Expansive Latent Planning for Sparse Reward Offline Reinforcement Learning

Robert Gieselmann
KTH Royal Institute of Technology
Sweden
robgie@kth.se

Florian T. Pokorny
KTH Royal Institute of Technology
Sweden
fpokorny@kth.se

Abstract: Sampling-based motion planning algorithms excel at searching global solution paths in geometrically complex settings. 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 by adding a latent planner that searches globally for feasible paths instead of being bound to a fixed prediction horizon.

Keywords: Reinforcement Learning, Planning, Robot Manipulation

1 Introduction

The acquisition of complex motor skills from raw sensory observations presents one of the main goals of robot learning. Reinforcement learning (RL) [1] provides a generic framework to obtain such decision-making policies through the interaction with an environment. Model-based RL [2] has recently gained much 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 equipped with planning methods. 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 [3, 4, 5, 6, 7].

The model-based reinforcement learning (RL) literature has investigated various planning methods in latent spaces, encompassing zero-order shooting-based approaches such as the Cross-Entropy Method (CEM) [8, 3] and Model-Predictive Path Integral (MPPI) [9, 10, 11, 7], first-order gradient-based optimization [12, 4], and more recently, trajectory collocation using second-order solvers [6]. Despite this methodological diversity, the majority of existing tools primarily facilitate local optimization within a fixed prediction horizon. Even with guidance from value heuristics, such as the one proposed in [7], local minima may still impede progress, particularly when estimating the optimal value function is difficult due to sparse reward feedback or limited training data. This paper argues that planning in latent state spaces can benefit from more global exploration strategies that seek solutions beyond a fixed prediction horizon to avoid convergence to local minima.

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

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 [13], provides a diverse range of algorithms for finding global paths between states in continuous and geometrically-complex environments. Recent works by [5, 14, 15] have proposed modifications of sampling-based planners in latent spaces. However, these approaches either rely on expert data or are not directly applicable to the reward-based learning setting. This paper explores the integration of sampling-based planning techniques in learned latent spaces, providing new avenues for model-based reinforcement learning. Specifically, we focus on the challenging scenario of offline RL [16], which is characterized by the amplified effects of value approximation errors [17]. Further, it allows us to better study the performance of planning in isolation by disentangling training and data collection. We introduce Value-guided Expansive Latent Planning (VELAP), which combines a sampling-based planning module with a suitable state embedding. Similar to [15], our search tree serves as a state memory and it used to guide the exploration towards undiscovered areas within the data support. Moreover, we leverage value heuristics obtained through temporal difference learning to accelerate the discovery of high-valued states. We present a comprehensive benchmark evaluation focusing on vision-based control. For this purpose, we adapt the robot manipulator control environments from the meta-world benchmark suite [18]. Our experiments reveal that VELAP surpasses existing approaches by a significant margin in terms of episode success rate. We attribute this performance gain to its ability to overcome local value optima through global exploration, in contrast to the prevalent approach of optimizing over a fixed horizons.

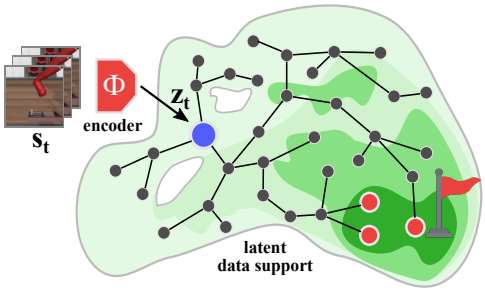


Figure 1: Our method grows a search tree in the latent space to globally explore reward-maximizing paths (blue:start, red:goal nodes, green: estimated values).

2 Related Work

Learning to control from visual input is becoming increasingly popular due to the wide availability of inexpensive sensors and the generic representational format of images. A considerable number of work uses deep generative models [19] to generate future images and plan actions via model predictive control [20, 21, 22, 23, 24]. These methods allow visual inspection of paths by humans, but this is accompanied by the difficulty of generating and evaluating high-resolution video sequences of many future steps. In this work, we instead follow the latent planning paradigm, which bypasses the need to synthesize high-dimensional samples and enables farsighted search with lower computational costs. Several works learn image distance metrics using unsupervised learning [25, 5] or RL [26, 27] to build an environment map that can be searched to generate visual paths. Map-based approaches have shown impressive results for navigation in static environments, but often do not scale well to complex settings with object interaction.

The work in [3] presents a model-based RL agent which uses a recurrent state space model to map image observations to a lower-dimensional latent space. Time-discrete state-action trajectories are optimized using the Cross-Entropy Method (CEM) [8] with a learned latent dynamics model. In a follow-up work, [4] uses a differentiable latent planner to efficiently learn behaviors by propagating analytic value gradients back in time. Similarly, [12, 28] implement a differentiable planner that optimizes latent trajectories via gradient descent. [29] trains a goal-conditioned RL agent which generates future subgoal states through trajectory optimization within the latent space of a Variational Autoencoder (VAE) [30]. [6] introduces the concept of collocation to model-based RL for visual robot manipulation and optimizes state-action sequences directly with a second-order optimizer. [10] solve dexterous manipulation tasks through latent trajectory optimization using a reward-weighted adaptation of MPPI [9]. Similarly, the method in [7] uses value estimates as cost within an MPPI-based latent policy. The line of work in [31, 32, 33] combines model-based RL

with Monte Carlo Tree Search (MCTS) [34] for long-horizon decision-making in vision-based tasks such as playing Atari games. While MCTS is mainly designed for discrete settings, [35] presents an adaptation for continuous action spaces where the search and the policy improvement are based on action sampling.

To address the offline RL setting, [36] modifies the MPPI agent in [10]. Their approach ensures adherence to the data support by sampling actions from an imitation learning policy, thus addressing the issue of out-of-distribution actions [17]. Similar, [11] presents a model-based agent for online and offline RL which uses MPPI to maximize the expected return of imagined trajectories while being guided by a learned policy. [37] designed a hierarchical agent for sparse reward manipulation tasks. A VAE-based manager policy predicts subgoal states which a worker policy must achieve within a fixed contingent of steps. In [38], a RRT-like [39] latent planner is presented for planning from high-dimensional data. Compared to ours, their method relies on collision checking data and is not designed for the more general reward-based setting. [14] leverages play data obtained from a human operator to train a task-conditioned policy which guides a tree search in a learned latent space. Our method is most related to the one recently presented in [15]. The authors introduce a sampling-based latent planner similar to the classical Expansive Space Trees (EST) algorithm [40]. Nevertheless, notable differences arise in terms of the problem types we tackle, resulting in the adoption of distinct sets of tools. While their approach is confined to goal-reaching navigation tasks, our method accommodates more general task specifications by leveraging sparse rewards. To accomplish this, we employ value-based reinforcement learning to (a) jointly optimize representations for planning alongside the control policy, (b) integrate learned heuristics for node and action selection during planning, and (c) identify suitable goals based on value estimates. These advancements significantly enhance the capabilities and versatility of our method, surpassing the scope of the previous work.

3 Preliminaries

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 [1] (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 [16] arises when training from a fixed dataset $\mathcal{D}$ consisting of trajectories generated by a behavior 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 [17].

Hindsight data relabeling Relabeling data has emerged as a popular technique in goal-conditioned off-policy RL [41, 42, 43, 44, 45] 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 [41]. This approach extends to offline trajectory datasets, where relabeling can be used to synthesize experiences for learning state-reaching behavior [46, 47]. Failed transitions are relabeled by designating the subsequent state as the desired goal and adjusting the corresponding reward accordingly. A connection between hindsight relabeling and contrastive learning was recently discussed in [48].

Sampling-based motion planning Sampling-based motion planners [13] compute feasible paths connecting two points in a robot configuration space. At the core, these methods explore and construct a graphical representation of the continuous search space. The rapidly-exploring random tree (RRT) [39] is a widely used single-query planner, particularly suitable for scenarios with varying environments. It incrementally expands a tree structure by alternately sampling collision-free states from the robot's configuration space and attempting to connect these to the nearest neighbor in the tree. Once a node reaches the vicinity of the target, a possible solution path is given by backtracing

to the root of the tree. Instead of sampling states from the configuration space, the expansive space trees (EST) planner [40] generates new states by expanding existing tree nodes using randomly-sampled actions.

4 Value-guided Expansive Latent Trees

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

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 which provides a positive signal only when reaching the final goal for which we terminate the episode. For training, we use an offline dataset $\mathcal{D}$ consisting of recorded transitions obtained from a sub-optimal policy.

Components To tackle the specified problem, we propose a model-based RL agent that incorporates a tree-based search, inspired by ESTs [40], within a learned representation space. Our preference for EST over an RRT-based approach [39] is driven by the reasoning that the expansion step in ESTs eliminates the need for a global state sampler. It should be noted that learning such generative models can be challenging, as they require high-fidelity predictions to prevent negative assessment of out-of-distribution samples. 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 our task goal. During planning, we will use Q^l to derive a distance proxy between states and Q^g to estimate the remaining number of steps to the goal. Among various actor-critic offline RL methods available, we select TD3-BC [49] due to its robustness and ease of 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 [50] (see App. B). 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 between two states z_i and z_j with respect to π^l. Finally, a conditional generative model, representing our action model g, enables sampling actions from the state-conditioned action distribution. We use $\mathbb{R}^m$ to denote the input noise used during the generation process.

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

Alignment of representation and planner To achieve long-horizon planning and control in $\mathcal{Z}$, we seek a state representation which favors accurate learning of dynamics in order to generate valid future waypoint states over many time steps. Secondly, the state encoding should facilitate the optimization of our value functions and control policies. Existing model-based RL approaches often rely on surrogate metrics for model learning, such as mean-squared prediction error or pixel-wise reconstruction. These metrics do not ensure alignment with actual control performance, leading to a mismatch between the environment model and the planner [51], which can adversely affect the controller's performance. To address the challenge of long-horizon predictions, we optimize our state encoder ϕ together with the latent dynamics h. In addition, we facilitate the approximation of the local and global value functions by training their models jointly with the encoding. Our model training objective $\mathcal{L}_{\text{model}}$ is shown in Eq. 2. Here, $\mathcal{L}_{Q^l_k}$ represents the temporal difference (TD) loss

for training Q_k^l, $\mathcal{L}_{Q^g}$ corresponds to the TD loss for training Q^g, and $\mathcal{L}_h$ denotes the loss function for the dynamics model h. The hyperparameters c_0 and c_1 act as weighting factors.

$$\mathcal{L}_{\text{model}} = \frac{1}{n_{\text{ens}}} \sum_k \mathcal{L}_{Q_k^l} + c_0 \cdot \mathcal{L}_{Q^g} + c_1 \cdot \mathcal{L}_h \tag{2}$$

$$\mathcal{L}_{Q_k^l} = \mathop{\mathbb{E}}_{\mathcal{D}'}[(Q_k^l(z_t, z_*, a_t) - (r_t + \gamma Q_{\text{min}}^{t+1,*}))^2] \tag{3}$$

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

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 $\mathcal{L}_{\pi^g}$ (Eq. 5). Note that this step is done by alternating between optimizing $\mathcal{L}_{\text{model}}$ and policy improvement while the encoder parameters are kept fixed during the policy update[1].

$$\begin{aligned} \mathcal{L}_{\pi^l} &= \mathop{\mathbb{E}}_{\mathcal{D}'}[-Q_{\text{min}}^{t,*}] + c_2 \cdot \mathop{\mathbb{E}}_{\mathcal{D}'}[(\pi^l(z_t, z_*) - a_t)^2] \\ \mathcal{L}_{\pi^g} &= \mathop{\mathbb{E}}_{\mathcal{D}}[-Q^g(z_t, \pi^g(z_t))] + c_3 \cdot \mathop{\mathbb{E}}_{\mathcal{D}}[(\pi^g(z_t) - a_t)^2] \end{aligned} \tag{5}$$

To provide data for training the local policy and value functions π^l and Q_k^l, we synthesize a dataset of state-reaching experiences $\mathcal{D}'$ by relabeling the transitions in $\mathcal{D}$. More specifically, we achieve this using hindsight goal relabeling [46, 47] to sample goals $s_* \in \mathcal{S}$ and use a binary reward to indicate success (see App. B.4). For training the dynamics model, we use the contrastive loss presented in [53]. In practice, we found this approach to work better in maintaining accurate long-term predictions compared to a standard mean-squared error objective (App. E.3).

Tree expansion Our aim is to solve the RL decision-making problem by searching the latent state space for the shortest connection towards valid goal states. Similar to [15], we follow the concept of EST planners [40] which 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_k^l estimates the return for reaching towards a particular node under sparse binary rewards, a temporal distance proxy is given by $\log_\gamma Q_k^l$. To account for value approximation errors [17], we will use the minimum value among the ensembles predictions to compute 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 $Q_{\text{min}}^{\text{exp,new}}$ is above a threshold $\tau_{\text{discard}}^{\text{low}}$. If it lies below this threshold, we discard z_{new}. Secondly, we also reject it if the corresponding value of $Q_{\text{std}}^{\text{exp,new}}$ is above a threshold $\tau_{\text{discard}}^{\text{std}}$. 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

Algorithm 1 Node sampling and tree expansion

Given: z_{init}, n_{iter}, n_{sim}, g, h, π^g, Q^g, Q_k^l, π^l
Initialize: $\mathcal{V} \leftarrow \{z_{\text{init}}\}$, $\mathcal{E} \leftarrow \emptyset$
for n_{iter} steps **do**
  Sample node z_{exp} from $\mathcal{V}$ given $P_{\text{node}}(\mathcal{V})$
  $z_{\text{new}} \leftarrow z_{\text{exp}}$
  Simulate forward using dynamics for n_{sim} steps
  for n_{sim} steps **do**
    Sample action $a \sim g(.|z_{\text{new}})$ (or $a = \pi^g(z_{\text{new}})$)
    $z_{\text{new}} \leftarrow h(z_{\text{new}}, a)$
  end for
  Reject node if too close to existing one in the tree or
  if the value uncertainty is too high
  if $Q_{\text{min}}^{\text{exp,new}} > \tau_{\text{discard}}^{\text{low}}$ and $Q_{\text{std}}^{\text{exp,new}} < \tau_{\text{discard}}^{\text{std}}$ **then**
    if $\max\{Q_{\text{min}}^{i,\text{new}} | z_i \in \mathcal{V}\} < \tau_{\text{discard}}^{\text{high}}$ **then**
      Add new node to tree
      $V \leftarrow V \cup \{z_{\text{new}}\}$; $E \leftarrow E \cup \{z_{\text{exp}\rightarrow\text{new}}\}$
    end if
  end if
end for

[1] We optimize the state representation during the critic update instead of the policy improvement step as motivated by the empirical analysis in [52].

data distribution. Lastly, we determine if the newly generated node is sufficiently novel given the existing ones in $\mathcal{T}$ and discard it otherwise. This sparsification step avoids redundant tree nodes and is important to keep the computations at a moderate level. More specifically, 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 $\mathcal{E} \leftarrow \mathcal{E} \cup \{z_{\text{exp}\rightarrow\text{new}}\}$. For more details on the expansion step see App. B.6.

Node sampling So far we haven't defined the node sampling distribution P_{node}. 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 Q^g. (a) leads to quick exploration of undiscovered latent states, while (b) drives the planner towards high-valued areas. For both parts, we use exponential weighting as shown in Eq. 6 and 7. In this regard, n^{neigh}_i corresponds to the number of incoming neighbors for a node ($\mathcal{V}^{\text{neigh}}_{\rightarrow \text{i}}$). We compose P_{node} by sampling according to P_{sparse} with probability p^{sparse} and from P_{value} with p^{value} (otherwise random uniform).

$$P_{\text{sparse}}(z_i) = \frac{e^{-n^{\text{neigh}}_i / T_{\text{sparsity}}}}{\sum_{z_j \in \mathcal{V}} e^{-n^{\text{neigh}}_j / T_{\text{sparsity}}}} \quad (6)$$

$$P_{\text{value}}(z_i) = \frac{e^{Q^g_i / T_{\text{value}}}}{\sum_{z_j \in \mathcal{V}} e^{Q^g_j / T_{\text{value}}}} \quad (7)$$

Action sampling The model g mimics the state-dependent action distribution of our data and is represented by a conditional VAE [30]. Sampling actions from g reduces the evaluation of undesired state-actions pairs which were not observed in $\mathcal{D}$. To help our planner discover task-relevant areas quicker, we further predict actions using π^g with a probability p^{policy}.

Planning objective and control Our planner builds a sparse tree representation in the latent space whose expansion is guided by value and sparsity heuristics. To choose the best path in $\mathcal{T}$, we must define an objective that ranks all explored paths. In practice, we first identify all nodes for which the value of Q^g surpasses a threshold τ_{goal} and gather the associated paths from the root z_{init} in a set G. Among the elements in G, we then choose the path g^* which has the shortest temporal length using $Q^{i,j}_{\text{min}}$ to derive a distance proxy between subsequent nodes (Eq. 8). If $G = \emptyset$, we simply pick the path that contains the node with the highest value of Q^g.

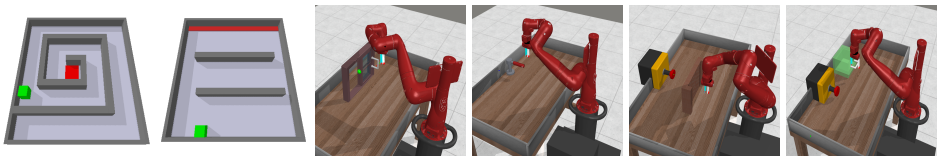
$$g^* = \arg\min_{g \in \mathcal{G}} c(g) \qquad \text{with} \qquad c(g) = \sum_{(i,j) \in \mathcal{E}_g} \log_\gamma Q^{i,j}_{\text{min}} \quad (8)$$

To use our planner for control, we embed it into a model-predictive control loop (Alg. 2). The controller queries our planner every n_{replan} steps and uses the local policy π^l to navigate 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 $Q^{i,j}_{\text{min}}$ against a threshold τ_{wp}.

5 Experiments

5.1 Empirical evaluation in simulation

Baselines To assess the effectiveness of `VELAP`, we measured its performance against the following baselines. *Behavioral cloning* (**BC**), a simple but often successful method that imitates the behavior policy using a supervised learning objective. We evaluated a second version of this approach only trained on the subset of successful trajectories ($\mathcal{D}^*$). **TD3-BC** [49], an adaptation of the Twin Delayed DDPG algorithm [54] which circumvents the effect value overestimation by adding an imitation objective to the policy update. **IQL** [55], a state-of-the-art offline RL baseline. **MPPI** [9] provides the base planning algorithm in various model-based RL methods ([10, 7]). We consider an implementation for the offline learning setup which uses TD3-BC within the cost update during optimization. **MBOP** [36], a model-based agent which uses an imitation learning policy to bias the action sampling in MPPI. **IRIS** [37], an offline RL method particularly designed for sparse reward settings. 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)



(a) SpiralMaze (b) ObstacleMaze (c) WindowClose (d) FaucetClose (e) ButtonWall (f) DrawerButton

Figure 2: Evaluation environments adapted from meta-world robotics benchmark.

Table 1: Success rates and std. deviations (%) on test cases with unseen object variations (except *SpiralMaze*).

METHOD	BC	BC ($\mathcal{D}^*$)	TD3-BC	IQL	MPPI	MBOP	IRIS	IRIS (MULTI-STEP)	VELAP
SPIRAL MAZE	0 ± 0	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	6 ± 3	83 ± 11	40 ± 25	50 ± 25	62 ± 14	$\mathbf{97 \pm 2}$
WINDOW	0 ± 0	34 ± 11	16 ± 8	9 ± 4	70 ± 7	23 ± 4	69 ± 3	43 ± 20	$\mathbf{78 \pm 4}$
FAUCET	0 ± 0	36 ± 6	13 ± 7	8 ± 4	41 ± 7	33 ± 2	10 ± 2	3 ± 1	$\mathbf{51 \pm 12}$
BUTTONWALL	0 ± 0	0 ± 0	2 ± 2	0 ± 0	9 ± 10	0 ± 0	35 ± 5	8 ± 8	$\mathbf{76 \pm 9}$
DRAWERBUTTON	0 ± 0	0 ± 0	0 ± 0	1 ± 0	0 ± 0	0 ± 0	5 ± 3	0 ± 0	$\mathbf{11 \pm 3}$

which a worker policy must achieve. We also examine **IRIS (multi-step)**, where the set of subgoals is generated by shooting a future state sequence using the cVAE. To establish a fair comparison and disentangle the effects of the representation and planner, we use the same representations, dynamics models across all methods. Further details are provided in App. D.

Tasks We consider the simulated visuomotor control tasks depicted in Fig. 2. In *SpiralMaze*, the velocity of a block robot is controlled in order to navigate from the outermost point of a maze to the inner region. This task was designed to require farsighted planning, as the temporal distance to the goal is approximately 150 steps. In the *ObstacleMaze* environment, the block robot must travel to the opposite wall of the room while two randomly positioned obstacles appear in the center of the workspace. Additionally, we evaluate the *WindowClose* and *FaucetClose* environments from the meta-world [18] benchmark. As done in [6], we use sparse binary rewards and render images from a static camera. Moreover, we propose two new settings *ButtonWall* and *DrawerButton*. In *ButtonWall*, the robot must first navigate around a wall (varying position) and then press a button. To solve the task *DrawerButton*, the agent must first close a drawer and then press a button (both randomly initialized). Our training data $\mathcal{D}$ consists of random trajectory data and a small number of noisy expert demonstrations. We use a latent space of size 32 and RGB images of resolution 64×64. For details on the environments, data collection and hyperparameters see App. B.

Results An overview of the numerical benchmark evaluation is given in Table 1. `VELAP` consistently outperform the baselines across all environments in terms of average episode success rate. The improvements are particularly visible in tasks which require far-sighted planning such as *SpiralMaze* and *ButtonWall*. These results support that our tree-based exploration strategy is indeed effective at planning for sparse-reward offline RL. Fig. 3 illustrates a 2D embedding of a latent path for the *SpiralMaze* task computed with `VELAP`. It suggests that our method explores global solutions and identifies one which reaches through the entire maze. Fig. 3 demonstrates similar capabilities for the *ObstacleEnv* task. Further, it supports that our representation accommodates for the different locations of obstacles, creating latent spaces that mirror the topology of the underlying state space. Ablation results concerning the embedding and dynamics model are provided in App. E.

5.2 Physical robot experiments

To test our method under realistic conditions, we designed two manipulation tasks using a low-cost robot arm (Fig. 4). In the first, the robot must push a sponge into the marked region. In the second task, the robot holds a rope which it must unravel from a cylindrical object. Then it needs to position the held end of the rope precisely onto a designated colored area. The robot is controlled by providing desired end effector position and wrist orientation displacements, resulting in a 4-dimensional action space. Both tasks are challenging in terms of perception and control as

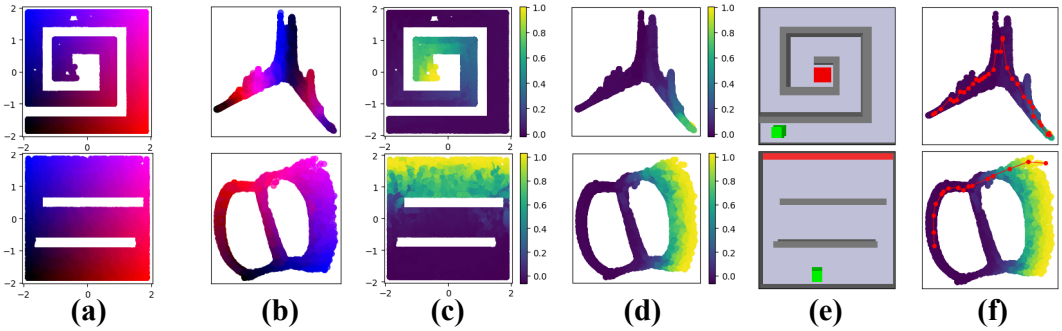


Figure 3: Supporting visualizations for *SpiralMaze* (top row) and *ObstacleMaze* (bottom row) tasks. (a) x-y robot positions for a uniformly sampled set of states (b). 2D Isomap embeddings of learned latent space (color encodes correspondence to robot positions in (a)).(c) approximated global Q-values for x-y robot positions. (d) approximated global Q-values for Isomap embedding of learned latent state space. (e) example image input frames. (f) planned latent paths (red) computed with `VELAP` and projected to Isomap embedding.

the initial configuration of the objects are randomized. The results of a comparison with BC, BC ($\mathcal{D}^*$) and IRIS are presented in Table 11 (more details in App. E.1). Our method surpasses the performance of its baselines achieving $14/20$ successful episode on the sponge manipulation and $8/20$ on the rope manipulation task. Video demonstrations can be found at https://sites.google.com/view/velap-corl/home.

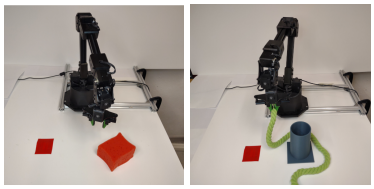


Figure 4: Physical manipulation of a sponge (left) and rope (right).

6 Limitations and Future Directions

Our method lays the groundwork for future enhancements. While `VELAP` was designed for offline RL, it can be adapted to the online setting by interleaving data collecting and learning [3, 6]. It could also enhance sample efficiency by improving policy and critic updates through planning. [56, 4, 31]. Currently, `VELAP` uses deterministic dynamics and encoder models, limiting it to fully-observable MDPs. By incorporating probabilistic transition models and state filtering approaches (similar to [3, 57]), it can be extended to partially observable and stochastic settings. While our method outperforms existing baselines, it still struggles with complex tasks which we account to the remaining difficulty in estimating accurate latent dynamics for long-horizon planning. Presently, our planner selects paths by minimizing the distance to high-valued states. To further minimize the effects of model inaccuracies, the planning strategy could incorporate uncertainty propagation and assessment in the tree branches. Notably, we discovered that in more complex tasks, such as *DrawerButton* and the rope manipulation scenario, failures often occurred when the agent moved to the final goal region without completing the required subtask, like first closing the drawer or maneuvering the rope around the pole. We believe that improved handling of uncertainties, along with risk-aware measures, could potentially lessen the planner's greedy exploitation of model errors. In this context, planning in belief spaces [58] provides another potential improvement avenue.

7 Conclusion

We introduced `VELAP`, an agent designed for model-based planning in sparse reward offline RL. Diverging from the usual model-based RL planners, our approach employs a tree-based exploration algorithm inspired by sampling-based planners commonly used in robot motion planning. Through empirical evaluation on visual control tasks, we showcased substantial enhancements achieved by our method compared to existing benchmarks. Our aim is that these outcomes will inspire additional exploration into the fusion of sampling-based planning, representation learning, and model-based RL

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

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

[2] T. M. Moerland, J. Broekens, and C. M. Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

[3] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In K. Chaudhuri and R. 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.

[4] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

[5] K. Liu, T. Kurutach, C. Tung, P. Abbeel, and A. Tamar. Hallucinative topological memory for zero-shot visual planning. In H. D. III and A. 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.

[6] O. Rybkin, C. Zhu, A. Nagabandi, K. Daniilidis, I. Mordatch, and S. Levine. Model-based reinforcement learning via latent-space collocation. In M. Meila and T. 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.

[7] N. Hansen, X. Wang, and H. Su. Temporal difference learning for model predictive control. In *ICML*, 2022.

[8] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L'Ecuyer. Chapter 3 - the cross-entropy method for optimization. In C. Rao and V. 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.

[9] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. 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.

[10] A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar. Deep Dynamics Models for Learning Dexterous Manipulation. In *Conference on Robot Learning (CoRL)*, 2019.

[11] H. Sikchi, W. Zhou, and D. Held. Learning off-policy with online planning. In *Conference on Robot Learning*, pages 1622–1633. PMLR, 2022.

[12] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In J. Dy and A. 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.

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

[14] B. Ichter, P. Sermanet, and C. 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.

[15] R. Gieselmann and F. T. Pokorny. Latent planning via expansive tree search. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=zSdz5scsnzU.

[16] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

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

[18] 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.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[20] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/d9d4f495e875a2e075a1a4a6e1b9770f-Paper.pdf.

[21] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control. *arXiv e-prints*, art. arXiv:1812.00568, Dec. 2018.

[22] L. Yen-Chen, M. Bauza, and P. Isola. Experience-embedded visual foresight. In *Conference on Robot Learning*, 2019.

[23] S. Nair and C. Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gzR2VKDH.

[24] S. Tian, S. Nair, F. Ebert, S. Dasari, B. Eysenbach, C. Finn, and S. Levine. Model-based visual planning with self-supervised functional distances. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=UcoXdfrORC.

[25] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SygwwGbRW.

[26] B. Eysenbach, R. Salakhutdinov, and S. Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, 2019.

[27] M. Laskin, S. Emmons, A. Jain, T. Kurutach, P. Abbeel, and D. Pathak. Sparse graphical memory for robust planning. In *Advances in Neural Information Processing Systems*, 2020.

[28] T. Yu, G. Shevchuk, D. Sadigh, and C. Finn. Unsupervised visuomotor control through distributional planning networks. In *Proceedings of Robotics: Science and Systems*, Freiburgim-Breisgau, Germany, June 2019. doi:10.15607/RSS.2019.XV.020.

[29] S. Nasiriany, V. Pong, S. Lin, and S. Levine. Planning with goal-conditioned policies. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/c8cc6e90ccbff44c9cee23611711cdc4-Paper.pdf.

[30] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2014.

[31] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. 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.

[32] J. Schrittwieser, T. K. Hubert, A. Mandhane, M. Barekatain, I. Antonoglou, and D. Silver. Online and offline reinforcement learning by planning with a learned model. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=HKtsGW-lNbw.

[33] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao. Mastering atari games with limited data. In *NeurIPS*, 2021.

[34] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. CG'06, page 72–83, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3540755373.

[35] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver. Learning and planning in complex action spaces. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4476–4486. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/hubert21a.html.

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

[37] A. Mandlekar, F. Ramos, B. Boots, S. Savarese, L. Fei-Fei, A. Garg, and D. Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4414–4420. IEEE, 2020.

[38] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094, 2018. doi:10.1109/ICRA.2018.8460730.

[39] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[40] 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.

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

[42] B. Eysenbach, X. Geng, S. Levine, and R. R. Salakhutdinov. Rewriting history with inverse rl: Hindsight inference for policy improvement. *Advances in neural information processing systems*, 33:14783–14795, 2020.

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

[44] T. Davchev, O. O. Sushkov, J.-B. Regli, S. Schaal, Y. Aytar, M. Wulfmeier, and J. 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.

[45] A. Li, L. Pinto, and P. Abbeel. Generalized hindsight for reinforcement learning. *Advances in neural information processing systems*, 33:7754–7767, 2020.

[46] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. C. Julian, C. Finn, and S. Levine. Actionable models: Unsupervised offline reinforcement learning of robotic skills. In M. Meila and T. 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.

[47] J. Li, C. Tang, M. Tomizuka, and W. 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.

[48] B. Eysenbach, T. Zhang, S. Levine, and R. Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=vGQiU5sqUe3.

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

[50] G. An, S. Moon, J.-H. Kim, and H. O. Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. In *Neural Information Processing Systems*, 2021.

[51] N. Lambert, B. Amos, O. Yadan, and R. Calandra. Objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2002.04523*, 2020.

[52] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. 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.

[53] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[54] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[55] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[56] R. 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.

[57] N. Srivastava, W. Talbott, M. B. Lopez, S. Zhai, and J. Susskind. Robust robotic control from pixels using contrastive recurrent state-space models. *arXiv preprint arXiv:2112.01163*, 2021.

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

[59] J. Bialkowski, S. Karaman, and E. 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.

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

[61] J. B. Tenenbaum, V. de Silva, and J. 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.

[62] E. Todorov, T. Erez, and Y. 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.

[63] M. Raisi, A. Noohian, L. McCutcheon, and S. Fallah. Value summation: A novel scoring function for mpc-based model-based reinforcement learning. *ArXiv*, abs/2209.08169, 2022.

A Additional material

Code examples and additional material will be uploaded here https://sites.google.com/view/velap-corl/home. All models were implemented in Python using the PyTorch library. The total training time including all models and baselines amounts to approximately 150 hours (wall clock) on a single GPU.

B Hyperparameters and algorithm details

Here we present a description of the hyperparameters of our trained models and planning module.

B.1 Model architectures

Table 2: 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 3: Hyperparameters of dynamics model h

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

Table 4: 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 5: Hyperparameters of policy networks π^l and π^g

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

Table 6: Hyperparameters of critic networks Q_k^l and Q^g

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

B.2 Training hyperparameters

Table 7: Model training hyperparameters

Parameter	Value
batch size	64
learning rate	0.0003
c_0	0.2
c_1 (*SpiralMaze*)	0.001
c_1 (*ObstacleMaze*)	0.01
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)	1.0
n_{ens}	3

B.3 Planner and controller hyperparameters

Table 8: Hyperparameters of planner

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_{neigh}	Q-value threshold to determine neighboring nodes	γ^3
τ_{goal}	Q-value threshold to determine goal nodes	γ^5
d_{neigh}	Euclidean distance threshold to determine candidate neighbors	3 x upper 5-percentile of Eucl. distances between encoding of subsequent states

Table 9: Hyperparameters of controller

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

B.4 Training of policy and value functions

We use TD3-BC [49] 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 further improve the accuracy of Q^l_k, we use n_{ens} 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 [50]). The ensemble further allows us to filter out unlikely or out-of-distribution transitions generated during the tree expansion. This is done by assessing the minimum predicted ensemble Q-value and the standard deviation among the predicted values (Sec. 4).

Our models π^l and Q^l_k describe goal-reaching 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 describe this particular setting, we can augment it using hindsight goal relabeling. In particular, we create a new dataset $\mathcal{D}'$ consisting of transitions $(z_t, a_t, r_t, z_{t+1}, z_*, \gamma) \in \mathcal{D}'$ by relabeling the values of r_t, γ (γ also indicates terminal condition, i.e. $\gamma = 0$) and adding a goal state z_*. We apply a combination of three different relabeling strategies (a) set goal z_* to be next state in the relabeled transition and set $\gamma = 0$; $r_t = 1$ (b) sample z_* from the set of future states within the same trajectory and set $r_t = 0$ (c) sample z_* from another trajectory in the data and set $r_t = 0$.

B.5 Training of dynamics model

Our dynamics model h is trained using the InfoNCE contrastive loss introduced in [53]. Given an initial state z_t and a sequence of actions $a_{t:t+k-1}$, we use h to generate predictions $\widetilde{z}_{t+k}$ for i=1..k. We compute the NCE loss at each each k with positive pairs $(\widetilde{z}_{t+k}, z_{t+k})$ and take negative examples z_j randomly from the batch. We use $f = e^{-||z_i - z_j||_2/T}$ to compute similarity between latent encodings. Our overall training loss for h (Eq. 9) takes the average over the contrastive loss terms computed at all k steps. For all experiments, we use $k = 3$.

$$\mathcal{L}_h = -\frac{1}{k} \sum_k \mathop{\mathbb{E}}_{\mathcal{D}} \Big[\log \frac{f(\widetilde{z}_{t+k}, z_{t+k})}{\sum_j f(\widetilde{z}_{t+k}, z_j)} \Big] \tag{9}$$

B.6 Additional details about planning method

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. Computing the goal-conditioned values with respect to all nodes results in an enormous computations overhead. Yet, we can significantly reduce the amount of 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. B.3).

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 one new nodes at a time. For discussions about highly-parallelized implementations of classical RRT-like planners, we refer to [59, 60].

B.7 Additional details about MPC controller

Alg. 2 provided below outlines the pseudocode for our MPC controller. The function `update_waypoint`(z_{curr}, g^*) is responsible for determining the subsequent waypoint z_{wp} that our

local policy aims to attain. Specifically, we estimate the value between the current state and waypoint and switch to the next element in g^* if the predicted value surpasses a threshold τ_{wp}, i.e $Q_{\text{min}}^{\text{curr,wp}} > \tau_{\text{wp}}$. As we approach the final goal, indicated by the proximity of our current state z_{curr}, we stop the planning and compute actions based on the policy π^g. To ascertain our proximity to the goal, we compare the predicted value of the global value function Q^g against a predefined threshold as τ_{goal}.

Algorithm 2 MPC controller

Given: s_{init}, n_{replan} $n_{\text{max_steps}}$, ϕ, π^l, π^g
$z_{\text{curr}} \leftarrow \phi(s_{\text{init}})$ ▷ Map state to latent encoding
$i \leftarrow 0$
while goal not achieved and $i < n_{\text{max_steps}}$ **do**
  if not $i \mod n_{\text{replan}}$ **then** ▷ Replan every n_{replan} steps
    Build tree $\mathcal{T}$ rooted at z_{curr} for n_{iter} steps (Alg. 1).
    Determine $g^* = \{z_{curr}, z_1, .., z_n\}$ given $\mathcal{T}$
  end if
  $z_{\text{wp}} \leftarrow \texttt{update_waypoint}(z_{\text{curr}}, g^*)$ ▷ Update waypoint if close enough
  $a \leftarrow \pi^l(z_{\text{curr}}, z_{\text{wp}})$ (or use π^g within proximity to the overall goal) ▷ Compute next action
  Execute a and observe updated state s_{curr}
  $z_{\text{curr}} \leftarrow \phi(s_{\text{curr}})$
  $i \leftarrow i + 1$
end while

C Evaluation Environments

C.1 Description of block environments

Similar to the evaluation environments in [15], we implement two long-horizon navigation tasks characterized by a comparatively low-dimensional underlying state space. This aspect facilitates a visual examination of the learned latent embeddings via dimensionality reduction methodologies such as Isomap [61]. Within both scenarios, a block-shaped robot's motion is controlled through velocity commands, while its movement remains confined to a two-dimensional plane.

C.1.1 SpiralMaze

To accomplish this objective, the block agent is tasked with maneuvering from the outer edge of a spiral-shaped corridor to the inner area colored in red (Fig. 2a). The episode's duration is capped at 300 steps. During data generation for training, the agent is initialized at collision-free positions within the workspace. Subsequently, random action sequences are executed by sequentially adding Gaussian noise to an initially sampled uniformly random action at the start of each episode. For testing purposes, the agent's position is uniformly sampled from a small area in proximity to the outermost point of the spiral-shaped corridor.

C.1.2 ObstacleMaze

Within this setting, the agent is required to move towards the upper wall of the workspace, highlighted in red (Fig. 2b). To successfully accomplish this objective, the agent needs to execute actions that navigate around two obstacles, positioned randomly near the workspace center at the beginning of each new episode. The maximum permissible number of steps in the environment is 100. During testing, the agent's initial configuration is randomly set in close proximity to the wall opposite to the goal. The same data collection policy as for the *SpiralMaze* task was utilized.

C.2 Description of manipulation environments

We have customized several environments of the meta-world robot benchmark tasks proposed by [18]. The underyling physics simulation engine is Mujoco, as introduced by [62]. To enable vi-

sual manipulation, similar to the problems studied in [6], we enable background rendering of RGB images from a static viewpoint. The robot is controlled by commanding desired end effector and gripper opening displacements resulting in a 4-dimensional action space. While *WindowClose* and *FaucetClose* were with small modifications adapted from [18], we evaluate two new environments *ButtonWall* and *DrawerButton*. These new scenarios were purposely designed to investigate tasks with long horizons under sparse rewards. Importantly, they require the integration ("stitching") of trajectory data originating from different regions of the workspace to determine viable solutions.

In our data collection process, we employ a suboptimal policy that predominantly applies random actions (using additive Gaussian noise). Infrequently, this policy selects actions from a pre-defined expert policy. Table 10 provides insight about the number of samples and trajectories in the training data and presents the portion of successful transitions (reward=1). Across all manipulation tasks, we set the maximum permitted number of environment steps at 150, with the exception of the *ButtonWall* scenario, where we allow up to 250 steps during the evaluation phase.

C.2.1 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 set the positioning of the end effector above the surface of the table. The sampling of expert actions is restricted to areas close to the goal region (window 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.

C.2.2 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.

C.2.3 ButtonWall

In this scenario, the robot's end effector is required to navigate around a wall structure before pressing a button. The location of the wall is randomly set at the beginning of each episode. Furthermore, a height limitation is imposed on the end effector to ensure that the agent takes a more extended path around the wall, as opposed to simply elevating the end effector. The dataset was produced by either placing the agent in front of the wall, near the button, or far behind the wall. However, expert samples in the dataset only exist for scenarios when starting closer to the goal. For testing purposes, the end effector is sampled within a restricted region behind the wall.

C.2.4 DrawerButton

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

C.3 Composition of training dataset

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

Table 10: 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 %

D Baseline methods

To enable a fair comparison between different methods, we use the same underlying representation/encoder ϕ and dynamics model h in the evaluation of all baselines. For assessing the quality and impact of our representation learner, please review the experimental ablation study in App. E.2.

D.1 BC and BC ($\mathcal{D}^*$)

Simple behavioral cloning baselines for which we use the same network architecture as our policy networks (see Table 5) and train using a mean-squared error objective on the predicted actions. For $\mathcal{D}^*$ we train only on the subset of successful episodes in the dataset. For each method, we train for $3 \cdot 10^5$ iterations using a learning rate of $3 \cdot 10^{-4}$ and batches of size 128.

D.2 TD3-BC [49]

This baselines resembles the underlying global policy π^g used in `VELAP`. It provides us with a baseline to assess how well pure offline RL performs without any additional planning methods.

D.3 IQL [55]

This method presents a state of the art model-free offline RL baseline which utilizes expectile regression to estimate state-conditional expectiles of the target values in order to avoid querying values of out-of-distribution actions during training. We train IQL for $3 \cdot 10^5$ iterations with a learning rate of $3 \cdot 10^{-4}$, batches of size 256 and using hyperparameters $\tau = 0.7$ and $\beta = 3.0$.

D.4 MPPI

We implement a trajectory optimization baseline similar to the model-based planning algorithm introduced in [7]. The method in [7] presents an adaption of MPPI specifically for the online reinforcement learning setting which optimizes the expected return of sampled trajectories. To estimate the return, a learned model is used to predict the reward for each trajectory node while a learned Q-function predicts the future return beyond the specified planning horizon. Since rewards in our evaluation environments are sparse, the predicted rewards carry little guidance for the trajectory optimization as most states have 0 reward. Therefore, we adapt the objective in [7] and instead use the accumulated sum of state-action values as the optimization criterion. This type of scoring function in model-based RL has recently been discussed in [63]. To implement this baseline, we utilize the Q-function of TD3-BC. For all environments, we use 1000 samples per iteration, a planning horizon of 50, elite size 64 and 5 iterations. Replanning is done every 5 environment steps.

D.5 MBOP [36]

MBOP presents an adaptation of MPPI which was particularly designed for the offline RL setting. It generates new candidate trajectories by addding a small amount of Gaussian noise to the actions predicted by a behavioral-cloned policy. To evaluate the quality of the rollouts it uses a truncated value function trained on the offline data. Due to the sparse nature of rewards in our experiments, we

found that both the behavioral-cloned policies and the truncated value function were insufficient to generated farsighted behaviors that solve our tasks. To accommodate for the long planning horizons, we instead sample actions from our TD3-BC policy and use the corresponding Q-values to assess candidate trajectories during the optimization (similar to our MPPI baseline). For all environments, we use 1000 samples per iteration, a planning horizon of 50, elite size 64, 5 iterations and $\beta = 0.7$. Replanning is done every 5 environment steps.

D.6 IRIS [37]

D.7 IRIS (multi-step)

We evaluate an extension of IRIS in which we use the state prediction model (conditional VAE) to generate multi-step rollouts of suitable subgoals. This strategy increases the exploration horizon and allows to choose the best subgoal from a larger and potentially more diverse set of states. This planning strategy can also be seen as random shooting of coarse subgoal sequences. In all experiments, we generate 256 different trajectories using rollouts of length 5 and a conditional generative model to predict states for a horizon of 5. In our evaluation, we found that this method sometimes performs worse than IRIS. We attribute this to the fact that the global policy doesn't align with the capabilities of the local one, which occasionally results in the selection of subgoal states that might not be attainable.

E Supplementary Experiments and Analysis

E.1 Physical hardware experiments

The physical evaluation was performed using the WidowX 200 low-cost robot platform. For the real-world validation of our method, we collected 200 episodes of data for the sponge ($\sim$ 15000 samples) and 150 episodes of data for the rope manipulation ($\sim$ 15000 samples) tasks. Training data was generated by operating the robot through a gamepad and took less than 1 hour per task. The collected dataset consist largely of suboptimal and entirely random trajectories. Successful transitions (positive reward + episode termination) were labeled manually during the data collection. Similar to our simulated datasets, we collect trajectories in such a way that successful episodes always start within the vicinity of the goal. Conversely, during testing, we deliberately position the agent distant from the goal region. Consequently, this configuration emphasizes the need for an approach capable of internally assessing the connectivity between distinct trajectory segments within the data. To construct states, we combine three sequential images captured by a stationary camera. The results of a comparison with BC, BC ($\mathcal{D}^*$) and IRIS are presented in Table 11.

Table 11: Results of phyiscal robot experiments (successful episodes)

Environment	BC	BC ($\mathcal{D}^*$)	IRIS	VELAP
Sponge	5/20	6/20	6/20	14/20
Rope	0/20	0/20	2/20	8/20

Sponge task In this setting, the robot needs to push a sponge object onto a marked goal region (Fig. 4 (left)). To increase the difficulty of this task, we initialize the robot end effector between the goal region and the sponge. Consequently, the robot must initially maneuver itself behind the sponge before it can proceed to push the sponge towards the goal. During both training and testing, the initial poses of the end effector and sponge object were subject to random selection.

Rope task In this particular setting, the robot's end effector grasps a green rope, requiring the robot to skillfully maneuver the rope around a central pole in the workspace before ultimately placing the held end within a designated goal area (as shown in Figure 4 (right)). To elevate the complexity of this scenario, the agent is required to first unwind the rope from the pole before moving towards the

goal region. To introduce a challenging long-horizon aspect, we initialize the end effector in close proximity to the goal region while the rope is partially wrapped around the pole.

E.2 Ablating the impact of the learned representation

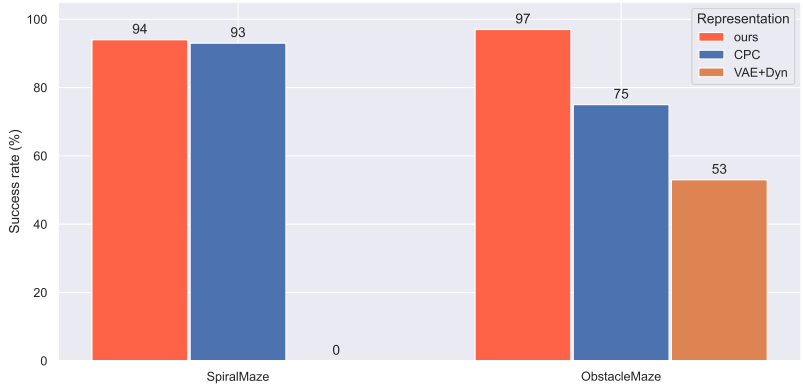


Figure 5: Impact of the type of representation on the performance of our method.

E.3 Influence of the dynamics loss

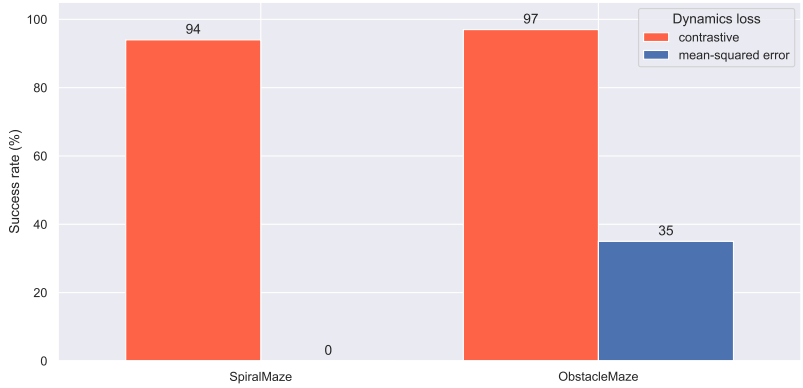


Figure 6: Impact of the type of dynamics loss on the performance of our method.

E.4 Computation time

Here, we provide an assessment of the computation time needed and the resulting success rates achieved by our approach in comparison to MPPI. Both algorithms were tested on hardware featuring a NVIDIA GeForce RTX 3090 graphics card. Despite utilizing GPU computation and implementing both methods in PyTorch, we didn't specifically fine-tune either for computational speed. It is worth mentioning that MPPI recalculates its plans every 5 steps within the environment, whereas our method follows a 25-step interval in the *SpiralMaze* and 15 steps in the *ObstacleMaze*.

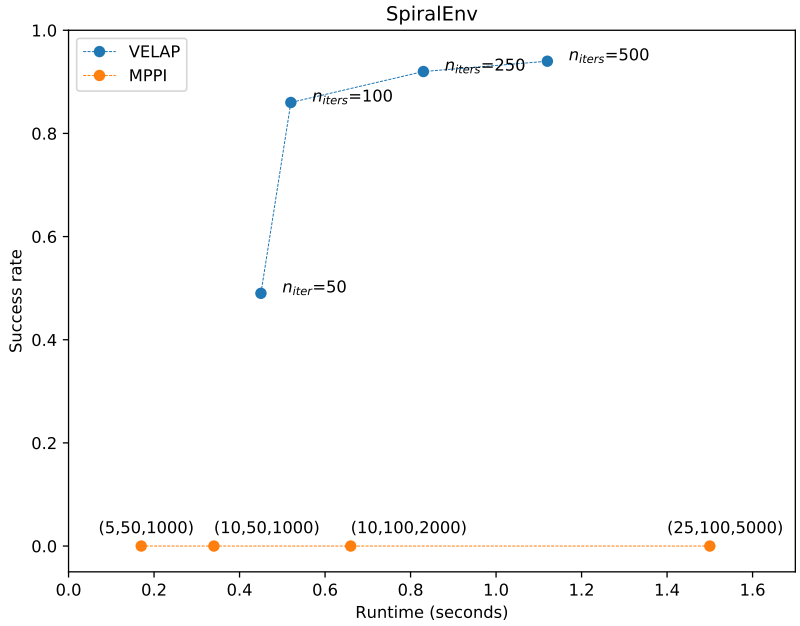


Figure 7: Relationship between average episode success rates and single planning query runtime (test scenarios) for different planning hyperparameters on *SpiralMaze* environment. For MPPI, we report results for varying (iterations, horizon, number of samples).

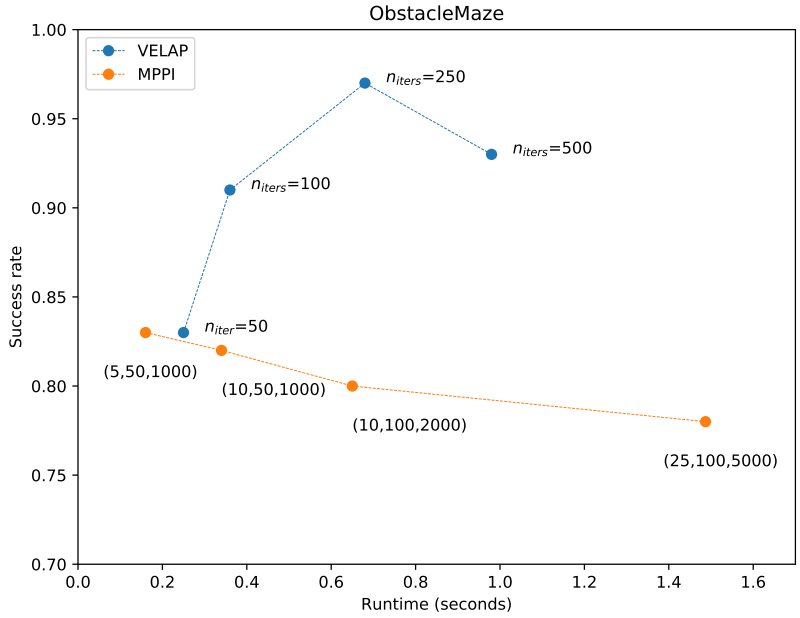


Figure 8: Relationship between average episode success rates and single planning query runtime (test scenarios) for different planning hyperparameters on *ObstacleMaze* environment. For MPPI, we report results for varying (iterations, horizon, number of samples).