Representing Positional Information in Generative World Models for Object Manipulation

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

The ability to predict outcomes of interactions between embodied agents and objects is paramount in the robotic setting. While model-based control methods have started to be employed for tackling manipulation tasks, they have faced challenges in accurately manipulating objects. As we analyze the causes of this limitation, we identify the cause of underperformance in the way current world models represent crucial positional information, especially about the target's goal specification for object positioning tasks. We propose two solutions for generative world models: position-conditioned (PCP) and latent-conditioned (LCP) policy learning. In particular, LCP employs object-centric latent representations that explicitly capture object positional information for goal specification. This naturally leads to the emergence of multimodal capabilities.

1 Introduction

Among RL algorithms, model-based approaches aim to provide greater data efficiency compared to their model-free counterparts [4, 6]. With the advent of world models (WM) [5], model-based agents have demonstrated impressive performance across various domains [8, 17, 10, 13], including real-world robotic applications [22, 20].

When considering robotic object manipulation tasks, it seems natural to consider an object-centric approach to world modeling. Object-centric world models, like FOCUS [3] learn a distinct dynamical latent representation per **object**. This contrasts with the popular Dreamer method [10], where a single **flat** representation, referring to the whole scene is extracted.

Model-based generative agents, like Dreamer and FOCUS, learn a latent model of the environment dynamics by reconstructing the agent's observations and use it to generate latent sequences for learning a behavior policy in imagination [8–10]. However, these kinds of agents have shown consistent issues in succeeding in object manipulation tasks, both from proprioceptive/vector inputs [11] and from images [19].

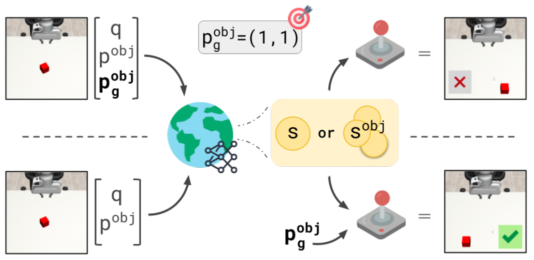

Figure 1: The world model compresses input observations into a single or per object latent state representation. The compressed representation serves as input to the policy for action selection. **(top)** Goal information is provided through the input state vector. **(bottom)**: Both single and object-centric representations can be paired to a **target-conditioned policy**.

After analyzing the causes of failure of generative agents, we propose two solutions to improve performance:

- a simpler solution, where the target is expressed as a vector of spatial coordinates, that presents no major changes to the model architecture and minimal changes to policy learning;
- a tailored solution employing an object-centric approach that integrates positional information about the objects into the latent space of the world model. This approach enables the possibility to specify goals through multimodal targets, e.g. vector inputs or visual goals.

2 Analysis of the Current Limitations

To provide insights into the limitations of current world model-based agents in object-positioning tasks, we consider the performance of Dreamer and FOCUS on a pose-reaching and an object-positioning task. For pose-reaching, we opted for the Reacher environment from the DMC suite [21]. In this task, we consider the end-effector of the manipulator as the entity to be positioned at the target location. For the more complex object positioning task, we opted for a cube-manipulation task from Robosuite [24]. The given cube has to be placed at the specified target location to succeed in the task.

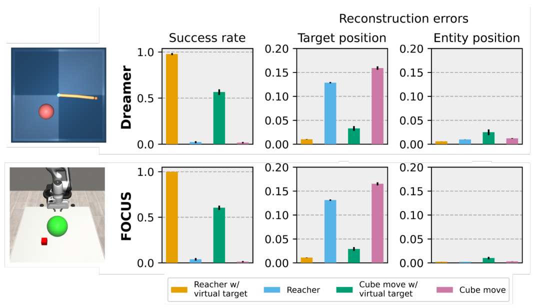

Figure 2: **(left)**: examples of virtual targets. **(top-right)**: Dreamer's success rate and reconstruction performance over target and entity position (end-effector position for reacher and cube position for the cube environment). **(bottom-right)**: Equivalent for the FOCUS object-centric model.

In both environments, the target position is uniformly sampled within the workspace at every new episode. We test the environments in two different scenarios: first, with a virtual visual target that is rendered in the environment, and second, without a visual target, where the target location is provided only as a vector in the agent's inputs. Training details are provided in appendix G. Based on Fig. 2, we highlight the significant gap in performance between the tasks with the virtual visual targets rendered in the environment and the tasks using only spatial coordinates as a target. The agents struggle to solve the tasks without a virtual target. It can also be noticed a negative correlation between the agents' ability to reconstruct positional information and the performance on the task. This is particularly evident for the target position, but it also seems to apply to the entity position.

There is a significant difference in the relative significance of the target information compared to the entire observation, in terms of their dimensionality. The information pertaining to a positional target comprises a maximum of three values (i.e., the *xyz* coordinates of the target). Conversely, when considering a visual cue, there are three values (i.e., RGB values) for each pixel that represents the target cue. Consequently, the relative significance of the target information is, at least, greater in the case of a large visual target, i.e. larger than a single pixel. This difference in the dimensionality affects the computation of the loss, and thus the weight of each component in the decoder's loss. For the entity, the agents have access to this information in the visual observation. Indeed, it's not surprising that both agents reconstruct the entity position accurately. To confirm our hypothesis that the improved predictions are due to the greater significance of the visual targets in the overall loss, we provide additional experiments in appendix C.

Discussion. A concurrent work [14] conducted an extensive study between the interplay of the reward and the observation loss in a world model. Our analysis provides an additional insight, as we identify within the observation loss, an unbalance between the different decoded components. In this work, rather than focussing on how to balance the losses (Appendix D), we consider different approaches to alleviate this issue. The central idea is to find alternative ways to provide positional information about the target directly to the reward computation and policy learning modules, rather than relying on the reconstruction of the targets obtained by the model.

3 Conditioned Policy

Position Conditioned Policy (PCP). The first declination of our proposed solutions is the conditioning of the policy directly on the positional coordinates of the desired target. By default, the world model encodes the target's positional information in the latent states, which are then fed to the policy for behavior learning. Instead, as shown in the bottom of Fig. 1, we propose to concatenate the object positional coordinates p_g^{obj} to the latent states s_t as an input to the policy network. We refer to this strategy as *Position-Conditioned Policy (PCP)*: $\pi_{PCP}(a_t|s_t, p_g^{obj})$

When employing PCP, the policy has direct access to the target's positional information p_g^{obj}. This can also be leveraged for reward computation. Rather than learning a reward head, we can use the world model's decoder to predict the object's position at time t, obtaining $\hat{p}_t^{obj}$. Then, the reward r_{PCP} can be estimated as the distance between the target given to the policy and the reconstructed position of the entity of interest: $r_{PCP} = dist(\hat{p}_t^{obj} - p_g^{obj})$

Latent Conditioned Policy (LCP). Conditioning the policy on explicit features has its limitations, particularly when extending features beyond positional ones, or when working with different goal specifications, e.g. visual ones. Therefore, expressing features implicitly could represent a more robust approach. To address this, we propose a latent conditioned method for behavior learning. This approach is analogous to the one adopted in LEXA [15] for goal-conditioned behavior learning. However, we tailor our strategy for object manipulation by designing an object-centric approach. We refer to our novel implementation as *Latent-Conditioned Policy (LCP)*.

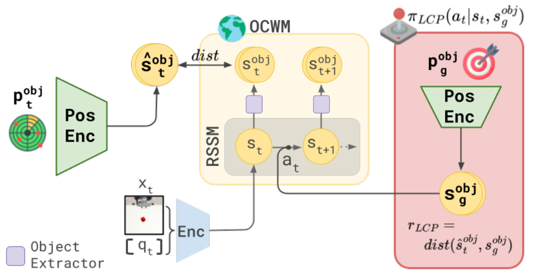

Figure 3: LCP leverages an object-centric representation. With the latent position encoder network, the agent learns to predict the latent of each object in the scene given the sole object position. The policy is then conditioned on an object latent target obtained from the target goal observation. Distance functions are expressed as cosine similarities.

In LEXA, policy conditioning occurs on the entire (flat) latent state, using either cosine or temporal distance methods. However, in manipulation tasks involving small objects, the cosine approach is inadequate because it prioritizes matching the robot's position over visually smaller aspects of the scene, such as an object's position, rather than on bigger visual components of the scene, e.g. the robot pose. The temporal approach was introduced to mitigate this issue. However, this approach generally requires a larger amount of data to converge, as the training signal is less informative, being based only on the temporal distance from the goal [15]. We argue that object-centric latent representations offer greater flexibility to condition the policy, thanks to the disentangled latent information. With LCP, we can condition the policy solely on the object's latent states, enabling fine-grained target conditioning focused exclusively on the entity of interest.

Latent Positional Encoder. To obtain object latent features for a given target position, we introduce the Latent Positional Encoder model, as shown in Fig. 3. This model enables inferring an object's latent state directly from the object's positional information, namely $p(\hat{s}_t^{obj}|p_t^{obj})$.

During training, the latent positional encoder is trained to minimize the negative cosine distance between the predicted and the reference object latent state: $\mathcal{L}_{pos} = -\frac{\hat{s}_t^{obj} \cdot s_t^{obj}}{\|\hat{s}_t^{obj}\| \|s_t^{obj}\|}$

Compared to the original loss function of FOCUS (defined in Appendix E), the world model loss becomes: $\mathcal{L}_{ocwm} = \mathcal{L}_{FOCUS} + \mathcal{L}_{pos}$

Latent-Conditioned Policy Learning. The introduction of the *latent positional encoder* enables the conditioning over the target object's latent. By encoding a desired target position p_g^{obj}, the target object's latent state s_g^{obj} is inferred. The latter serves as the conditioning factor for the policy network: $\pi_{LCP}(a_t|s_t, s_g^{obj})$. To incentivize the policy to move the entity of interest to the target location, we maximize the negative latent distance between $\hat{s}_t^{obj}$ and s_g^{obj}. The distance function used is cosine similarity. r_{LCP} becomes then: $r_{LCP} = \frac{\hat{s}_t^{obj} \cdot s_g^{obj}}{\|\hat{s}_t^{obj}\| \|s_g^{obj}\|}$

Visual targets. Additionally with respect to PCP, LCP enables conditioning the policy on visual targets. In this case, the agent does not use the latent position encoder. Instead, given a visual observation representing the goal target position for the object, the world model can infer the corresponding world model state, using the encoder and the posterior. Then given such a state, the object extractor allows extracting the target latent state s_g^{obj}, which is used in the reward computation.

4 Results

We now present the evaluation of the trained models (training details in Appendix G) for a set of 4 environments (Appendix F). The score function considered is presented in Appendix, Eq. 2 .

	Dreamer	FOCUS	Dreamer + PCP	FOCUS + PCP	FOCUS + LCP
Reacher	0.27 ± 0.11	0.29 ± 0.1	0.8 ± 0.08	0.87 ± 0.04	0.91 ± 0.02
Cube move	0.35 ± 0.05	0.35 ± 0.08	0.54 ± 0.04	0.74 ± 0.04	0.61 ± 0.05
Shelf place	0.4 ± 0.06	0.3 ± 0.1	0.58 ± 0.08	0.59 ± 0.1	0.65 ± 0.08
Pick&Place	0.26 ± 0.13	0.22 ± 0.12	0.48 ± 0.15	0.47 ± 0.17	0.45 ± 0.17
Overall	0.32 ± 0.08	0.29 ± 0.09	0.6 ± 0.09	0.67 ± 0.09	0.65 ± 0.08

Table 1: Average score for 100 goal points equally distributed over the workspace. Performance is averaged over 3 seeds, ± indicates the std. error.

Spatial-coordinates goal specification. By providing the different agents with goals uniformly distributed in the workspace we extract the overall performance of each method. Results are presented in Table 1. Overall, the FOCUS agent equipped with PCP or LCP gives the best performance, followed by Dreamer + PCP. In the "Cube move" setting where the cube object is close to the camera, PCP has an edge, we think this is influenced by the accurate position prediction coming from the world model, which is trained using more accurate position information (bigger segmentation mask $\rightarrow$ better granularity in position). Instead, in the "Shelf place" environment, the latent representation of LCP represents best. Given that the camera is further away from the scene, we believe the agent is better able to deal with the inaccuracies that come from the inaccurate position readings.

Visual goal specification. An emergence property of FOCUS + LCP is the possibility to define goals via different modalities. The policy π_{LCP} can be conditioned on the goal object latent $\hat{s}_g^{obj}$ coming from the encoding of the visual goal x_g.

We compare our method with visual goal conditioning against LEXA cosine and temporal. The goal locations are provided to the simulator which renders the corresponding goal observations by "teleporting" the object to the correct location. The agent is then asked to matched the visual goal, after resetting the environment. Results are shown in Fig. 4, where the positional conditioning results are shown for reference.

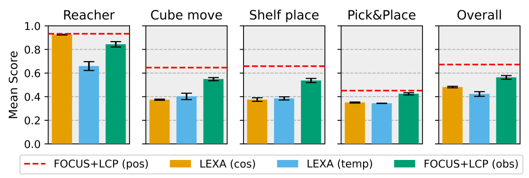

Figure 4: The mean score was achieved over 10 episodes with goal observations for latent conditioning. The performance of our method with spatial-coordinate goals (pos) is shown as a reference. Performance is averaged over 3 seeds.

As stated before, LEXA matches the flat latent vector to the goal one. This proves helpful in the Reacher environment, where the only part that moves is the agent, and thus LEXA cosine achieves the best performance. LEXA cosine fails in the other tasks, given the presence of multiple entities in the observations and visual goals, i.e. the robotic arm and the object. where the model focuses on matching the visually predominant features i.e. the robotic arm. FOCUS+LCP performs better than both LEXA with cosine and temporal distance in all environments but the Reacher. When compared to the performance of FOCUS+LCP with spatial-coordinates goal specification, there is a decrease of only $\sim$10% in performance.

5 Conclusion

We analyzed the challenges in solving visual robotic positional tasks using generative world model-based agents. We found these systems suffer from information bottleneck issues when considering positional information for task resolution (i.e. goal position).

The approaches we presented overcome this issue by providing the policy network with more direct access to the target information. Positional Conditioning Policy (PCP), allows direct conditioning on the target spatial coordinates. We showed PCP improves performance for any class of world models, including Dreamer-like "flat" world models and FOCUS-like object-centric world models. Latent Conditioning Policy (LCP), is an object-centric approach that we implement on top of FOCUS. This allows the conditioning of the policy on object-centric latent targets, enabling multimodal goal definition. Results show the promise of this approach as a multimodal goal-specification method.

References

[1] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2020. URL `https://arxiv.org/abs/1912.01588`.

[2] J. Fan. A review for deep reinforcement learning in atari:benchmarks, challenges, and solutions, 2023. URL `https://arxiv.org/abs/2112.04145`.

[3] S. Ferraro, P. Mazzaglia, T. Verbelen, and B. Dhoedt. Focus: Object-centric world models for robotics manipulation, 2023.

[4] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018.

[5] D. Ha and J. Schmidhuber. World models. 2018. doi: 10.5281/ZENODO.1207631. URL `https://zenodo.org/record/1207631`.

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[7] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *ICML*, pages 2555–2565, 2019.

[8] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. 2020. URL `https://arxiv.org/pdf/1912.01603.pdf`.

[9] D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. In *ICLR*, 2021.

[10] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[11] N. Hansen, H. Su, and X. Wang. Td-mpc2: Scalable, robust world models for continuous control, 2024.

[12] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[13] P. Lancaster, N. Hansen, A. Rajeswaran, and V. Kumar. Modem-v2: Visuo-motor world models for real-world robot manipulation, 2024.

[14] H. Ma, J. Wu, N. Feng, C. Xiao, D. Li, J. Hao, J. Wang, and M. Long. Harmonydream: Task harmonization inside world models, 2024. URL `https://arxiv.org/abs/2310.00344`.

[15] R. Mendonca, O. Rybkin, K. Daniilidis, D. Hafner, and D. Pathak. Discovering and achieving goals via world models, 2021.

[16] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. A. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. M. Zhang. Solving rubik's cube with a robot hand. *ArXiv*, abs/1910.07113, 2019.

[17] S. Rajeswar, P. Mazzaglia, T. Verbelen, A. Piché, B. Dhoedt, A. Courville, and A. Lacoste. Mastering the unsupervised reinforcement learning benchmark from pixels. 2023.

[18] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak. Planning to explore via self-supervised world models. In *ICML*, 2020.

[19] Y. Seo, D. Hafner, H. Liu, F. Liu, S. James, K. Lee, and P. Abbeel. Masked world models for visual control, 2022.

[20] Y. Seo, J. Kim, S. James, K. Lee, J. Shin, and P. Abbeel. Multi-view masked world models for visual robotic manipulation, 2023.

[21] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022. URL `https://www.sciencedirect.com/science/article/pii/S2665963820300099`.

[22] P. Wu, A. Escontrela, D. Hafner, K. Goldberg, and P. Abbeel. Daydreamer: World models for physical robot learning, 2022.

[23] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL `https://arxiv.org/abs/1910.10897`.

[24] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

Appendix

A Preliminaries

The agent is a robotic manipulator that, at each discrete timestep t receives an input x_t from the environment. The goal of the agent is to move an object in the environment from its current position p_t^{obj} to a target goal position p_g^{obj}.

In this work, we focus on observations composed of both visual and vector entities. Thus, $x_t = (o_t, v_t)$ is composed of the visual component o_t and of the vector v_t. The latter is a concatenation of proprioceptive information of the robotic manipulator q_t, the object's position p_t^{obj}, and the target position p_g^{obj}. The target position can also be expressed through a visual observation x_g, from which the agent should infer the corresponding p_g^{obj} to succeed in the positioning task.

A.1 Generative World Models

Generative world models learn a latent representation of the agent inputs using a variational auto-encoding framework [12]. Dreamer-like agents [9, 10] implement the world model as a Recurrent State-Space Model (RSSM) [7]. The encoder $f(\cdot)$ is instantiated as the concatenation of the outputs of a CNN for high-dimensional observations and an MLP for low-dimensional proprioception. Through the encoder network, the input x_t is mapped to an embedding e_t, which then is integrated with dynamical information with respect to the previous RSSM state and the action taken a_t, resulting in s_t features.

$$\begin{aligned} \text{Encoder:} \quad & e_t = f(x_t) \\ \text{Posterior:} \quad & p_\phi(s_{t+1}|s_t, a_t, e_{t+1}), \\ \text{Prior:} \quad & p_\phi(s_{t+1}|s_t, a_t), \\ \text{Decoder:} \quad & p_\theta(\hat{x}_t|s_t). \end{aligned}$$

Generally, the system either learns to predict the expected reward given the latent features [8], using a reward predictor $p_\theta(\hat{r}_t|s_t)$. Alternatively, some world-model based methods adopt specialized ways to compute rewards in imagination, as the goal-conditioned objectives in LEXA [15].

Rewards are computed on rollouts of latent states generated by the model and are used to learn the policy π and value network v in imagination [8–10].

In our experiments, we consider a world model with a discrete latent space [9]. We also implement advancements of the world model representation introduced in DreamerV3 [10], such as the application of the symlog transform to the inputs, KL balancing, and free bits to improve the predictions of the vector inputs and the robustness of the model.

A.2 Object-centric World Models

Compared to Dreamer-like *flat* world models, the world model of FOCUS [3] introduces the following object-centric components:

$$\begin{aligned} \text{Object latent extractor:} \quad & p_\theta(s_t^{obj}|s_t, c^{obj}), \\ \text{Object decoder:} \quad & p_\theta(\hat{x}_t^{obj}, \hat{m}_t^{obj}|s_t^{obj}). \end{aligned}$$

Here, $x_t^{obj} = (o_t^{obj}, p_t^{obj})$ represents the object-centric inputs and it is composed of segmented RGB images o_t^{obj} and object positions p_t^{obj}. The variable c^{obj} indicates which object is being considered.

Thanks to the *object latent extractor* unit, object-specific information is separated into distinct latent representations s_t^{obj}. Two decoding units are present. The introduced object-centric decoder $p_\theta(\hat{x}_t^{obj}, \hat{m}_t^{obj}|s_t^{obj})$ reconstructs each object's related inputs x_t^{obj} and segmentation mask m_t^{obj}. The original Dreamer-like decoder takes care of the reconstruction of the remaining vector inputs, i.e. proprioception q_t and given goal targets p_g^{obj}.

We provide additional descriptions of the world model and policy learning losses, hyperparameters, and training details in the Appendix.

A.3 Object Positioning Tasks

In general terms, we consider positioning tasks the ones where an entity of interest has to be moved to a specific location. Two positioning scenarios are considered in this analysis: *pose reaching* and *object positioning*. Pose-reaching tasks can be seen as simplified positioning tasks where the entity of interest is part of the robotic manipulator itself. Pose-reaching tasks are interesting because these only require the agent to have knowledge of the proprioceptive information to infer their position in space and reach a given target. When interacting with objects instead, there is the additional necessity of knowing the position of the object entity in the environment. Then, the agent needs to be able to manipulate and move the entity to the provided target location.

For object positioning tasks, especially when considering a real-world setup, there is a significant advantage in relying mainly on visual inputs. It is convenient because it avoids the cost and difficulty associated with tracking additional state features, such as the geometrical shape of objects in the scene or the presence of obstacles. Some synthetic benchmarks additionally make use of "virtual" visual targets for positioning tasks [21, 23], which strongly facilitates the learning of these tasks, leveraging rendering in simulation. However, applying such "virtual" targets in real-world settings is not often feasible. Non-visual target locations can be provided as spatial coordinates. Alternatively, an image showing the target location could be used to specify the target's position.

Rewards and evaluation criteria. When applying RL algorithms to a problem, a heavily engineered reward function is generally necessary to guide the agent's learning toward the solution of the task [16]. The object positioning setup allows us to consider a natural and intuitive reward definition that scales across different agents and environments. We define the reward as the negative distance between the position of the entity of interest and the goal target position:

$$r_t = -\mathtt{distance}(\text{object}, \text{target}) = -\|p_t^{obj} - p_g^{obj}\|_2. \tag{1}$$

In the spirit of maintaining a setup that is as close as possible to a real-world one, to retrieve positional information p_t of the objects we rely on image segmentation information, rather than using the readings provided from the simulator. For each entity of interest, the related position is extracted by computing the centroid of the segmentation mask and subsequently transformed according to the camera extrinsic and intrinsic matrices to obtain the absolute position with respect to the workspace.

For evaluation purposes, we use the goal-normalized score function:

$$\mathtt{normalized\ score} = \exp\left(-\frac{\|p_t^{obj} - p_g^{obj}\|_2}{\|p_g^{obj}\|_2} \right) \tag{2}$$

As detailed in the Appendix, the above function allows us to rescale performance between 0 and 1, where 1 = expert performance, a common evaluation strategy in RL [1, 2].

B Normalized score

Scaling performance using expert performance is a common evaluation strategy in RL [1, 2]. In our problem, we define the reward as the negative distance:

$$r_t = -r(p_t^{obj}) = -\|p_t^{obj} - p_g^{obj}\|_2. \tag{3}$$

For a given goal p_g^{obj}, $r_t \in]-\inf, 0]$. In order to compare different tasks, where distances may have different magnitudes, we divide the rewards r_t by the typical reward range. This is given by $r_{max} - r_{min}$, where $r_{min} = r(p_0^{obj})$, with p_0 being the initial position of the object (this is normally around the origin, and $r_{max} = r(p_g^{obj}) = 0$.

Thus, we obtain:

$$s_t = r_t/(r_{max} - r_{min}) \tag{4}$$
$$= r(p_t^{obj})/(0 - r(p_0^{obj})) = \tag{5}$$
$$= -\|p_t^{obj} - p_g^{obj}\|_2/(0 + \|0 - p_g^{obj}\|_2) \tag{6}$$
$$= -\|p_t^{obj} - p_g^{obj}\|_2/\|p_g^{obj}\|_2 \tag{7}$$

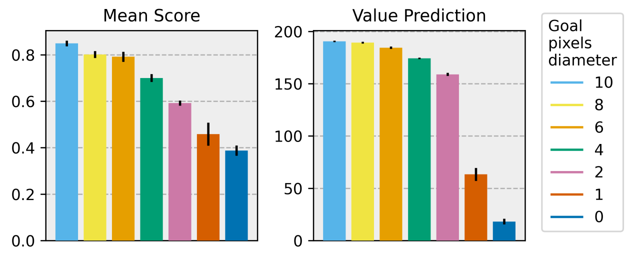

Figure 5: Dreamer virtual visual goal modulation experiments on the Reacher environment. Value prediction from the value network is shown to highlight the policy's awareness of the lack of information with respect to the target goal.

Finally, we apply the exp operator, to make values positive and bring them in the $[0, 1]$ range, where 1 is the expert score:

$$\texttt{normalized score} = \exp\left(-\frac{\|p_t^{obj} - p_g^{obj}\|_2}{\|p_g^{obj}\|_2}\right) \tag{8}$$

C Target size ablation

In Figure 5, we present a study where the Dreamer model is trained on the Reacher environment with varying visual target sizes.

We observe that the reduction in pixel information regarding the target adversely affects the target representation within the model, resulting in a deficiency of this information being conveyed to the policy network. The policy struggles to learn to position the entity at the correct location, and we observe that this is correctly reflected in the value function's predictions. This means the policy is aware that is not being able to reach the goal. With small targets (< 5 pixels diameters), the representation tends to put more attention on other visually predominant aspects of the environment, struggling to predict the position of the target. In the case of a single pixel target, the amount of target information equals the one of a positional vector and, as expected, the task performance is equally low.

D Loss rescaling ablation

To overcome the identified information bottleneck, different strategies can be considered. The simplest one is the re-scaling of the loss components in the decoder to incentivize the model's encoding of the target information. This approach requires finding the optimal scaling factor between the different decoding components, given the complexity of the environment at hand (i.e. 2D or 3D) and the amount of relevant pixels. In Figure 6, we present supporting experiments based on Dreamer, where we vary the importance of the target in the loss of the world model, using different coefficients. We observe that very high coefficients improve the target's reconstruction and thus allow the agent to learn the task. However, the optimal loss coefficient may vary, depending on the complexity of the

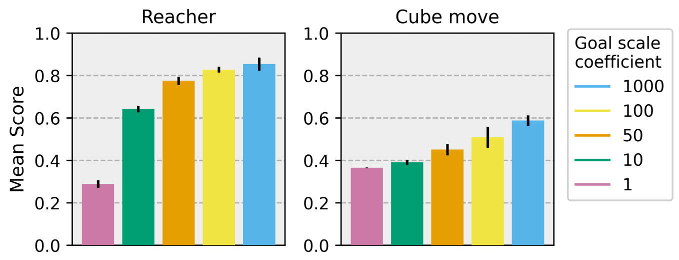

Figure 6: Dreamer trained with goal scaling modulation on the Reacher and Cube move environments.

environment and the presence of information-rich observations. As this naive solution may require extensive hyperparameter tuning for each new scenario, we aim to find more robust strategies for overcoming this issue.

E FOCUS objective

Training of the FOCUS architecture is guided by the following loss function:

$$\mathcal{L}_{\text{FOCUS}} = \mathcal{L}_{\text{dyn}} + \mathcal{L}_{\text{state}} + \mathcal{L}_{\text{obj}}. \tag{9}$$

$\mathcal{L}_{\text{dyn}}$ refers to the dynamic component of the RSSM, and equals too:

$$\mathcal{L}_{\text{dyn}} = D_{\text{KL}}[p_\phi(s_{t+1}|s_t, a_t, e_{t+1})||p_\phi(s_{t+1}|s_t, a_t)]. \tag{10}$$

the backpropagation is balanced and clipped below 1 nat as in DreamerV3 [10].

The object loss component is instantiated as the composition of NLL over the mask and RGB mask reconstructions:

$$\mathcal{L}_{\text{obj}} = -\log \underbrace{p(\hat{m}_t)}_{\text{mask}} - \log \sum_{\text{obj}=0}^{N} \underbrace{m_t^{\text{obj}} p_\theta(\hat{x}_t{}^{\text{obj}}|s_t^{\text{obj}})}_{\text{masked reconstruction}} \tag{11}$$

Finally, the decoder learns to reconstruct the rest of vector state information v_t by minimization of the negative log-likelihood (NLL) loss:

$$\mathcal{L}_{\text{state}} = -\log p_\theta(\hat{q}_t, p_g^{obj}|s_t) \tag{12}$$

F Baselines and Environments

For the evaluation of the proposed method we consider several manipulation environments (Figure 7):

- **Reacher** (DMControl): which, as described previously, represents a pose-reaching positioning task.
- **Cube move** (Robosuite): where considered target locations are on the 2D plane of the table, no height placement is considered.
- **Shelf place** and **Pick&Place** (Metaworld): The robotic manipulator has to place the cube at the given target location. Considered target locations are on the 2D space in front of the robotic arm.

In all environments, the reward signal is defined as the distance between the entity of interest (in the Reacher environment, this is the end-effector) and the target location. All considered environments lack any visual target; the target is provided as an input vector containing spatial coordinates.

We benchmark our methods against various baselines:

- **Dreamer**: based on a PyTorch DreamerV2 implementation, but integrated with input vector symlog transformation and KL balancing of the latent dynamic representation, from the DreamerV3 paper.

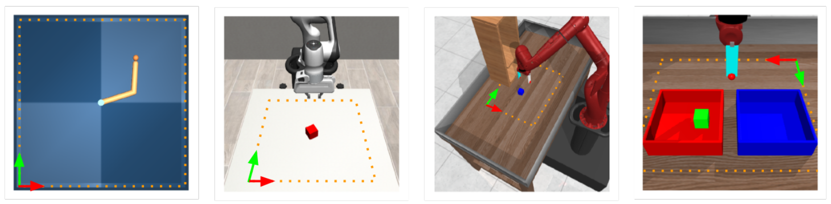

Figure 7: Simulation environments with relative workspace, delimited by an orange dotted line, and the reference frames indicated with arrows.

- **FOCUS**: An object-centric world model implementation based on DreamerV2, also integrated with input vector symlog transformation and KL balancing of the latent dynamic representation.
- **LEXA**: Based on DreamerV2, this is a latent goal-conditioned method. The conditioning is based on the full latent target. Both proposed distance methods (cosine and temporal) are considered. We adopted our own PyTorch implementation for LEXA.

G Training details and Hyperparameters

All methods are trained following an offline RL training scheme. The offline datasets contain 1M steps in the environment, which are collected using the object-centric exploration strategy proposed in [3]. The datasets are loaded in the replay buffer of the offline agents, and the training is conducted for 250K steps. Both world model and agent are updated at every training step. V100-16GB GPUs have been used for all experiments. Our proposed methods (i.e. Dreamer/FOCUS + PCP, FOCUS + LCP) took roughly 18 hours to complete each training run.

The hyperparameters used for the main implementation of the world models and agent are the same used in DreamerV2 [9] official implementation. Symlog function is applied at every input. KL balancing as in DreamerV3 [10] is implemented.

With reference to FOCUS model, we have the following additional parameters:

- Object-extractor: MLP composed of 2 layers, 512 units, ReLU activation;

With reference to FOCUS + LCP model, we have the following additional parameters:

- Object-encoder: MLP composed of 4 layers, 400 units, ReLU activation;
- Distance method object-encoder objective: Cosine similarity (also tested MSE)
- Distance method actor policy objective: Cosine similarity (also tested MSE)

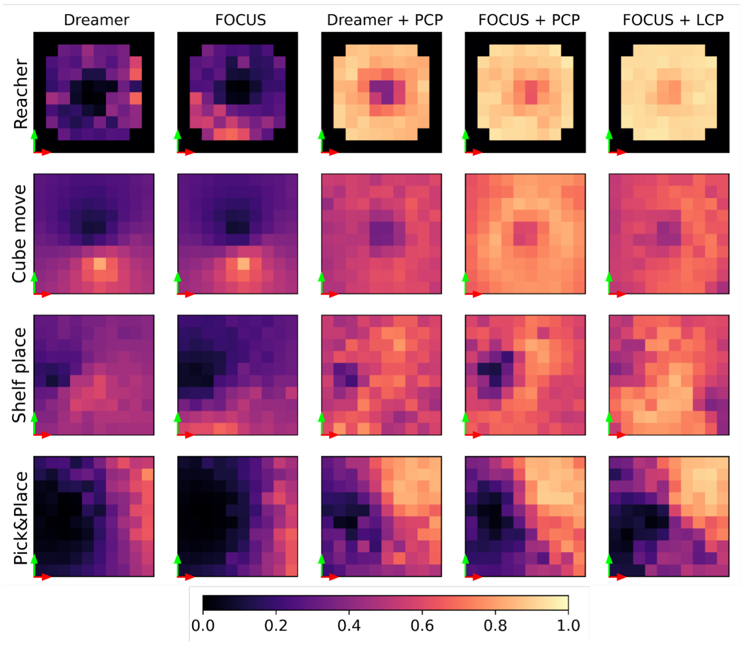

Figure 8: Heatmaps of the mean achieved score for uniformly spread targets in the workspace. References frames refers to the one presented in the figures of Table 1. The score notation is expressed as the notation presented in Eq. 2. Results are averaged over 3 seeds.

H Heatmaps positioning tasks

To highlight the performance distribution over the different goals in the environment, in Fig. 8 we present heatmaps with the score function for each target location in the workspace. Results are presented for all the different tasks. As expected, both Dreamer and FOCUS have poor performances, resulting in only a few positions being reached with a high score. All the proposed methods have a similar distribution, reaching goals spread all over the environment.

I Offline Training Curves

Offline training curves are presented in Figure 9. In general FOCUS + PCP/LCP have faster convergence when compared to all other methods. Only for the Reacher environment, LEXA cosine converge faster.

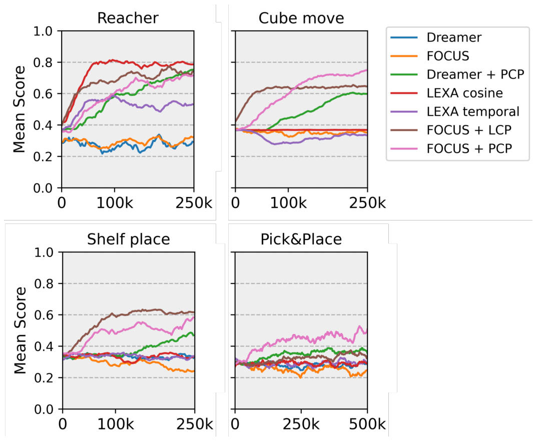

Figure 9: Offline training curves. Standard deviation is omitted for graphical reasons. Mean score refers to eq. 2 and is computed over 5 evaluation episodes, performed during the offline training. For each episode, a random goal is selected out of a pool of 10 manually engineered ones.

J Explorations strategies

In the presented work each model is trained offline from a pre-recorded dataset. The dataset of choice is obtained from pure exploration behavior. In Fig. 10 we compare the general performance of LCP when trained on datasets acquired using different exploration strategies. We consider the object-centric entropy maximization method proposed by Ferraro et al. [3] and Plan2Explore [18].

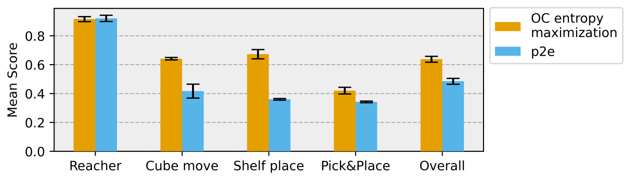

Figure 10: Mean score achieved over 10 episodes for models trained with both datasets obtained from FOCUS exploration method (Object-Centric entropy maximization) and Plan2Explore. The score is expressed according to equation 2.

Overall exploring by maximizing the entropy over the object's latent, gives better performance in the downstream task. We hypothesize this is related to the focus the exploration strategy puts on the object of interest while disregarding background aspects in the scene.