An Expansive Latent Planner for Long-horizon Visual Offline Reinforcement Learning

Robert Gieselmann
KTH Royal Institute of Technology
Stockholm, Sweden
Email: robgie@kth.se

Florian T. Pokorny
KTH Royal Institute of Technology
Stockholm, Sweden
Email: fpokorny@kth.se

***Abstract*—Sampling-based motion planning algorithms are highly effective in finding global paths in geometrically-complex environments. However, classical approaches, such as RRT, are difficult to scale beyond low-dimensional search spaces and rely on privileged knowledge e.g. about collision detection and underlying state distances. In this work, we take a step towards the integration of sampling-based planning into the reinforcement learning framework to solve sparse-reward control tasks from high-dimensional inputs. Our method, called `VELAP`, determines sequences of waypoints through sampling-based exploration in a learned state embedding. Unlike other sampling-based techniques, we iteratively expand a tree-based memory of visited latent areas, which is leveraged to explore a larger portion of the latent space for a given contingent of search iterations. We demonstrate state-of-the-art results in learning control from offline data in the context of vision-based manipulation under sparse reward feedback. Our method extends the set of available planning tools in model-based reinforcement learning to include a latent planner that searches for global solutions paths, rather than being bound to a fixed prediction horizon.**

I. Introduction

The acquisition of complex motor skills from raw sensory observations presents one of the main goals of robot learning. Reinforcement learning (RL) [38] provides a generic framework to obtain such decision-making policies through the interaction with an environment. Especially model-based RL methods [30] have recently gained attention due to benefits in terms of sample-efficiency and robustness in long-horizon scenarios. To address the issue of short-sighted decisions, model-based agents are often combined with planning algorithms. However, effective planning with high-dimensional inputs, such as video data, is often challenging due to the increased complexity of the search space and the difficulty in generating accurate long-term predictions. Consequently, a growing body of research has explored the utilization of representation learning to simplify the decision-making problem by mapping it to an abstract and lower-dimensional latent state space [14, 15, 28, 33, 16].

The model-based RL literature has investigated various planning methods in latent spaces, encompassing zero-order shooting-based approaches such as the Cross-Entropy Method (CEM) [5, 14] and Model-Predictive Path Integral (MPPI) [41, 31, 16], first-order gradient-based optimization [36, 15], and more recently, trajectory collocation using second-order solvers [33]. Despite this methodological diversity, the majority of existing tools primarily facilitate local optimization within a fixed prediction horizon. In this paper, we argue that long-horizon planning in learned latent state spaces can pose significant geometric challenges, necessitating methods that strive for global solutions. Even with guidance from value heuristics, such as the one proposed in [16], local minima may still impede progress, particularly when estimating the optimal value function is difficult due to sparse reward feedback or limited training data.

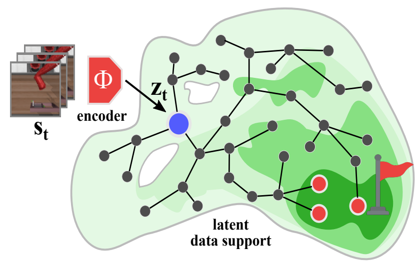


Fig. 1. We propose a sampling-based planning approach which grows a search tree in the latent space to globally explore reward-maximizing paths (blue:start, red:goal nodes, green: estimated values).

The limitations observed in existing methods raise the need for more sophisticated planning strategies that can seamlessly integrate with learned state and dynamics models. Sampling-based motion planning [23], a well-established field in classical robotics, provides a diverse range of algorithms for finding global paths between states in continuous and geometrically-complex environments. However, these methods typically require the definition of suitable distance metrics, state samplers, and rely on other task-specific information, such as collision checks. Recent works by [28, 19, 13] have proposed modifications of sampling-based planners for planning in latent spaces. However, these approaches either rely on expert data or are not directly applicable to reward-based learning settings. It is worth noting that, for many problems, defining objectives through rewards offers greater practicality compared to specifying explicit goal states. For instance, in the context of wrapping a deformable cloth around an object, the set of successful goal states may be extensive due to the vast underlying configuration space of deformable objects.

This paper explores the integration of sampling-based planning techniques into learned latent spaces, providing new avenues for model-based reinforcement learning. Specifically, we focus on the challenging scenario of offline RL [24], which is characterized by the amplified effects of value approximation errors [12]. Offline learning has gained significant attention due to its potential in leveraging logged trajectory data. Moreover, it allows us to better study the performance of planning in isolation by disentangling training and data collection. We introduce a novel method, termed Value-guided Expansive Latent Planning (`VELAP`), which combines a long-horizon planning module with a learned state embedding which is optimized to facilitate efficient generation and task-specific evaluation of future predictions. Similar to [13], we draw inspiration from sampling-based motion planners in robotics and construct a tree-based representation that grows by probing the continuous latent space. This search tree serves as a memory of state coverage and guides the planning process towards unexplored regions within the data support. Moreover, we demonstrate that leveraging value estimates obtained through temporal difference learning as sampling heuristics during planning significantly accelerates the discovery of suitable solution paths. To benchmark our method, we adapt the robot manipulator control environments from the meta-world benchmark suite [43]. Our experiments reveal that our proposed method surpasses existing approaches by a significant margin in terms of episode success rate. We attribute this performance gain to `VELAP`'s ability to overcome local value optima through global exploration, in contrast to the prevalent approach of optimizing over a fixed horizons.

II. Preliminaries

a) MDPs and Offline RL: A Markov decision process (MDP) is defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces, $\mathcal{P}(s'|s,a)$ are state dynamics, $r(s,a)$ is a scalar reward function, and γ is a discount factor. The goal of reinforcement learning [38] (RL) is to find a policy $\pi(a|s)$ that maximizes the expected discounted future reward $R[\tau]$ over all trajectories τ given an initial state distribution p_0 and induced by π, i.e., to optimize $\mathbb{E}_\pi[R[\tau]]$. The problem of offline RL [24] arises when training from a fixed dataset $\mathcal{D}$ consisting of trajectories generated by a behavioral policy π_β. Due to the limited coverage of $\mathcal{D}$ across the state-action space, effectively addressing the adverse consequences of poor approximations outside the data support becomes crucial in the development of offline RL methods [12].

b) Hindsight data relabeling: Relabeling data has emerged as a popular technique in goal-conditioned off-policy RL [2, 8, 25, 7, 26] for the purpose of enhancing training efficiency. The underlying idea behind hindsight relabeling is to transform unsuccessful trajectories into successful ones by retrospectively modifying their goals [2]. This approach extends to offline trajectory datasets, where relabeling can automatically generate experiences for learning state-reaching behaviors [6, 27]. Specifically, failed transitions can be relabeled by designating the subsequent state as the desired goal and adjusting the corresponding reward accordingly. To introduce diversity into the dataset, negative examples of hindsight goals can be sampled from future steps within the same trajectory or from alternative trajectories. A connection between hindsight relabeling and contrastive learning was recently discussed in [9].

$$\begin{aligned}
\text{State encoder: } & \phi : \mathcal{S} \rightarrow \mathcal{Z} \\
\text{Dynamics: } & h : \mathcal{Z} \times \mathcal{A} \rightarrow \mathcal{Z} \\
\text{Action model: } & g : \mathcal{Z} \times \mathbb{R}^m \rightarrow \mathcal{A} \\
\text{Local policy: } & \pi^l : \mathcal{Z}\times\mathcal{Z}\rightarrow\mathcal{A} \quad Q^l : \mathcal{Z}\times\mathcal{Z}\times\mathcal{A}\rightarrow\mathbb{R} \\
\text{Global policy: } & \pi^g : \mathcal{Z}\rightarrow\mathcal{A} \qquad Q^g : \mathcal{Z}\times\mathcal{A}\rightarrow\mathbb{R}
\end{aligned} \tag{1}$$

III. Value-guided Expansive Latent Trees

In this section, we detail the elements that comprise `VELAP`, our proposed offline RL planning agent.

a) Problem definition: We are interested in solving sparse reward continuous control tasks from high-dimensional inputs. For this purpose, we choose the example of visual control for a state space $\mathcal{S}$ and action space $\mathcal{A} = \mathbb{R}^{d_{\text{action}}}$. $\mathcal{S} = \mathbb{R}^{W\times H\times C\times N}$ describes sequential image data where W is the image width, H the height, C the channel dimension and N the number of frames. Note that we employ the MDP formulation, hence assume that states $s \in \mathcal{S}$ are informative to predict the distribution of future states. A sparse binary reward $r : \mathcal{S}\times\mathcal{A}\rightarrow\{0,1\}$ is designed to provide a positive value only upon successful completion of the task. To train our model, we are provided with an offline dataset $\mathcal{D}$ consisting of recorded transitions obtained from a sub-optimal policy.

b) Components overview: To tackle the specified problem, we propose a novel model-based reinforcement learning agent that incorporates a tree-based search, inspired by ESTs [17], within a learned representation space. Our approach involves several key components outlined in Eq. 1. The encoder ϕ maps input states to latent encodings, while the dynamics model h predicts future latent states based on actions, serving as a tool for expanding the search tree during planning. A local policy π^l is trained to navigate between neighboring states in the tree. The global policy π^g determines optimal actions with respect to the MDP task objective. Both policies are learned using actor-critic RL algorithms. Among various actor-critic offline RL methods available, we select TD3-BC [10] due to its robustness and simple implementation. To improve the predictions of Q^l and measure value uncertainty, we employ an ensemble of n_{ens} Q-heads $\{Q^l_1, ..Q^l_{n_{\text{ens}}}\}$ similar to [1] (see App. E). For the following, we use k to denote the k-th ensemble member and define $Q^{i,j}_{\text{min}} := \min\{Q^l_k(z_i, z_j, \pi^l(z_i, z_j))\}^{n_{\text{ens}}}_{k=1}$ as the minimum and $Q^{i,j}_{\text{std}} := \text{std}\{Q^l_k(z_i, z_j, \pi^l(z_i, z_j))\}^{n_{\text{ens}}}_{k=1}$ as the standard deviation of the ensemble predictions at a particular step. Finally, we incorporate a conditional generative model g to facilitate sampling actions from the state-conditioned action distribution.

c) Alignment of representation and planner: To compute feasible path in $\mathcal{Z}$, our state representations must favor the approximation of long-horizon dynamics. In particular, our model should not only predict the next state with high accuracy but provide useful future waypoint predictions over several time steps. At the same time, our goal is to solve the MDP control task, hence a good representation should also learn relevant features which ease optimization of the control behaviors π^g and π^l. Existing work on model-based RL often use surrogate metrics for model learning (e.g. mean-squared error on dynamics or pixel reconstruction loss) which do not enforce compliance with the actual control performance. In the literature this misalignment between the environment model and planner [22] has been shown to hamper the performance of the controller. To overcome those issues, we train our state representation encoder ϕ through joint optimization with the latent dynamics, local and global RL policies, leading to the overall training objective $\mathcal{L}_{\text{model}}$ (Eq. 2). In this regard, $\mathcal{L}_{Q^l}$ describes the temporal difference (TD) value loss for training Q^l, $\mathcal{L}_{Q^g}$ the TD loss for training Q^g and $\mathcal{L}_h$ the loss function for the dynamics model h.

$$\mathcal{L}_{\text{model}} = \mathcal{L}_{Q^l} + c_0 \cdot \mathcal{L}_{Q^g} + c_1 \cdot \mathcal{L}_h \tag{2}$$

$$\mathcal{L}_{Q^l} = \mathop{\mathbb{E}}_{\mathcal{D}'}[(Q^l(z_t, z^g, a_t) - (r_t + \gamma Q^l(z_{t+1}, z^g, \pi^l(z_{t+1}, z^g))))^2] \tag{3}$$

$$\mathcal{L}_{Q^g} = \mathop{\mathbb{E}}_{\mathcal{D}}[(Q^g(z_t, a_t) - (r_t + \gamma Q^g(z_{t+1}, \pi^g(z_{t+1}))))^2] \tag{4}$$

In accordance to the standard TD3-BC training objective, we simultaneously optimize the corresponding policies π^l and π^g in $\mathcal{L}_{\pi^l}$ and $\mathcal{L}_{\pi^g}$ (App. F, Eq. 6). Note that this step is done by alternating between optimizing $\mathcal{L}_{\text{model}}$ and policy improvement while the encoder parameters are not optimized during the policy update [1].

To provide data for training π^l and Q^l, we synthesize a set of state-reaching experiences $\mathcal{D}'$ by relabeling the transitions in $\mathcal{D}$. More specifically, we employ hindsight goal relabeling similar to [6, 27] to sample goals $z^g \in \mathcal{Z}$ and use a binary reward to indicate success (detailed information on relabeling strategy in App. E). For training the dynamics model, we choose a contrastive loss objective (CPC) [32] similar to [13]. In practice, we found this approach to work better in maintaining accurate long-term predictions compared to a standard mean-squared error objective.

d) Exploration strategy: We formulate our RL decision-making task as a geometric planning problem and seek a planner that efficiently explores the latent space searching for high-valued states. Similar to [13], we follow the concept of ESTs [17] and iteratively expand the current set of nodes through action sampling. The tree $\mathcal{T}=(\mathcal{V}, \mathcal{E})$ can be seen as a growing memory of latent nodes $\mathcal{V} \subset \mathcal{Z}$ and transitions $\mathcal{E} \subset \mathcal{Z} \times \mathcal{Z}$. The core mechanism behind our expansion strategy is summarized in Alg. 1. We first initialize $\mathcal{T} = (\mathcal{V}=\{z_{\text{init}}\}, \mathcal{E}=\emptyset)$ where $z_{\text{init}} \in \mathcal{Z}$ is the latent encoding of the current state $s_{\text{init}} \in \mathcal{S}$ obtained from ϕ. For n_{iter} steps, a node z_{expand} is drawn using a categorical distribution P_{node} defined over $\mathcal{V}$. Starting from z_{expand}, the dynamics h rolls out a short n_{sim}-step state sequence given actions drawn from our generative model g (or π^g). Since Q^l_k estimates the return for trying to reach a node under sparse rewards, a temporal distance proxy is given by $\log_\gamma Q^l_k$. To account for value approximation errors [12], we will use the minimum value among the ensembles predictions to obtain a conservative distance estimate. After every n_{sim}-step expansion with h, we determine if the transition from z_{exp} to z_{new} is feasible by checking if $Q^{\text{exp,new}}_{\text{min}}$ is above a threshold $\tau^{\text{low}}_{\text{discard}}$. If it lies below this threshold, we discard z_{new}. Secondly, we also reject it if the corresponding value of $Q^{\text{exp,new}}_{\text{std}}$ is above a threshold $\tau^{\text{std}}_{\text{discard}}$. The purpose of this second rejection step is filter states in which the epistemic uncertainty, i.e. model uncertainty, is high and thereby avoid the evaluation of high-uncertainty areas, for example outside the support of the latent data distribution. Lastly, we also want to determine if the newly generated node is sufficiently novel from the existing ones $\mathcal{T}$ and discard it otherwise. We found this step to be necessary to keep computation at a moderate level by sparsifying the tree. For that, we discard z_{new} if $\max\{Q^{i,\text{new}}_{\text{min}} | z_i \in \mathcal{V}\}$ is above a threshold $\tau^{\text{high}}_{\text{discard}}$. In other words, we find the closest neighbor z_{neigh} in the tree and reject z_{new} if there already exists a node which can transition to it within few steps. If z_{new} passes the previous stages, it is added to $\mathcal{T}$, i.e. $\mathcal{V} \leftarrow \mathcal{V} \cup \{z_{\text{new}}\}$ and $\mathcal{E} \leftarrow \mathcal{E} \cup \{z_{\text{exp}\to\text{new}}\}$.

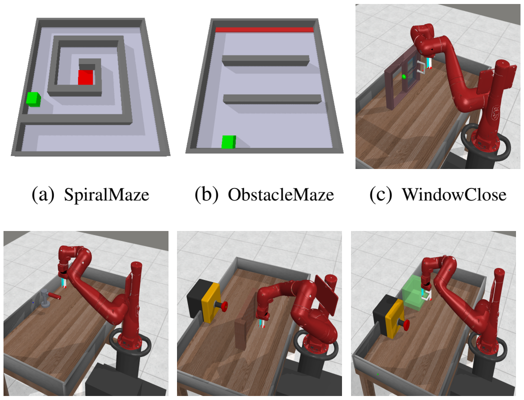
Algorithm 1 Node sampling and tree expansion

```
Given: z_init, n_iter, n_sim, τ^neigh_discard, τ^std_discard g, h, π^g, Q^g, Q^l_k, π^l
Initialize: V ← {z_init}, E ← ∅
for n_iter steps do
    Sample node z_exp from V given P_node(V)
    z_new ← z_exp
    Simulate forward using dynamics for n_sim steps
    for n_sim steps do
        Sample action a∼g(.|z_new) (or a=π^g(z_new))
        z_new ← h(z_new, a)
    end for
    Reject node if too close to existing one in the tree, too far
    from expansion node or if the value uncertainty is too high
    if Q^exp,new_min > τ^low_discard and Q^exp,new_std < τ^std_discard then
        if max{Q^i,new_min | z_i ∈ V} < τ^high_discard then
            Add new node to tree
            V←V ∪ {z_new}; E←E ∪ {z_exp→new}
        end if
    end if
end for
```

e) Node sampling heuristics: To achieve fast and task-oriented exploration, we combine two sampling heuristics based on (a) the inverse number of neighbors around each node and (b) the state-action value of Q^g. (a) leads to quick exploration of undiscovered latent states, while (b) drives the planner towards high-rewarding states. For both parts, we use exponential weighting as shown in Eq. 7 and 8 (App. G). In this regard, n^{neigh}_i corresponds the number of incoming neighbors for a node ($\mathcal{V}^{\text{neigh}}_{\to\text{i}}$). We compose P_{node} by sampling according to P_{sparse} with probability p^{sparse} and from P_{value} with p^{value} (otherwise random uniform).

f) Action sampling: Our action-generative model g mimics the state-dependent action distribution in the data and is learned using a standard conditional VAE [21]. Sampling

[1] We optimize the state representation during the critic update instead of the policy improvement step as motivated by the empirical analysis in [42].



(a) SpiralMaze (b) ObstacleMaze (c) WindowClose

(d) FaucetClose (e) ButtonWall (f) DrawerButton

Fig. 2. Visualizations of evaluation control environments. The tasks (c) - (f) presents adaptations from the meta-world benchmark [43] for which we render images using a static camera.

actions from g (e.g. instead of uniformly from $\mathcal{A}$) avoids the evaluation of undesired state-actions pairs for which our models have not seen any data. To help our planner discover task-relevant areas quicker, we also sample with probability p^{policy} actions from π^g.

g) Planning Objective and Control: We presented a planner which builds a sparse tree representation in the latent space while being guided by value and sparsity heuristics. To pick the best path, we must define an objective that ranks all paths $\mathcal{T}$. In practice, we first identify G, the set of trajectories in $\mathcal{T}$ reaching the goal which we determine by checking the leaf node values predicted by Q^g against a threshold τ_{goal}. Among the elements in G, we then choose the path g^* which is associated with the minimal path length computed based on using Q^l as a distance proxy between subsequent states (Eq. 5). If $G = \emptyset$, we simply pick we one in $\mathcal{T}$ that leads to the highest-valued state (based on Q^g).

$$g^* = \arg\min_{g \in \mathcal{G}} c(g) \qquad \text{with} \qquad c(g) = \sum_{(i,j) \in \mathcal{E}_g} \log_\gamma Q^{i,j}_{\text{min}} \tag{5}$$

To use our planner in a control setting, we embed it into a model-predictive control loop. The controller queries our planner every n_{replan} steps and locally uses the local policy π^l to steer between nodes in the planned sequences of latent states. If close enough, the controller switches to the next waypoint, which we determine by checking the value of Q^l.

IV. Experiments

a) Environments: For our quantitative evaluation, we consider the simulated vision-based control tasks shown in Fig. 2. A detailed description of the evaluation environments in App. G. Our training data $\mathcal{D}$ was collected using a combination of random actions and a small number of noisy expert demonstrations. For all environment, we form states by concatenating three consecutive image frames of resolution 64x64. We use a latent space of dimension 32 for all experiments. Further information on the training procedure, baselines and data collection can be found in the appendix.

TABLE I
SUCCESS RATES (%) ON TEST SCENARIOS.

Method	BC	BC ($\mathcal{D}^*$)	TD3-BC	MPPI	MBOP	IRIS	IRIS (multi-step)	VELAP
Spiral Maze	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	15 ± 31	$\mathbf{94 \pm 3}$
Obstacle Maze	0 ± 0	15 ± 6	35 ± 22	83 ± 11	40 ± 25	50 ± 25	62 ± 14	$\mathbf{97 \pm 2}$
Window	0 ± 0	34 ± 11	16 ± 8	70 ± 7	23 ± 4	69 ± 3	43 ± 20	$\mathbf{78 \pm 4}$
Faucet	0 ± 0	36 ± 6	13 ± 7	41 ± 7	33 ± 2	10 ± 2	3 ± 1	$\mathbf{51 \pm 12}$
ButtonWall	0 ± 0	0 ± 0	2 ± 2	9 ± 10	0 ± 0	35 ± 5	8 ± 8	$\mathbf{76 \pm 9}$
DrawerButton	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	5 ± 3	0 ± 0	$\mathbf{11 \pm 3}$

b) Experimental Results: The results for the quantitative evaluation are presented in Table I. As shown, `VELAP` consistently outperform the baselines across all environment in terms of average episode success rate. Interestingly, the improvements due to our method are particularly visible in tasks which require far-reaching planning such as the *SpiralMaze* environment and *ButtonWall*. These results support that our tree-based memory and expansion strategy is indeed effective at improving upon learned model-free offline RL policies in sparse-reward settings. To further support that our method is able to compute feasible latent paths over many time steps, we illustrate a planned solution path for the *SpiralMaze* task in Fig. 3 (App. L). It can be seen that the path corresponds to a global solution which covers the entire space reaching the far-distant goal region. Fig. 4 (App. L) presents similar visualizations for the *ObstacleEnv* environment. As the figure suggests, our state embedding correctly identifies the positions of the obstacles and enables our planner to find a feasible path towards the goal region. Further visualization are provided in App. L.

V. Discussion

a) Limitations: Our method provides the basis that allows interesting future extensions. Firstly, `VELAP` is currently implemented for the offline RL scenario. Similar to e.g. [14, 33], it could be adapted to the online setting by interleaving online data collecting and model learning. Moreover, the planning could be integrated into the RL training to provide better updates for policy and critic [37, 15, 34]. At the moment, our method is geared towards fully-observable environments. A promising future direction could be to solve partially-observable MDPs by planning in belief spaces [20]. Decision-making could be improved in that way by considering the dynamics and perceptual uncertainty which is propagated along the predicted states. Another exciting direction could be the integration of language-specified goal as recently been done e.g. in [29] [35].

VI. Conclusion

We present `VELAP`, a model-based planning agent for tasks with sparse rewards from offline data. Unlike most existing planning tools currently used in model-based RL, we propose a novel tree-based search algorithm similar to the type of sampling-based planners used in robot motion planning. An empirical comparison, which included high-dimensional robot manipulation tasks, demonstrated significant improvements of our method over the state-of-the-art. We hope that our results will stimulate further research on the integration of classical planning tools and data-driven approaches.

REFERENCES

[1] Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. In *Neural Information Processing Systems*, 2021.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[3] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=OMNB1G5xzd4.

[4] Joshua Bialkowski, Sertac Karaman, and Emilio Frazzoli. Massively parallelizing the rrt and the rrt. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3513–3518, 2011. doi: 10.1109/IROS.2011.6095053.

[5] Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Pierre L'Ecuyer. Chapter 3 - the cross-entropy method for optimization. In C.R. Rao and Venu Govindaraju, editors, *Handbook of Statistics*, volume 31 of *Handbook of Statistics*, pages 35–59. Elsevier, 2013. doi: https://doi.org/10.1016/B978-0-444-53859-8.00003-5. URL https://www.sciencedirect.com/science/article/pii/B9780444538598000035.

[6] Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jacob Varley, Alex Irpan, Benjamin Eysenbach, Ryan C Julian, Chelsea Finn, and Sergey Levine. Actionable models: Unsupervised offline reinforcement learning of robotic skills. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1518–1528. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/chebotar21a.html.

[7] Todor Davchev, Oleg Olegovich Sushkov, Jean-Baptiste Regli, Stefan Schaal, Yusuf Aytar, Markus Wulfmeier, and Jon Scholz. Wish you were here: Hindsight goal selection for long-horizon dexterous manipulation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=FKp8-pIRo3y.

[8] Ben Eysenbach, Xinyang Geng, Sergey Levine, and Russ R Salakhutdinov. Rewriting history with inverse rl: Hindsight inference for policy improvement. *Advances in neural information processing systems*, 33:14783–14795, 2020.

[9] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=vGQiU5sqUe3.

[10] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[11] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[12] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

[13] Robert Gieselmann and Florian T. Pokorny. Latent planning via expansive tree search. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=zSdz5scsnzU.

[14] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565, 09–15 Jun 2019.

[15] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

[16] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. In *ICML*, 2022.

[17] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2719–2726 vol.3, 1997. doi: 10.1109/ROBOT.1997.619371.

[18] Brian Ichter. *Massive Parallelism and Sampling Strategies for Robust and Real-Time Robotic Motion Planning*. PhD thesis, Stanford University, 2018.

[19] Brian Ichter, Pierre Sermanet, and Corey Lynch. Broadly-exploring, local-policy trees for long-horizon task planning. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=yhy25u-DrjR.

[20] V. Indelman, L. Carlone, and F. Dellaert. Towards planning in generalized belief space. In *International Symposium on Robotics Research (ISRR)*, December 2013.

[21] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[22] Nathan Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. Objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2002.04523*, 2020.

[23] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[24] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[25] Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryzECoAcY7.

[26] Alexander Li, Lerrel Pinto, and Pieter Abbeel. Generalized hindsight for reinforcement learning. *Advances in neural information processing systems*, 33:7754–7767, 2020.

[27] Jinning Li, Chen Tang, Masayoshi Tomizuka, and Wei Zhan. Hierarchical planning through goal-conditioned offline reinforcement learning. *IEEE Robotics and Automation Letters*, 7(4):10216–10223, 2022. doi: 10.1109/LRA.2022.3190100.

[28] Kara Liu, Thanard Kurutach, Christine Tung, Pieter Abbeel, and Aviv Tamar. Hallucinative topological memory for zero-shot visual planning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6259–6270. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/liu20h.html.

[29] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems*, 2021. URL https://arxiv.org/abs/2005.07648.

[30] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

[31] Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. Deep Dynamics Models for Learning Dexterous Manipulation. In *Conference on Robot Learning (CoRL)*, 2019.

[32] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[33] Oleh Rybkin, Chuning Zhu, Anusha Nagabandi, Kostas Daniilidis, Igor Mordatch, and Sergey Levine. Model-based reinforcement learning via latent-space collocation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9190–9201. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/rybkin21b.html.

[34] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, dec 2020. doi: 10.1038/s41586-020-03051-4. URL https://doi.org/10.1038%2Fs41586-020-03051-4.

[35] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

[36] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4732–4741. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/srinivas18b.html.

[37] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4): 160–163, jul 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL https://doi.org/10.1145/122344.122377.

[38] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[39] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500): 2319–2323, 2000. doi: 10.1126/science.290.5500.2319. URL https://www.science.org/doi/abs/10.1126/science.290.5500.2319.

[40] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[41] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, 2016. doi: 10.1109/ICRA.2016.7487277.

[42] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10674–10681, 2021.

[43] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Metaworld: 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.

Appendix

A. Model architectures

TABLE II
Hyperparameters of the encoder ϕ

Parameter	Value
Batch-norm.	yes
Filters	[32,32,64,64]
Kernels	[4,4,4,4]
Strides	[2,2,2,2]
Activation	LeakyRelu
Dense layers	[256, 128, 32]

TABLE III
Hyperparameters of dynamics model h

Parameter	Value
Batch-norm.	yes
Activation	LeakyRelu
Dense layers	[128,128,128,128,32]

TABLE IV
Hyperparameters of action sampler g (β-VAE)

Parameter	Value
Batch-norm.	yes
Activation	LeakyRelu
Latent dimension	16
β (kl-weight)	0.01
Encoder dense layers	[128,128,128, 2*16]
Decoder dense layers	[128,128,128, d_{action},]

TABLE V
Hyperparameters of policy networks π^l and π^g

Parameter	Value
Batch-norm.	yes
Activation	LeakyRelu
Dense layers	[128,128,128, d_{action}]

TABLE VI
Hyperparameters of critic networks Q^l and Q^g

Parameter	Value
Batch-norm.	yes
Activation	LeakyRelu
Dense layers	[128,128,128, 1]

B. Training hyperparameters

TABLE VII
HYPERPARAMETERS DURING TRAINING (MODEL LEARNING)

Parameter	Value
batch size	64
learning rate (all models)	0.0003
c_0	0.2
c_1 (*SpiralMaze*)	0.001
c_1 (*ObstacleMaze*)	1.0
c_1 (metaworld tasks)	0.01
c_2	0.001
c_3	0.001
c_3 (expert)	0.5
γ (discount factor)	0.96
$d_{\mathcal{Z}}$	32
T (temperature parameter)	1.0
n_{ens}	3

C. Planner and controller hyperparameters

TABLE VIII
HYPERPARAMETERS DURING INFERENCE (PLANNING)

Name	Description	Value
n_{iter}	Number of planner iterations	250 (500 in *ButtonWall*)
n_{sim}	Number of simulation steps during tree expansion	5 (10 in *SpiralMaze*, *ButtonWall* and *DrawerButton*)
$\tau_{\text{discard}}^{\text{high}}$	Q-value threshold for discarding node if too close to existing nodes in the tree	γ^2
$\tau_{\text{discard}}^{\text{low}}$	Q-value threshold for discarding node if too far from expansion node	$\gamma^{n_{\text{sim}}}$
$\tau_{\text{discard}}^{\text{std}}$	Q-value threshold for discarding node if standard deviation of ensemble prediction is too high	$1.0 - \gamma$
τ_{goal}	Q-value threshold to determine goal nodes	γ^1
d_{neigh}	Euclidean distance threshold to determine candidate neighbors	3 x upper 5-percentile of Eucl. distances between encoding of subsequent states

TABLE IX
HYPERPARAMETERS DURING INFERENCE (CONTROLLER)

Parameter	Description	Value
n_{replan}	Planning frequency	15 (25 in *SpiralMaze*, *ButtonWall*)
ϵ_{goal}	Q-threshold to determine vicinity to the goal	γ^5
ϵ_{wp}	Q-threshold for switching to the next waypoint	γ^3

D. Additional details about planning method

a) Neighbor computation: To determine if a newly sampled node z_{new} is novel, we check its similarity to existing nodes in the tree by evaluating the state-action value function. Yet, evaluating the value network for all nodes in the tree results in an enormous computational overhead. Yet, we can significantly reduce this computation by first determining a set of candidate neighbors around z_{new} using the Euclidean metric and a distance threshold d_{neigh}. In practice, we found it useful to define d_{neigh} based on the statistics of Euclidean distances between subsequent states in the dataset (see App. C).

b) Batch processing: The method in Alg. 1 describes an iterative schema for which at every expansion step one new node is generated and evaluated. Yet, some steps can be computed in parallel on a GPU in order to speed up the planning time. For a practical implementation, we therefore suggest to parallelize the tree expansion by sampling multiple expansion nodes at once and generating new nodes by passing batches through the neural network dynamics model. Similarly, we can compute state-action values in batches instead of assessing a single nodes at a time. For a discussion about parallelized implementations of classical RRT-like planners, we refer to [4, 18].

E. Training of policy and value functions

We use TD3-BC [10] as the base offline RL algorithm to train our local and goal policies π^l and π^g, respectively state-action value functions a Q^l_k and Q^g. Within our planning framework Q^l_k takes an important role as it provides us with a distance proxy. To improve the accuracy of the value estimates, we use k Q networks (instead of 2 usually used in TD3). During the training update of the Q-network, we then determine the Q-target by taking the minimum value among the predictions given by the ensemble of Q-networks (similar to [1]). The ensemble further allows us to filter out unlikely or out-of-distribution transitions generated during the tree expansion by thresholding the resulting Q-values based on the minimum predicted ensemble value and the standard deviation among the predicted values.

Our models π^l and Q^l_k describe goal-reaching navigation policy and state-action value functions which require a set of goal-conditioned reaching experiences for training. Since our original dataset $\mathcal{D}$ might not provide such data, we augment it using data augmentation via hindsight relabeling. In particular, we create a new dataset $\mathcal{D}'$ creating transitions (z_t, a_t, r_t, z_{t+1}, z^g, γ) $\in \mathcal{D}'$ based on the existing transitions in $\mathcal{D}$ by relabeling the values of r_t, γ (γ also indicates terminal condition, i.e. $\gamma = 0$) and adding a new goal state z^g. In this regard, we apply a combination of three different relabeling strategies (a) set goal z^g to be next state of the original transitions and set $\gamma = 0$ and $r_t = 1$ (b) sample z^g from the set of future states within the same trajectory and set $r_t = 0$ (c) sample z^g from another trajectory in the data and $r_t = 0$.

F. RL policies training objectives

$$
\begin{aligned}
\mathcal{L}_{\pi^l} &= \mathop{\mathbb{E}}_{\mathcal{D}'}[-Q^l(z_t, z^g, \pi^l(z_t, z^g))] + c_2 \cdot \mathop{\mathbb{E}}_{\mathcal{D}'}[(\pi^l(z_t, z^g) - a_t)^2] \\
\mathcal{L}_{\pi^g} &= \mathop{\mathbb{E}}_{\mathcal{D}}[-Q^g(z_t, \pi^g(z_t))] + c_3 \cdot \mathop{\mathbb{E}}_{\mathcal{D}}[(\pi^g(z_t) - a_t)^2]
\end{aligned} \tag{6}
$$

G. Node sampling heuristics

$$
P_{\text{sparse}}(z_i) = \frac{e^{-n_i^{\text{neigh}}/T_{\text{sparsity}}}}{\sum_{z_j \in \mathcal{V}} e^{-n_j^{\text{neigh}}/T_{\text{sparsity}}}} \tag{7}
$$

$$
P_{\text{value}}(z_i) = \frac{e^{Q_i^g/T_{\text{value}}}}{\sum_{z_j \in \mathcal{V}} e^{Q_j^g/T_{\text{value}}}} \tag{8}
$$

H. Description of block environments

Similar to the evaluation environments in [13], we implement two long-horizon navigation tasks whose underlying state space is relatively low-dimensional in order to facilitate visual inspection of learned latent representations using dimensionality reduction techniques (e.g. Isomap [39]). In both environments, a block robot is controlled using velocity commands while its movements are limited to a planar surface.

1) SpiralMaze: To solve this task, the block agent must navigate form the outer end of the spiral-shaped corridor to the inner region (colored in red; see Fig. 2). The maximum allowed number of episode steps is limited to 300. To generate training data, the agent is placed randomly at a collision free position in the workspace and random actions sequences are applied by subsequently adding Gaussian noise to a randomly sampled initial action. For testing, the agent's position is sampled uniformly within a small region close to the outer end of the spiral-shaped corridor.

2) ObstacleMaze: In this environment, the agent must navigate towards the upper wall of the workspace (color in red; see Fig. 2). To achieve this goal, the agent must take actions around two obstacles which are randomly placed within the center of the workspace at the beginning of each new episode. The maximum allowed number of environment steps is set to 100. For testing, the agent is initialized at a random configuration close to the wall which is on the opposite side of the goal. We used the same random data collection policy as for the *SpiralMaze* task.

I. Description of manipulation environments

We adapted and implemented several robot manipulation environments based on the Metaworld [43] robot benchmark tasks. The underlying physics simulator in this regard is Mujoco [40]. To enable visual manipulation, similar to the problems studied in [33], we render RGB images from a static viewpoint. The robot is controlled by commanding desired endeffector and gripper opening displacements resulting in a 4-dimensional action space. *WindowClose* and *FaucetClose* were with small adaptions modified from [43]. Moreover, we evaluate two new scenarios *ButtonWall* and *DrawerButton* which are specifically designed to evaluate our method in extremely sparse reward conditions and over a lengthy temporal horizon. These scenarios necessitate the use of trajectory "stitching" techniques to discover a solution policy.

For data collection, we implemented a suboptimal policy that takes random actions (additive Gaussian noise) most of the time and with a low probability takes an action generated by a scripted expert policy. Table. X shows the number of transition samples

TABLE X
COMPOSITION OF TRAINING DATASETS FOR EACH ENVIRONMENT

Environment	Num. contexts	Traj. per context	Max. traj. length	Successful transitions
SpiralMaze	1	1000	20	0.12 %
ObstacleMaze	250	20	20	0.11 %
WindowClose	200	10	50	0.48 %
ButtonWall	200	10	50	0.16 %
FaucetClose	200	10	50	0.31 %
DrawerButton	150	20	50	0.16 %

and trajectories in the training data and further presents the portion of successful actions (reward=1). For all manipulation tasks, we set the maximum permitted environmental steps at 150, with the exception of the "ButtonWall" scenario, where we allow up to 250 steps during the evaluation phase.

1) WindowClose: In order to accomplish this task, the robotic arm must successfully open a window by shifting a specific handle sideways. We implement environmental variability by randomly determining the x-y location of the window object in each episode. During the data collection stage, we randomly position the end-effector above the surface of the table. However, we restrict the sampling of expert actions to areas close to the handle. This approach is intended to guarantee that the strategy employed necessitates to "stitch" different trajectories together to reach the objective and complete the task when starting from states that are farther away. To ensure challenging planning situations during testing, we initiate the robot at a significant distance away from the target.

2) FaucetClose: This task is similar to the *WindowClose* task, but it requires the agent to use its end-effector to close a faucet instead. In addition, we employ analogous strategies for data gathering and scenario creation as those used in the *WindowClose* environment.

3) ButtonWall: In this particular scenario, the robot's end-effector is tasked with maneuvering around a wall structure before reaching a button to press. The exact position of the wall is randomized at the beginning of each episode. Additionally, there is a height constraint imposed on the end-effector to ensure that the agent follows a longer path around the wall instead of simply raising the end-effector. The dataset was generated by placing the agent either in front of the wall, near the button, or far behind the wall. However, expert demonstrations in the dataset are only available for scenarios where the agent starts in proximity to the goal. To enhance the complexity of the planning task during testing, the end-effector is sampled within an area located behind the wall.

4) DrawerButton: In this scenario, the agent is tasked to first close a drawer using its end-effector and then press a button. To train the agent, we develop a dataset by separately collecting trajectories for each subtask. Again, this approach necessitates a method capable of combining different trajectories in the data to devise a solution that achieves the overall task goal.

J. Composition of training dataset

The table below presents the composition of our training datasets. Each context in this regards, refers to a new environment initialization (excl. agent) such as the position of obstacles.

K. Baselines

To assess the effectiveness of `VELAP`, we compare it with existing offline RL methods and consider the following baselines. **Behavioral cloning** (**BC**): A simple yet often effective baseline which is trained to imitate the behavioral policy π_β using a supervised learning objective. We also assessed a variant of this method using a subset of only successful trajectories ($\mathcal{D}^*$). **TD3-BC** [10]: An adaptation of the Twin Delayed DDPG algorithm [11] which mitigates the negative effects of value overestimation by adding an imitation objective to the policy update. **MPPI**: A sampling-based trajectory optimization method provides the base planning algorithm in various state-of-the-art model-based RL methods (e.g. [31, 16]). We consider a variant of this method for the offline learning setup which uses TD3-BC within the cost update during optimization. **MBOP** [3]: A model-based planning method which uses an adaptation of MPPI to optimize paths particularly in the offline RL setting. **IRIS**: An offline RL method particularly designed for sparse reward settings. In essence, it uses a hierarchical decomposition of the policy for which a manager predicts feasible subgoals given future candidate states (n-step horizon) sampled from a generative model (cVAE) which a worker policy must achieve. We also examined an adaptation, which we call **IRIS (multi-step)**, where the set of potential subgoals is generated by randomly shooting future state sequences given the state prediction model. To establish a fair comparison and disentangle the effects of the representation and planner, we use the same base representations and dynamics models across all methods.

L. Visualizations

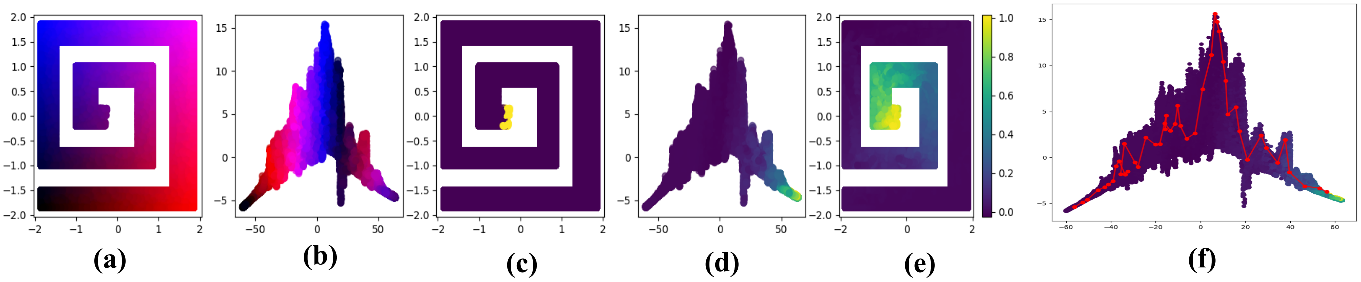


Fig. 3. Latent space visualizations for *SpiralMaze* (a) xy coordinates of block robot (b) Isomap embeddings of latent spaces (c) environment reward for xy-coordinates (d) predicted Q-values for latent space (e) predicted Q-values for xy-coordinates (f) planned latent path computed using `VELAP`

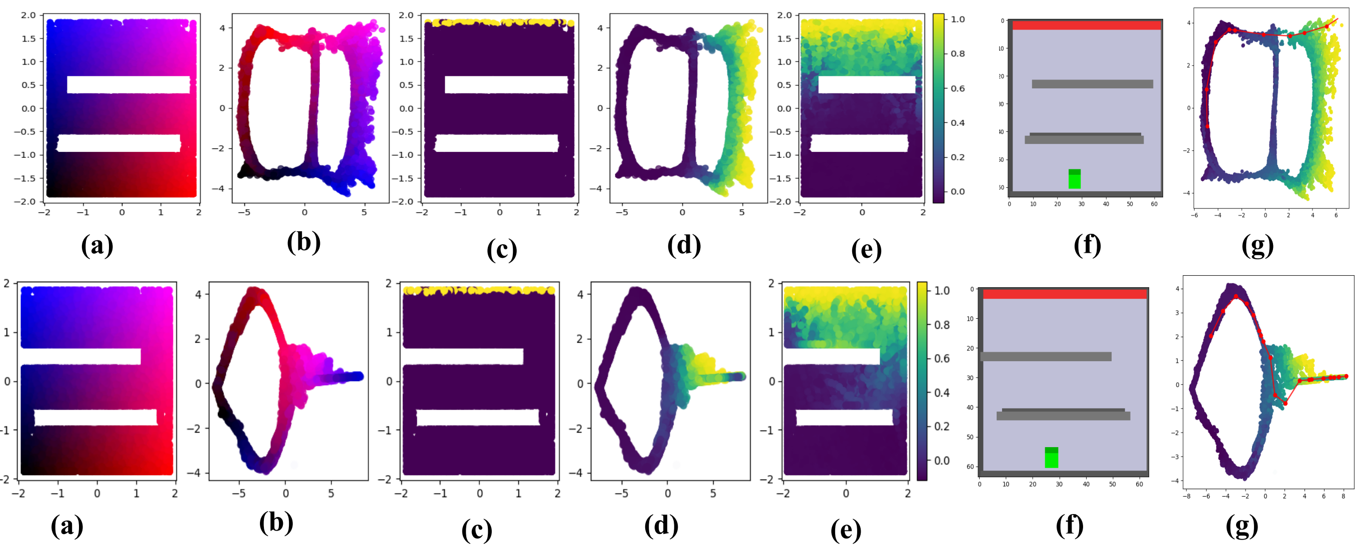


Fig. 4. Latent space visualizations for *ObstacleMaze* tasks (a) xy coordinates of block robot (b) Isomap embeddings of latent spaces (c) environment reward for xy-coordinates (d) predicted Q-values for latent space (e) predicted Q-values for xy-coordinates (f) example visual inputs for different contexts (g) corresponding latent path computed using `VELAP`

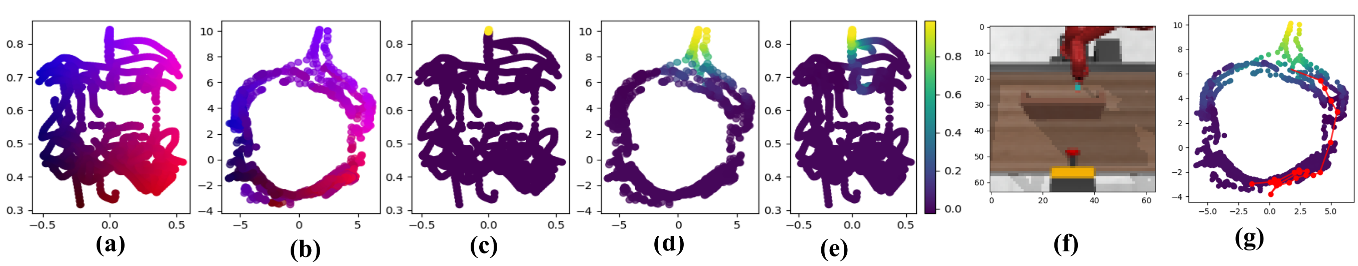


Fig. 5. Latent space visualizations for *ButtonWall* tasks (a) xy coordinates of block robot (b) Isomap embeddings of latent spaces (c) environment reward for xy-coordinates (d) predicted Q-values for latent space (e) predicted Q-values for xy-coordinates (f) example visual inputs for different contexts (g) corresponding latent path computed using `VELAP`