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 [35,15], and more recently, trajectory collocation using second-order solvers [83]. 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.

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 $\gamma$ 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 $R[^{t}]$ over all trajectories $t$ given an initial state distribution $p_0$ and induced by $\pi$, i.e., to optimize $E_\pi[R[^{t}]]$. The problem of offline RL [24] arises when training from a fixed dataset $\mathcal{D}$ consisting of trajectories generated by a behavioral policy $\pi_\beta$. 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] [20] 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] [22]. 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].

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 $\phi$ 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 $\pi_l$ is trained to navigate between neighboring states in the tree. The global policy $\pi_g$ 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 $n_{\text{ens}}$ 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 $Q_{\text{min}}^{i,j} := \min_{k=1}^{n_{\text{ens}}}(Q_k^l(z_i,z_j,\pi^l(z_i,z_j)))$ as the minimum and $Q_{\text{std}}^{i,j} := \text{std}(Q_k^l(z_i,z_j,\pi^l(z_i,z_j)))_{k=1}^{n_{\text{ens}}}$ 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 \( \pi^g \) and \( \pi^h \). 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 \( \phi \) through joint optimization with the latent dynamics, local and global RL policies, leading to the overall training objective \( L_{\text{model}} \) (Eq. 2). In this regard, \( L_{Q^l} \) describes the temporal difference (TD) value loss for training \( Q^l \), \( L_{Q^g} \) the TD loss for training \( Q^g \) and \( L_h \) the loss function for the dynamics model \( h \).

\[
L_{\text{model}} = L_{Q^l} + c_0 \cdot L_{Q^g} + c_1 \cdot L_h
\]

\[
L_{Q^l} = \frac{E}{|D|}[Q^l(z_t, z^g, a_t) - (r_t + \gamma Q^l(z_{t+1}, z^g, \pi^l(z_{t+1}, z^g)))]^2
\]

\[
L_{Q^g} = \frac{E}{|D|}[Q^g(z_t, a_t) - (r_t + \gamma Q^g(z_{t+1}, \pi^g(z_{t+1})))]^2
\]

In accordance to the standard TD3-BC training objective, we simultaneously optimize the corresponding policies \( \pi^l \) and \( \pi^g \) in \( L_{\pi^l} \) and \( L_{\pi^g} \) (App. 5 Eq. 6). Note that this step is done by alternating between optimizing \( L_{\text{model}} \) and policy improvement while the encoder parameters are not optimized during the policy update \(^1\).

To provide data for training \( \pi^l \) and \( Q^l \), we synthesize a set of state-reaching experiences \( D' \) by relabeling the transitions in \( D \). More specifically, we employ hindsight goal relabeling similar to \(^6, 27\) to sample goals \( z^g \in Z \) and use a binary reward to indicate success (detailed information on relabeling strategy in App. 3). For training the dynamics model, we choose a contrastive loss objective (CPC) \(^{12}\) 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 \( T = (V, E) \) can be seen as a growing memory of latent nodes \( V \subset Z \) and transitions \( E \subset Z \times Z \). The core mechanism behind our expansion strategy is summarized in Alg. 1. We first initialize \( T = (V = \{z_{\text{init}}\}, E = \emptyset) \) where \( z_{\text{init}} \in Z \) is the latent encoding of the current state \( s_{\text{init}} \in S \) obtained from \( \phi \). For \( n_{\text{iter}} \) steps, a node \( z_{\text{expand}} \) is drawn using a categorical distribution \( P_{\text{node}} \) defined over \( V \). Starting from \( z_{\text{expand}} \), the dynamics \( h \) rolls out a short \( n_{\text{sim}} \)-step state sequence given actions drawn from our generative model \( g \) (or \( \pi^g \)). Since \( Q^l_k \) estimates the return for trying to reach a node under sparse rewards, a temporal distance proxy is given by \( \log_+, Q^l_k \). 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 \( n_{\text{sim}} \)-step expansion with \( h \), we determine if the transition from \( z_{\text{expand}} \) to \( z_{\text{new}} \) is feasible by checking if \( Q^l_{\text{expand,new}} \) is above a threshold \( \tau^\text{low}_\text{discard} \). If it lies below this threshold, we discard \( z_{\text{new}} \). Secondly, we also reject it if the corresponding value of \( Q^\text{std} \) is above a threshold \( \tau^\text{std}_\text{discard} \). The purpose of this second rejection step is to 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 \( T \) 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 \( z_{\text{new}} \) if \( \max\{Q^\text{new}_k \mid z_i \in V\} \) is above a threshold \( \tau^\text{high}_\text{discard} \). In other words, we find the closest neighbor \( z_{\text{neigh}} \) in the tree and reject \( z_{\text{new}} \) if there already exists a node which can transition to it within few steps. If \( z_{\text{new}} \) passes the previous stages, it is added to \( T \), i.e. \( V \leftarrow V \cup \{z_{\text{new}}\} \) and \( E \leftarrow E \cup \{z_{\text{expand,new}}\} \).

**Algorithm 1 Node sampling and tree expansion**

1. Given: \( z_{\text{init}}, n_{\text{iter}}, n_{\text{sim}}, n_{\text{neigh}}^\text{expand}, \tau^\text{low}_\text{discard}, \tau^\text{high}_\text{discard}, h, \pi^g, Q^l, \pi^h, \pi^l\)
2. Initialize: \( V \leftarrow \{z_{\text{init}}\}, E = \emptyset \)
3. for \( n_{\text{iter}} \) steps do
4. Sample node \( z_{\text{expand}} \) from \( V \) given \( P_{\text{node}}(V) \)
5. \( z_{\text{new}} \leftarrow z_{\text{expand}} \)
6. Simulate forward using dynamics for \( n_{\text{sim}} \) steps
7. for \( n_{\text{sim}} \) steps do
8. Sample action \( a \sim g(\cdot, z_{\text{new}}) \) (or \( a = \pi^g(\cdot, z_{\text{new}}) \))
9. \( z_{\text{new}} \leftarrow h(z_{\text{new}}, a) \)
10. for end
11. Reject node if too close to existing one in the tree, too far from expansion node or if the value uncertainty is too high
12. if \( Q^l_{\text{expand,new}} > \tau^\text{low}_\text{discard} \) and \( Q^\text{std}_{\text{new}} < \tau^\text{std}_\text{discard} \) then
13. if \( \max\{Q^l_k \mid z_i \in V\} < \tau^\text{high}_\text{discard} \) then
14. Add new node to tree
15. \( V \leftarrow V \cup \{z_{\text{new}}\} \)
16. \( E \leftarrow E \cup \{z_{\text{expand,new}}\} \)
17. for end
18. end if
19. for end

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. C). In this regard, \( n_{\text{neigh}} \) corresponds the number of incoming neighbors for a node \( \langle V^{\text{neigh}} \rangle \). We compose \( P_{\text{node}} \) by sampling according to \( P_{\text{spare}} \) with probability \( p_{\text{spare}} \) and from \( P_{\text{value}} \) with \( p_{\text{value}} \) (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
actions from \( g \) (e.g. instead of uniformly from \( A \)) 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 \( p_{\text{policy}} \) actions from \( \pi^g \).

\( g \) 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 \( T \). In practice, we first identify \( G \), the set of trajectories in \( T \) reaching the goal which we determine by checking the leaf node values predicted by \( Q^g \) against a threshold \( \tau_{\text{goal}} \). 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 one in \( T \) that leads to the highest-valued state (based on \( Q^g \)).

\[
g^* = \arg \min_{g \in G} c(g) \quad \text{with} \quad c(g) = \sum_{(s,j) \in E_g} \log \tau_{Q_{\min}^l} \tag{5}
\]

To use our planner in a control setting, we embed it into a model-predictive control loop. The controller queries our planner every \( n_{\text{replan}} \) steps and locally uses the local policy \( \pi^l \) 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. [C] Our training data \( D \) 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.

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. 3 (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, 53]. 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


APPENDIX

A. Model architectures

<table>
<thead>
<tr>
<th>TABLE II</th>
<th>HYPERPARAMETERS OF THE ENCODER $\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>Batch-norm.</td>
<td>yes</td>
</tr>
<tr>
<td>Filters</td>
<td>[32,32,64,64]</td>
</tr>
<tr>
<td>Kernels</td>
<td>[4,4,4,4]</td>
</tr>
<tr>
<td>Strides</td>
<td>[2,2,2,2]</td>
</tr>
<tr>
<td>Activation</td>
<td>LeakyRelu</td>
</tr>
<tr>
<td>Dense layers</td>
<td>[256, 128, 32]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE III</th>
<th>HYPERPARAMETERS OF DYNAMICS MODEL $h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>Batch-norm.</td>
<td>yes</td>
</tr>
<tr>
<td>Activation</td>
<td>LeakyRelu</td>
</tr>
<tr>
<td>Dense layers</td>
<td>[128,128,128,128,32]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE IV</th>
<th>HYPERPARAMETERS OF ACTION SAMPLER $g$ ($\beta$-VAE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>Batch-norm.</td>
<td>yes</td>
</tr>
<tr>
<td>Activation</td>
<td>LeakyRelu</td>
</tr>
<tr>
<td>Latent dimension</td>
<td>16</td>
</tr>
<tr>
<td>$\beta$ (kl-weight)</td>
<td>0.01</td>
</tr>
<tr>
<td>Encoder dense layers</td>
<td>[128,128,128, 2*16]</td>
</tr>
<tr>
<td>Decoder dense layers</td>
<td>[128,128,128, $d_{\text{action}}$]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE V</th>
<th>HYPERPARAMETERS OF POLICY NETWORKS $\pi^l$ AND $\pi^g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>Batch-norm.</td>
<td>yes</td>
</tr>
<tr>
<td>Activation</td>
<td>LeakyRelu</td>
</tr>
<tr>
<td>Dense layers</td>
<td>[128,128,128, $d_{\text{action}}$]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE VI</th>
<th>HYPERPARAMETERS OF CRITIC NETWORKS $Q^l$ AND $Q^g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>Batch-norm.</td>
<td>yes</td>
</tr>
<tr>
<td>Activation</td>
<td>LeakyRelu</td>
</tr>
<tr>
<td>Dense layers</td>
<td>[128,128,128, 1]</td>
</tr>
</tbody>
</table>
B. Training hyperparameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch size</td>
<td>64</td>
</tr>
<tr>
<td>learning rate (all models)</td>
<td>0.0003</td>
</tr>
<tr>
<td>$c_0$</td>
<td>0.2</td>
</tr>
<tr>
<td>$c_1$ (SpiralMaze)</td>
<td>0.001</td>
</tr>
<tr>
<td>$c_1$ (ObstacleMaze)</td>
<td>1.0</td>
</tr>
<tr>
<td>$c_1$ (metaworld tasks)</td>
<td>0.01</td>
</tr>
<tr>
<td>$c_2$</td>
<td>0.001</td>
</tr>
<tr>
<td>$c_3$</td>
<td>0.001</td>
</tr>
<tr>
<td>$c_3$ (expert)</td>
<td>0.5</td>
</tr>
<tr>
<td>$\gamma$ (discount factor)</td>
<td>0.96</td>
</tr>
<tr>
<td>$d_z$</td>
<td>32</td>
</tr>
<tr>
<td>$T$ (temperature parameter)</td>
<td>1.0</td>
</tr>
<tr>
<td>$n_{ens}$</td>
<td>3</td>
</tr>
</tbody>
</table>

C. Planner and controller hyperparameters

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{replan}$</td>
<td>Planning frequency</td>
<td>15 (25 in SpiralMaze, ButtonWall)</td>
</tr>
<tr>
<td>$\epsilon_{goal}$</td>
<td>Q-threshold to determine vicinity to the goal</td>
<td>$\gamma^2$</td>
</tr>
<tr>
<td>$\epsilon_{wp}$</td>
<td>Q-threshold for switching to the next waypoint</td>
<td>$\gamma^3$</td>
</tr>
</tbody>
</table>

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 node 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 \(\pi^l\) and \(\pi^g\), respectively state-action value functions \(Q^l_k\) and \(Q^g_k\). 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 \([11]\)). 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 \(\pi^l\) 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 \(D\) might not provide such data, we augment it using data augmentation via hindsight relabeling. In particular, we create a new dataset \(D'\) creating transitions \((z_t, a_t, r_t, z_{t+1}, z^g, \gamma) \in D'\) based on the existing transitions in \(D\) by relabeling the values of \(r_t\), \(\gamma\) (\(\gamma\) 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

\[
L_{\pi^l} = \mathbb{E}_{D'}[-Q^l(z_t, z^g, \pi^l(z_t, z^g))] + c_2 \cdot \mathbb{E}_{D'}[(\pi^l(z_t, z^g) - a_t)^2]
\]

\[
L_{\pi^g} = \mathbb{E}_{D'}[-Q^g(z_t, \pi^g(z_t))] + c_3 \cdot \mathbb{E}_{D'}[(\pi^g(z_t) - a_t)^2]
\]

G. Node sampling heuristics

\[
P_{\text{sparse}}(z_i) = \frac{e^{-n_{z_i}^{\text{sparsity}}/T_{\text{sparsity}}}}{\sum_{z_j \in V} e^{-n_{z_j}^{\text{sparsity}}/T_{\text{sparsity}}}}
\]

\[
P_{\text{value}}(z_i) = \frac{e^{Q^g_i/T_{\text{value}}}}{\sum_{z_j \in V} e^{Q^g_j/T_{\text{value}}}}
\]

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 from the outer end of the spiral-shaped corridor to the inner region (colored in red; see Fig. 4). 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. 4). 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 adaptations 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
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 \( \pi_\beta \) using a supervised learning objective. We also assessed a variant of this method using a subset of only successful trajectories \( D^* \).
- **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.