese, E. Schmerling, and M. Pavone, "Online distribution shift detection via recency prediction," *arXiv preprint arXiv:2211.09916*, 2022.

[13] P. KJ, N. Singh, P. Dayama, A. Agarwal, and V. Pandit, "Change point detection for compositional multivariate data," *Applied Intelligence*, pp. 1–26, 2021.