# World Model as a Graph: Learning Latent Landmarks for Planning

**Lunjun Zhang** [1 2] **Ge Yang** [3] **Bradly Stadie** [4]

## Abstract

Planning, the ability to analyze the structure of a problem in the large and decompose it into interrelated subproblems, is a hallmark of human intelligence. While deep reinforcement learning (RL) has shown great promise for solving relatively straightforward control tasks, it remains an open problem how to best incorporate planning into existing deep RL paradigms to handle increasingly complex environments. One prominent framework, Model-Based RL, learns a world model and plans using step-by-step virtual rollouts. This type of world model quickly diverges from reality when the planning horizon increases, thus struggling at long-horizon planning. How can we learn world models that endow agents with the ability to do temporally extended reasoning? In this work, we propose to learn graph-structured world models composed of sparse, multi-step transitions. We devise a novel algorithm to learn latent landmarks that are scattered (in terms of reachability) across the goal space as the nodes on the graph. In this same graph, the edges are the reachability estimates distilled from Q-functions. On a variety of high-dimensional continuous control tasks ranging from robotic manipulation to navigation, we demonstrate that our method, named $L^3P$, significantly outperforms prior work, and is oftentimes the only method capable of leveraging both the robustness of model-free RL and generalization of graph-search algorithms. We believe our work is an important step towards scalable planning in reinforcement learning.

[1]University of Toronto [2]Vector Institute [3]MIT [4]Toyota Technological Institute at Chicago. Correspondence to: Lunjun Zhang <lunjun@cs.toronto.edu>.

*Proceedings of the $38^{th}$ International Conference on Machine Learning*, PMLR 139, 2021. 

## 1. Introduction

An intelligent agent should be able to solve difficult problems by breaking them down into sequences of simpler problems. Classically, planning algorithms have been the tool of choice for endowing AI agents with the ability to reason over complex long-horizon problems (Doran & Michie, 1966; Hart et al., 1968). Recent years have seen an uptick in monographs examining the intersection of classical planning techniques – which excel at temporal abstraction – with deep reinforcement learning (RL) algorithms – which excel at state abstraction. Perhaps the ripest fruit born of this relationship is the AlphaGo algorithm, wherein a model free policy is combined with a MCTS (Coulom, 2006) planning algorithm to achieve superhuman performance on the game of Go (Silver et al., 2016a).

In the field of robotics, progress on combining planning and reinforcement learning has been somewhat less rapid, although still resolute. Indeed, the laws of physics in the real world are vastly more complex than the simple rules of Go. Unlike board games such as chess and Go, which have deterministic and known dynamics and discrete action space, robots have to deal with a probabilistic and unpredictable world. Moreover, the action space for robotics is often continuous. As a result of these difficulties, planning in robotics presents a much harder problem. One general class of methods (Sutton, 1991) seeks to combine model-based planning and deep RL. These methods can be thought of as an extension of model-predictive control (MPC) algorithms, with the key difference being that the agent is trained over hypothetical experience in addition to the actually collected experience. The primary shortcoming of this class of methods is that, like MCTS in AlphaGo, they resort to planning with action sequences – forcing the robot to plan for each action at every hundred milliseconds. Planning on the level of action sequences is fundamentally bottlenecked by the accuracy of the learned dynamics model and the horizon of a task, as the learned world model quickly diverges over a long horizon. This limitation shows that world models in the traditional Model-based RL (MBRL) setting often fail to deliver the promise of planning.

Another general class of methods, Hierarchical RL (HRL), introduces a higher-level learner to address the problem of

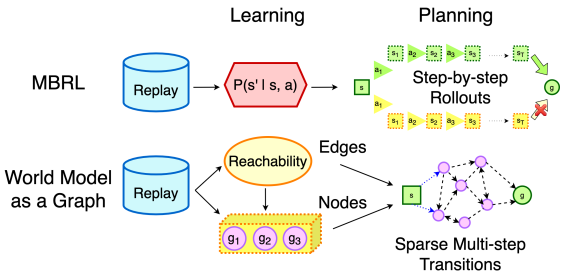


*Figure 1.* MBRL versus $L^3P$ (World Model as a Graph). MBRL does step-by-step virtual rollouts with the world model and quickly diverges from reality when the planning horizon increases. $L^3P$ models the world as a graph of sparse multi-step transitions, where the nodes are learned latent landmarks and the edges are reachability estimates. $L^3P$ succeeds at temporally extended reasoning. **Code** for $L^3P$ is available at: `https://github.com/LunjunZhang/world-model-as-a-graph`.

planning (Dayan & Hinton, 1993; Vezhnevets et al., 2017; Nachum et al., 2018). In this case, a goal-based RL agent serves as the worker, and a manager learns what sequences of goals it must set for the worker to achieve a complex task. While this is apparently a sound solution to the problem of planning, hierarchical learners neither explicitly learn a higher-level model of the world nor take advantage of the graph structure inherent to the problem of search.

To better combine classical planning and reinforcement learning, we propose to learn graph-structured world models composed of sparse multi-step transitions. To model the world as a graph, we borrow a concept from the navigation literature – the idea of landmarks (Wang et al., 2008). Landmarks are essentially states that an agent can navigate between in order to complete tasks. However, rather than simply using previously seen states as landmarks, as is traditionally done, we will instead develop a novel algorithm to learn the landmarks used for planning. Our key insight is that by mapping previously achieved goals into a latent space that captures the temporal distance between goals, we can perform clustering in the latent space to group together goals that are easily reachable from one another. Subsequently, we can then decode the latent centroids to obtain a set of goals scattered (in terms of reachability) across the goal space. Since our learned landmarks are obtained from latent clustering, we call them *latent landmarks*. The chief algorithmic contribution of this paper is a new method for planning over learned latent landmarks for high-dimensional continuous control domains, which we name Learning Latent Landmarks for Planning ($L^3P$).

The idea of reducing planning in RL to a graph search problem has enjoyed some attention recently (Savinov et al., 2018a; Eysenbach et al., 2019; Huang et al., 2019; Liu et al., 2019; Yang et al., 2020; Emmons et al., 2020). A key difference between those works and $L^3P$ is that our use of *learned* latent landmarks allows us to substantially reduce the size of the search space. What's more, we make improvements to the graph search module and the online planning algorithm to improve the robustness and sample efficiency of our method. As a result of those decisions, our algorithm is able to achieve superior performance on a variety of robotics domains involving both navigation and manipulation. In addition to the results presented in Section 5, **videos** of our algorithm's performance and a more detailed analysis of the sub-tasks discovered by the latent landmarks can be found at: `https://sites.google.com/view/latent-landmarks/`.

## 2. Related Works

The problem of learning landmarks to aid in robotics problems has a long and rich history (Gillner & Mallot, 1998; Wang & Spelke, 2002; Wang et al., 2008). Prior art has been deeply rooted in the classical planning literature. For example, traditional methods would utilize (Dijkstra et al., 1959) to plan over generated waypoints, SLAM (Durrant-Whyte & Bailey, 2006) to simultaneously integrate mapping, or the RRT algorithm (LaValle, 1998) for explicit path planning. The A* algorithm (Hart et al., 1968) further improved the computational efficiency of Dijkstra. Those types of methods often heavily rely on a hand-crafted configuration space that provides prior knowledge.

Planning is intimately related to model-based RL (MBRL), as the core ideas underlying learned models and planners can enjoy considerable overlap. Perhaps the most clear instance of this overlap is Model Predictive Control (MPC), and the related Dyna algorithm (Sutton, 1991). When combined with modern techniques (Kurutach et al., 2018; Luo et al., 2018; Nagabandi et al., 2018; Ha & Schmidhuber, 2018; Hafner et al., 2019; Wang & Ba, 2019; Janner et al., 2019), MBRL is able to achieve some level of success. (Corneil et al., 2018) and (Hafner et al., 2020) also learn a discrete latent representation of the environment in the MBRL framework. As discussed in the introduction, planning on action sequences will fundamentally struggle to scale in robotics.

Our method makes extensive use of a parametric goal-based RL agent to accomplish low-level navigation between states. This area has seen rapid progress recently, largely stemming from the success of Hindsight Experience Replay (HER) (Andrychowicz et al., 2017). Several improvements to HER augment the goal relabeling and sampling strategies to boost performance (Nair et al., 2018; Pong et al., 2018; 2019; Zhao et al., 2019; Pitis et al., 2020). There have also been attempts at incorporating search as inductive biases within the value function (Silver et al., 2016b; Tamar et al., 2016; Farquhar et al., 2017; Racanière et al., 2017; Lee et al., 2018; Srinivas et al., 2018). The focus of this line of work is to improve the low-level policy and is thus orthogonal to our work.

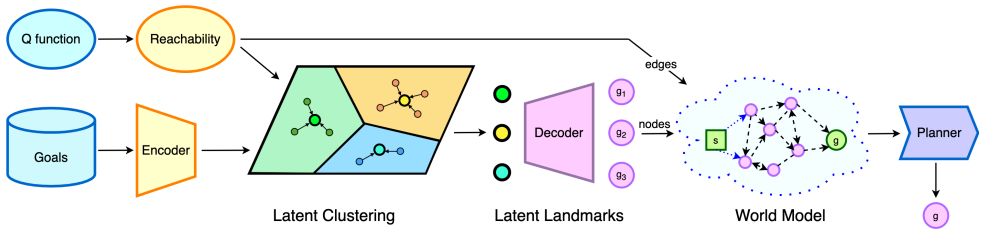


*Figure 2.* An overview of $L^3P$, which learns a small number of latent landmarks for planning. The main components of our method are: learning reachability estimates (via Q-learning and regression), learning a latent space (via an auto-encoder with reachability constraints), learning latent landmarks (via clustering in the latent space), graph search on the world model and online planning.

Recent work in Hierarchical RL (HRL) builds upon goal-based RL by learning a high-level parametric manager that feeds goals to the low-level goal-based agent (Dayan & Hinton, 1993; Vezhnevets et al., 2017; Nachum et al., 2018). This can be viewed as a parametric alternative to classical planning, as discussed in the introduction. Recently, (Jurgenson et al., 2020; Pertsch et al., 2020) have derived HRL methods that are intimately tied to tree search algorithms. These papers are further connected to a recent trend in the literature wherein classical search methods are combined with parametric control (Savinov et al., 2018a; Eysenbach et al., 2019; Huang et al., 2019; Liu et al., 2019; Yang et al., 2020; Emmons et al., 2020). Several of these articles will be discussed throughout this paper. LEAP (Nasiriany et al., 2019) also considers the problem of proposing sub-goals for a goal-conditioned agent: it uses a VAE (Kingma & Welling, 2013) and does CEM on the prior distribution to form the landmarks. Our method constrains the latent space with temporal reachability between goals, a concept previously explored in (Savinov et al., 2018b), and uses latent clustering and graph search rather than sampling-based methods to learn and propose sub-goals.

## 3. Background

We consider the problem of Multi-Goal RL under a Markov Decision Process (MDP) parameterized by $(S, A, \mathbb{P}, G, \Psi, R, \rho_0)$. $S$ and $A$ are the state and action space. The probability distribution of the initial states is given by $\rho_0(s)$, and $\mathbb{P}(s'|s, a)$ is the transition probability. $\Psi : S \mapsto G$ is a mapping from the state space to the goal space, which assumes that *every* state $s$ can be mapped to a corresponding *achieved* goal $g$. The reward function $R$ can be defined as $R(s, a, s', g) = -\mathbb{1}\{\Psi(s') \neq g\}$. We further assume that each episode has a fixed horizon $T$.

A multi-goal policy is a probability distribution $\pi : S \times G \times A \to \mathbb{R}^+$, which gives rise to trajectory samples of the form $\tau = \{s_0, a_0, g, s_1, \cdots s_T\}$. The purpose of the policy $\pi$ is to learn how to reach the goals drawn from the goal distribution $p_g$. With a discount factor $\gamma \in (0, 1)$, it maximizes $\mathcal{J}(\pi) = \mathbb{E}_{g\sim p_g, \tau\sim\pi(g)}[\sum_{t=0}^{T-1} \gamma^t \cdot R(s_t, a_t, s_{t+1}, g)]$.

Q-learning provides a sample-efficient way to optimize the above objective by utilizing off-policy data stored in a replay buffer $B$. $Q(s, a, g)$ estimates the reward-to-go under the current policy $\pi$ conditioned upon the given goal. An additional technique, called Hindsight Experience Replay, or HER (Andrychowicz et al., 2017), uses hindsight relabelling to drastically speed up training. This relabeling crucially relies upon the mapping $\Psi : S \mapsto G$ in the multi-goal MDP setting. We can write the the joint objective of multi-goal Q-learning with HER as minimizing (with $Q$ being the online network and $\widehat{Q}$ being the target network):

$$\left(Q(s_t, a_t, g) - \left(R(s_{t+1}, g) + \gamma \cdot \widehat{Q}(s_{t+1}, a', g)\right)\right)^2 \quad (1)$$

where $\tau \sim B, t \sim \{0 \cdots T-1\}, (s_t, a_t, s_{t+1}) \sim \tau, k \sim \{t+1 \cdots T\}, g = \Psi(s_k), a' \sim \pi(\cdot \mid s_{t+1}, g)$.

## 4. The $L^3P$ Algorithm

Our overall objective in this section is to derive an algorithm that learns a small number of landmarks scattered across goal space in terms of reachability and use those learned landmarks for planning. There are three chief difficulties we must overcome when considering such an algorithm. First, how can we group together goals that are easily reachable from one another? The answer is to embed goals into a latent space, where the latent representation captures some notion of temporal distance between goals – in the sense that goals that would take many timesteps to navigate between are further apart in latent space. Second, we need to find a way to learn a sparse set of landmarks used for planning. Our method performs clustering on the constrained latent space, and decodes the learned centroids as the landmarks we seek. Finally, we need to develop a non-parametric planning algorithm responsible for selecting sequences of landmarks the agent must traverse to accomplish its high-level goal. The proposed online planning algorithm is simple, scalable, and robust.

### 4.1. Learning a Latent Space

Let us consider the following question: "How should we go about learning a latent space of goals where the metric re-

flects reachability?" Suppose we have an auto-encoder (AE) in the agent's goal space, with deterministic encoder $f_E$ and decoder $f_D$. As usual, the reconstruction loss is given by $\mathcal{L}_{rec}(g) = \Big\| f_D\big(f_E(g)\big) - g \Big\|_2^2$. We want to make sure that the distance between two latent codes would roughly correspond to the number of steps it would take the policy to go from one goal to another. Concretely, for any pair of goals $(g_1, g_2)$, we optimize the following loss $\mathcal{L}_{latent}(g_1, g_2)$:

$$\Big( \big\| f_E(g_1) - f_E(g_2) \big\|_2^2 - \frac{1}{2}\big(V(g_1, g_2) + V(g_2, g_1)\big) \Big)^2 \tag{2}$$

Where $V : G \times G \to \mathbb{R}^+$ is a mapping that estimates how many steps it would take the policy $\pi$ to go from one goal to another goal on average. By adding this constraint and solving a joint optimization $\mathcal{L}_{rec} + \lambda \cdot \mathcal{L}_{latent}$, the encoding-decoding mapping can no longer be arbitrary, giving more structure to the latent space. Goals that are close by in terms of reachability will be naturally clustered in the latent space, and interpolations between latent codes will lead to meaningful results.

Of course, the constraint in Equation 2 is quite meaningless if we do not have a way to estimate the mapping $V$. We will proceed towards this objective by noting the following interesting connection between multi-goal Q-functions and reachability. In the multi-goal RL framework considered in the background section, the reward is binary in nature. The agent receives a reward of $-1$ until it reaches the goal, and then $0$ when it reaches the desired goal. In this setting, the Q-function is implicitly estimating *the number of steps* it takes to reach the goal $g$ from the current state $s$ *after* the action $a$ is taken. Denote this quantity as $D(s, a, g)$, the Q-function can be re-written as:

$$\begin{aligned} Q(s, a, g) &= \sum_{t=0}^{D(s,a,g)-1} \gamma^t \cdot (-1) + \sum_{t=D(s,a,g)}^{T-1} \gamma^t \cdot 0 \\ &= -\frac{1 - \gamma^{D(s,a,g)}}{1 - \gamma} \end{aligned} \tag{3}$$

Choosing to parameterize Q-functions in this way disentangles the effect of $\gamma$ on multi-goal Q-learning. It also provides us with access the direct distance estimation function $D(s, a, g)$. We note that this *distance* is not a mathematical distance in the sense of a metric. Instead, we use the word *distance* to refer to the number of steps the policy $\pi$ needs to take in the environment.

Given our tractable estimate of $D$, it is now a straightforward matter to estimate the desired quantity $V$, which approximates how many steps it takes the policy to transition between goals. To get the desired estimate, we regress $V$ towards $D$ by minimizing

$$\min_V \left( D\big(s_t, a_t, \Psi(s_k)\big) - V\big(\Psi(s_{t+1}), \Psi(s_k)\big) \right)^2 \tag{4}$$

with $\tau \sim B, t \sim \{0 \cdots T-1\}, (s_t, a_t, s_{t+1}) \sim \tau, k \sim \{t+1 \cdots T\}$, and $\Psi$ being given by the environment to map the states to the goal space. One crucial detail is the use of $\Psi(s_{t+1})$ rather than $\Psi(s_t)$ in the inputs to $V$. This is due to the fact that $D : S \times A \times G \to \mathbb{R}$ outputs the number of steps to go *after* an action is taken, when the state has transitioned into $s_{t+1}$. The objective above provides an unbiased estimate of the average number of steps between two goals.

The estimates $D$ and $V$ will prove useful beyond helping to optimize the auto-encoder in Equation 2. They will prove essential in weighting and planning over latent landmark nodes in Section 4.3.

### 4.2. Learning Latent Landmarks

Planning on a graph can be expensive, as the number of edges can grow quadratically with the number of nodes. To battle this issue in scalability, we use the constrained latent space to learn a sparse set of landmarks. A landmark can be thought of as a waypoint that the agent can pass through enroute to achieve a desired goal. Ideally, *goals that are easily reachable from one another should be grouped to form one single landmark*. Since our latent representation captures the temporal reachability between goals, this can be achieved by doing clustering in the latent space. The cluster centroids, when decoded from the decoder, will be precisely the latent landmarks we are seeking.

Clustering proceeds as follows. For $N$ clusters to be learned, we define a mixture of Gaussians in the latent space with $N$ trainable latent centroids, $\{\mathbf{c}_1 \cdots \mathbf{c}_N\}$, and a shared trainable variance vector $\boldsymbol{\sigma}$. We maximize the evidence lower bound (ELBO) with a uniform prior $p(\mathbf{c})$:

$$\begin{aligned} &\log p\Big(z = f_E(g)\Big) \\ &\geq \mathbb{E}_{q(\mathbf{c}|z)}\Big[\log p(z \mid \mathbf{c})\Big] - D_{KL}\Big(q(\mathbf{c} \mid z) \parallel p(\mathbf{c})\Big) \end{aligned} \tag{5}$$

Ideally, we would like each batch of data given to the latent clustering model to be representative of the whole replay buffer, such that the centroids will quickly learn to scatter out. To this end, we propose to use the Greedy Latent Sparsification (GLS) algorithm (see the Appendix) on each batch of data sampled from the replay before taking a gradient step with the batch. GLS is inspired by kmeans++ (Arthur & Vassilvitskii, 2007), with several key differences: this sparsification process is used for both training and initialization, it uses a neural metric for determining the distance between data points, and that it is compatible with mini-batch-style gradient-based training.

### 4.3. Planning with Latent Landmarks

Having derived a latent encoding algorithm and an algorithm for learning latent landmarks, we at last turn our attention to search and planning. $L^3P$ is agnostic to the graph search algorithm being used. In practice, we use a variant of the Floyd algorithm, where our relaxation operations use a soft max rather than hard max for better stability (see the Appendix for more details). To construct a weight matrix that provides raw distance estimates between latent landmarks in the first place, we begin by decoding the learned centroids in the latent space into the nodes in the graph $\{f_D(\mathbf{c}_1) \cdots f_D(\mathbf{c}_N)\}$. To build the graph, we add two edges directed in reverse orders for every pair of latent landmarks. For instance, for an edge going from $f_D(\mathbf{c}_i)$ to $f_D(\mathbf{c}_j)$, the weight on that edge is $w_{i,j} = -V(f_D(\mathbf{c}_i), f_D(\mathbf{c}_j))$. Notice that the distances are negated. At the start of an episode, the agent receives a goal $g$, and we construct matrix $W$:

$$W = \begin{pmatrix} 0 & \dots & w_{1,N} & -V(f_D(\mathbf{c}_1), g) \\ \vdots & \ddots & \vdots & \vdots \\ w_{N,1} & \dots & 0 & -V(f_D(\mathbf{c}_N), g) \\ -\infty & \dots & -\infty & 0 \end{pmatrix} \tag{6}$$

**Algorithm 1** Online Planning in $L^3P$

**Given**: Environment `env`, initial state $s$, goal $g$.

`Cnt` = 0. `SubG` = `None`.
Solve for $\boldsymbol{d}_{\boldsymbol{c}\to g}$ with **graph search** using $W$.
**for** $t$ = 1 to $T$ **do** ▷ One episode
  **if** `Cnt` $\geq 1.0$ **then**
    `Cnt` = `Cnt` $- 1$
  **else** ▷ We do **not** re-plan at every step
    Calculate $\boldsymbol{d}_{s\to\boldsymbol{c}}$.
    $\boldsymbol{d} \leftarrow \boldsymbol{d}_{s\to\boldsymbol{c}} + \boldsymbol{d}_{\boldsymbol{c}\to g}$
    ▷ Remove the **immediate** previous landmark
    **if** `SubG` $\neq$ `None` **then**
      $\boldsymbol{d}$[`SubG`] $\leftarrow -\infty$
    **end if**
    `SubG`, `Cnt` $\leftarrow \arg\max \boldsymbol{d}, -\max \boldsymbol{d}$
  **end if**
  $a \sim \pi(s,$ `SubG`$)$; $s \leftarrow$ `env.step(a)`.
**end for**

For online planning, when the agent receives a goal at the start of an episode, we use graph search to solve for $\boldsymbol{d}_{\boldsymbol{c}\to g}$ (which is fixed throughout an episode). For an observation state $s$, the algorithm calculates $\boldsymbol{d}_{s\to\boldsymbol{c}}$:

$$\boldsymbol{d}_{s\to\boldsymbol{c}} = \begin{pmatrix} -D\big(s, \pi(s, f_D(\mathbf{c}_1)), f_D(\mathbf{c}_1)\big) \\ \vdots \\ -D\big(s, \pi(s, f_D(\mathbf{c}_N)), f_D(\mathbf{c}_N)\big) \\ -D\big(s, \pi(s, g), g\big) \end{pmatrix} \tag{7}$$

The chosen landmark is `subgoal` $\leftarrow \arg\max(\boldsymbol{d}_{s\to\boldsymbol{c}} + \boldsymbol{d}_{\boldsymbol{c}\to g})$. To further provide temporal abstraction and robustness, the agent will be asked to consistently pursue `subgoal` for $K = -\boldsymbol{d}_{s\to\boldsymbol{c}}$[`subgoal`] number of steps, which is *how many steps it thinks it will need*. The proposed goal does **not** change during this period. In this way, $L^3P$ makes sure that the agent does not re-plan at every step, and this mechanism for temporal abstraction is crucial to its robustness. This mechanism is detailed in Algorithm 1.

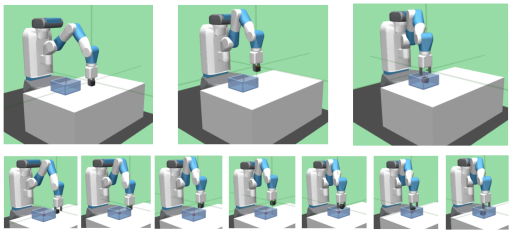

*Figure 3.* We consider two environments involving a fetch robot, a block, and a box. In Box-Distractor-PickAndPlace, the fetch must learn to pick and place the block while avoiding collision with the box. In Place-Inside-Box, the fetch must pick the block and place it inside the box. We visualize the fetch states corresponding to learned landmarks in the second row of images.

After this $K$ many steps, the agent will decide on the next landmark to pursue by re-calculating $\boldsymbol{d}_{s\to\boldsymbol{c}}$, but the *immediate* previous landmark will not be considered as a candidate landmark. The reason is that, if the agent has failed to reach a self-proposed landmark within the reachability limit it has set for itself, then the agent should try something new for the immediate next goal rather than stick to the immediate previous landmark for another round. We have found that this simple algorithm helps the agent avoid getting stuck and improves the overall robustness of the agent.

In summary, we have derived an algorithm that learns a sparse set of latent landmarks scattered across goal space in terms of reachability, and uses those learned landmarks for robust planning.

## 5. Experiments and Evaluation

We investigate the impact of $L^3P$ in a variety of robotic manipulation and navigation environments. These include standard benchmarks such as Fetch-PickAndPlace, and more difficult environments such as AntMaze-Hard and Place-Inside-Box that have been engineered to require test-time generalization. Videos of our algorithm in action are available at: `https://sites.google.com/view/latent-landmarks/`.

### 5.1. Baselines

We compare our method with a variety of baselines. HER (Andrychowicz et al., 2017) is a model-free RL algorithm.

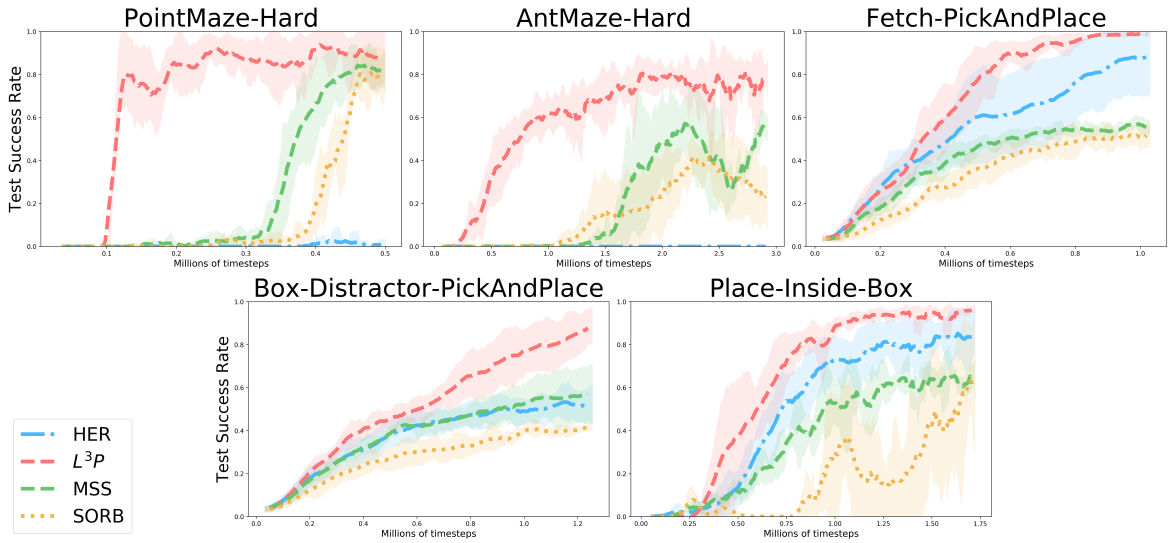


*Figure 4.* Test time success rate vs. total number of timesteps, on a variety of challenging robotic navigation and manipulation environments. $L^3P$ demonstrates better sample efficiency, higher asymptotic performance, and in some cases, the ability to generalize to longer horizons.

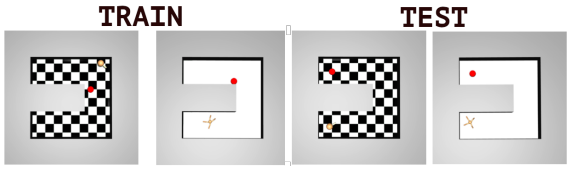


*Figure 5.* For both Point and Ant, during training, the initialization state distribution and the goal proposal distribution are *uniform* around the maze. During test time, the agent is asked to traverse the longest path in the maze, which is not seen during training. Importantly, the map of the environment is not given to the agent at any given point; the agent has to learn the structure of the environment purely through interaction. The success rate during test is reported in Figure **??**. This environment demonstrates $L^3P$'s ability to generalize to longer horizon goals during test time.

SORB (Eysenbach et al., 2019) is a method that combines RL and graph search by using the entire replay buffer. Mapping State Space (MSS Huang et al. 2019) reduces the number of vertices by sub-sampling the replay buffer. $L^3P$, SORB, and MSS all use the same hindsight relabelling strategy proposed in HER. All of the domains are continuous control tasks, so we adopt DDPG (Lillicrap et al., 2015) as the learning algorithm for the low-level actor.

### 5.2. Generalization to Longer Horizons

The PointMaze-Hard and AntMaze-Hard environments introduced in Figure 6 are designed to test an agent's ability to generalize to longer horizons. While PointMaze and AntMaze have been previously used in (Duan et al., 2016; Huang et al., 2019; Pitis et al., 2020), we make slight changes to those environments in order to increase their difficulty. We use a short, 200-timestep time horizon dur-

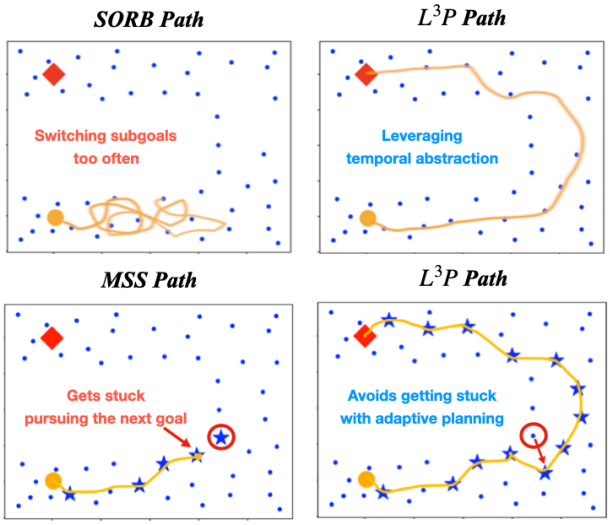


*Figure 6.* Visualizing the paths taken by SORB, MSS and $L^3P$ on AntMaze at test time. The blue dots in the backgrounds are the learned landmarks using $L^3P$. The orange dot is the starting location of the Ant. The red dot is the final goal. The blue stars indicate the landmarks chosen by the planning algorithms. As illustrated in the figure above, $L^3P$ addresses two major failure modes of graph-based planning with RL. Firstly, graph-based methods tend to switch proposed subgoals too frequently and fall into a loop due to wormholes in distance estimates, whereas $L^3P$ leverages temporal abstraction in both landmark learning and online planning to avoid this pitfall. Secondly, when the agent pursues a subgoal unsuccessfully (due to obstacles, etc), other methods tend to get stuck by continuing proposing the same subgoal, whereas $L^3P$ can adapt to the encountered failure and propose different subgoals in the event of getting stuck.

ing training and a $\rho_0$ that is uniform in the maze. At test time, we always initialize the agent on one end of the maze, and set the goal on the other end. The horizon of the test environment is 500 steps. Crucially, no prior knowledge on the shape of the maze is given to the agent. We also set a much stricter threshold for determining whether an agent has reached the goal. In Figure **??**, we see $L^3P$ is the only algorithm capable of solving AntMaze-Hard consistently.

We observe an interesting trend where the success rates for some of other graph search methods crash and then slowly recover after making some initial progress. We postulate this occurs because methods that are based on using the entire replay or sub-sampling the replay for landmark selection will struggle as the buffer size increases. For instance, in the AntMaze-Hard environment, MSS and SORB use 400 and tens of thousands of landmarks respectively, whereas $L^3P$ obtains a lean graph that only contain 50 learnable landmarks. The result suggests that *learning latent landmarks* is significantly more sample efficient and stable than either directly using or sub-sampling the replay buffer to build the graph. The online planning algorithm in $L^3P$, which effectively leverages temporal abstraction to improve robustness, also contributes to the asymptotic success rate. As explained in Figure 6, $L^3P$ successfully addresses the common failure modes of graph-based RL methods. The result convincingly shows that, at least on the navigation tasks considered, $L^3P$ is most effective at taking advantage of the problem's inherent graph structure (without any prior knowledge of the map or environment configurations) and generalizing to longer horizons during test time.

### 5.3. Robotic Manipulation Tasks

We also benchmark challenging robotic manipulations tasks with a Fetch robot introduced in (Plappert et al., 2018; Andrychowicz et al., 2017). Besides the PickAndPlace task, we also evaluate our method on two additional Fetch tasks involving a box on a table, as illustrated in Figure 3. In Box-Distractor-PickAndPlace environment, the agent needs to perform the pick-and-place task with a box in the middle of the table serving as a distractor. The Place-Inside-Box environment aims to teach the agent to place an object with randomly initialized locations into the box and has a simple curriculum. During training, the goal distribution has 80% regular pick-and-place goals, enabling the agent to first learn how to fetch in general. Meanwhile, only 20% of the goals are inside the box, which is the harder part of the task. During testing, we evaluate the agent's ability to pick up the object from the table and place it inside the box. Our method achieves dominant performance in both learning speed and test-time generalization on those three robotic manipulation environments. We note that on those manipulation tasks considered, many prior planning methods *hurt* the performance of the model-free agent. $L^3P$ is the only method that is able to help the model-free agent learn faster and perform better on all three tasks.

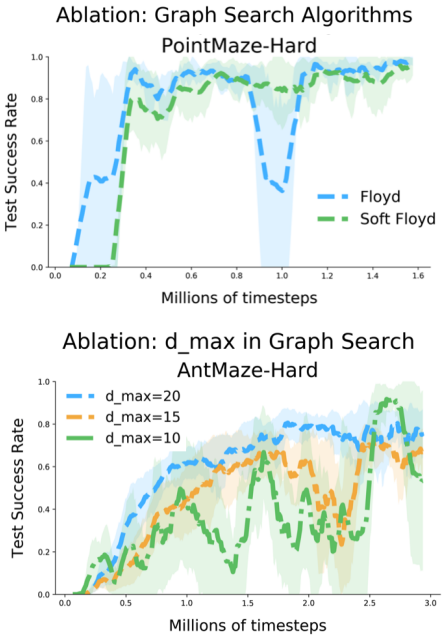


*Figure 7.* Ablation studies on the graph search module, including the choice of graph search algorithms and a key hyper-parameter in graph search: the edge weight cutoff threshold.

### 5.4. Understanding Model Choices in $L^3P$

We investigate $L^3P$'s sensitivity to different design choices and hyper-parameters via a set of ablation studies. More specifically, we study how the following four factors affect the performance of $L^3P$: the choice of graph search algorithms, and edge weight cutoff threshold in graph search (a key hyper-parameter in the graph search module); the choice of online planning algorithms, and the number of latent landmarks being learned (a key hyper-parameter in the planning module).

While $L^3P$ is agnostic to the graph search algorithm being used, we study the effect of two possible choices: Floyd algorithm and a soft version of Floyd (soft Floyd). As shown in Figure 7, the choice seems to have a relatively small effect on learning. During the early phase of experimentation, we find that having a *soft* operation for relaxation in Floyd leads to better overall training stability. A hard version of relaxation helps the learning curve take off faster but suffers from greater instability during policy improvement. The likely reason is that neural distance estimates are not entirely accurate, and in the presence of occasional bad edges, using *softmax* rather than hard max improves robustness. We therefore use soft relaxation in $L^3P$.

In the graph search module, a very sensitive hyper-parameter

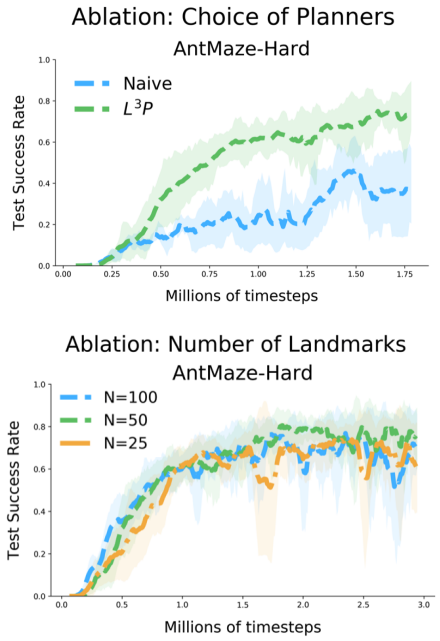


*Figure 8.* Ablation studies on the online planning module, including the choice of planners and a key hyper-parameter in graph-based planning: the number of nodes (landmarks).

is the edge weight cutoff threshold, denoted as $d\_max$. This clipping threshold is commonly used in prior works such as (Savinov et al., 2018a; Eysenbach et al., 2019; Huang et al., 2019; Emmons et al., 2020). It essentially means that if the weight of an edge is bigger than $d\_max$, then it is set to be infinity during the graph search process. The motivation for introducing this common hyper-parameter is two-fold. Firstly, we only trust distance estimates when they are *local*, because value iterations are inherently local. Secondly, we want the next sub-goal to be relatively *nearby* in terms of reachability. The $d\_max$ value determines the maximum (perceived) distance from the current state to next proposed subgoal. As shown in Figure 7, our current approach is still quite sensitive to this hyper-parameter; changes to $d\_max$ can have a considerable impact on learning. As this weakness is common to this class of approaches, we believe that further research is required to discover more principled ways of encouraging the search results to be local.

For online planning, the $L^3P$ planner introduced in Algorithm 1 is essential to the success of $L^3P$. Our planning algorithm can take advantage of the temporal abstraction provided by the graph-structured world model. As previously shown in Figure 6, the design of $L^3P$ planner avoids many common pitfalls. It does not re-plan at every step, but instead uses the reachability estimates to dynamically decide when to re-plan, striking a balance between adaptability and consistency in planning. This planner is also more tolerant of errors: it removes the immediate previous landmark when it re-plans, so that the agent will be less prone to getting stuck. In Figure 8, we compare the $L^3P$ planner to a naive planner, which simply re-calculates the shortest path at every step. The result shows that our planning algorithm is crucial to the success of $L^3P$.

An important hyper-parameter in graph-based planning is the number of landmarks being used. Intuitively, since $L^3P$ is *learning* the nodes on the graph, it should be robust to the changes in the number of nodes (landmarks) being learned. In Figure 8, we show that this is indeed the case: $L^3P$ is robust to the number of latent landmarks. In contrast to prior methods, $L^3P$ is able to *learn* the nodes (landmarks) used for graph search from the agent's own experience. We vary this hyper-parameter in the challenging AntMaze-Hard environment, and we find that $L^3P$ is robust against a variety of values. This is expected, because the landmarks in the latent space of $L^3P$ will try to be equally scattered across the goal space according to the learned reachability metric. As the number of landmarks decreases, the learning procedure will automatically push the landmarks to be further away from one another.

## 6. Closing Remarks

In this work, we introduce a way of learning graph-structured world models that endow agents with the ability to do temporally extended reasoning. The algorithm, $L^3P$, learns a set of latent landmarks scattered across the goal space to enable scalable planning. We demonstrate that $L^3P$ achieves significantly better sample efficiency, higher asymptotic performance, and generalization to longer horizons on a range of challenging robotic navigation and manipulation tasks. Here we briefly discuss two promising future directions. First, how can an agent quickly generate a set of plausible landmarks in a previously unseen environment? A lot of progress has been made on the topics of meta reinforcement learning and learning to explore; can $L^3P$ be combined with meta learning techniques for fast landmarks generation? Second, can we learn graph-structured world models from offline datasets? Batch RL is a more realistic setting for many RL applications, since online interaction can be expensive in the real world. Applying $L^3P$ to offline datasets might require a notion of uncertainty in different parts of the graph.

## Acknowledgements

We thank the anonymous reviewers for providing helpful comments on the paper. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence (www.vectorinstitute.ai/partners).

## References


Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017.

Arthur, D. and Vassilvitskii, S. k-means++: The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.

Corneil, D., Gerstner, W., and Brea, J. Efficient model-based deep reinforcement learning with variational state tabulation. *arXiv preprint arXiv:1802.04325*, 2018.

Coulom, R. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

Dijkstra, E. W. et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Doran, J. E. and Michie, D. Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 294(1437):235–259, 1966.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Durrant-Whyte, H. and Bailey, T. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

Emmons, S., Jain, A., Laskin, M., Kurutach, T., Abbeel, P., and Pathak, D. Sparse graphical memory for robust planning. *arXiv preprint arXiv:2003.06417*, 2020.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15246–15257, 2019.

Farquhar, G., Rocktäschel, T., Igl, M., and Whiteson, S. Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. October 2017. URL http://arxiv.org/abs/1710.11417.

Gillner, S. and Mallot, H. A. Navigation and acquisition of spatial knowledge in a virtual maze. *Journal of cognitive neuroscience*, 10(4):445–463, 1998.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pp. 2450–2462, 2018.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Huang, Z., Liu, F., and Su, H. Mapping state space using landmarks for universal goal reaching. In *Advances in Neural Information Processing Systems*, pp. 1942–1952, 2019.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pp. 12519–12530, 2019.

Jurgenson, T., Avner, O., Groshev, E., and Tamar, A. Sub-goal trees–a framework for goal-based reinforcement learning. *arXiv preprint arXiv:2002.12361*, 2020.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

LaValle, S. M. Rapidly-exploring random trees: A new tool for path planning. 1998.

Lee, L., Parisotto, E., Chaplot, D. S., Xing, E., and Salakhutdinov, R. Gated path planning networks. June 2018. URL http://arxiv.org/abs/1806.06408.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Liu, K., Kurutach, T., Tung, C. K.-C., Abbeel, P., and Tamar, A. Hallucinative topological memory for zero-shot visual planning. *ArXiv*, 2019. URL https://arxiv.org/abs/2002.12336.

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pp. 9191–9200, 2018.

Nasiriany, S., Pong, V., Lin, S., and Levine, S. Planning with goal-conditioned policies. In *Advances in Neural Information Processing Systems*, pp. 14843–14854, 2019.

Pertsch, K., Rybkin, O., Ebert, F., Finn, C., Jayaraman, D., and Levine, S. Long-horizon visual planning with goal-conditioned hierarchical predictors. *arXiv preprint arXiv:2006.13205*, 2020.

Pitis, S., Chan, H., Zhao, S., Stadie, B., and Ba, J. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. *arXiv preprint arXiv:2007.02832*, 2020.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Pong, V., Gu, S., Dalal, M., and Levine, S. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., and Wierstra, D. Imagination-augmented agents for deep reinforcement learning. *Neural Information Processing Systems*, 2017.

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

Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., and Gelly, S. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018b.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016a.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. The predictron: End-to-end learning and planning. December 2016b. URL http://arxiv.org/abs/1612.08810.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. Universal planning networks. April 2018. URL http://arxiv.org/abs/1804.00645.

Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. February 2016. URL http://arxiv.org/abs/1602.02867.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Wang, R. F. and Spelke, E. S. Human spatial representation: Insights from animals. *Trends in cognitive sciences*, 6(9): 376–382, 2002.

Wang, T. and Ba, J. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

Wang, Y., Mulvaney, D., Sillitoe, I., and Swere, E. Robot navigation by waypoints. *Journal of Intelligent and Robotic Systems*, 52(2):175–207, 2008.

Yang, G., Zhang, A., Morcos, A. S., Pineau, J., Abbeel, P., and Calandra, R. Plan2vec: Unsupervised representation learning by latent plans. In *Proceedings of The 2nd Annual Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pp. 1–12, 2020. arXiv:2005.03648.

Zhao, R., Sun, X., and Tresp, V. Maximum entropy-regularized multi-goal reinforcement learning. *arXiv preprint arXiv:1905.08786*, 2019.