Learning to Navigate in Mazes with Novel Layouts using Abstract Top-down Maps

Linfeng Zhao
zhao.linf@northeastern.edu
Khoury College of Computer Sciences
Northeastern University

Lawson L.S. Wong
lsw@ccs.neu.edu
Khoury College of Computer Sciences
Northeastern University

Abstract

Learning navigation capabilities in different environments has long been one of the major challenges in decision-making. In this work, we focus on *zero-shot navigation ability* using given abstract 2-D top-down maps. Like human navigation by reading a *paper* map, the agent reads the map as an *image* when navigating in a novel layout, after learning to navigate on a set of training maps. We propose a model-based reinforcement learning approach for this multi-task learning problem, where it jointly learns a *hypermodel* that takes top-down maps as input and predicts the weights of the transition network. We use the DeepMind Lab environment and customize layouts using generated maps. Our method can adapt better to novel environments in zero-shot and is more robust to noise.

1 Introduction

If we provide a rough solution of a problem to a robot, can the robot learn to follow the solution effectively? In this paper, we study this question within the context of maze navigation, where an agent is situated within a maze whose layout has never been seen before, and the agent is expected to navigate to a goal without first training on or even exploring this novel maze. This task may appear impossible without further guidance, but we will provide the agent with additional information: an abstract 2-D top-down map, treated as an image, that illustrates the rough layout of the 3-D environment, as well as indicators of its start and goal locations ("abstract map" in Figure 1). This is akin to a tourist attempting to find a landmark in a new city: without any further help, this would be very challenging; but when equipped with a 2-D map of environment layout, the tourist can easily plan a path to reach the goal without needing to explore or train excessively.

In our case, we are most concerned with *zero-shot navigation* in novel environments, where the agent cannot perform further training or even exploration of the new environment; all that is needed to accomplish the task is technically provided by the abstract 2-D map. This differs from the vast set of approaches based on simultaneous localization and mapping (SLAM) typically used in robot navigation (Thrun et al., 2005), where the agent can explore and build an accurate *but specific* occupancy map of *each* environment *prior* to navigation. Recently, navigation approaches based on deep reinforcement learning (RL) approaches have also emerged, although they often require extensive training in the same environment (Mirowski et al., 2017; 2018). Some deep RL approaches are even capable of navigating novel environments with new goals or layouts without further training; however, these approaches typically learn the strategy of efficiently exploring the new environment to understand the layout and find the goal, then exploiting that knowledge for the remainder of the episode to repeatedly reach that goal quickly (Jaderberg et al., 2017). In contrast, since the solution is essentially provided to the agent via the abstract 2-D map, we require a more stringent version of zero-shot navigation, where it should *not* explore the new environment; instead, we expect the agent to produce a near-optimal path in its *first* (and only) approach to the goal.

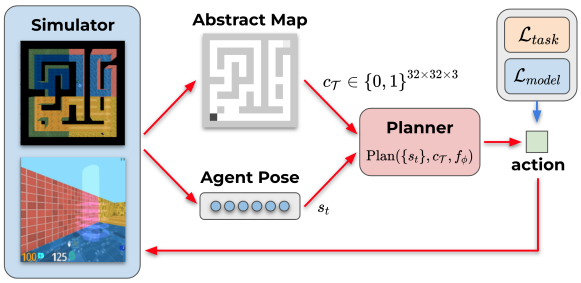

Figure 1: We develop an agent that can perform zero-shot navigation on unseen maps $\mathcal{T}$ (in DeepMind Lab, blue box), without needing to first explore the new 3-D environment. Instead, the agent is given the *top-down view* as additional guidance: an abstract 2-D occupancy map, and a goal and start position (bottom-left black dot and top-right gray dot). The map provides a rough solution solution, the path cannot be directly followed due to the continuous nature of the agent's environment, as well as unknown map scale, inaccuracies in the map, and noisy localization.

The solution to navigation using the provided abstract map seems obvious: we should localize ourselves on the abstract map (image), plan a path, and simply follow it. However, this approach suffers from a key difficulty: determining the correspondence between 2-D image maps and 3-D environments. It is not obvious how to execute the abstract plan in practice because the state and action spaces are completely different, and may even be discrete in the abstract map but continuous in the real environment.

Instead, in this paper we explore an alternative approach that avoids explicitly localizing and planning on the abstract map. The key idea is to plan in a learned model that only considers the abstract map (and start/goal information) as contextual input, but does not directly plan on the map image itself. Specifically, we propose learning a *task-conditioned hypermodel* that uses the abstract map context to produce the *environment-specific parameters* (weights) of a latent-state transition dynamics model. We then perform planning by using sampling-based forward search on this task-specific dynamics model. Importantly, although the learned transition model operates in latent state space, it uses the agent's original action space, so that planned trajectories can be directly executed in the environment, without needing to solve the aforementioned correspondence problem. The hypermodel and the state encoder are learned in an end-to-end fashion, using loss functions that assess whether the learned components were able to support effective planning.

We refer to our method as the Map-conditioned Multi-task Navigator (**MMN**). We start with a model-based RL algorithm, MuZero (Schrittwieser et al., 2020), and introduce the above task-conditioned hypermodel based on HyperNetworks (Ha et al., 2017). To tackle challenges in training, we additionally introduce an n-step generalization of Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) and an auxiliary hypermodel loss. Additionally, we introduce a model-free RL baseline, named Map-conditioned Ape-X HER DQN (**MAH**). This method builds upon DQN (Mnih et al., 2015; Horgan et al., 2018) and augments the input with the provided abstract map, and uses standard single-step HER.

In experiments performed in DeepMind Lab (Beattie et al., 2016), a 3-D maze simulation environment shown in Figure 1, we show that both approaches achieve effective zero-shot navigation in novel environment layouts, though the model-based **MMN** is significantly better at long-distance navigation. Additionally, whereas a baseline approach using deterministic path planning and reactive navigation quickly fails when the map is inaccurate or localization is noisy, our experiments suggest that **MMN** is significantly more robust to such noise.

2 Related work

Navigation is widely studied in robotics, vision, RL, and beyond; to limit the scope, we focus on zero-shot navigation in novel environments, which is most relevant to this work. This excludes

traditional approaches based on SLAM (Thrun et al., 2005), since those methods need to explicitly build a map before navigation, and the map can only be used for the corresponding environment and cannot be transferred to other layouts. Learning-based methods (e.g., Mirowski et al. (2017; 2018)) also require extensive training data from the same environment; they demonstrate generalization to new goals in the environment, but not transfer to new layouts. Jaderberg et al. (2017); Chen et al. (2019); Gupta et al. (2020); Chaplot et al. (2020) demonstrate agents that learn strategies to explore the new environment and potentially build maps of the environment during exploration; in contrast, we are interested in agents that do not need to explore the new environment. Gupta et al. (2020) learns to exploit semantic cues from its rich visual input, which is orthogonal to our work since we use the state directly. Other domains such as first-person-shooting games also involve agents navigating in novel environments (Lample & Chaplot, 2017; Dosovitskiy & Koltun, 2017; Zhong et al., 2020), but since navigation is not the primary task in those domains, the agents may not need to actually reach the specified goal (if any). Most closely related to our work is Brunner et al. (2018), who also use 2-D occupancy maps as additional input and perform experiments in DeepMind Lab. Their approach is specific to map-based navigation, whereas our methodology aims to be less domain specific. Huang et al. (2021) also use HyperNetworks on robot manipulation tasks.

Our work is an instance of *end-to-end model-based planning* (Tamar et al., 2016; Oh et al., 2017; Schrittwieser et al., 2020). It has also been referred to as *implicit model-based planning* since the model is learned implicitly. It rolls out trajectories using a learnable transition model and jointly trains the *value and policy* networks along with the *transition* network. This is different from *decoupled* model learning and planning, such as *Dyna-style* (Pong et al., 2018). One important distinction in end-to-end planning is whether the gradients are passed through the *planning computation*. For example, MuZero (Schrittwieser et al., 2020) uses sampling-based search method, Monte Carlo tree search (MCTS), that is hard to differentiate though. Other sampling-based approaches include (Hafner et al., 2019; Chua et al., 2018). Another thread of work includes Value Iteration Networks and its variants (Tamar et al., 2016; Lee et al., 2018; Zhao et al., 2023b;a), which iteratively applies Bellman operators and is easily differentiable. They have also been used in end-to-end navigation, including CMP (Gupta et al., 2020) and DAN (Karkus et al., 2019). However, they are limited to grid-like structure as the VIN backbone is 2-D convolution. Additionally, a body of work (Parisotto & Salakhutdinov, 2018; Banino et al., 2018; Fortunato et al., 2019; Wayne et al., 2018; Ma et al., 2020) studies learning structured latent models or representations useful for planning.

Our method is based on MuZero (Schrittwieser et al., 2020), which has only been used on single-map/goal navigation because it learns purely from rewards. We augment the approach with task conditioning (map and goal) to generalize to new layouts. Moro et al. (2022) also introduced goal-relabeling for AlphaZero and applied it in 2-D navigation; however, AlphaZero requires a given model, whereas MuZero jointly learns and plans with a model.

3 Problem statement

We consider a distribution of navigation tasks $\rho(\mathcal{T})$. Each task is different in two aspects: map layout and goal location. (1) *Abstract map.* The layout of each navigation task is specified by an abstract map. Specifically, an abstract map $m \in \mathbb{R}^{N\times N}$ is a 2-D occupancy grid, where cell with 1s (black) indicate walls and 0s (white) indicate nagivable spaces. A cell does not correspond to the agent's world, so the agent needs to learn to localize itself on an abstract 2-D map (i.e., to know which part of map it is currently at). We generate a set of maps and guarantee that any valid positions are reachable, i.e., there is only one connected component in a map. (2) *Goal position.* Given a map, we can then specify a pair of start and goal position. Both start and goal are represented as a "one-hot" occupancy grid $g \in \mathbb{R}^{2\times N\times N}$ provided to the agent. For simplicity, we use g to refer to both start and goal, and we denote the provided map and start-goal positions $c = (m, g)$ as the *task context*.

We formulate each navigation task as a goal-reaching *Markov decision process* (MDP), consisting of a tuple $\langle \mathcal{S}, \mathcal{A}, P, R_{\mathcal{G}}, \rho_0, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition probability function $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, $\rho_0 = \rho(s_0)$ is the initial state distribution, and $\gamma \in (0, 1]$

is the discount factor. In the learning, we assume transitions are deterministic. For each task, the objective is to reach a subset of state space $\mathcal{S}_\mathcal{G} \subset \mathcal{S}$ indicated by a reward function $R_\mathcal{G} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We denote a task as $\mathcal{T} = \langle P, R_\mathcal{G}, \rho_0 \rangle$, since a map and goal specify the dynamics and reward function of a MDP, respectively. In the episodic goal-reaching setting, the objective is typically not discounted ($\gamma = 1$) and the reward is -1 for all non-goal states, i.e., $R_\mathcal{G}(s, a) = -\mathbb{I}[s \neq g]$ for $g \in \mathcal{S}_\mathcal{G}$.

We emphasize that although the abstract map's occupancy grid corresponds to the environmental layout, the correspondence between abstract "states" (grid cells) and agent states (pose and velocity) is not known in advance, and likewise for actions (grid-cell transitions vs. forward/backward/rotate). Furthermore, the learned correspondence may not be reliable due to inaccuracies in the abstract map and localization error.

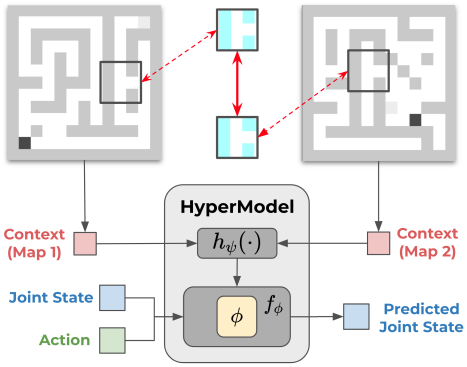

Figure 2: Applying the hypermodel h_ψ on map m_1 and m_2 outputs two sets of transition network weights $\phi_1 = h_\psi(m_1, g_1)$ and $\phi_2 = h_\psi(m_2, g_2)$. Each transition network uses their weight ϕ_i to predict the next state $f(s, a; \phi_i) = s'$, illustrated at the bottom. Since the maps may share local patterns at some scales (illustrated by the cropped 3×3 patches in light blue), they can be captured by the hypermodel h_ψ.

4 Learning to navigate using abstract maps

This section presents an approach that can effectively use abstract maps (in image form) by end-to-end model-based planning based on MuZero (Schrittwieser et al., 2020). We expect the agent to be able to efficiently *train* on multiple maps as well as *generalize* to new maps.

This poses several technical challenges. *(i)* A local change in map may introduce entirely different environment structure, so we need the model and planner to adapt to the task context in a different way than conditioning on state, and not directly condition on the entire task context. *(ii)* During training, we can only rely on a very small proportion of training tasks (e.g., 20 of 13×13 maps). This requires compositional generalization from existing map patches to novel combinations of patches. *(iii)* The reward signal is sparse, but model learning is done jointly and purely relies on reward signal. To this end, we first introduce the idea of using a *hypermodel* that learns to predict weights of transition model, instead of state output directly, to tackle *(i)* and *(ii)*. For challenge *(iii)*, we use the idea from Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) to reuse failure experience and also add an auxiliary loss of predicting transitions (described in Appendix A).

4.1 Task-conditioned hypermodel

Our goal is to create a transition model that accurately handles various map inputs, enabling planning in 3D environments with arbitrary layouts. In a single-task training schema, a straightforward approach would be to learn a parameterized transition function $f_i(s, a)$ for each individual map. However, we aim to leverage shared knowledge between navigation tasks, where maps often exhibit common local patterns and require the ability to generalize to recombination of known patterns. For instance, in Figure 2, moving right on the center of the box in the left map shares computation with the right map. By enabling the agent to recognize these local computational patterns, it can transfer to new tasks by compositional generalization.

We propose to build a *meta* network h_ψ, or *hypermodel*, to learn the "computation" of the transition model f_ψ simultaneously for all maps with abstract 2-D maps as input. The transition model for task $\mathcal{T}$ (map-goal pair) is a function f_i that maps current (latent) state and action to a next (latent) state. We parameterize a transition function f_i as a neural network with its parameter vector ϕ_i. The set $\{f_i\}$ represents transition functions of all tasks belonging to a navigation schema (e.g., a certain size of map), and these tasks have similar structure. This set of transition functions/networks are characterized by the context variables $c = (m, g)$, i.e., the abstract 2-D map and goal.[1] This implies that parameter vectors ϕ_i live in a low-dimensional manifold. Thus, we define a mapping $h : \mathcal{C} \to \Phi$ that maps the context of a task to the parameter vector ϕ_i of its transition function f_i, predicting state s' and reward r. We parameterize h also as a network with parameter ψ:[2]

$$h_\psi : c \mapsto \phi, \quad f_\phi : s, a \mapsto s', r.$$

This can be viewed as soft weight sharing between multiple tasks. It efficiently maps low-dimensional structure in the MDP, specified by the map, to computation of the transition model. It may also be viewed as a structured *learned* "dot-product" between task context $c_\mathcal{T}$ and state and action s_t, a_t to predict the next state. The idea of predicting the weights of a main network using another *meta*-network is also known as HyperNetworks (Ha et al., 2017; von Oswald et al., 2020).

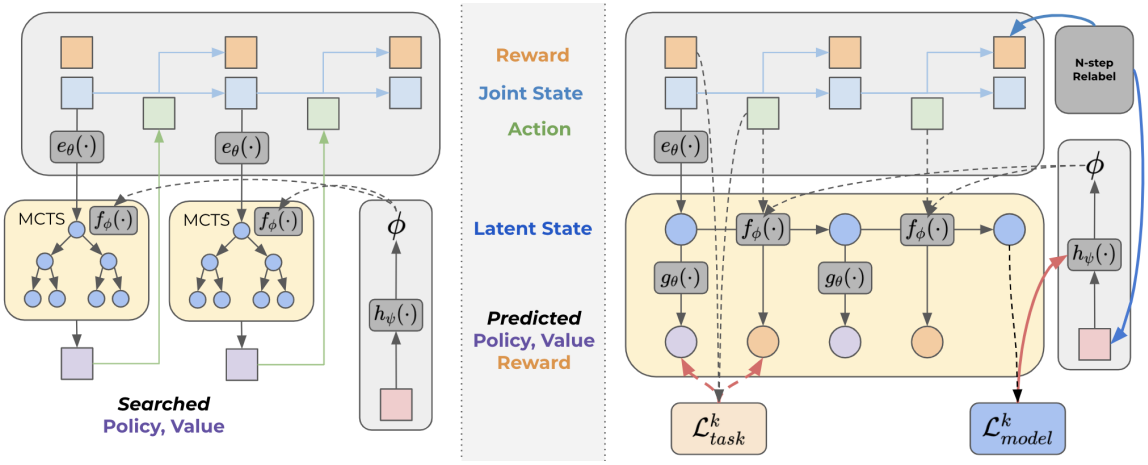

Figure 3: The planning/learning process. Yellow boxes indicate predictions; grey boxes come from actual interactions. *(Left) Inference: search with learned model.* Applying MCTS with hypermodel to search for policy and value, and act with a sampled action. *(Right) Training: building learning targets.* Computing targets and backpropagating from loss. The dark blue line indicates n-step relabelling. We only illustrate backpropagation for one reward node for simplicity. The solid red line shows the gradient flow from auxiliary model loss to the meta-network's weight ψ. The dashed red line is the gradient from task loss.

4.2 Planning using a learned hypermodel

Equipped with a map-conditioned model, we use it to search for actions according to the map layout and goal location: $(a^1, ..., a^k) = \text{Plan}(\{s_i\}, c, f_\phi)$. We follow MuZero Schrittwieser et al. (2020) to use Monte-Carlo tree search (MCTS) to search with the learned hypermodel f_ϕ. The planner needs to act based on different task inputs, which necessitates a task-dependent value function that differs from the single-task setup in MuZero. Consequently, the planner $\text{Plan}(s_i, c, f_\phi)$ must strongly correlate its computation with the map and goal input $c = (m, g)$, which presents a challenge for model-free reactive agents. As shown in Figure 3 (left), we begin by encoding the observed joint state o_t into a latent space s_t using the learned encoder $e_\theta(o_t)$. This serves as the root node of the search

[1] Concretely, a task context $c \in \mathbb{R}^{4\times N\times N}$ has four components: downsampled global occupancy map, cropped local occupancy map, and one-hot goal and start occupancy maps; N is downsampled size.

[2] We only predict weights of the transition model $f_\phi : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ which operates on a latent state space. The mapping from environment observations to latent states $e : \mathcal{O} \to \mathcal{S}$ is not predicted by a meta network. Since the latent space is low-dimensional, it is feasible to predict weight matrices of a transition network for it.

tree. To predict the next state given a latent state and a candidate action, we use the hypermodel f_ϕ. For each state (blue circle nodes), we use another network $g_\theta(s_t, c)$ to predict the policy π_t and value function v_t (not shown). These networks guide the search, where the value network estimates the future value and the policy network provides candidate actions for rollout in MCTS (blue circles), as described in Schrittwieser et al. (2020). During training, they are trained to minimize the loss with searched values and actions. Once a number of MCTS simulations are completed (yellow rounded boxes), we backup the statistics to the root node and sample an action (green boxes) from the searched action distribution (purple boxes). The trajectory and corresponding abstract map and goal $(c_\mathcal{T}, \{s_t, a_t, r_t, s_{t+1}\}_t)$ are saved to a centralized replay buffer for training.

At **zero-shot evaluation** time, given a new abstract map, we plan with the trained hypermodel: (1) given a map and goal $c_\mathcal{T} = (m_\mathcal{T}, g_\mathcal{T})$, at the beginning of the episode, compute the hypermodel weights $\phi = h(c; \psi)$ by applying the meta-network on the task context $c_\mathcal{T}$, (2) start MCTS simulations using the hypermodel $f(s, a; \phi)$ for latent state predictions, (3) get an action and transit to next state, and go to step (2) and repeat. Moreover, if we assume access to a *landmark oracle* on given maps, we can perform **hierarchical navigation** by generating a sequence of local subgoals $\{(m, g_i)\}_{i=1}^n$, and plan to sequentially achieve each landmark; see Section 5.3 for more details.

Figure 3 (right) shows our goal-relabeling scheme and loss functions; see Appendix A for details.

5 Experiments

In the experiments, we assess our method and analyze its performance on DeepMind Lab (Beattie et al., 2016) maze navigation environment. We focus on zero-shot evaluation results.

5.1 Experimental setup

We perform experiments on DeepMind Lab (Beattie et al., 2016), an RL environment suite supporting customizing 2-D map layout. As shown in Figure 1, we generate a set of abstract 2-D maps, and use them to generate 3-D environments in DeepMind Lab. Each cell on the abstract map corresponds to 100 units in the agent world. In each generated map, all valid positions are reachable, i.e., there is only one connected component in the map. Given a sampled map, we then generate a start-goal position within a given distance range. Throughout each task, the agent receives the abstract map and start/goal location indicators, the joint state vector $o \in \mathbb{R}^{12}$ (consisting of position $\mathbb{R}^3$, orientation $\mathbb{R}^3$, translational and rotational velocity $\mathbb{R}^6$), and reward signal r. The action space is {forward, backward, strafe left, strafe right, look left, look right}, with an action repeat of 10. This means that, at maximum forward velocity, the agent can traverse a 100×100 block in two steps, but typically takes longer because the agent may slow down for rotations.

Training settings We train a set of agents on a variety of training settings, which have several key options: (1) *Map size.* We mainly train on sets of $13 \times 13, 15 \times 15, 17 \times 17, 19 \times 19, 21 \times 21$ maps. One cell in the abstract map is equivalent to a 100×100 block in the agent's world. (2) *Goal distance.* During training, we generate local start-goal pairs with distance between 1 and 5 in the abstract map. (3) *Map availability.* For each map size, we train all agents on the same set of 20 generated maps, with different randomly sampled start-goal pairs in each episode.

Evaluation settings We have several settings for evaluation: (1) *Zero-shot transfer.* We mainly study this type of generalization, where the agent is presented with 20 unseen evaluation maps, and has to navigate between randomly generated start-goal pairs of varying distances. (2) *Goal distance* on abstract map. We consider both *local* navigation and *hierarchical* navigation. In the *local* case, we evaluate on a range of distances ($[1, 15]$) on a set of maps, while in the *hierarchical* case, we generate a set of landmarks with a fixed distance of 5 between them and provide these to agents sequentially. (3) *Perturbation.* To understand how errors in the abstract map and in localization affects performance, we evaluate agents with maps and poses perturbed by different strategies.

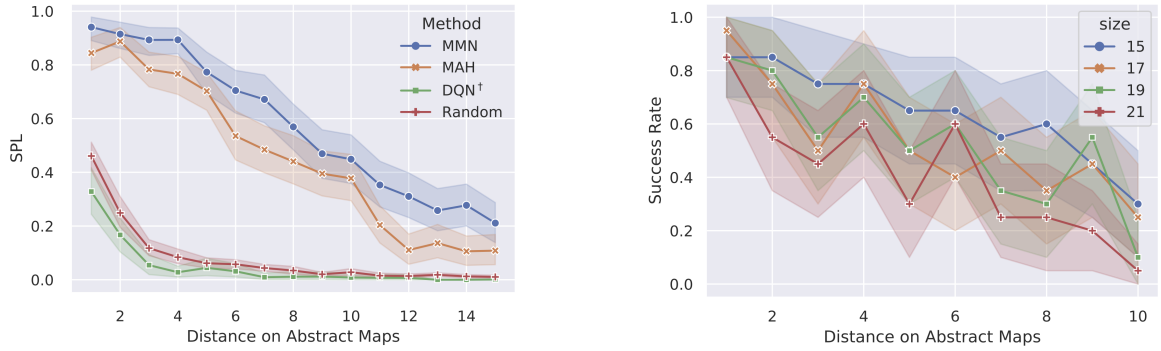

Figure 4: **(Left)** Zero-shot evaluation performance on 13×13 maps. Local navigation with different distances between start and goal, from 1 to 15. **(Right)** Performance of our method on larger maps.

Evaluation metrics We mainly report success rate and (approximate) SPL metric (Anderson et al., 2018) with 95% confidence intervals (higher SPL is better). We report results from fully trained agents to compare asymptotic performance; no training is performed on evaluation maps.

Methods We compare our model-based approach against two model-free baselines.

1. *Map-conditioned Multi-task Navigator* (**MMN**), model-based. Our map-conditioned planner based on MuZero and improved with n-step HER and multi-task training.
2. *Map-conditioned Ape-X HER DQN* (**MAH**), model-free. Based on Ape-X DQN (Horgan et al., 2018) and single-step HER (Andrychowicz et al., 2017), *conditioned* on map and goal.
3. *Single-task Ape-X HER DQN* (**DQN**†). Similar to above, but no task context c provided.
4. *Random*, a reference of the navigation performance.

5.2 Zero-shot local navigation in novel layouts

For zero-shot generalization of *locally trained agents*, we *train* all four agents on 20 of 13×13 maps with randomly generated local start-goal pairs with distance $[1, 5]$ in each episode. We train the agents until convergence; MAH typically takes $3\times$ more training episodes and steps. We *evaluate* all agents on 20 *unseen* 13×13 maps and generate 5 start-goal pairs for each distance from 1 to 15 on each map. The results are shown in Figure 4 left. MMN and MAH generally outperforms the other two baselines. MMN has better performance especially over longer distances, both in success rate and successful-trajectory length (not shown), even though it was only trained on distances ≤ 5. Since we compare fully trained agents, we found MMN performs asymptotically better than MAH. Additionally, as shown in Figure 4 right, we also train and evaluate MMN on larger maps from 15×15 to 21×21. Observed with similar trend to 13×13 maps, when trained with start-goal distance ≤ 5, the agent will find distant goals and larger maps more difficult.

5.3 Hierarchical navigation in novel layouts

We also performed a hierarchical navigation experiment, which requires an additional *landmark oracle* to generate sequences of subgoals between long-distance start-goal pairs, and evaluate the performance of hierarchical navigation. The agent is trained on 13×13 maps, and evaluate on 20 unseen 13×13 maps. On each map, we use the top-right corner as the global start position and the bottom-left corner as the global goal position, then plan a shortest path in the abstract 2-D map, and generate a sequence of subgoals with distance 5 between them; this typically results in 3 to 6 intermediate subgoals. The choice of distance 5 is motivated by our previous experiment, and because the agent is trained on distances ≤ 5. Consecutive subgoal pairs are provided sequentially to the agent as local start-goal pairs to navigate. The navigation is considered successful only if the agent reaches the global goal by the end.

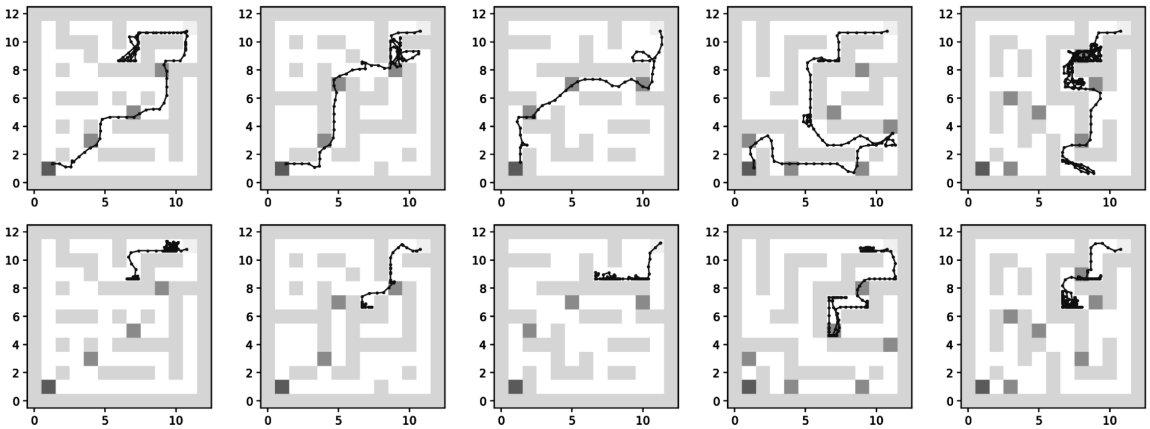

Figure 5: Trajectories from hierarchical navigation in zero-shot on 13×13 maps. The top row is for MMN and bottom row is for MAH. Since there is a fixed scaling factor from maps to environments, we can compute the corresponding location on the abstract map and visualize trajectories, although this information is *not* known to the agent. The top-right corner is the start, and the bottom-left is the goal. Darker cells indicate provided subgoals from the landmark oracle. For the first 4 tasks (columns), MMN successfully reached the goals, while MAH failed. Both methods failed in the last task (right-most column).

We evaluated MMN and MAH on the 20 evaluation maps. We provide the next subgoal when the current one is *reached* or until *timeout*. As shown in Table 1, our model-based MMN outperforms the model-free counterpart by a large margin. MMN can reach 16 out of 20 global goals, which include all 9 successful cases of MAH. We visualize five trajectories of zero-shot hierarchical navigation in Figure 5. The model-based MMN is more robust to the intermediate failed subgoals by navigating to the new subgoal directly, where the model-free MAH gets stuck frequently.

Table 1: Hierarchical navigation performance for various distances between the *landmarks*, measured by SPL and success rate (SR only shown for distance 5). Landmarks are provided subgoals between fixed start-goal pairs on 20 maps. SPL performance is not monotonic because it reflects (lack of) optimality.

Landmark Distance	1	2	3	4	5	5 (SR)
MMN	**0.61**	**0.59**	**0.68**	**0.45**	**0.63**	**0.80**
MAH	0.24	0.42	0.45	0.41	0.28	0.45
DQN†	0.00	0.00	0.00	0.00	0.00	0.00
Random	0.00	0.00	0.00	0.00	0.00	0.00

In Appendix B, we provide further experiments studying the robustness of our method to various perturbations, including situations where the abstract map contains inaccuracies and where the agent is only provided a noisy version of its location. In general, our learning-based agent is robust to these changes, though performance gradually degrades as the magnitude of perturbation increases.

6 Conclusion

In this work, we have presented an end-to-end model-based approach, MMN, for enabling agents to navigate in environments with novel layouts. By using provided abstract 2-D maps and start/goal information, MMN does not require further training or exploration (zero-shot). Compared to the map-conditioned model-free counterpart MAH, both approaches performed well in zero-shot navigation for short distances; for longer distances (with access to a landmark oracle), our model-based approach MMN performed significantly better. In future work, we will explore learned subgoal generators, handle visual observation input, and perform navigation in rich visual environments.

References

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents. *arXiv:1807.06757*, 2018.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Neural Information Processing Systems*, 2017.

Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J. Chadwick, Thomas Degris, Joseph Modayil, Greg Wayne, Hubert Soyer, Fabio Viola, Brian Zhang, Ross Goroshin, Neil Rabinowitz, Razvan Pascanu, Charlie Beattie, Stig Petersen, Amir Sadik, Stephen Gaffney, Helen King, Koray Kavukcuoglu, Demis Hassabis, Raia Hadsell, and Dharshan Kumaran. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433, 2018.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittweiser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. DeepMind lab. *arXiv:1612.03801*, 2016.

Gino Brunner, Oliver Richter, Yuyi Wang, and Roger Wattenhofer. Teaching a machine to read maps with deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2018.

Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural SLAM. In *International Conference on Learning Representations*, 2020.

Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. In *International Conference on Learning Representations*, 2019.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Neural Information Processing Systems*, 2018.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *International Conference on Learning Representations*, 2017.

Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttimore, Charlie Deck, Joel Z. Leibo, and Charles Blundell. Generalization of reinforcement learners with working and episodic memory. In *Neural Information Processing Systems*, 2019.

Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *International Journal on Computer Vision*, 128:1311–1330, 2020.

David Ha, Andrew Dai, and Quoc V Le. HyperNetworks. In *International Conference on Learning Representations*, 2017.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, 2019.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *IEEE International Conference on Robotics and Automation*, 2021.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*, 2017.

Peter Karkus, Xiao Ma, David Hsu, Leslie Pack Kaelbling, Wee Sun Lee, and Tomás Lozano-Pérez. Differentiable algorithm networks for composable robot learning. In *Robotics: Science and Systems*, 2019.

Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2017.

Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated path planning networks. In *International Conference on Machine Learning*, 2018.

Xiao Ma, Peter Karkus, David Hsu, Wee Sun Lee, and Nan Ye. Discriminative particle filter reinforcement learning for complex partial observations. In *International Conference on Learning Representations*, 2020.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *International Conference on Learning Representations*, 2017.

Piotr Mirowski, Matthew Koichi Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Koray Kavukcuoglu, Andrew Zisserman, and Raia Hadsell. Learning to navigate in cities without a map. In *Neural Information Processing Systems*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Lorenzo Moro, Amarildo Likmeta, Enrico Prati, and Marcello Restelli. Goal-directed planning via hindsight experience replay. In *International Conference on Learning Representations*, 2022.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Neural Information Processing Systems*, 2017.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *International Conference on Learning Representations*, 2018.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep RL for model-based control. In *International Conference on Learning Representations*, 2018.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Neural Information Processing Systems*, 2016.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, MA, 2005.

Johannes von Oswald, Christian Henning, Benjamin F. Grewe, and João Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, Davod Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised predictive memory in a goal-directed agent. *arXiv:1803.10760*, 2018.

Linfeng Zhao, Huazhe Xu, and Lawson L.S. Wong. Scaling up and stabilizing differentiable planning with implicit differentiation. In *International Conference on Learning Representations*, 2023a.

Linfeng Zhao, Xupeng Zhu, Lingzhi Kong, Robin Walters, and Lawson L.S. Wong. Integrating symmetry into differentiable planning with steerable convolutions. In *International Conference on Learning Representations*, 2023b.

Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. RTFM: Generalising to new environment dynamics via reading. In *International Conference on Learning Representations*, 2020.

A Further details on our approach

A.1 n-step goal relabelling: Denser reward

Jointly training a planner with learned model can suffer from lack of reward signal, especially when the model training entirely relies on reward from *multiple tasks*, which is common in model-based agents based on *value gradients* (Schrittwieser et al., 2020; Oh et al., 2017). Motivated by this, we introduce a straightforward strategy to enhance the reward signal by implicitly defining a learning curriculum, named *n-step hindsight goal relabelling*. This generalizes the single-step version of *Hindsight Experience Replay* (HER) (Andrychowicz et al., 2017) to *n-step return relabeling*.

Motivation. As shown in Figure 3 (right), we sample a trajectory of experience $(c_{\mathcal{T}}, \{s_t, a_t, r_t, s_{t+1}\}_t)$ on a specific map and goal $c_{\mathcal{T}} = (m_{\mathcal{T}}, g_{\mathcal{T}})$ from the replay buffer. Observe that, if the agent does not reach the goal area $\mathcal{S}_{\mathcal{G}}$ (a 100×100 cell in the agent's 3-D environment, denoted by a 2-D position $g_{\mathcal{T}}$ on the abstract 2-D map), it will only receive reward $r_t = -1$ during the entire episode until timeout. In large maps, this hinders the agent to learn effectively from the current map $m_{\mathcal{T}}$. Even if the agent partially understands a map, it would rarely experiences a specific goal area on the map again.[3] This is more frequent on larger maps in which possible navigable space is larger.

Relabelling n-step returns. Motivated by single-step HER, we relabel failed goals to randomly sampled *future* states (visited area) from the trajectory, and associating states with the relabelled n-step return. Concretely, the *task-conditioned* bootstrapped n-step return is

$$G_t^{\mathcal{T}} \doteq r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n v_n^{\mathcal{T}}, \quad \left[v_n^{\mathcal{T}}, \pi_n^{\mathcal{T}}\right] = g_\theta(s_t, c_{\mathcal{T}})$$

where $v_n^{\mathcal{T}}$ is the state-value function *bootstrapping* n steps into the future from the search value and *conditional* on task context $c_{\mathcal{T}}$. This task-conditioned value function is *asymmetric* since $\mathbb{R}^{12} = \mathcal{S} \neq \mathcal{S}_g = \mathbb{R}^2$.

Steps. To relabel the task-conditioned bootstrapped n-step return, there are three steps, demonstrated by the blue lines from "N-step Relabel" box. (1) *Goal (red boxes).* Randomly select a reached state $s_t \in \mathbb{R}^{12}$ from the trajectories, then take the 2-D position $(x, y) \in \mathbb{R}^2$ in agent world and convert it to a 2-D goal support grid $g_{\mathcal{T}_S}$. Then, relabel the *goal* in task context $c_{\mathcal{T}_S} = (m_{\mathcal{T}}, g_{\mathcal{T}_S})$, keeping the abstracted map and start position unchanged. (2) *Reward (orange boxes).* Recompute the rewards along the n-step segment. In episodic case, we need to terminate the episode if the agent can reach the relabelled goal area $g_{\mathcal{T}_S}$, by marking "done" at the certain timestep or assigning zero discount after that step $\gamma_t = 0$ to mask the remaining segment. (3) *Value (purple circles).* Finally, we need to recompute the bootstrapping task-conditioned value $v_n^{\mathcal{T}_S}, \pi_n^{\mathcal{T}_S} = g_\theta(s_t, c_{\mathcal{T}_S})$.

Empirically, this strategy significantly increases the efficiency of our multi-task training by providing smoothing gradients when sampling a *mini-batch* of n-step targets from successful or failed tasks. It can also be applied to other multi-task agents based on n-step return.

A.2 Joint optimization: Multi-task value learning

Our training target has two components. The first component is based on the value gradient objective in MuZero (Schrittwieser et al., 2020; Oh et al., 2017), using relabelled experiences from proposed n-step HER. It is denoted by $\mathcal{L}^k_{\text{task}}$ for step $k = 1, \ldots, K$. However, this loss is only suitable for single-task RL.

Thus, we propose an auxiliary model prediction loss, denoted by $\mathcal{L}^k_{\text{model}}$ in Figure 3 (right). The motivation is to regularize that the hypermodel $f_\phi(s, a, h_\psi(c_{\mathcal{T}}))$ should predict trajectory based on the information of given abstract map and goal $c_{\mathcal{T}}$. The objective corresponds to maximizing the mutual information between task context $c_{\mathcal{T}}$ and *predicted* trajectories $\hat{\tau}_{\mathcal{T}}$ from the hypermodel on

[3] In our *extremely* low data regime, the agent only has one start-goal pair on a small set of map. While on *low* data regime, the agent can train on randomly sampled pairs on the maps. See the Setup for more details.

sampled tasks $\mathcal{T} \sim \rho(\mathcal{T})$:

$$\max_{h_\psi} \mathbb{E}_{\mathcal{T}\sim\rho(\mathcal{T})} \left[I(c_\mathcal{T}; \hat{\tau}_\mathcal{T}) \right],$$

where $h_\psi(c_\mathcal{T}) = \phi$ is the meta network predicting the weight of transition network f_ϕ. Observe that: $I(\tau; c) = H(\tau) - H(\tau|c) \geq H(\tau) + \mathbb{E}_{\tau,c} [\log q(\tau|c)]$, we can equivalently optimize the RHS $\max_h \mathbb{E}_\mathcal{T} [\log q(\tau|c)] \iff \max_h \mathbb{E}_{(s,a,s')} [\log q(s'|s, a; h(c))]$ (subscripts omitted). This objective is equivalent to minimizing the loss between predicted states and true states from environment, for all transition tuples across all tasks. The final loss is given by the sum over multiple steps:

$$\mathcal{L}(\psi, \phi, \theta) = \sum_{k=1}^{n} \mathcal{L}^k_{\text{task}} + \mathcal{L}^k_{\text{model}},$$

where $k = 1, ..., K$, and K is the length of training segment.

B Further experiments and results

B.1 Robustness to map and localization errors

To further study the robustness of our method and the importance of each component, we considered breaking three components in closed-loop map-based navigation: Map – (1) → Path – (2) → Environment – (3) → Map (repeat). In general, our learning-based agent is robust to these changes. To illustrate the difficulty of the problem, we considered a hard-coded strategy (hand-crafted deterministic planner) based on perfect information of the environment (e.g., can plan on map) for comparison correspondingly: (1) known perfect maps and intermediate landmarks, (2) scaling factor (unavailable to MMN), and (3) world position on map. Since we assume that it has perfect localization and landmarks, the key step is to reach a landmark given current location, which consists of several procedures: (a) change the orientation to the target landmark, (b) move forward along the direction, and (c) stop at the target cell as soon as possible.

Perturbing planning We try to break the implicit assumption of requiring perfect abstract map information. We adopt the hierarchical setting, but generating subgoals on perturbed maps, where some proportion of the map's occupancy information is flipped. In Figure 6 (left), as the perturbation level increases, MMN's performance gradually decreases, but it still navigates successfully with significant noise levels.

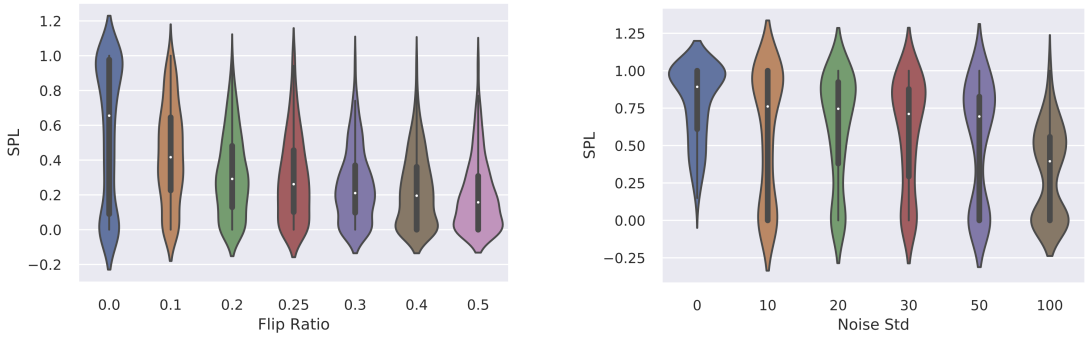

Figure 6: Violin plots show the SPL of MMN with different map flip ratio **(left)** and localization noise level **(right)**. The two figures clearly show the negative impact of imperfect information, which also justify the importance of the guidance.

Perturbing action mapping We break the implicit requirement of known scaling between map and environment. We provide the agent with randomly transformed maps with random perspective transformation, where the ratio (in both x and y directions) is different. As shown in Table 2,

perturbed MMN's performance decreases gracefully compared to unperturbed one, which shows that our agent rely little on this knowledge or any perfect relation.

Table 2: Success rate for perturbing action mapping, comparing with unperturbed MMN for reference.

Goal Distance	2	4	6	8	10
MMN (Perturbed)	0.80	0.85	0.71	0.40	0.36
MMN (Default)	0.91	0.90	0.71	0.58	0.43

Perturbing location We break the identifiability of agent position (a part of its joint state) by applying random noise to given position. We aim to show that our agent does not rely on the position to understand the map, since providing position in the agent world has no relation with localizing on abstract maps and our learning-based method can adapt to the noise. In Figure 6 (right), even though MMN is trained without noise, it tolerates some amount of noise and maintains relatively high SPL even at 50 units of noise (corresponding to 0.5 cell width). In Figure 7, we visualize the trajectories of MMN and the deterministic planner to qualitatively demonstrate MMN's robustness to noise.

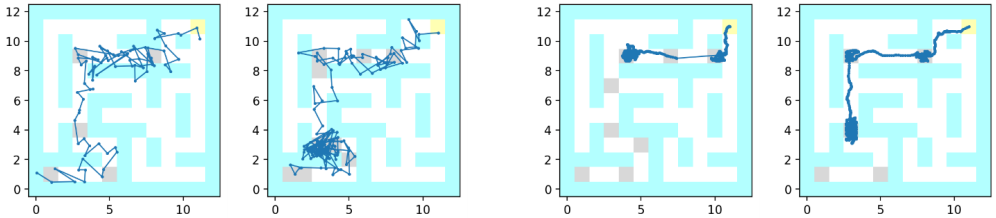

Figure 7: (**left pair**) MMN visualized with *perturbed* locations; even though the provided state is noisy, MMN successfully reaches the goal. (**right pair**) Deterministic planner is unable to reach the goal when the provided state is noisy. (We only show the *unperturbed* locations in this case for clarity in visualization.) MMN still reaches the goal with 50 units of noise (0.5 cell), while the deterministic planner gets stuck at some subgoals or runs out of budget.