PALMER: Perception-Action Loop with Memory for Long-Horizon Planning

Onur Beker, Mohammad Mohammadi, Amir Zamir
School of Computer and Communication Sciences
Swiss Federal Institute of Technology (EPFL)

Abstract: To achieve autonomy in a priori unknown real-world scenarios, agents should be able to: i) act from high-dimensional sensory observations (e.g., images), ii) learn from past experience to adapt and improve, and iii) be capable of long horizon planning. Classical planning algorithms (e.g. PRM, RRT) are proficient at handling long-horizon planning. Deep learning based methods in turn can provide the necessary representations to address the others, by modeling statistical contingencies between observations. In this direction, we introduce a general-purpose planning algorithm called PALMER that combines classical sampling-based planning algorithms with learning-based perceptual representations. For training these representations, we combine Q-learning with contrastive representation learning to create a latent space where the distance between the embeddings of two states captures how easily an optimal policy can traverse between them. For planning with these perceptual representations, we re-purpose classical sampling-based planning algorithms to retrieve previously observed trajectory segments from a replay buffer and restitch them into approximately optimal paths that connect any given pair of start and goal states. This creates a tight feedback loop between representation learning, memory, reinforcement learning, and sampling-based planning.

Keywords: Representation Learning, Memory, Planning, Reinforcement Learning

1 Introduction[1]

Animals and humans operate on high-dimensional stimuli (e.g., vision) to achieve diverse and ever-changing goals necessary for their survival [1, 2, 3, 4, 5]. 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 by trying every possible action for achieving every possible goal is intractable. The complexity of this search motivates memory-based mechanisms for compositional thinking. Examples of such mechanisms include : i) remembering relevant segments of past experience, ii) recomposing them in new counterfactual ways to form plans, and iii) executing such plans as part of a targeted search strategy. Such mechanisms for recycling past successful behavior can significantly accelerate trial-and-error compared to uniformly sampling all possible actions. This is because the same behavior (i.e., sequence of actions) can remain valid for different goals and in different contexts, due to the inherent compositional structure of real-world goals as well as the commonality of the physical laws that govern real-world environments.

Contribution: We describe a long-horizon planning method that directly operates on high dimensional sensory input observable by an agent on its own (e.g., images from an onboard camera). Our method combines classical sampling-based planning algorithms with learning-based perceptual representations, to retrieve and recompose previously observed sequences of state transitions in a replay buffer. This is enabled by a two-step process. *First*, we learn a latent space where the distance between two states captures how many timesteps it takes for an optimal policy to go from one to the

[1]This paper has been accepted for publication at NeurIPS 2022.

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

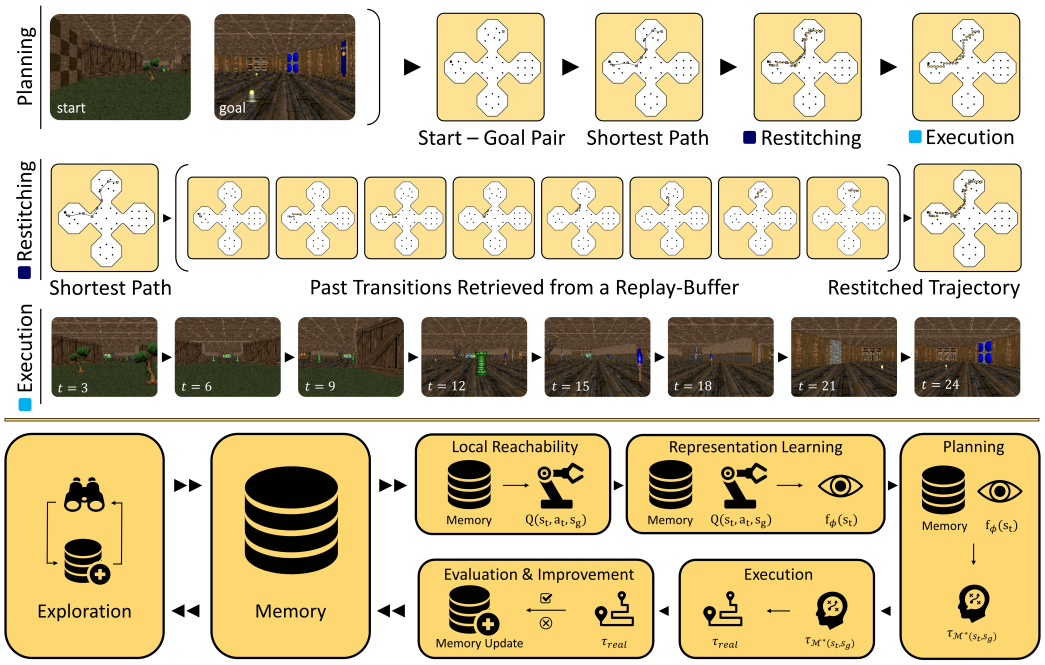

Figure 1: **Top**: Given a start-goal image pair, PALMER plans a path between them by concatenating the endpoints of past trajectory segments retrieved from a provided replay buffer. This is enabled by a state embedding function f_ϕ that can identify close-by states, and results in robust long-horizon planning. **Bottom**: To achieve this : i) it uses offline Q-learning to obtain *local reachability* estimates between states, ii) uses these Q-values for *representation learning* to train f_ϕ, iii) uses f_ϕ to *plan* over the replay buffer, iv) *executes* these plans, v) *evaluates* the resulting trajectories and inserts them back into the replay buffer to *improve* its contents.

other. To achieve this, we use goal-conditioned Q-values learned through offline hindsight relabelling [11] for contrastive representation learning. *Second*, we threshold this learned latent distance metric to define a neighborhood criterion between states. We then define sampling-based planning algorithms that search over the replay buffer [12] to retrieve and stitch together trajectory segments (i.e., past sequences of observed transitions) whose endpoints are neighboring states. This trajectory stitching approach allows for creating planning graphs to connect any pair of start and goal states that were observed before (as depicted in Fig.1). Our approach operates on offline unlabeled data, and can therefore be combined with any exploration method to populate the replay buffer. Our experiments implement 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

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. τ_0 , τ_{-1} , τ_i denote the first, last, and i'th states in τ 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.

2.1 Perceptual Representations that Capture Local Reachability

A key component of our framework is a perceptual encoder $f_\phi(s) : \mathcal{S} \to \mathbb{R}^d$ 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 (i.e., how many timesteps it takes for the optimal policy to go from one state to another). To discuss this more rigorously, we follow the work of [13, 12] and define a goal-conditioned reward function $r(s_t, a, s_{t+1}, s_g) = -\mathbb{1}_{s_{t+1} \neq s_g}$ that returns -1 for all steps before reaching a goal. This means goal-conditioned Q-values for the optimal policy correspond to negative shortest-path distances

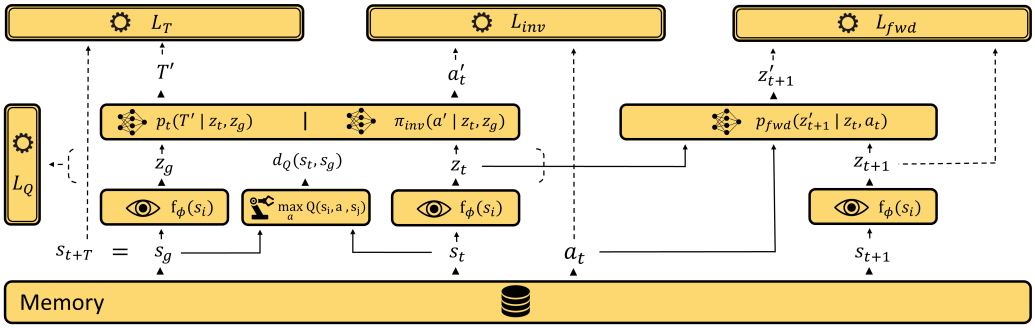

Figure 2: An overview of the functions, inputs, and losses used in our method (see Sec.2.2 for details). We aim to train a perceptual encoder f_ϕ with two properties: i) representations of two states should be close if they were observed to be easily reachable from each other within a low number of timesteps, ii) the representation of a state should capture a minimal sufficient statistic to inform an agent about the actions needed to reach nearby states.

(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 $d_Q(s_c, s_g) := max(\ -V(s_c, s_g), -V(s_g, s_c)\ )$, as proposed in [14]. 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 Representation Learning via 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 for the upcoming sections 2.3 and 2.4. This section discusses *one possible way* by using goal-conditioned Q-values for contrastive representation learning. We propose a model (depicted in Fig.2) that includes 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 [15, 16]; **iii)** $\pi_{inv}(a'_t \mid z_t, z_g)$, defining a distribution of actions to reach a goal state; **iv)** $p_t(T' \mid z_t, z_g)$, modelling the distribution of timesteps necessary to reach a goal state [17]; **v)** $Q(s_t, a_t, s_g)$, providing local reachability estimates between pairs of states.

We train $Q(s_t, a_t, s_g)$ over an offline replay buffer $\mathcal{M}$, using hindsight relabelling [11] 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 [18] 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 [19]. 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 [16, 17].

2.3 Perceptual Experience Retrieval (PER)

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 neighboring states, and ii) retrieving neighboring trajectories.

i) Retrieving Neighboring States: 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 neighboring states amounts to computing the set $\mathcal{N}_{d_p}(s_c) = \{s_n \mid d_\phi(s_c, s_n) \leq d_p\}$, which can be achieved by a straightforward L2 distance computation and thresholding. The number of neighbors $|\mathcal{N}_{d_p}(s_c)|$ of a query state s_c is an approximate measure of

how many times the agent has visited around s_c, which also makes it a good visitation-count that is applicable to both discrete and continuous state spaces.

ii) Retrieving Neighboring Trajectories: Given a starting state s_c and a goal state s_g, we can search the replay buffer for the highest reward trajectory segment τ that starts from a state τ_0 in $\mathcal{N}_{d_p}(s_c)$ and ends in a state τ_{-1} in $\mathcal{N}_{d_p}(s_g)$. This corresponds to the following optimization problem:

$$\tau_{\mathcal{M}(s_c,s_g)} := \arg\max_{\tau \in \mathcal{M}} \mathcal{R}(\tau) \quad \text{s.t.} \quad \tau_0 \in \mathcal{N}_{d_p}(s_c)\ ,\ \tau_{-1} \in \mathcal{N}_{d_p}(s_g) \tag{1}$$

To find $\tau_{\mathcal{M}(s_c,s_g)}$, we first select all state pairs $(s_i, s_j) \in \mathcal{N}_{d_p}(s_c) \times \mathcal{N}_{d_p}(s_g)$. 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, end at s_j, and are below a length threshold in terms of timesteps. We sort them based on $\mathcal{R}(\tau_{ij})$, and return the trajectory with the highest reward. We call this trajectory retrieval process 'Perceptual Experience Retrieval' (PER). We use PER only to retrieve short trajectory segments between close-by states (s_c, s_g) (i.e., hence the length threshold on τ_{ij}). These are then stitched together into long global trajectories using the planning algorithms defined in the next section.

2.4 Long-Horizon Planning Through Stitching Trajectory Segments

This section discusses how PER can be employed for long-horizon planning. Classical sampling-based planning algorithms such as RRT [20] or PRM [21] 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 PER (eq.1) and stored in that edge. Our new definitions for these subroutines directly mirror the original ones given in [22]:
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 $len(\tau_{\mathcal{M}(s_c,s_g)})$.

$$Nearest(V, s_g) := \arg\min_{s_c \in V} len(\tau_{\mathcal{M}(s_c,s_g)})$$

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

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 PER by adjusting the reward function (i.e., if τ has such an event, this reflects on $\mathcal{R}(\tau)$).

Algorithm 1 R-PRM (Roadmap Construction)

Input: $f_\phi, \mathcal{M}$
$V \leftarrow \{SampleFree_i\}_{i=1,...,num\_vertices}$; $E \leftarrow \emptyset$ ▷ 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 PER 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 $G = (V, E)$

Using these subroutines directly in-place of their originals, we reimplement retrospective 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 explicitly describe R-PRM as an example[2]. We note that our proposed planning algorithms 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 planning graph G).

[2]Similar descriptions for R-RRT and R-RRT* can be obtained by directly replacing subroutines 1-4 in [22].

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 PER 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 PER trajectories along the shortest path
  $\tau_{stitched} \leftarrow \tau_{stitched} \circ \tau_{\mathcal{M}(s_{i-1},s_i)}$
return $\tau_{\mathcal{M}^*(s_c,s_g)} = \tau_{stitched}$

2.5 Refining Memory Contents via Forming and Executing Plans

We iteratively form and execute $\tau_{\mathcal{M}^*(s_c,s_g)}$, and whenever execution is successful, we insert the resulting new trajectories back into $\mathcal{M}$. This creates the following perception-action loop: **i)** $\mathcal{M}$ with refined 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 chances of successful execution to further refine $\mathcal{M}$, thus looping back to step **i**.

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) [23] and Search on the Replay Buffer (SoRB) [12], which we compare to in our experiments. The key difference is that when setting the edges of the planning graph, our approach 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, 14]. In our approach, eq.1 naturally addresses this problem, since it requires an actual short trajectory in the dataset that connects 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 neighbor queries and as a threshold to create edges) from the downstream task reward $\mathcal{R}(\tau)$ (used to set edge distances), unlike previous work.
Robot Motion Planning: A common approach to motion planning is to first run a sampling-based planning algorithm [24, 22], and then refine the result through trajectory optimization [25, 26, 27] to satisfy constraints [28, 29, 30]. 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 [31] can autonomously construct high-fidelity geometric maps [32, 33], 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

Setup: Our experiments are performed in ViZDoom [34] and Habitat [35]. The VizDoom environment consists of a clover shaped maze. States solely consist of four images $I_{North/East/South/West}$ that

form a panorama, 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 (62 m2), and ii) Annawan ($75m2$). States consist of single 150 FOV images. There are 3 actions: $\{turn\_left\_30\_deg, turn\_right\_30\_deg, move\_forward\_\Delta\}$. In both environments, an offline training dataset is collected by a uniform random walk exploring the environment. This offline training dataset consists of only 300k and 150k timesteps for ViZDoom and Habitat respectively (i.e., compared to sample complexities around the orders of magnitude 1e6-1e7 common in RL).

4.1 Experiments in Vizdoom

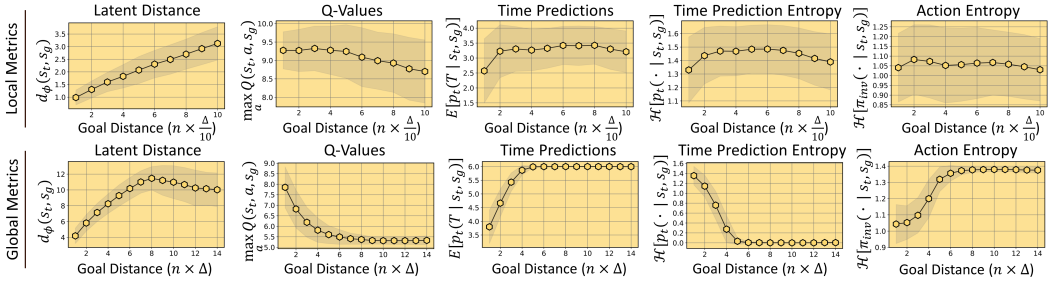

Figure 3: A comparison between perceptual distances d_ϕ and other suitable metrics from Sec.2.2. While all of these metrics are reasonably monotonic with physical reachability (i.e., goal distance), only perceptual distances d_ϕ do not saturate when evaluated locally (i.e., for close by goals). In addition, 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 nearest-neighbor criterion for perceptual experience retrieval and trajectory stitching, compared to the other metrics.

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 $\mathcal{N}_{d_p}(s_t)$ from $\mathcal{M}$ using d_ϕ (i.e., NN retrieval) returns physically close states.

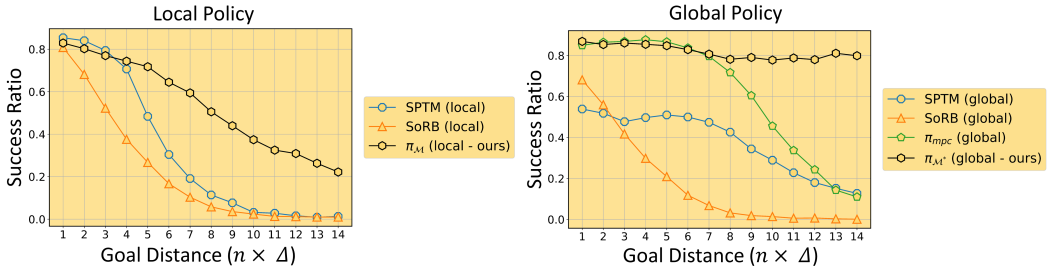

Figure 4: Comparisons of our local policy $\pi_{\mathcal{M}}$ and global policy $\pi_{\mathcal{M}^*}$ with SPTM and SoRB. $\pi_{\mathcal{M}}$ performs well because it avoids getting stuck (as such events are filtered by eq.1), while $\pi_{\mathcal{M}^*}$ performs well because it builds robust roadmaps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

Perceptual Experience Retrieval (PER): Fig.5 shows visualizations of trajectories retrieved with PER. 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 SORB [12] and SPTM [23] as baselines. Fig.4 shows the results. The main mode of failure for both baselines is that they get stuck in column-like structures. $\pi_{\mathcal{M}}$ avoids this, since eq.1 retrieves collision free $\tau_{\mathcal{M}(s_t,s_g)}$.

Proposed Planning Algorithms: Fig.6 shows the 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. We

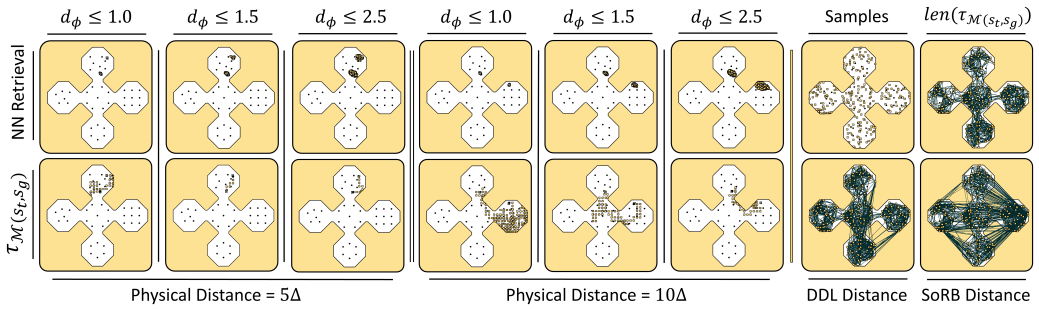

Figure 5: At the core of PALMER is a process called *perceptual experience retrieval* (PER). Given a query pair of current-goal states, PER searches the replay buffer to retrieve the highest scoring trajectory $\tau_{\mathcal{M}(s_t,s_g)}$ whose first and last states are close to the query pair according to the perceptual distance d_ϕ. **Left, Middle**: Visualizations of $\tau_{\mathcal{M}(s_t,s_g)}$ retrieved using PER and nearest neighbor states $\mathcal{N}_{d_p}(s_t)$ 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 [17]. We found that distance estimates from baselines are prone to setting false edges that cross map boundaries.

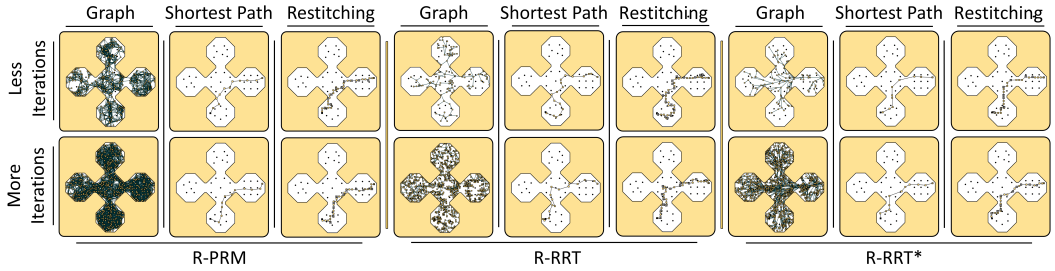

Figure 6: We repurpose conventional sampling-based planning algorithms as memory search mechanisms, by altering their graph building subroutines so that whenever an edge is created a trajectory $\tau_{\mathcal{M}(s_t,s_g)}$ is retrieved through PER and stored in that edge. We visualize the resulting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

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 $argmax_a\ Q(s_t, a, \tau_{\mathcal{M}^*(s_t,s_g)}, s, 1)$. Fig.4 shows the results. A new mode of failure for both baselines is that false distance estimates throw-off graph search by setting hallucinated shortcuts. An additional 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.

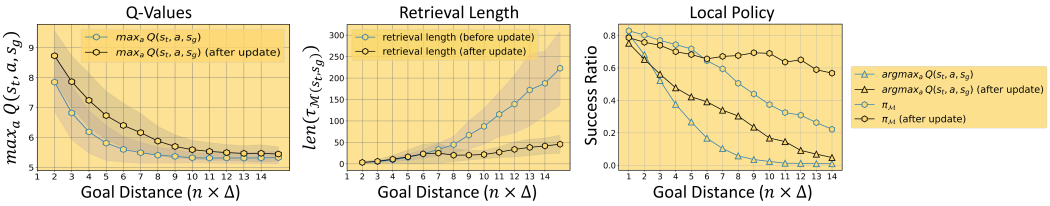

Figure 7: Memory Refinement: In PALMER, a policy has three groups of parameters: $Q(s_t, a_t, s_g)$, f_ϕ, and the contents of $\mathcal{M}$. Iteratively forming plans through PER and executing them creates a feedback loop between these components, where: i) *actions inform perception* during the training of f_ϕ, ii) *perception facilitates actions* through the formation plans, 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**).

Refining Memory Contents: We refine 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 unrefined $\mathcal{M}$. 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 $\tau_{\mathcal{M}^*(s_c,s_g)}$. These results highlight that refining memory contents improves the quality of future plans.

4.2 Experiments in Habitat

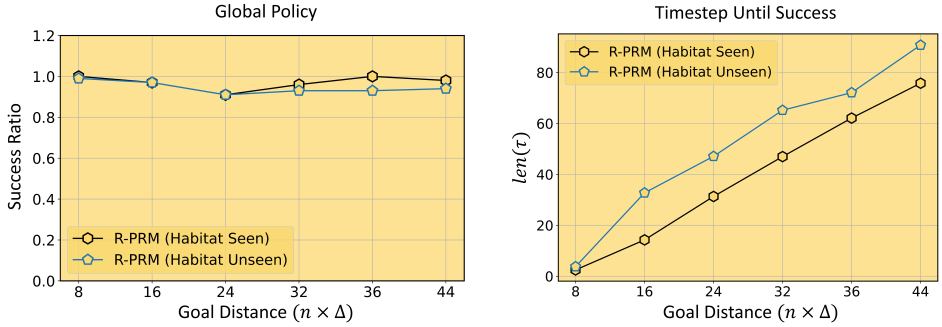

Figure 8: We evaluate our R-PRM based policy $\pi_{\mathcal{M}^*}$ in Habitat for image-based navigation. **Top Left:** Success ratios in training and test apartments. **Top Right:** Number of timesteps until reaching the goal. ("Habitat seen" refers to the training apartment Roxbox, while "habitat unseen" refers to the test apartment Annawan.

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 experience retrieval and trajectory stitching 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 plot the SPTM and SORB 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 $\{turn\_left\_30\_deg, turn\_right\_30\_deg, move\_forward\_\Delta\}$.

5 Conclusion and Future Directions

We presented PALMER, a long-horizon planning method that combines learning-based perceptual representations with classical sampling-based planning algorithms. It operates by retrieving and restitching previously observed trajectory segments in a replay buffer. This results in an experiential framework for long-horizon planning that is significantly more robust and sample efficient compared to baselines. Our memory-based planning perspective highlights a number of 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, 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. 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*, 24(5):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*, 299: 103535, 2021.

[6] E. L. Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements*, 2(4):i, 1898.

[7] E. L. Thorndike. The law of effect. *The American journal of psychology*, 39(1/4):212–222, 1927.

[8] D. T. Campbell. Perception as substitute trial and error. *Psychological review*, 63(5):330, 1956.

[9] R. J. Herrnstein. On the law of effect. *Journal of the experimental analysis of behavior*, 13(2): 243–266, 1970.

[10] S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

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

[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*, 32, 2019.

[13] L. P. Kaelbling. Learning to achieve goals. In *IJCAI*, volume 2, pages 1094–8. Citeseer, 1993.

[14] 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*, 33:5251–5262, 2020.

[15] 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*, 29, 2016.

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

[17] 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.

[18] 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)*, volume 1, pages 539–546. IEEE, 2005.

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

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

[21] 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*, 12(4):566–580, 1996.

[22] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

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

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

[25] J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[26] J. T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

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

[28] R. Tedrake. *Underactuated Robotics*. 2022. URL http://underactuated.mit.edu.

[29] P. Abbeel. *Advanced Robotics*. 2019. URL https://people.eecs.berkeley.edu/~pabbeel/cs287-fa19/.

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

[31] S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[32] 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)*, pages 6351–6358. IEEE, 2019.

[33] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[34] 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)*, pages 1–8. IEEE, 2016.

[35] 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*, pages 9339–9347, 2019.