PALMER: Perception-Action Loop with Memory Retrieval

Onur Beker **Mohammad Mohammadi** **Amir Zamir**

Swiss Federal Institute of Technology (EPFL)

Abstract

To achieve full autonomy in a priori unknown real-world scenarios, agents should be able to operate without assuming auxiliary instrumentation in the environment, act from high-dimensional sensory observations (e.g., images), learn from past experience to adapt and improve, and be capable of long horizon planning. Classical search algorithms are proficient at addressing the last requirement, while deep learning methods can provide the necessary flexibility to address the others. Therefore, recent work has proposed combining learning based local-policies (i.e., skills) with classical planning algorithms for subgoal (i.e., subtask) generation. Following this line of work, we introduce a general-purpose planning algorithm called PALMER that creates a tight feedback loop between representation learning, a non-parametric memory, reinforcement learning, and sampling-based motion planning. We introduce two main ideas: **i)** learning a perception function that projects high-dimensional observations into a representation space where latent distances correlate with *reachability* between states, **ii)** using these perceptual representations as a way to *remember* transitions stored in a replay buffer, to explicitly search and recompose past experiences. We discuss how this joint consideration of *perception*, *action*, and *memory* can address two of the main bottlenecks in reinforcement learning, namely visitation-counts and long-horizon planning.

1 Introduction

Animals and humans operate on high-dimensional stimuli to achieve diverse and ever-changing goals necessary to fulfill the ecological requirements of survival [1, 2, 3, 4, 5]. Experiential learning through trial-and-error plays a fundamental role in this [6, 7, 8, 9, 10, 5]. Even in simplest environments a brute-force approach to trial-and-error based search is intractable. This in turn motivates memory-based mechanisms for retrospective reasoning and compositional thinking, such as: i) remembering relevant segments of past experience, ii) recomposing them in new counterfactual ways to form hypotheses, and iii) executing such hypotheses as part of a targeted search strategy. Such memory-based mechanisms can significantly accelerate the trial-and-error process compared to a uniform search method. This is because the same behavior can remain valid for different goals and in different contexts due to the compositional structure of real-world environments.

What principles can allow for memory mechanisms to remember and recompose bits of experience? The concept of dynamic programming (DP) is directly related to this discussion, as it greatly reduces the computational cost of trial-and-error by employing the principle of optimality [11]. This principle can be colloquially stated as treating new and complex problems as a recomposition of old and simpler sub-problems that were already solved before. Recent work [12, 13, 14] employs this perspective to build hierarchical reinforcement learning (RL) algorithms for goal-reaching tasks. Such methods set edges between states using a distance regression model to build a planning graph, perform shortest

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

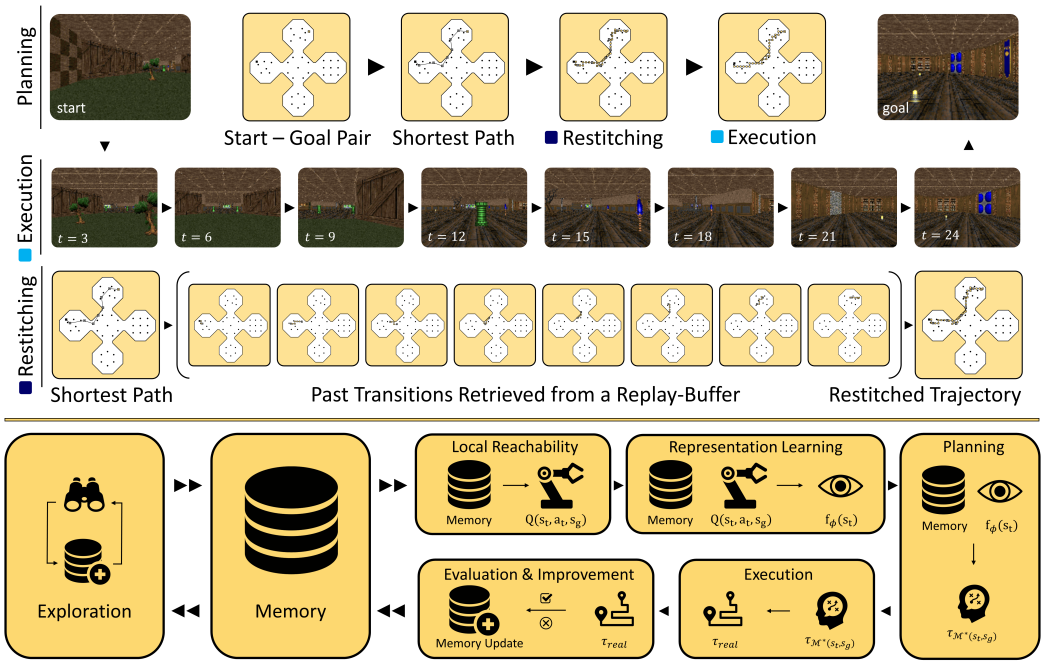

Figure 1: **Top**: Given a start-goal state pair, PALMER creates a trajectory that connects them by restitching the endpoints of past transitions in a replay buffer. This is enabled by a perception model f_ϕ that can identify close-by states, and results in robust long-horizon planning. **Bottom**: To achieve this : i) it uses any *exploration* process to populate a replay buffer, ii) uses off-policy Q-learning to obtain *local reachability* estimates, iii) uses them for *representation learning* to train f_ϕ, iv) uses f_ϕ to *plan* over the replay buffer, v) *executes* these plans, vi) *evaluates* the resulting transitions and inserts them back into the replay buffer to *improve* its contents.

path computations over it using DP-based graph search, and follow the resulting shortest paths with a learning-based local policy. Our paper builds upon this line of work.

Contribution: *We describe a method for an RL policy to explicitly remember and retrieve previously observed sequences of state transitions from its replay buffer using perceptual representations*. This is achieved through a two-step process. *First*, we learn a latent space where the distance between two states captures how much time it takes for the optimal policy to go from one to the other. *Second*, given a query state and a replay buffer, we use this distance metric to retrieve nearest neighbor states from the replay buffer that are easily reachable from the query state. We show that this perception enabled memory mechanism can have two important uses. *First one* addresses the exploration problem, by defining an approximate visitation-count for any given state (i.e., discrete or continuous) based on its number of neigboring states in the replay buffer. *Second one* addresses the long-horizon planning problem, by defining sampling-based planning algorithms that search over the replay buffer [12] to stitch together trajectories whose endpoints are neighboring states. This trajectory stitching approach allows for creating robust roadmaps to connect any two distant states (as depicted in Fig.1).

Our approach operates entirely on offline, unlabeled data [15, 16, 17], and can therefore be flexibly combined with any auxiliary data collection method to populate the replay buffer. Our experiments present a proof-of-concept by implementing an image-based navigation policy in simulation, using an offline replay buffer populated with uniform random-walk exploration data.

2 Perception-Action Loop with Memory Retrieval[1]

Nomenclature: An environment is represented as a tuple $\langle \mathcal{S}, \mathcal{A}, p_{env} \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, and $p_{env}(s'|s,a)$ is the markovian transition dynamics. A trajectory $\tau \in \mathcal{T}$ is any sequence of states and actions. $\tau_{i,s,0}$; $\tau_{i,s,-1}$; $\tau_{i,s,j}$ denote the first, last, and j'th states in τ_i respectively. The length of a trajectory in terms of timesteps is denoted as $len(\tau)$, and concatenation of two trajectories is denoted as $\tau_{cat} = \tau_1 \circ \tau_2$. We assume an additive reward function $\mathcal{R} : \mathcal{T} \to \mathbb{R}$ where $\mathcal{R}(\tau) = \sum_{(s,a)\in\tau} r(s,a)$. We call a finite set of trajectories $\mathcal{M} = \{\tau_i\}$ a replay buffer.

[1]Most sub-sections have a corresponding section in the supplementary for further elaboration.

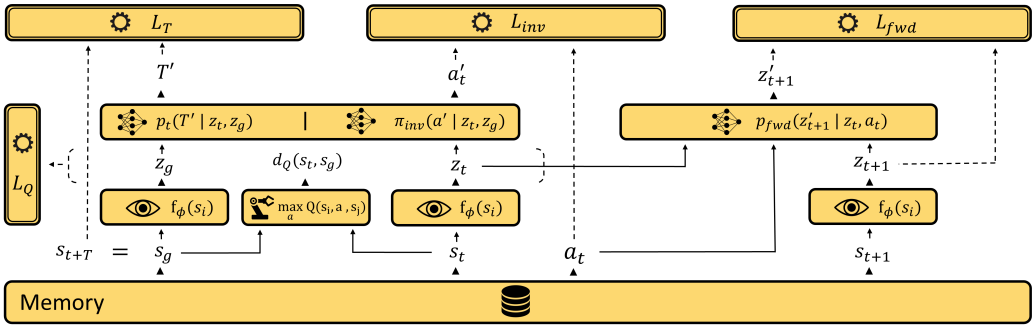

Figure 2: An overview of the functions, inputs, and losses used in our method (see Sec.2.2 for details). We aim to train a perceptual encoder f_ϕ with two properties: i) representations of two states should be close if they were observed to be easily reachable from each other, therefore implementing a form of *visual memory*, ii) representation of a state should capture a minimal sufficient statistic to inform an agent when and how it can reach nearby states, therefore implementing a form of *visual feature extraction*.

2.1 Perceptual Representations that Capture Local Reachability

A key component of our framework is a perceptual encoder $f_\phi(s) : \mathcal{S} \rightarrow \mathbb{R}^d$ that maps states into a representation space where L2 distance $d_\phi(s_t, s_g) := \|f_\phi(s_t) - f_\phi(s_g)\|$ captures local reachability. To discuss what local reachability means more rigorously, we follow the work of [18, 12] and define a goal-conditoned reward function $r(s_t, a, s_{t+1}, s_g) = -\mathbb{1}_{s_{t+1} \neq s_g}$ that returns -1 for all timesteps before reaching a goal (i.e., and 0 afterwards). This means goal-conditioned Q-values [18, 19] for the optimal policy correspond to negative shortest-path distances (i.e., $max_a Q(s_t, a, s_g) = V(s_i, s_j) = -len(\tau_{sp})$). We can then define a symmetric distance metric between states as $d_Q(s_c, s_g) := max(\ -V(s_c, s_g), -V(s_g, s_c)\ )$. This in turn corresponds to the two-way consistency criterion proposed in [13]. What we want from $f_\phi(s)$ is for $d_\phi(s_c, s_g)$ and $d_Q(s_c, s_g)$ to roughly correlate.

2.2 Combining Representation Learning with Goal-conditioned Reinforcement Learning

Any perceptual encoder f_ϕ whose latent representations satisfy the local reachability property defined in Sec.2.1 can be used to implement the nearest neighbor retrieval and trajectory stitching mechanisms from upcoming sections 2.3 and 2.4. This section discusses *one possible way* to obtain such a perceptual encoder, by using goal-conditioned Q-values for contrastive representation learning.

We propose a model (depicted in Fig.2) that combines the following standard components from the literature: **i)** $z = f_\phi(s)$, projecting a state into a latent representation; **ii)** $p_{fwd}(z'_{t+1} \mid z_t, a_t)$, modelling the transition distribution induced by $p_{env}(s'|s, a)$ over the latent space $z = f_\phi(s)$, as discussed in [20, 21]; **iii)** $\pi_{inv}(a'_t \mid z_t, z_g)$, defining a distribution of actions to reach a goal state, as discussed in [20, 21, 14]; **iv)** $p_t(T' \mid z_t, z_g)$, modelling the distribution of timesteps necessary to reach a goal state, as discussed in [22]; **v)** $Q(s_t, a_t, s_g)$, a Q-value function that provides local reachability estimates between pairs of states, as discussed in [12, 18].

Following [12, 18], we train $Q(s_t, a_t, s_g)$ over an offline replay buffer $\mathcal{M}$, using hindsight relabelling [23, 18] with a reward function $r(s_t, a, s_{t+1}, s_g) = -\mathbb{1}_{s_{t+1} \neq s_g}$. After training $Q(s_t, a_t, s_g)$ in isolation, we freeze its parameters and use it to define a contrastive loss function [24] L_Q as explained below. We then train the remaining components using the same replay buffer $\mathcal{M}$. We randomly sample a transition (s_t, a_t, s_{t+1}) and a time difference T, and set the goal state as $s_g := s_{t+T}$, as in hindsight relabelling. We then minimize the following losses:

- $L_Q(s_t, s_g) = l_{hinge}(d_\phi(s_t, s_g) - d_p)\ \mathbb{1}_{d_Q(s_t, s_g) \leq c_Q} + l_{hinge}(d_p - d_\phi(s_t, s_g))\ \mathbb{1}_{d_Q(s_t, s_g) \geq c_Q}$, where l_{hinge} is the hinge loss [25]. This contrastive loss dictates that perceptual representations should be close together (i.e., $d_\phi(s_t, s_g) \leq d_p$ holds) if and only if two states are close to each other in terms of reachability (i.e., $d_Q(s_t, s_g) \leq c_Q$ holds). d_p and c_Q are hyperparameters.
- $L_T(T', T)$, $L_{inv}(a'_t, a_t)$, and $L_{fwd}(z'_{t+1}, z_{t+1})$ are MSE and cross-entropy losses [21, 22]. L_T and L_{inv} dictate that perceptual representations should capture enough information to know when and how an agent can reach from one state to another, while L_{fwd} dictates that they should capture only a minimal-sufficient statistic for doing so ([21] presents a more elaborate discussion).

2.3 Perceptual Memory Retrieval (PMR)

Given a perceptual encoder f_ϕ that captures local reachability, we go over all states $s_i \in \mathcal{M}$ in the replay buffer and compute their projections $z_i = f_\phi(s_i)$, which are stored alongside the states themselves. We then employ z_i to implement two retrieval mechanisms from the replay buffer: i) retrieving neighbor states, and ii) retrieving neighbor trajectories.

Given a query state s_c and radius d_p (i.e., the same one used in the contrastive loss L_Q in Sec.2.2), retrieving neighbor states amounts to computing the set $\mathcal{N}_{d_p}(s_c) = \{s_n \mid d_\phi(s_c, s_n) \leq d_p\}$. The d_ϕ computations and comparison operations involved in this computation can be vectorized. Furthermore, computing $\mathcal{N}_{d_p}(s_c)$ is embarrassingly parallelizable across different query states s_c. The number of neighbors $|\mathcal{N}_{d_p}(s_c)|$ of a query state s_c is a good measure of how many times the agent has visited around s_c. This, combined with how cheap and easy it is to compute $|\mathcal{N}_{d_p}(s_c)|$, makes it a good visitation-count that is applicable to both discrete and continuous state spaces.

Given a starting state s_c and a goal state s_g, we can also search the replay buffer for the highest reward trajectory τ that starts from a state $\tau_{s,0}$ in $\mathcal{N}_{d_p}(s_c)$ and ends in a state $\tau_{s,-1}$ in $\mathcal{N}_{d_p}(s_g)$. This corresponds to the following optimization problem:

$$\tau_{\mathcal{M}(s_c,s_g)} := \underset{\tau \in \mathcal{M}}{\arg\max}\, \mathcal{R}(\tau) \quad \text{s.t.} \quad \tau_{s,0} \in \mathcal{N}_{d_p}(s_c)\ ,\ \tau_{s,-1} \in \mathcal{N}_{d_p}(s_g) \tag{1}$$

To find $\tau_{\mathcal{M}(s_c,s_g)}$, we first select those state pairs $(s_i, s_j) \in \mathcal{N}_{d_p}(s_c) \times \mathcal{N}_{d_p}(s_g)$ that come from the same episode. We then take all sequences of transitions $\tau_{ij} = \{s_i, a_i, s_{i+1}, ..., s_{j-1}, a_{j-i}, s_j\}$ that start from s_i and end at s_j. We sort them based on $\mathcal{R}(\tau_{ij})$, and return the trajectory with the highest reward. We call this trajectory retrieval process perceptual memory retrieval (PMR). We note that all operations involved in PMR can be vectorized, and it takes $\sim 8e^{-4}$ seconds to compute $\tau_{\mathcal{M}(s_c,s_g)}$ for each query pair. Again, PMR is embarrassingly parallelizable across different query pairs (s_c, s_g). In following sections, we discuss how PMR can be employed for long-horizon planning.

2.4 'Experiential Planning' Through Trajectory Restitching

Sampling-based planning algorithms such as RRT [26], RRT* [27], and PRM [28] connect points sampled from obstacle-free space with line segments in order to build a planning graph. We instead reimagine them as memory search mechanisms by altering their subroutines so that whenever an edge is created, a trajectory is retrieved from the replay buffer through PMR (eq.1) and stored in that edge. Our new definitions for these subroutines directly mirror the original ones given in [27]:
1) Sampling: Sampling originally returns a point from obstacle free space. We instead return a state s_c from the replay buffer $\mathcal{M}$ using any distribution (e.g., uniform, or based on visitation-counts).
2) Lines and Their Cost: The equivalent of drawing a line segment in our framework is retrieving a trajectory $\tau_{\mathcal{M}(s_c,s_g)}$, and its length and cost are $len(\tau_{\mathcal{M}(s_c,s_g)})$ and $-\mathcal{R}(\tau_{\mathcal{M}(s_c,s_g)})$ respectively.
3) Nearest State and Neighborhood Queries: Given a query point s_i, these subroutines return the closest point or a neighborhood of points within a distance, among a set of vertices $V = \{s_j\}$. We preserve these definitions, and only replace the metric from euclidean distance to $len(\tau_{\mathcal{M}(s_c,s_g)})$.

$$Nearest(V, s_g) := \underset{s_c \in V}{\arg\min}\ len(\tau_{\mathcal{M}(s_c,s_g)})$$
$$Near(V, s_g, r) := \{s_c \in V \mid len(\tau_{\mathcal{M}(s_c,s_g)}) \leq r\}$$

4) Collision Tests: Collision tests originally prevent the sampling and line drawing subroutines from intersecting obstacles. Since we are planning in retrospect, any such undesirable event can be handled during PMR by adjusting the reward function (i.e., if τ has such an event, this reflects on $\mathcal{R}(\tau)$).

Using these subroutines directly in-place of their originals, we reimplement experiential equivalents of PRM, RRT, and RRT*, which we call R-PRM, R-RRT, R-RRT*. We denote the resulting planned trajectory as $\tau_{\mathcal{M}^*(s_c,s_g)}$. Algorithms 1, 2 describe R-PRM as an example, and the supplementary contains descriptions for R-RRT, R-RRT*.

We note two things about our proposed 'experiential planning' algorithms. First, they can optimize any general reward function $\mathcal{R}$. As the number of sampled vertices increases, $\mathcal{R}(\tau_{\mathcal{M}^*(s_c,s_g)})$ gets optimized through dynamic programming (i.e., by minimizing the Bellman error between vertices of the roadmap G), therefore employing the same mechanism as classical sampling-based planning

Algorithm 1 R-PRM (Roadmap Construction)

Input: $f_\phi, \mathcal{M}$
$V \leftarrow \{SampleFree_i\}_{i=1,\ldots,num_vertices}$; $E \leftarrow \emptyset$ ▷ Initialize vertices and edges
for each $s_i \in V$ **do**
  $U \leftarrow Near(V, s_i, r) \setminus \{s_i\}$
  for each $s_j \in U$ **do** ▷ Place PMR trajectories in edges
    $E \leftarrow E \cup \{(s_i, s_j) : \tau_{edge} = \tau_{\mathcal{M}(s_i,s_j)},\ d_{edge} = -\mathcal{R}(\tau_{\mathcal{M}(s_i,s_j)})\}$
return $G = (V, E)$

Algorithm 2 R-PRM (Trajectory Restitching Given the Constructed Roadmap)

Input: $s_c, s_g, G = (V, E), \mathcal{R}(\tau), f_\phi, \mathcal{M}$
for each $s_i \in V$ **do** ▷ Insert s_c and s_g into the PRM graph
  if $len(\tau_{\mathcal{M}(s_c,s_i)}) \leq r$ **then** ▷ Place PMR trajectories in edges
    $E \leftarrow E \cup \{(s_c, s_i) : \tau_{edge} = \tau_{\mathcal{M}(s_c,s_i)},\ d_{edge} = -\mathcal{R}(\tau_{\mathcal{M}(s_c,s_i)})\}$
  if $len(\tau_{\mathcal{M}(s_i,s_g)}) \leq r$ **then**
    $E \leftarrow E \cup \{(s_i, s_g) : \tau_{edge} = \tau_{\mathcal{M}(s_i,s_g)},\ d_{edge} = -\mathcal{R}(\tau_{\mathcal{M}(s_i,s_g)})\}$

$\tau_{stitched} \leftarrow \emptyset$
$\{s_j\} \leftarrow ShortestPath(s_c, s_g, G, \mathcal{R}(\tau))$ ▷ Trajectory stitching by dynamic programming
for $0 < i < |\{s_j\}|$ **do** ▷ Concatenate PMR trajectories along the shortest path
  $\tau_{stitched} \leftarrow \tau_{stitched} \circ \tau_{\mathcal{M}(s_{i-1},s_i)}$
return $\tau_{\mathcal{M}^*(s_c,s_g)} = \tau_{stitched}$

algorithms [27]. Second, they operate on an offline dataset of unlabeled transitions which solely consists of high-dimensional on-board sensory data (e.g. images), *without assuming any auxiliary instrumentation in the environment or oracle information that cannot be sensed by the agent on its own*. They therefore aim to relax the assumptions classical sampling-based planning methods make about what constitutes a model (e.g., replacing a geometric environment model with sensory experience) and what constitutes a state (e.g., enabling search and planning directly over images).

2.5 PALMER: Closing the Perception-Action Loop to Optimize Memory Contents

We iteratively form and execute $\tau_{\mathcal{M}^*(s_c,s_g)}$, and whenever execution is successful, we insert the resulting transitions back into $\mathcal{M}$ to optimize its contents through trial-and-error. This creates the following perception-action loop: **i)** $\mathcal{M}$ with optimized contents is used to train a more accurate $Q(s_t, a, s_g)$, **ii)** a more accurate $Q(s_t, a, s_g)$ creates a more accurate distance metric d_ϕ, **iii)** a better d_ϕ generates better $\tau_{\mathcal{M}^*(s_c,s_g)}$, **iv)** better $\tau_{\mathcal{M}^*(s_c,s_g)}$ result in higher frequencies of successful execution to further optimize $\mathcal{M}$, therefore looping back to step i (see the supplementary for an algorithmic description). This is essentially a generalized policy-iteration process [29], where generating and executing $\tau_{\mathcal{M}^*(s_c,s_g)}$ is equivalent to policy-evaluation, and putting only successfully executed trajectories back into $\mathcal{M}$ and then updating Q and f_ϕ is equivalent to policy improvement.

3 Related work

Self-supervised goal reaching: Our approach is closely related to goal-reaching methods that combine learning-based distance-regression with graph search, particularly Semi-parametric Topological Memory (SPTM) [14] and Search on the Replay Buffer (SoRB) [12], which we compare to in our experiments. The key difference of our approach is that when setting the edges of the planning graph, it retrieves transitions that *actually happened* rather than relying on learned distance regression. This brings two main benefits. First is robustness. Local reachability estimates are susceptible to overestimation when evaluated between pairs of states that are far apart or unreachable. This is because such states rarely occur together and are therefore out of distribution for the distance regression model. This creates 'hallucinated' shortcuts in the planning graph that corrupt shortest path queries [12, 13]. To address this, [14] employs temporally consistent localization and adaptive

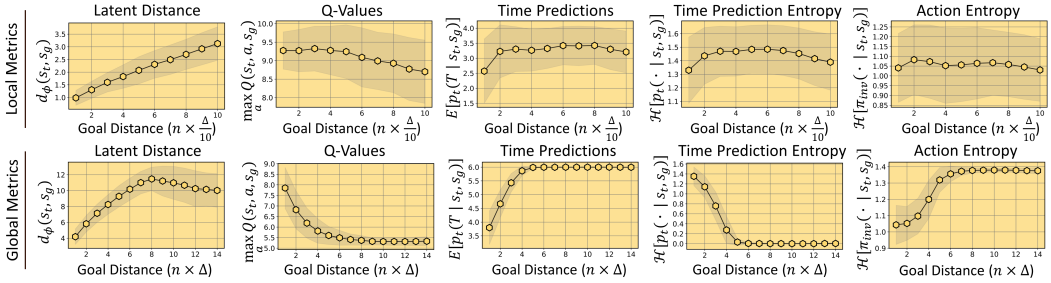

Figure 3: A comparison between perceptual distances d_ϕ and other suitable metrics from Sec.2.2. While all of these metrics are reasonably monotonic with physical reachability (i.e., goal distance), only perceptual distances d_ϕ do not saturate when evaluated locally (i.e., for close by goals). In addition, the ratio between the variance of d_ϕ and the slope of its mean is much smaller compared to other sensible metrics (i.e., d_ϕ has a high signal-to-noise ratio). This means that perceptual distances can implement a more accurate approximate-overlap criterion for perceptual memory retrieval and experiential planning, compared to the other metrics.

waypoint selection, while [12] employs distributional Q-learning and an ensemble of Q-functions. In our approach, eq.1 naturally adresses this problem, since it requires an actual short trajectory in the dataset approximately connecting two states before marking them as close. The second benefit of our approach is that it can optimize general reward functions. This is because it decouples the reachability metric $len(\tau)$ (used in nearest neigbor queries and as a threshold to create edges) from the downstream task reward $\mathcal{R}(\tau)$ (used to set edge distances), unlike previous work.

Image-Based Navigation: [30, 31, 32] present learning-based navigation systems that incrementally build roadmaps through online operation. Our approach has two main differences: **i)** it builds a roadmap entirely using raw offline data, therefore allowing applications like multi-robot learning without additional loop-closure mechanisms to fuse graphs from multiple agents, **ii)** our approach can optimize general reward functions, therefore it is not limited to navigation.

Robot Motion Planning: A common approach to motion planning is to first run a sampling-based planning algorithm [33, 27], and then refine the result through trajectory optimization [34, 35, 36] to satisfy constraints [37, 38, 39]. An important bottleneck is that sampling-based planning algorithms require a precomputed map of the environment, and our approach extends such algorithms in a way that relaxes this requirement by replacing a precomputed map with raw exploration experience.

SLAM and Geometric Maps: SLAM based methods [40] can autonomously construct high-fidelity geometric maps [41, 42], therefore alleviating the bottleneck of precomputing environment maps. The downside of such approaches is that they can abstract away useful physical and semantic affordances. For example, a purely geometric map cannot plan a path through a traversable field of tall-grass, while our approach can learn such affordances as long as they are represented in past experiences.

4 Experiments[2]

Setup: Our experiments are performed in ViZDoom [43], Habitat [44], and the Maze2D benchmark [15]. The VizDoom environment consists of a clover shaped maze. States solely consist of four images $I_{North/East/South/West}$ that form a panorama (i.e., $4 \times 3 \times 160 \times 120$ dimensions), and actions move the agent North/South/East/West by a fixed distance Δ. The maze contains many long-thin column-like obstructions (shown as dots in visualizations). Habitat experiments contain demonstrations on two large-scale scans of real-world apartments: i) Roxboro, with a total area of 62 m2, and ii) Annawan, which has a total-area of $75m2$. States consist of a single 150 FOV image (i.e., $3 \times 256 \times 256$ dimensions). There are 3 actions: $\{turn_left_30_deg, turn_right_30_deg, move_forward_\Delta\}$. Maze2D is a continuous control task, where states consist of the 2D position and velocity of a point mass, and actions correspond to 2D accelerations. In all environments, an offline training dataset is collected by a uniform random walk exploring the environment. For VizDoom and Habitat, this offline training dataset consists of only 300k and 150k timesteps respectively, while for Maze2D there are 1e6 timesteps. Supplementary contains further details.

[2] All experiments have a corresponding section in the supplementary providing further implementation details.

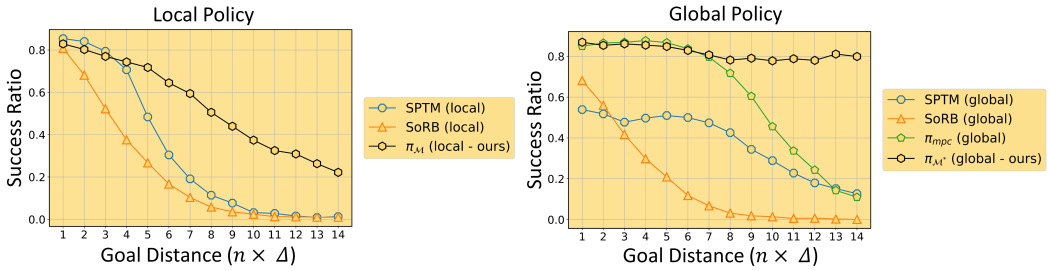

Figure 4: Comparisons of our local policy $\pi_{\mathcal{M}}$ and global policy $\pi_{\mathcal{M}^*}$ with SPTM and SoRB. $\pi_{\mathcal{M}}$ performs well because it avoids getting stuck (as such events are filtered by eq.1), while $\pi_{\mathcal{M}^*}$ performs well because it builds robust roadmaps without hallucinated shortcuts; therefore avoiding the main failure modes of the baselines.

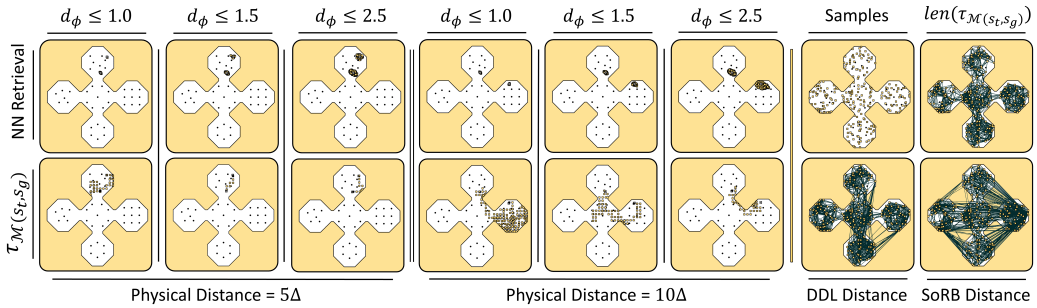

Figure 5: At the core of PALMER is a process called *perceptual memory retrieval* (PMR). Given a query pair of current-goal states, PMR searches the replay buffer to retrieve the highest scoring trajectory $\tau_{\mathcal{M}(s_t,s_g)}$ whose first and last states are close to the query pair according to the perceptual distance d_ϕ. **Left, Middle**: Visualizations of $\tau_{\mathcal{M}(s_t,s_g)}$ retrieved using PMR and nearest neighbor states $\mathcal{N}_{d_p}(s_c)$ retrieved using d_ϕ. **Right**: Setting edges of a roadmap using $len(\tau_{\mathcal{M}(s_t,s_g)})$, compared with distance estimates used in SORB and DDL [22]. We found that distance estimates from baselines are prone to setting false edges that cross map boundaries.

4.1 Experiments in Vizdoom

Validating Perceptual Representations: Fig.3 shows that $d_\phi(s_t, s_g)$ obtained from our model captures a suitable notion of local reachability. Fig.5 in turn shows that retrieving nearest neighbor states $\mathcal{N}_{d_p}(s_c)$ from $\mathcal{M}$ using d_ϕ (i.e., NN retrieval) returns physically close states.

Validating Perceptual Memory Retrieval (PMR): Fig.5 shows visualizations of trajectories retrieved with PMR. We implement a retrieval policy $\pi_{\mathcal{M}}$ that computes $\tau_{\mathcal{M}(s_t,s_g)}$ through eq.1 at each timestep t and executes $argmax_a\ Q(s_t, a, \tau_{\mathcal{M}(s_t,s_g),s,1})$, therefore forming a model predictive control (MPC) loop. We evaluate $\pi_{\mathcal{M}}$ in an image-based navigation task where start/goal images are sampled randomly to have an euclidean distance $n \times \Delta$ in between, and a trial is considered successful if the agent can get within Δ proximity of the goal position within $4 \times n$ time-steps. We use the local policies from SORB [12] and SPTM [14] as baselines. Fig.4 shows the results. The main mode of failure for both SPTM and SORB local policies is that they get stuck in column-like structures. $\pi_{\mathcal{M}}$ avoids this, since eq.1 retrieves collision free $\tau_{\mathcal{M}(s_t,s_g)}$.

Robust Distances: PMR also helps avoid hallucinations in local distance regression. Fig.5 illustrates this point by setting edges between sampled states by thresholding $len(\tau_{\mathcal{M}(s_t,s_g)})$, where methods of [12, 22] are used as baselines. It can be seen that edges set by $len(\tau_{\mathcal{M}(s_c,s_g)})$ are more robust.

Experiential Planning Algorithms: Fig.6 shows visualizations of planning graphs and $\tau_{\mathcal{M}^*(s_c,s_g)}$ produced by R-PRM, R-RRT, and R-RRT*. It can be seen that R-PRM doesn't contain any hallucinated edges, while R-RRT and R-RRT* maintain the visual characteristics of their classical counterparts (i.e., R-RRT has jagged branches with uniform coverage, while R-RRT* has straight branches shooting out from the root). We implement an MPC policy $\pi_{\mathcal{M}^*}$ that replans at each timestep t using Algorithm 2 to return $\tau_{\mathcal{M}^*(s_t,s_g)}$, and executes $argmax_a\ Q(s_t, a, \tau_{\mathcal{M}^*(s_t,s_g),s,1})$. We again use SORB [12] and SPTM [14] as baselines.[3] Fig.4 shows the results. In addition to the local policy

[3]For a comparison without confounders, we train SoRB with DDQN [45] rather than distributional Q-learning [46], and we do not employ temporally consistent localization for SPTM, as such fixes are equally applicable to our method and orthogonal to the discussion. The supplementary provides further elaboration.

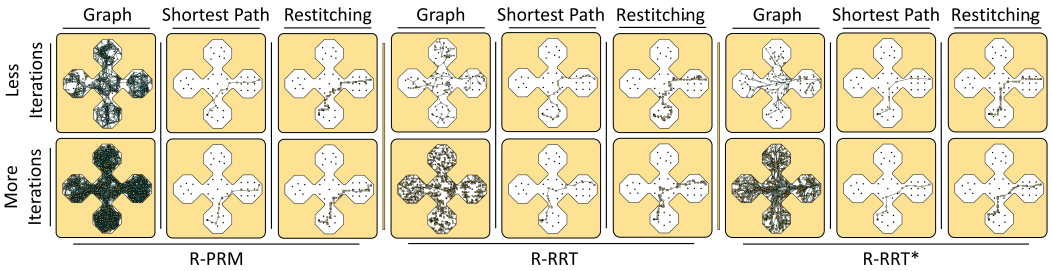

Figure 6: We repurpose conventional sampling-based planning algorithms as memory search mechanisms, by altering their graph building subroutines so that whenever an edge is created a trajectory $\tau_{\mathcal{M}(s_t,s_g)}$ is retrieved through PMR and stored in that edge. We visualize the resulting planning graphs produced by our proposed algorithms R-PRM, R-RRT, R-RRT*.

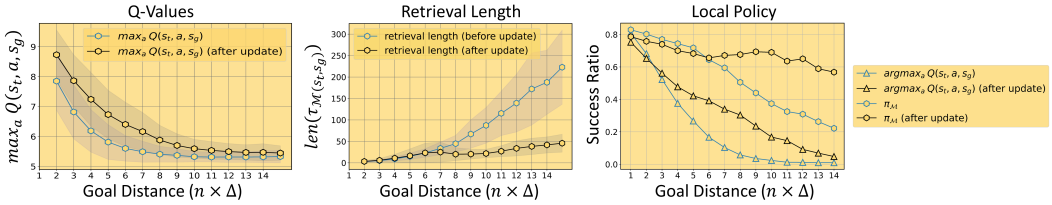

Figure 7: In PALMER, a policy has three groups of parameters: $Q(s_t, a_t, s_g)$, f_ϕ, and the contents of $\mathcal{M}$. Iteratively forming plans through PMR and executing them creates a feedback loop between these components, where: i) *actions inform perception* during the training of f_ϕ, ii) *perception facilitates actions* through the formation roadmaps, and iii) *memory serves as the medium* for this reciprocal interaction. As a result, trajectories produced by explicit planning are gradually internalized as implicit behavior encoded in the model parameters. This leads to: Q-values propagating further into distant goals (**Left**), memory contents getting closer to optimal (**Middle**), and performances of local policies showing significant improvement (**Right**).

getting stuck, a new mode of failure for both baselines is that false distance estimates throw-off graph search by setting hallucinated shortcuts. A new baseline is π_{mpc}, which extends the SPTM local policy by using p_{fwd} and p_t from Sec.2.2 to implement an MPC loop with n-step look-ahead. π_{mpc} avoids getting stuck in columns thanks to n-step lookahead, but still isn't sufficient for global navigation as the accuracy of simulated rollouts from p_{fwd} decreases with the number of timesteps.

Closing the Loop with PALMER: We optimize the contents of $\mathcal{M}$ by iteratively generating and executing $\tau_{\mathcal{M}^*(s_c,s_g)}$. We then retrain all model components only on the resulting new data that is equal in size to the initial unoptimized $\mathcal{M}$. Fig.7 shows the results. When $\pi_{\mathcal{M}}$, and $argmax_a Q(a_t, a, s_g)$ are used as policies, their success ratio increases significantly if they are trained on the optimized $\mathcal{M}$. Q-value estimates trained on the optimized $\mathcal{M}$ also propagate better to goals further away. The scaling of $len(\tau_{\mathcal{M}(s_c,s_g)})$ with goal-distance changes from an exponential trend to an approximately linear one, due to the inclusion of transitions from successfully executed $\tau_{\mathcal{M}^*(s_c,s_g)}$.

4.2 Experiments in Habitat

As shown in Fig.8, we find that our method allows image-based navigation in this new domain with significantly different visuals and layouts (i.e., real-world apartments), action space (i.e., turn-left, turn-right, go-forward), and state space (i.e., single 256×256 RGB images with 150 FOV). Perhaps more surprisingly, we find that training f_ϕ only on exploration data from a *single* apartment generalizes substantially well to any unseen apartment, which directly allows perceptual memory retrieval and experiential planning when provided with a corresponding replay buffer. For a quantitative evaluation, we randomly pick two apartments, named Roxbox and Annawan. In both apartments, we collect an exploration dataset using a uniform random walk sequence of only 150k timesteps. We train the model components solely on data from Roxbox. We then use them to implement our $\pi_{\mathcal{M}^*}$ policy from Sec.4.1, which we then evaluate on both apartments. For $n \in \{8, 16, 24, 32, 36, 44\}$, we randomly sample 100 pairs of start and goal-states in a way that the geodesic distance between them lies within $n \times \Delta$ and $(n + 8) \times \Delta$ through rejection sampling. A policy is considered successful if it can get within $2 \times \Delta$ proximity of the goal-state. We do not include SPTM and SORB as baselines, because we found that the models $\pi_{inv}(a|s_t, s_g)$ and $argmax_a\ Q(s_t, a, s_g)$ that they use as local navigation

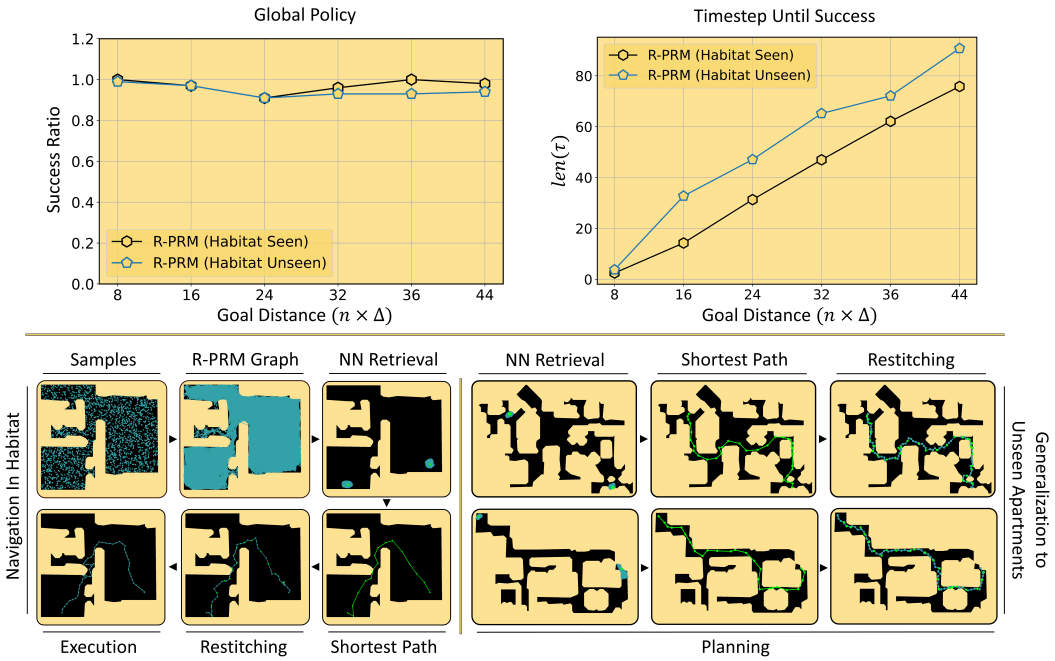

Figure 8: We evaluate our R-PRM based policy $\pi_{\mathcal{M}^*}$ in the Habitat simulator for image-based navigation. **Top Left:** Success ratios in training and test apartments. **Top Right:** Number of timesteps until reaching the goal. ("Habitat seen" refers to the training apartment Roxbox, while "habitat unseen" refers to the test apartment Annawan. **Bottom:** We found that training the perception model f_ϕ on a *single* apartment generalizes sufficiently well to allow perceptual memory retrieval and experiential planning in any unseen apartment.

policies achieved almost zero percent success rate in reaching local goals beyond $\sim 2 \times \Delta$ distance. We empirically observed that most of the time these policies get stuck in repetitive rotational motions without moving forward. This is most likely due to the difficulty of offline RL training with hindsight relabelling over random-walk data obtained with a much more challenging non-cartesian action space $\{turn_left_30_deg, turn_right_30_deg, move_forward_\Delta\}$.

4.3 Experiments in Maze2D

	SAC	SAC-off	BEAR	AWR	BCQ	CQL	IQL	Diffuser	$\pi_{\mathcal{M}^*}$ (Ours)
maze2d-umaze	110.4	145.6	28.6	25.2	41.5	31.7	89.6	**182.1**	131.76
maze2d-medium	69.5	82.0	89.8	33.2	35.0	26.4	105.2	332.9	**416.28**
maze2d-large	14.1	1.5	19.0	70.1	23.2	40	159.9	328.1	**361**

Table 1: Total rewards on the Maze2D benchmark, which is a continuous control task that requires long-horizon planning. Our R-PRM based $\pi_{\mathcal{M}^*}$ policy achieves comparatively strong performance.

To test our method on a continuous control task, we perform additional experiments on the Maze2D benchmark. As shown in Table.1, we find that the same $\pi_{\mathcal{M}^*}$ policy from sections 4.1 and 4.2 achieves strong performance, and can solve mazes of all three complexities.

5 Discussion and Future Directions

Is PALMER less expressive than standard deep Q-learning: Two important premises of deep Q-learning [47, 38] are: i) minimizing Bellman error through temporal-difference (TD) updates can restitch observed transitions in new optimal ways [15, 48], ii) a neural network can learn to extrapolate Q-values to unobserved but close-by states in high-dimensional spaces (e.g. images) [49]. Both arguments are equally valid for our approach, since it can: i) restitch transitions at arbitrary resolutions (i.e., anywhere from one-step transitions to multi-step trajectories) by virtue of sampling-based planning, ii) group together close-by states through d_ϕ. Therefore, PALMER is an RL algorithm that: **i)** optimizes Bellman error through sampling-based optimal planning rather than gradient-based TD-updates [49], **ii)** performs extrapolation between states using a perceptual-backbone f_ϕ rather

than a deep Q-network, and **iii)** replaces the greedy-policy $argmax_a\ Q(s_t, a, s_g)$ and value estimate $max_a\ Q(s_t, a, s_g)$ with $argmax_a\ Q(s_t, a, \tau_{\mathcal{M}^*(s_t,s_g),s,1})$ and $\mathcal{R}(\tau_{\mathcal{M}^*(s_c,s_g)})$ respectively. The key benefits of these alterations come into play when s_t and s_g are far apart, and these benefits are: **i)** the PMR mechanism in eq.1 that prevents hallucinations in $Q(s_t, a, s_g)$, **ii)** global propagation of value estimates by virtue of employing sampling-based planning methods, which are known to be particularly proficient at searching high-dimensional state spaces across long-horizons [39, 33].
Combining PALMER with standard deep Q-learning: Our approach can also be flexibly combined with any traditional Q-learning method [49, 50, 51], by using our retrospective-planning algorithms (Sec.2.4) as experience replay methods [52]. This alternative approach stitches together $\tau_{\mathcal{M}^*(s_c,s_g)}$ during training, and perform backwards TD-updates over this trajectory starting from $s_g = \tau_{\mathcal{M}^*(s_c,s_g),s,-1}$ and ending at $s_c = \tau_{\mathcal{M}^*(s_c,s_g),s,0}$. As suggested by Fig.7, this can allow value estimates $Q(s_t, a, s_g)$ to propagate more globally. Our proof-of-concept experiments identify this as a promising direction, and we leave a further extensive evaluation to future work.

6 Conclusion and Limitations

We presented PALMER, which uses *perceptual representations that correlate with local reachability as a way to explicitly remember previously visited states and trajectories in the replay buffer*. We have shown how this memory retrieval mechanism can be employed to implement 'experiential planning' algorithms, which sit as an interface between any given local controller π and any given offline dataset of state transitions $\mathcal{M}$. Given a goal state s_g and reward function $\mathcal{R}$, these algorithms search the contents of $\mathcal{M}$ to stitch together a sequence of transitions $\tau_{\mathcal{M}^*(s_c,s_g)} = \{s_1, a_1, s_2, ...\}$ that reaches s_g while maximizing $\mathcal{R}$. This sequence is then communicated to π to be executed, which communicates back the resulting real transitions to update the contents of $\mathcal{M}$.

Our experiments show that PALMER can solve long-horizon planning tasks from continuous high-dimensional inputs. In particular, we show that given an offline dataset of only 150k transitions (i.e., compared to sample complexities around the orders of magnitude 1e6-1e7 common in RL) obtained from an entirely uniform random-walk (i.e., which is significantly less structured compared to on-policy rollouts), memory-based experiential planning allows image-based navigation between any two points in large-scale scans of real-world apartments.

The biggest limitation of our current exposition as well as the most immediate direction for future work is to extend our experiments to more complex planning problems [53, 54] and real-world robotic systems [55, 56, 57]. We believe that our memory-based experiential planning perspective highlights a number of interesting questions for future research. First, which transitions should be kept in the replay buffer $\mathcal{M}$, and which ones should be discarded? $\mathcal{M}$ cannot be infinitely expanded after deployment, and it is critical to distill away redundancies between stored experiences. Second, when the environment undergoes a change, which transitions in the replay buffer remain valid and can still be used for planning, and which ones become invalid? A mechanism that can answer this question can allow quick and sample-efficient adaptation to environmental changes. Third, how can we extend f_ϕ to allow more abstract associations and functional equivariances between states? This can improve generalization by defining a more flexible notion of experience retrieval that can recycle past behavior in new contexts and for new tasks. We leave these questions to future work.

References

[1] S. E. Palmer, *Vision science: Photons to phenomenology*. MIT press, 1999.

[2] J. J. Gibson, *The ecological approach to visual perception: classic edition*. Psychology Press, 2014.

[3] J. K. O'regan and A. Noë, "A sensorimotor account of vision and visual consciousness," *Behavioral and brain sciences*, vol. 24, no. 5, pp. 939–973, 2001.

[4] J. D. Co-Reyes, S. Sanjeev, G. Berseth, A. Gupta, and S. Levine, "Ecological reinforcement learning," *arXiv preprint arXiv:2006.12478*, 2020.

[5] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, vol. 299, p. 103535, 2021.

[6] E. L. Thorndike, "Animal intelligence: An experimental study of the associative processes in animals." *The Psychological Review: Monograph Supplements*, vol. 2, no. 4, p. i, 1898.

[7] ——, "The law of effect," *The American journal of psychology*, vol. 39, no. 1/4, pp. 212–222, 1927.

[8] D. T. Campbell, "Perception as substitute trial and error." *Psychological review*, vol. 63, no. 5, p. 330, 1956.

[9] R. J. Herrnstein, "On the law of effect," *Journal of the experimental analysis of behavior*, vol. 13, no. 2, pp. 243–266, 1970.

[10] S. D. Whitehead and D. H. Ballard, "Learning to perceive and act by trial and error," *Machine Learning*, vol. 7, no. 1, pp. 45–83, 1991.

[11] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 1.

[12] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[13] S. Emmons, A. Jain, M. Laskin, T. Kurutach, P. Abbeel, and D. Pathak, "Sparse graphical memory for robust planning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5251–5262, 2020.

[14] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," *arXiv preprint arXiv:1803.00653*, 2018.

[15] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2020.

[16] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *CoRR*, vol. abs/2005.01643, 2020. [Online]. Available: https://arxiv.org/abs/2005.01643

[17] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement learning*. Springer, 2012, pp. 45–73.

[18] L. P. Kaelbling, "Learning to achieve goals," in *IJCAI*, vol. 2. Citeseer, 1993, pp. 1094–8.

[19] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International conference on machine learning*. PMLR, 2015, pp. 1312–1320.

[20] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," *Advances in neural information processing systems*, vol. 29, 2016.

[21] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International conference on machine learning*. PMLR, 2017, pp. 2778–2787.

[22] K. Hartikainen, X. Geng, T. Haarnoja, and S. Levine, "Dynamical distance learning for semi-supervised and unsupervised skill discovery," *arXiv preprint arXiv:1907.08225*, 2019.

[23] 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*, vol. 30, 2017.

[24] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 539–546.

[25] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[26] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning."

[27] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[28] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[29] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[30] D. Shah, B. Eysenbach, N. Rhinehart, and S. Levine, "Recon: Rapid exploration for open-world navigation with latent goal models," *arXiv preprint arXiv:2104.05859*, 2021.

[31] D. Shah and S. Levine, "Viking: Vision-based kilometer-scale navigation with geographic hints," *arXiv preprint arXiv:2202.11271*, 2022.

[32] X. Meng, N. Ratliff, Y. Xiang, and D. Fox, "Scaling local control to large-scale topological navigation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 672–678.

[33] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[34] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[35] ——, *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[36] D. E. Kirk, *Optimal control theory: an introduction*. Courier Corporation, 2004.

[37] R. Tedrake, *Underactuated Robotics*, 2022. [Online]. Available: http://underactuated.mit.edu

[38] P. Abbeel, *Advanced Robotics*, 2019. [Online]. Available: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa19/

[39] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*, 2005.

[40] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[41] M. Bujanca, P. Gafton, S. Saeedi, A. Nisbet, B. Bodin, M. F. O'Boyle, A. J. Davison, P. H. Kelly, G. Riley, B. Lennox *et al.*, "Slambench 3.0: Systematic automated reproducible evaluation of slam systems for robot vision challenges and scene understanding," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6351–6358.

[42] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[43] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE conference on computational intelligence and games (CIG)*. IEEE, 2016, pp. 1–8.

[44] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik *et al.*, "Habitat: A platform for embodied ai research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9339–9347.

[45] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[46] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 449–458.

[47] S. Levine, *Deep Reinforcement Learning*, 2022. [Online]. Available: http://rail.eecs.berkeley.edu/deeprlcourse/

[48] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn *et al.*, "Actionable models: Unsupervised offline reinforcement learning of robotic skills," *arXiv preprint arXiv:2104.07749*, 2021.

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[50] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[51] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[52] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[53] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[54] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," 2018.

[55] X. Meng, N. Ratliff, Y. Xiang, and D. Fox, "Neural autonomous navigation with riemannian motion policy," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8860–8866.

[56] C. Paxton, N. Ratliff, C. Eppner, and D. Fox, "Representing robot task plans as robust logical-dynamical systems," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 5588–5595.

[57] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "Pampc: Perception-aware model predictive control for quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–8.

[58] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: a new approach for hard-exploration problems," *arXiv preprint arXiv:1901.10995*, 2019.

[59] ——, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, 2021.

[60] S. Tian, S. Nair, F. Ebert, S. Dasari, B. Eysenbach, C. Finn, and S. Levine, "Model-based visual planning with self-supervised functional distances," *arXiv preprint arXiv:2012.15373*, 2020.

[61] A. Van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv e-prints*, pp. arXiv–1807, 2018.

[62] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[63] G. Konidaris and A. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," *Advances in neural information processing systems*, vol. 22, 2009.

[64] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Supplementary Material

This document presents additional experimental results, implementation details, visualizations, and conceptual discussions that were excluded or briefed due to space limitations in the main paper. The organization mirrors the sections from the main paper (with the exact same order and numbering), and the exposition generally maintains a question-answer format. It aims to provide significantly more details, at a degree of clarity sufficient for re-implementation. As such, we recommend referring to this document whenever any section in the main paper can benefit from further elaboration. The supplementary material consists of:

- `vizdoom_nav` — navigation videos in VizDoom.
- `habitat_roxbox_nav` — navigation videos in the training apartment Roxbox in Habitat.
- `habitat_annawan_nav` — navigation videos in the test apartment Annawan in Habitat.
- `maze2d_experiments` — goal reaching videos in Maze2D.
- `planning_vis` — videos visualizing the execution of R-RRT and R-RRT*.
- `code` — source code for PALMER.

Table of Contents

1. Introduction

- Why learn a latent distance metric for nearest neighbor retrieval?
- What does stitching transitions together mean?

2. Perception-Action Loop with Memory Reorganization

2.1. Perceptual Representations that Capture Local Reachability and Functional Similarity

- What does local reachability mean?

2.2. Combining Representation Learning with Goal-conditioned Reinforcement Learning

- How are the model components trained, and what are their exact inputs and outputs?
- What do the terms in the contrastive loss-function L_Q mean?

2.3. Combining Representation Learning with Goal-conditioned Reinforcement Learning

- How do we solve the optimization problem in equation 1 from the main paper?
- What do the retrieved trajectories $\tau_{\mathcal{M}(s_c,s_g)}$ look like?

2.4. Trajectory Restitching Through 'experiential planning'

- How does R-PRM work in detail?
- How does R-RRT work in detail?
- How does R-RRT* work in detail?
- What does optimizing the Bellman error on the roadmap mean?
- What does restitching transitions at arbitrary resolution mean?

2.5. PALMER: Closing the Perception-Action Loop to Optimize Memory Contents

- What does optimizing memory contents mean?
- How does the perception-action loop in PALMER work in detail?

3. Related Work

- What is the main reason why memory-based reasoning over actually observed transitions is necessary? Why are methods like SPTM or SoRB that solely rely on learning-based distance estimates are inherently prone to false predictions?

- What are the details for our implementations of SoRB and SPTM?

4. Experiments

Setup

- What are the details for the experimental setup in VizDoom?

Validating Perceptual Memory Retrieval (PMR)

- What is the exact evaluation process that produced Fig.4 in the main paper?

Robust Distances

- What is the exact evaluation process for the right panel of Fig.5 in the main paper?

experiential planning Algorithms

- Why does the policy $\pi_{\mathcal{M}^*}$ use R-PRM for experiential planning?
- What are the details for the π_{mpc} baseline?

Additional Experiments in Habitat and Generalization

- What are the details for the experimental setup in Habitat?
- Additional monte-carlo evaluations in Habitat
- Why does the agent occasionally take random-looking actions in the habitat navigation trials?

5. Discussion and Future Directions

- How is PALMER related to the "Options Framework (Sutton et al.)" and "Skill-Chaining (Konidaris et al.)"?
- How is PALMER related to "LQR-Trees (Tedrake et al.)?

1 Introduction

Why learn a latent distance metric for nearest neighbor retrieval: In a low-dimensional state space such as 3D positions, L2 distance (i.e., euclidean distance) directly correlates with *local* physical-reachability (i.e., we emphasize local, because euclidian distances still do not match geodesic distances globally). Therefore in such state-spaces, grouping together two nearby states and treating them as the same single state for downstream global planning should still result in a feasible planned trajectory. By feasible, we mean that the gaps and approximations introduced by state grouping are functionally inconsequential and can be handled reasonably well by a local policy tracking the global planned trajectory. This property doesn't hold in high-dimensional state spaces such as images, since the L2 distance doesn't correlate with physical reachability. The main purpose of f_ϕ is to project such high-dimensional spaces into a low-dimensional representation space where this property holds, so that nearby states can be fused together to make sampling-based planning computationally tractable.

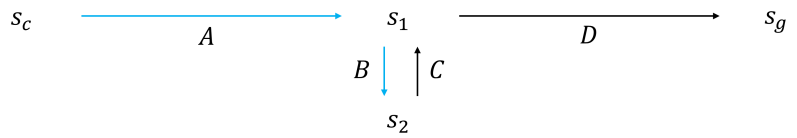

Figure 9: Visualization of stitching together trajectories. If an agent has previous experience of separately going through segments (A, B) and (C, D), it should be able to go from s_c to s_g through the segments (A, D).

What does stitching transitions together mean: As shown in Fig9, if there are two separate sequences of transitions in an offline memory buffer that traverse segments (A, B) and (C, D), an agent should

be capable of going from s_c to s_g through the segments (A, D) event if such a direct path of transitions was never actually observed. Traditional deep Q-learning methods achieve this by combining and propagating value estimates through TD-updates (i.e., $argmax_a Q(s_1, a, s_g)$ points to segment D after TD updates over the path (C, D), therefore the argmax policy would follow the path (A, D) when going from s_c to s_g. [15] provides a further discussion). In our approach, this is achieved by setting edge distances for (A, B, C, D) in a planning graph through perceptual memory retrieval, and then performing a shortest path computation to retrieve the path (A, D).

2 Perception-Action Loop with Memory Reorganization

2.1 Perceptual Representations that Capture Local Reachability

What does local reachability mean: At a high-level (and for the special case of image-based navigation) what the term 'local reachability' intends to convey is that if two images I_1 and I_2 are from physically close positions, $d_\phi = |f_\phi(I_1) - f_\phi(I_2)|$ should be small. This in turn provides a metric for grouping together states that is better than the L2 distance in image space, which has no correlation with physical reachability. Such a metric is necessary to make search and sampling-based planning planning over the memory buffer computationally tractable. This learned metric d_ϕ serves the exact same purpose as the hand-crafted image compression criterion employed in [58, 59] to initialize cells from states.

2.2 Combining Representation Learning with Goal-conditioned Reinforcement Learning

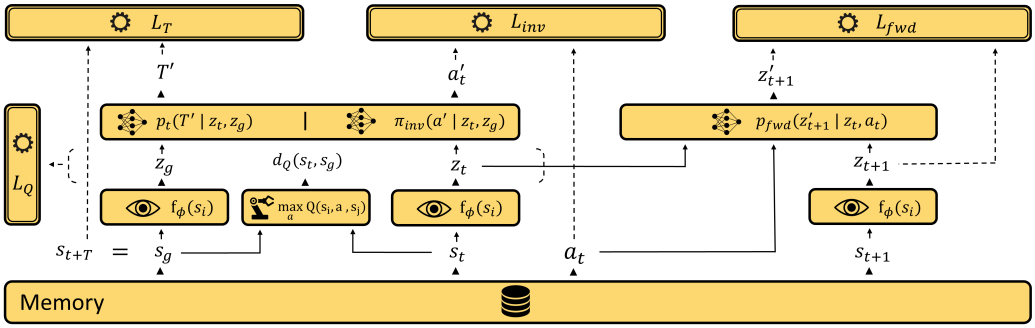

Figure 10: Visualization of the model architecture, reproduced here for ease of reference.

How are the model components trained, and what are their exact inputs and outputs: More detailed descriptions for all model components are given below (Fig.10 presents a visualization):

- The architecture and training of $Q(s_t, a_t, s_g)$ is completely decoupled from the other components. It consists of cascaded convolutional and fully-connected layers with batch normalization and ReLU activations between each layer. It takes as input the concatenated images for current and goal states (i.e., shape $B \times C \times H \times W$), and outputs a vector of Q-values for each action (i.e., shape $B \times num_actions$). It is trained through offline DDQN [45] with hindsight goal-relabelling [23]. We first sample $t \sim Uniform(0, dataset_size)$ and $T \sim Geom(p)$, and then retrieve from the replay buffer a transition and a goal state as $(s_t, a_t, s_{t+1}, s_g := s_{t+T})$. We then minimize the TD error $[\, Q_\theta(s_t, a_t, s_g) - (\mathbb{1}_{s_{t+1}=s_g} + \gamma \mathbb{1}_{s_{t+1} \neq s_g}\, max_a\, Q_{\theta^-}(s_{t+1}, a, s_g))\,]^2$, as in [60].
- The perceptual backbone $f_\phi(s)$ uses a standard Resnet-18 architecture. It takes as input the images for a given state (i.e., $B \times C \times H \times W$), and outputs a low-dimensional representation vector $z = f_\phi(s)$ (i.e., shape $B \times D$). All other components take as input these low-dimensional representations, rather than operating over images.
- $p_{fwd}(z'_{t+1} \mid z_t, a_t)$, $\pi_{inv}(a'_t \mid z_t, z_g)$, and $p_t(T' \mid z_t, z_g)$ all consist of fully-connected layers with ReLU activations. To train them, we first sample $t \sim Uniform(0, dataset_size)$. We then sample T according to $T \sim Uniform(0, T_{max})$ or $T \sim Uniform(T_{max}, dataset_size - t)$, half the time from the former distribution, half the time from the latter. We retrieve from the replay buffer a transition and a goal state as $(s_t, a_t, s_{t+1}, s_g := s_{t+T})$, and project them into low dimensional representations (z_t, a_t, z_{t+1}, z_g) using f_ϕ. These are concatenated and passed

to models $p_{fwd}(z'_{t+1} \mid z_t, a_t), \pi_{inv}(a'_t \mid z_t, z_g), p_t(T' \mid z_t, z_g)$ in a way compatible with their arguments. p_{fwd} outputs the mean for the predicted next state distribution (i.e., shape $B \times D$), and is trained using the MSE loss L_{fwd} with z_{t+1} as the target. π_{inv} outputs a vector of probabilities over actions (i.e., shape $B \times num_actions$), and is trained with the cross-entropy loss L_{inv} with a_t as the target. p_t also outputs a discrete probability distribution over $[0, T_{max}]$ that predicts the distribution of time-steps to reach the goal, where the last bin T_{max} serves as a catch-all for all values above it. It is trained using the cross entropy loss L_T, with T as the target. As mentioned in the main paper, all components $f_\phi, p_{fwd}(z'_{t+1} \mid z_t, a_t), \pi_{inv}(a'_t \mid z_t, z_g), p_t(T' \mid z_t, z_g)$ are trained jointly, with an additional loss function L_Q that regularizes f_ϕ.

What do the terms in the contrastive loss-function L_Q mean: The loss function $L_Q(s_t, s_g) = l_{hinge}(d_\phi(s_t, s_g) - d_p)\ \mathbb{1}_{d_Q(s_t,s_g) \leq c_Q} + l_{hinge}(d_p - d_\phi(s_t, s_g))\ \mathbb{1}_{d_Q(s_t,s_g) \geq c_Q}$ consists of two penalty terms $l_{hinge}(d_\phi(s_t, s_g) - d_p)$ (i.e., only active when $d_\phi \geq d_p$) and $l_{hinge}(d_p - d_\phi(s_t, s_g))$ (i.e., only active when $d_\phi \leq d_p$). These penalty terms are gated through two complementary indicator functions $\mathbb{1}_{d_Q(s_t,s_g) \leq c_Q}$ and $\mathbb{1}_{d_Q(s_t,s_g) \geq c_Q}$. This essentially means that for L_Q to be zero, $d_\phi \leq d_p$ should hold (i.e., perceptual representations are close) if and only if $d_Q(s_t, s_g) \leq c_Q$ holds (i.e., states are physically close). The reason for employing such a conservative switching mechanism with a hinge loss in L_Q (i.e., rather than a continuous penalty term as in [24, 61]) is because Q-value estimates are quite inaccurate (especially when s_c and s_g are far apart), and their exact value is generally unreliable (i.e., they can indicate whether two-states are close sufficiently well, but cannot robustly answer how close they are). To pick the hyperparameter c_Q, we compute the average Q-value between states in the replay buffer that were observed to be within one-step proximity, and use a fraction of this value to as a conservative estimate. While conceptually the choice for the hyperparameter d_p is arbitrary, we heuristically pick it by examining the average d_ϕ distance between subsequent states in the replay buffer, obtained from a preliminary f_ϕ backbone trained without L_Q.

2.3 Perceptual Memory Retrieval (PMR)

How do we solve the optimization problem in equation 1 from the main paper: Our experiments use $-\mathcal{R}(\tau) = len(\tau)$. We first compute the perceptual representations z for all states in the replay buffer, and stack them into a tensor block of shape $(dataset_size, D)$. Given s_c and s_g, we search this tensor with vectorized masking operations to retrieve a set of neighboring states $N(s_c, d_p)$ and $N(s_g, d_p)$ (i.e., sets of states within a perceptual distance threshold d_p of s_c and s_g), to address cons.4. We sort the resulting pairs of states $(s_i, s_j) \in N(s_c, d_p) \times N(s_g, d_p)$ in terms of $j - i$, and filter these pairs using cons.5. We then pick the first (i.e., closest) pair, and return all the states with indices between i, j from the replay buffer as the resulting trajectory $\tau_{\mathcal{M}(s_c,s_g)}$. The important thing to emphasize about all of these operations is that they can be trivially vectorized, and therefore the optimization problem in eq.3-5 can be solved in less time than a forward pass of f_ϕ.

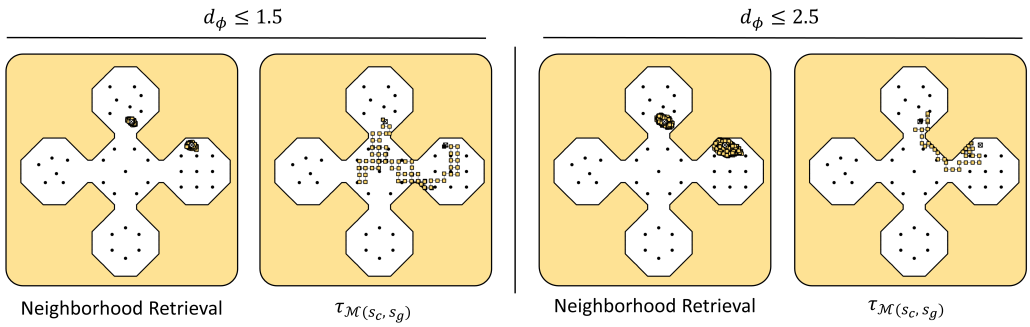

Figure 11: Visualization of retrieved trajectories, with different perceptual distance threshold. Query states s_c and s_g are represented as white squares with a cross in the center, while the start and end points of the retrieved trajectory $\tau_{\mathcal{M}(s_c,s_g)}$ are denoted with a black square and a yellow square with a diagonal dash respectively.

What do the retrieved trajectories $\tau_{\mathcal{M}(s_c,s_g)}$ look like: As the perceptual distance threshold for d_ϕ increases, the physical radii spanned by the nearest neighbor sets $N(s_c, d_p)$ and $N(s_g, d_p)$ increase, as shown in Fig.11. As a result, constraint 4 in the PMR equation gets looser, more trajectories satisfy

constraints 4 and 5 (because their start and end points are allowed to deviate further from the query pair s_c, s_g), and therefore the optimization in equation 3 returns a shorter trajectory $\tau_{\mathcal{M}(s_c,s_g)}$.

2.4 'Experiential Planning' Through Trajectory Restitching

Algorithm 3 Classic PRM (Roadmap Construction)

$V \leftarrow \{SampleFree_i\}_{i=1,\ldots,num_vertices};\ E \leftarrow \emptyset$ ▷ Initialize vertices and edges
for each $s_i \in V$ **do**
  $U \leftarrow Near(V, s_i, r) \setminus \{s_i\}$
  for each $s_j \in U$ **do** ▷ Draw lines as edges
    if $CollisionFree(s_i, s_j)$ **then** $E \leftarrow E \cup \{(s_i, s_j), (s_j, s_i)\}$
return $G = (V, E)$

Algorithm 4 Classic PRM (Shortest-Path Queries Over the Roadmap)

Input: $s_c, s_g, G = (V, E), \mathcal{R}(\tau), f_\phi, \mathcal{M}$
for each $s_i \in V$ **do** ▷ Insert s_c and s_g into the PRM graph
  if $CollisionFree(s_c, s_i)$ **then**
    $E \leftarrow E \cup \{(s_c, s_i), (s_i, s_c)\}$
  if $CollisionFree(s_i, s_g)$ **then**
    $E \leftarrow E \cup \{(s_i, s_g), (s_g, s_i)\}$
return $\{s_j\} \leftarrow ShortestPath(s_c, s_g, G)$ ▷ Shortest path through graph search

Algorithm 5 R-PRM (Roadmap Construction)

Input: $f_\phi, \mathcal{M}$
$V \leftarrow \{SampleFree_i\}_{i=1,\ldots,num_vertices};\ E \leftarrow \emptyset$ ▷ Initialize vertices and edges
for each $s_i \in V$ **do**
  $U \leftarrow Near(V, s_i, r) \setminus \{s_i\}$
  for each $s_j \in U$ **do** ▷ Place PMR trajectories in edges
    $E \leftarrow E \cup \{(s_i, s_j) :\ \tau_{edge} = \tau_{\mathcal{M}(s_i,s_j)},\ d_{edge} = -\mathcal{R}(\tau_{\mathcal{M}(s_i,s_j)})\}$
return $G = (V, E)$

How does R-PRM work in detail: Alg.3, 4 give step-by-step descriptions for the classical PRM algorithm (adapted from [27]), while Alg.5, 6 describe our new definitions for R-PRM. It can be seen that there are two main differences:

- In R-PRM, whenever an edge is created, a trajectory $\tau_{\mathcal{M}(s_c,s_i)}$ is retrieved through perceptual memory retrieval and stored in a field τ_{edge}, while its reward $-\mathcal{R}(\tau_{\mathcal{M}(s_c,s_i)})$ is stored in a different field d_{edge}.
- In PRM the length and cost of a line segment are the same (i.e., euclidian distance), whereas in R-PRM the length of a trajectory $len(\tau_{\mathcal{M}(s_c,s_i)})$ and its reward $-\mathcal{R}(\tau_{\mathcal{M}(s_c,s_i)})$ are decoupled. This means that a shortest path query in R-PRM returns a sequence of nodes and edges that optimize the reward function $\mathcal{R}$. An additional step in R-PRM is that at the end of the shortest-path query, all trajectories $\tau_{edge} = \tau_{\mathcal{M}(s_{i-1},s_i)}$ stored in the returned edges are concatenated into a single trajectory $\tau_{\mathcal{M}^*(s_c,s_g)} = \tau_{stitched}$.

How does R-RRT work in detail: Alg.7 gives a step-by-step description for the classical RRT algorithm (adapted from [27]), while Alg.8 describes our new definition for R-RRT. In addition to the two previous differences between PRM and R-PRM, there is one additional difference between RRT and R-RRT. In RRT, there is a steering sub-routine that draws a line segment of length r starting from $s_{nearest}$ and extending towards s_{rand}, to create a new vertex s_{new}. In R-RRT, this is replaced by retrieving a trajectory $\tau_{\mathcal{M}(s_{nearest},s_{rand})}$ starting from $s_{nearest}$ and ending at s_{rand}, and its r 'th state is used to create the new vertex s_{new}.

Algorithm 6 R-PRM (Trajectory Restitching Given the Constructed Roadmap)

Input: $s_c, s_g, G = (V, E), \mathcal{R}(\tau), f_\phi, \mathcal{M}$
for each $s_i \in V$ **do** ▷ Insert s_c and s_g into the PRM graph
  if $len(\tau_{\mathcal{M}(s_c,s_i)}) \leq r$ **then** ▷ Place PMR trajectories in edges
    $E \leftarrow E \cup \{(s_c, s_i) : \tau_{edge} = \tau_{\mathcal{M}(s_c,s_i)},\ d_{edge} = -\mathcal{R}(\tau_{\mathcal{M}(s_c,s_i)})\}$
  if $len(\tau_{\mathcal{M}(s_i,s_g)}) \leq r$ **then**
    $E \leftarrow E \cup \{(s_i, s_g) : \tau_{edge} = \tau_{\mathcal{M}(s_i,s_g)},\ d_{edge} = -\mathcal{R}(\tau_{\mathcal{M}(s_i,s_g)})\}$

$\tau_{stitched} \leftarrow \emptyset$
$\{s_j\} \leftarrow ShortestPath(s_c, s_g, G, \mathcal{R}(\tau))$ ▷ Trajectory stitching by dynamic programming
for $0 < i < |\{s_j\}|$ **do** ▷ Concatenate PMR trajectories along the shortest path
  $\tau_{stitched} \leftarrow \tau_{stitched} \circ \tau_{\mathcal{M}(s_{i-1},s_i)}$
return $\tau_{\mathcal{M}^*(s_c,s_g)} = \tau_{stitched}$

Algorithm 7 Classic RRT

$V \leftarrow \{s_{init}\};\ E \leftarrow \emptyset$ ▷ Initialize vertices and edges
for $i = 1, ..., n$ **do**
  $s_{rand} \leftarrow SampleFree_i$ ▷ Sample random vertex
  $s_{nearest} \leftarrow Nearest(V, s_{rand})$ ▷ Find the nearest vertex in V
  $s_{new} \leftarrow Steer(s_{nearest}, s_{rand}, r)$ ▷ Draw a line segment of length r
  if $CollisionFree(s_{nearest}, s_{new})$ **then**
    $V \leftarrow V \cup \{s_{new}\}\ ;\ E \leftarrow E \cup \{(s_{nearest}, s_{new}), (s_{new}, s_{nearest})\}$
return $G = (V, E)$

How does R-RRT work in detail:* Alg.9 gives a step-by-step description for the classical RRT* algorithm (adapted from [27]), while Alg.10 describes our new definition for R-RRT*. R-RRT* almost exactly maintains the tree rewiring machinery employed in RRT*, the only difference being that the line costs (i.e., euclidian distance) are replaced with $-\mathcal{R}(\tau_{\mathcal{M}(s_i,s_j)})$ (i.e., in addition to the previous three differences from R-PRM and R-RRT).

What does optimizing the Bellman error on the roadmap mean: Dynamic programming based graph-search algorithms like Dijkstra or (discrete) value-iteration generally employ a cost caching mechanism to iteratively update cost-to-come values, and these updates reduce Bellman error (i.e., the difference between the cost-to-come values before and after the update) during forward-search [33]. R-PRM uses Dijkstra for shortest-path search, while R-RRT* employs the same dynamic programming based tree-rewiring mechanism as the original RRT*, therefore both algorithms are optimizing the Bellman error between the vertices of their graphs through dynamic programming.

What does restitching transitions at arbitrary resolution mean: The retrospective-planning algorithms R-PRM, R-RRT, R-RRT* are sampling based. This means that if two consequent states s_t and s_{t+1} are retrieved during their $SampleFree_i$ routines, then these algorithms will set an edge using $\tau_{\mathcal{M}(s_t,s_{t+1})}$, which is simply the single transition (s_t, a_t, s_{t+1}). Therefore, given enough samples, these algorithms can restitch trajectories down to the level of individual transitions.

2.5 PALMER: Closing the Perception-Action Loop to Optimize Memory Contents

What does optimizing memory contents mean: A replay buffer $\mathcal{M}$ is a collection of trajectories. Optimizing its contents means adding trajectories to $\mathcal{M}$ that achieve higher total reward.

How does the perception-action loop in PALMER work in detail: Alg.11 gives a step-by-step description of the overall perception-action loop implemented in PALMER (where *experiential planning* refers to any of the algorithms R-PRM, R-RRT, R-RRT*). **What PALMER does is essentially bridging any auxiliary exploration method with any auxiliary exploitation method**, by reorganizing exploration experience in $\mathcal{M}$ into $\tau_{\mathcal{M}^*(s_c,s_g)}$ that can be used for exploitation. The particular way in which the trajectories $\tau_{\mathcal{M}^*(s_c,s_g)}$ can be executed for exploitation has a great deal of flexibility, for example: all actions in $\tau_{\mathcal{M}^*(s_c,s_g)}$ can be executed sequentially in an open-loop manner, the first

Algorithm 8 R-RRT

$V \leftarrow \{s_{init}\};\ E \leftarrow \emptyset$ ▷ Initialize vertices and edges
for $i = 1, ..., n$ **do**
  $s_{rand} \leftarrow SampleFree_i$ ▷ Sample random vertex
  $s_{nearest} \leftarrow Nearest(V, s_{rand})$ ▷ Find the nearest vertex in V
  $s_{new} \leftarrow \tau_{\mathcal{M}(s_{nearest}, s_{rand}), r}$ ▷ Get the r 'th state in $\tau_{\mathcal{M}(s_{nearest}, s_{rand})}$
  if $len(\tau_{\mathcal{M}(s_{nearest}, s_{new})}) \leq r$ **then**
    $V \leftarrow V \cup \{s_{new}\}$
    $E \leftarrow E \cup \{(s_{nearest}, s_{new}) :\ \tau_{edge} = \tau_{\mathcal{M}(s_{nearest}, s_{new})},$
      $d_{edge} = -\mathcal{R}(\tau_{\mathcal{M}(s_{nearest}, s_{new})})\}$
return $G = (V, E)$

Algorithm 9 Classic RRT*

$V \leftarrow \{s_{init}\};\ E \leftarrow \emptyset$ ▷ Initialize vertices and edges
for $i = 1, ..., n$ **do**
  $s_{rand} \leftarrow SampleFree_i$ ▷ Sample random vertex
  $s_{nearest} \leftarrow Nearest(V, s_{rand})$ ▷ Find the nearest vertex in V
  $s_{new} \leftarrow Steer(s_{nearest}, s_{rand}, r)$ ▷ Draw a line segment of length r

  if $CollisionFree(s_{nearest}, s_{new})$ **then**
    $S_{near} \leftarrow Near(V, s_{new}, r)$
    $V \leftarrow V \cup \{s_{new}\}$
    $s_{min} \leftarrow s_{nearest}$
    $c_{min} \leftarrow Cost(s_{nearest}) + Cost(Line(s_{nearest}, s_{new}))$

    for each $s_{near} \in X_{near}$ **do** ▷ Connect along a minimum-cost path
      $c_{near} \leftarrow Cost(s_{near}) + Cost(Line(s_{near}, s_{new}))$
      if $CollisionFree(s_{near}, s_{new})$ and $c_{near} \leq c_{min}$ **then**
        $s_{min} \leftarrow s_{near}\ ;\ c_{min} \leftarrow c_{near}$
    $E \leftarrow E \cup \{(s_{min}, s_{new})\}$

    for each $s_{near} \in X_{near}$ **do** ▷ Rewire the tree
      $c_{near,new} \leftarrow Cost(s_{new}) + Cost(Line(s_{new}, s_{near}))$
      if $CollisionFree(s_{new}, s_{near})$ and $c_{near,new} \leq c_{near}$ **then**
        $s_{parent} \leftarrow Parent(s_{near})$
        $E \leftarrow (E \backslash \{(s_{parent}, s_{near})\} \cup \{s_{new}, s_{near}\})$

return $G = (V, E)$

actions of $\tau_{\mathcal{M}^*(s_c, s_g)}$ generated at each timestep can be executed in a model predictive control (MPC) manner, states in $\tau_{\mathcal{M}^*(s_c, s_g)}$ can be tracked by an auxiliary local feedback controller, or the entirety of $\tau_{\mathcal{M}^*(s_c, s_g)}$ can be used to initialize a separate trajectory optimization method.

3 Related Work

What is the main reason why memory-based reasoning over actually-observed transitions is necessary? Why are methods like SPTM or SoRB that solely rely on learning-based distance estimates are inherently prone to false predictions: By definition, states that are far apart from each other in-terms of physical reachability are rarely observed together in an experiential learning framework (e.g., within the same RL episode, or within close by time-steps during random exploration). Therefore, for any learning-based prediction model that is conditioned on current-goal state pairs (e.g., $Q(s_t, a, s_g),\ \pi_{inv}(a'|s_t, s_g),\ p_t(T'|s_t, s_g)$), if it is trained solely on the observed distribution of experiential data, far apart states will be out of distribution (i.e., they have a low probability of being sampled from the experiential data distribution, therefore they are underrepresented in the replay buffer). This means that predictions for such far apart states will inevitably be inaccurate, unless a

Algorithm 10 R-RRT*

$V \leftarrow \{s_{init}\};\ E \leftarrow \emptyset$ ▷ Initialize vertices and edges
for $i = 1, ..., n$ **do**
  $s_{rand} \leftarrow SampleFree_i$ ▷ Sample random vertex
  $s_{nearest} \leftarrow Nearest(V, s_{rand})$ ▷ Find the nearest vertex in V
  $s_{new} \leftarrow \tau_{\mathcal{M}(s_{nearest}, s_{rand}), r}$ ▷ Get the r 'th state in $\tau_{\mathcal{M}(s_{nearest}, s_{rand})}$

  if $len(\tau_{\mathcal{M}(s_{nearest}, s_{new})}) \leq r$ **then**
    $S_{near} \leftarrow Near(V, s_{new}, r)$
    $V \leftarrow V \cup \{s_{new}\}$
    $s_{min} \leftarrow s_{nearest}$
    $c_{min} \leftarrow Cost(s_{nearest}) + -\mathcal{R}(\tau_{\mathcal{M}(s_{nearest}, s_{new})})$

    for each $s_{near} \in X_{near}$ **do** ▷ Connect along a minimum-cost path
      $c_{near} \leftarrow Cost(s_{near}) + -\mathcal{R}(\tau_{\mathcal{M}(s_{near}, s_{new})})$
      if $len(\tau_{\mathcal{M}(s_{near}, s_{new})}) \leq r$ and $c_{near} \leq c_{min}$ **then**
        $s_{min} \leftarrow s_{near}$; $c_{min} \leftarrow c_{near}$
    $E \leftarrow E \cup \{(s_{min}, s_{new}) :\ \tau_{edge} = \tau_{\mathcal{M}(s_{min}, s_{new})}, d_{edge} = -\mathcal{R}(\tau_{\mathcal{M}(s_{min}, s_{new})})\}$

    for each $s_{near} \in X_{near}$ **do** ▷ Rewire the tree
      $c_{near,new} \leftarrow Cost(s_{new}) + -\mathcal{R}(\tau_{\mathcal{M}(s_{new}, s_{near})})$
      if $len(\tau_{\mathcal{M}(s_{new}, s_{near})}) \leq r$ and $c_{near,new} \leq c_{near}$ **then**
        $s_{parent} \leftarrow Parent(s_{near})$
        $E \leftarrow (E \backslash \{(s_{parent}, s_{near})\} \cup \{s_{new}, s_{near}\})$

return $G = (V, E)$

hard-negative sampling mechanism is implemented to explicitly push their reachability-estimates lower. The problem with this is that there is no inherent signal solely contained in perceptual input (e.g., images) that can guide the resampling process in a way that is generally applicable to all tasks.

For example, [60] employs a hard-negative sampling mechanism for a manipulation task using joint pose labels for guidance, but such a mechanism has two bottlenecks: i) it is specific to their particular manipulation task, ii) it assumes auxiliary labels. Similarly, the temporally consistent localization mechanism used in SPTM is essentially a heuristic fix specific to navigation. For the case of SoRB, there is no inherent mechanism in distributional RL that addresses this out-of-distribution issue either. While employing an ensemble of Q-functions could potentially capture the epistemic uncertainty for out-of-distribution pairs to directly address this problem, we used an ensemble of Q-functions in our implementation of SoRB and empirically observed that it was insufficient. A similar conclusion can be drawn from the Fig.8 of the original SoRB paper [12], as the bulk of the performance increase appears to be due to distributional RL, and ensembles only provide a moderate benefit.

All of these considerations highlight the importance of memory as a robustification mechanism. To summarize the discussions from above, there are two main reasons that cause false reachability predictions: i) there is no robust and general signal solely contained in the isolated instances of perceptual input (s_c, s_g) (i.e., without the trajectory of states in between that connect them) that can identify whether two states are physically far apart, ii) both physically close and far apart states can occur with a long temporal distance in between. Therefore, an agent needs to rely on memory: in order to identify whether two states are close or not, it should try to remember if it ever actually observed those two states close-by in a segment of past experience.

What are the details for our implementations of SoRB and SPTM: Tha main difference of our SPTM implementation is that it doesn't employ temporally consistent localization and adaptive waypoint selection. The main differences of our SoRB implementation are: i) we use an ensemble of Q-functions, but they are trained with DDQN rather than distributional RL, ii) we train the Q-function on offlline random-walk data, rather than an online episodic training setup with resets and a reward oracle as employed in the original paper. We acknowledge and emphasize that for SoRB, these differences are the most likely reason for the lower performance level we observed in our evaluations

Algorithm 11 PALMER: Perception-Action Loop with Memory Retrieval

Input: $\mathcal{R}(\tau)$
$f_\phi.init(), Q.init(), \mathcal{M} \leftarrow \emptyset$ ▷ Initialize policy parameters
while $t \leq max_timestep$ **do**

  while $i \leq num_exploration_steps$ **do** ▷ Exploration
    i) Using [*any suitable method*]: explore the environment to obtain an exploration trajectory τ_{new}
    ii) Using [τ_{new}]: **update** $\mathcal{M}$ ▷ Memory expansion

  while $i \leq num_updates$ **do**
    Using [$\mathcal{M}$]: **update** $Q(s_t, a_t, s_g)$ ▷ Train value function
    Using [$\mathcal{M}$ and $Q(s_t, a_t, s_g)$]: **update** $f_\phi(s_t, s_g)$ ▷ Train perception model

  while $i \leq num_exploitation_steps$ **do** ▷ Exploitation
    i) Using [$\mathcal{M}$]: sample a random goal $s_g \sim \mathcal{M}$
    ii) Using [$f_\phi(s_c, s_g)$, $\mathcal{R}(\tau)$, and *experiential planning*]: generate $\tau_{\mathcal{M}^*(s_c,s_g)}$
    iii) Using [*any suitable method*]: execute $\tau_{\mathcal{M}^*(s_c,s_g)}$ to obtain a real trajectory τ_{real}
    iv) Using [τ_{real}]: **update** $\mathcal{M}$ ▷ Memory optimization

compared to the original paper, as they inevitably reduce the accuracy of Q-values. We however note that our method also employs the same Q-values, and generally these implementation differences in the baselines were chosen to facilitate a clear understanding of our approach without confounders, because: i) they do not directly address the root cause of the false prediction problem (as discussed above), ii) one of the main benefits of our method is that it operates over arbitrary offline data without any resets or reward oracles (and it uses DDQN to train the related Q-function).

4 Experiments

Setup

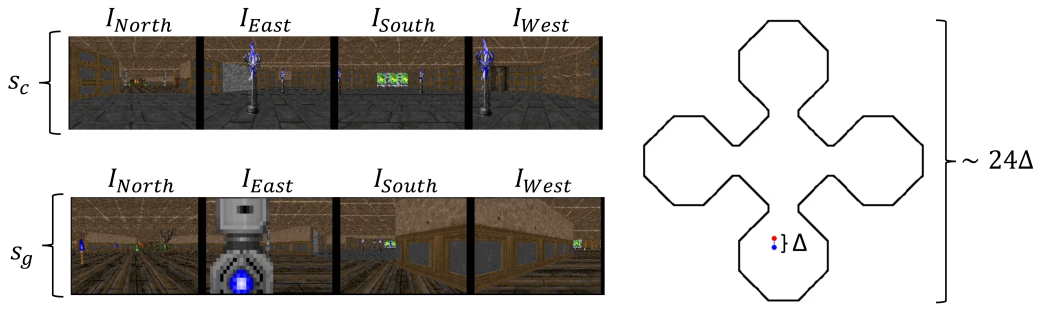

Figure 12: Visualization of the map used in VizDoom experiments. **Further video visuals of image-based navigation can be found in the folder vizdoom_nav provided in the supplementary alongside this document**.

What are the details for the experimental setup in VizDoom: As shown in Fig.12, states solely consist of four images $I_{North/East/South/West}$ that form a panorama (i.e., $4 \times 3 \times 160 \times 120$ dimensions), and actions move the agent North/South/East/West by a fixed distance Δ. The geodesic distances scale approximately by a factor of $\times 3$ compared to euclidian distances (e.g., If a goal has an euclidian distance of 14Δ, it takes approximately 42 timesteps for an optimal policy to reach it). The map contains many long-thin column-like obstructions (e.g., torches, pillars, trees), as we found that image-based navigation policies are prone to getting stuck in such obstacles. These obstacles have dynamically changing appearances (e.g., flickering flames on torches, glowing lights on pillars), and can completely block the field of view of the agent after a collision (as shown in I_{East} in Fig.12). The replay buffer $\mathcal{M}$ consists of 300k images obtained from a uniform random walk exploring the map in a single continuous sequence of actions, without resets and rewards.

Validating Perceptual Memory Retrieval (PMR)

What is the exact evaluation process that produced Fig.4 in the main paper: For every integer value $n \in [0, 14]$, we randomly sample 1000 pairs of start and goal-states in a way that the euclidian distance between them lies within $n \times \Delta$ and $(n+1) \times \Delta$ through rejection sampling, and a policy is considered successful if it can get within Δ proximity of the goal-state.

Robust Distances

What is the exact evaluation process for the right panel of Fig.5 in the main paper: To produce the roadmap visualizaitons, we randomly sample 250 states from the replay buffer and set the edges between them by thresholding the distance estimates from all methods. Thresholds were calibrated individually and by hand for each baseline, by picking the threshold with the lowest number of false edges until a further reduction in the threshold resulted in splitting the roadmap into a large number of isolated subgraphs (i.e., therefore making it impossible to use it for global planning).

Experiential Planning Algorithms

Why does the policy $\pi_{\mathcal{M}^*}$ use R-PRM for experiential planning: PRM and R-PRM are multi-query methods [27], meaning that the full roadmap only needs to be constructed once. For any query pair of current-goal states (s_c, s_g), the same roadmap can be reutilized by inserting (s_c, s_g) in the roadmap and performing a shortest path query. In contrast, RRT and RRT* require recreating a full roadmap for every new query pair. Since $\pi_{\mathcal{M}^*}$ replans at each timestep with a different current state s_t, PRM based approaches are computationally much cheaper. We also note that planning graphs for all methods in this experiment contain 500 vertices.

What are the details for the π_{mpc} baseline: The π_{mpc} policy uses the p_{fwd} model to obtain simulated rollouts, and uses the p_t model to rank those rollouts in terms of how close they get to the goal. This allows an MPC optimization loop that picks and implements the first action from the most successful simulated rollout. As previously discussed above, the main bottleneck for SPTM and SoRB is the difficulty of estimating reachability metrics solely using the two states s_c, s_g without any consideration of the states in between. Simulated rollouts in π_{mpc} naturally address this problem by generating and evaluating entire trajectories. The main bottleneck for π_{mpc} is that the accuracy of state predictions in simulated rollouts from p_{fwd} degrade with the rollout length.

Additional Experiments in Habitat and Generalization

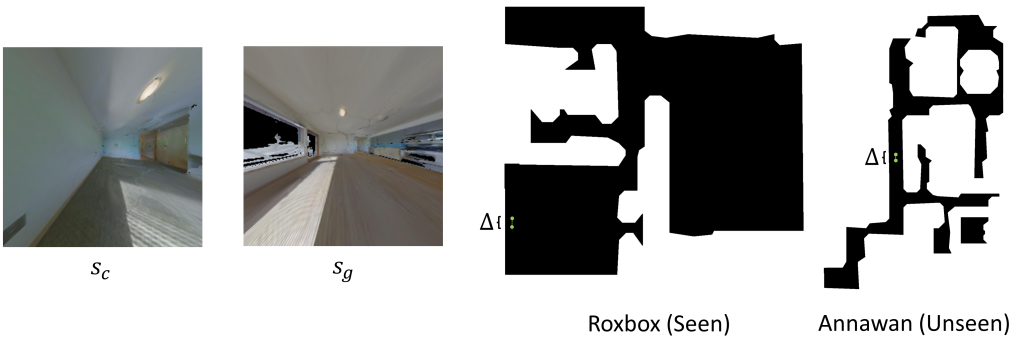

Figure 13: Visualization of the apartments used in Habitat experiments. **Further video visuals of image-based navigation can be found in the folders habitat_roxbox_nav and habitat_annawan_nav provided in the supplementary alongside this document**.

What are the details for the experimental setup in Habitat: As shown in Fig.13, states solely consist of a single 150 FOV image (i.e., $3 \times 256 \times 256$ dimensions). There are 3 actions: $\{turn_left_30_deg, turn_right_30_deg, move_forward_\Delta\}$. We run evaluations on two randomly picked apartments: Roxbox, and Annawan. In both apartments, we collect a replay buffer $\mathcal{M}$ that consists of 150k images obtained from a uniform random walk exploring the map in a single

continuous sequence of actions, without resets and rewards. We use only the memory buffer from Roxbox to train the perception model f_ϕ, and use this same model to do perceptual memory retrieval and experiential planning on replay buffers from both apartments. We have observed that the latent distances from f_ϕ generalize well, and can directly allow perceptual memory retrieval and experiential planning without any fine-tuning on the images from the new test apartment.

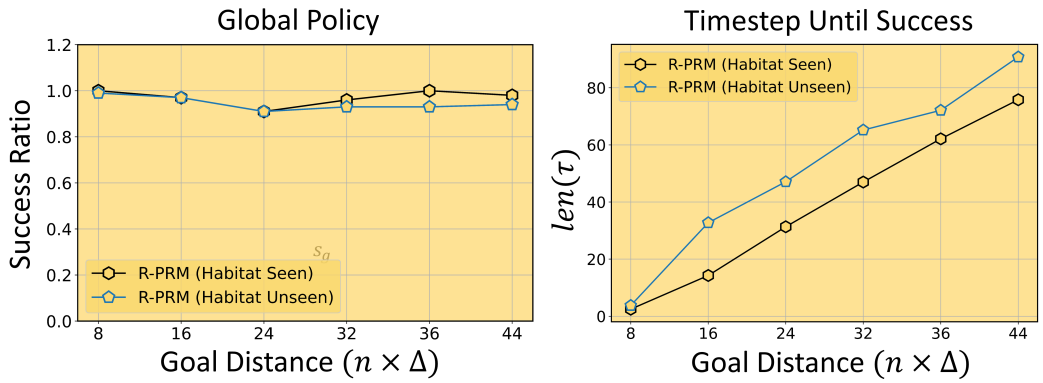

Figure 14: We evaluate our R-PRM based policy $\pi_{\mathcal{M}^*}$ in the Habitat simulator for image-based navigation. We find that training the perception model f_ϕ on a single apartment generalizes well to test apartments. **Left:** Success ratios in training and test apartments. **Right:** Number of timesteps until reaching the goal. ("Habitat seen" refers to the training apartment Roxbox, while "habitat unseen" refers to the test apartment Annawan.

Additional monte-carlo evaluations in Habitat: Following up on the corresponding section from the main paper, Fig.14 plots the performance metrics for our $\pi_{\mathcal{M}^*}$ policy in the Habitat simulation environment. In this evaluation, for $n \in \{8, 16, 24, 32, 36, 44\}$, we randomly sample 100 pairs of start and goal-states in a way that the geodesic distance between them lies within $n \times \Delta$ and $(n+8) \times \Delta$ through rejection sampling. A policy is considered successful if it can get within $2 \times \Delta$ proximity of the goal-state. We see that our method can allow image-based navigation in this new and more challenging domain. We do not include SPTM and SORB as baselines, because we found that the models $\pi_{inv}(a|s_t, s_g)$ and $argmax_a\ Q(s_t, a, s_g)$ that they use as local navigation policies achieved almost zero percent success rate in reaching local goals beyond $\sim 2 \times \Delta$ distance. We empirically observed that most of the time these policies get stuck in repetitive rotational motions without moving forward. This is most likely due to the difficulty of offline RL training with hindsight relabelling over random-walk data obtained with a much more challenging non-cartesian action space $\{turn_left_30_deg, turn_right_30_deg, move_forward_\Delta\}$.

Why does the agent occasionally take random-looking actions in the habitat navigation trials: This is due to a combination of two main factors. First, our MPC loop replans from scratch at each timestep using Algorithm.6. This frequent replanning has a destabilizing effect on the control loop, similar to employing a large derivative action in a PID controller (i.e., a strong anticipatory term causes frequent switches in the actions). This first factor is exacerbated by the second main factor: the poor performance of $argmax_a\ Q(s_t, a, s_g)$ as mentioned above. The restitched trajectories $\tau_{\mathcal{M}^*(s_t,s_g),s,1})$ produced by R-PRM at each time-step are converted to actions by following their first state $\tau_{\mathcal{M}^*(s_t,s_g),s,1}$ using $argmax_a\ Q(s_t, a, \tau_{\mathcal{M}^*(s_t,s_g),s,1})$, and inaccurate Q-values occasionally cause random-looking actions.

There are multiple ways to counter this. The most direct way is to train a better Q-function. Our current Q-function is trained in a particularly challenging setting, as: i) it is trained on entirely offline data with hindsight goal relabelling, ii) this data is collected through uniformly random actions, and iii) it consists of only 150k environment steps. A second way is to instead counter the large derivative action by reducing the replanning rate and introducing a momentum mechanism to the controller. For example, we can replan through R-PRM only every n'th timestep (i.e., hence reducing the replanning rate), and act according to $argmax_a\ Q(s_t, a, \tau_{\mathcal{M}^*(s_t,s_g),s,n})$ for the timesteps inbetween (i.e., hence introducing momentum to the controls). We couldn't get this fix to work well, because the Q-values are only accurate up to states that are $\sim 2 \times \Delta$ distance away (hence acting according to $argmax_a\ Q(s_t, a, \tau_{\mathcal{M}^*(s_t,s_g),s,n})$ isn't possible for $n \geq 2$ in our case). We note however that our method can still navigate from any point to any point in a challenging 3D reconstruction of a

real-world apartment in Habitat using poor Q-value estimates, highlighting the robustness introduced by memory-based planning.

5 Discussion and Future Directions

How is PALMER related to the "Options Framework (Sutton et al.)" and "Skill-Chaining (Konidaris et al.)": The idea of restitching (i.e., chaining) transition sequences from a replay buffer has direct connections to the options framework [62] and skill-chaining [63]. Essentially, PALMER can be thought of as a framework for converting every possible sequence of transitions $\tau \in \mathcal{M}$ in memory into an option $o = \{\pi_o, I_o, \beta_o\}$, where the option policy π_o is implemented by simply executing all the actions in τ in an open-loop manner, the initiation set is $I_o = \{s \in \mathcal{S} : d_\phi(s, \tau_{s,0}) \leq d_p)\}$, and the termination condition is $\beta_o = \{s \in \mathcal{S} : d_\phi(s, \tau_{s,-1}) \leq d_p)\}$. Therefore, PALMER can be interpreted as a skill-chaining algorithm that converts unstructured transitions in a replay buffer into a set of executable options to be chained.

How is PALMER related to "LQR-Trees (Tedrake et al.)": A central idea in PALMER is repurposing the edge creation subroutines of sampling-based planning algorithms so that whenever an edge is created some additional processing is done to connect the endpoints (i.e., particularly, perceptual memory retrieval in our case). This approach is directly inspired by the method of LQR-Trees [64], which instead creates a trajectory stabilizing LQR controller to connect the endpoints of each edge. This results in a roadmap of local controllers, rather then a roadmap of memories as in PALMER.