UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title
Learning, Planning, and Acting with Models

Permalink
https://escholarship.org/uc/item/9n09s2xc

Author
Kurutach, Thanard

Publication Date
2021

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Learning, Planning, and Acting with Models

by

Thanard Kurutach

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Co-chair
Professor Stuart Russell, Co-chair
Professor Alison Gopnik

Spring 2021

Learning, Planning, and Acting with Models

Copyright 2021
by
Thanard Kurutach

Abstract

Learning, Planning, and Acting with Models

by

Thanard Kurutach

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Co-chair

Professor Stuart Russell, Co-chair

While the classical approach to planning and control has enabled robots to achieve various challenging control tasks, it requires domain experts to specify transition dynamics as well as inferring hand-designed symbolic states from raw observations. Therefore, bringing such method into a diverse, unstructured environment is still a grand challenge. Recent successes in computer vision and natural language processing have shed light on how robot learning could be pivotal in tackling such complexity. However, there are many challenges in deploy learning-based systems such as (1) data efficiency – how to minimize the amount of training data required, (2) generalization – how to handle tasks that the robots are not explicitly trained on, and (3) long-horizon tasks – how to simplify the optimization complexity when presented with temporal-extended tasks.

In this thesis, we present learning-and-planning methods that utilize deep neural networks to *model* the environments in different forms in order to facilitate planning and acting. We validate the efficacy of our methods in data efficiency, generalization, and long-horizon tasks on simulated locomotion benchmarks, navigation tasks, and real robot manipulation tasks.

To my parents, my siter, my wife, and my mentor

Acknowledgments

I am incredibly grateful to my opportunity in the past five years at UC Berkeley, during which I have received invaluable training, developed life-long friendship, and freely pursued my goals.

First and foremost, I would like to thank my advisors Pieter Abbeel and Stuart Russell for their commitment and support. Not only did they provide ample resources for pursuing my research goals, but also gave encouragement, feedback, and guidance along the way. I would like to thank my postdoc mentor Aviv Tamar who taught me and showed me how exciting and enjoyable research is amidst uncertainty and challenges. Thanks to their patience developing an undergrad student who knew little about research, here I am today proudly completing my PhD. thesis.

Crucially, I would like to thank my collaborators who are part of this thesis: Angelina Wang, Kara Liu, Julia Peng, Christine Tung, Rocky Duan, Ignasi Clavera, Misha Laksin, Ajay Jain, Scott Emmons, Ge Yang, Deepak Pathak, Yang Gal, Kimin Lee, Jinwoo Shin, Younggyo Seo, Sergey Levine, Chelsea Finn, Dumitru Erhan, Mohammad Babaeizadeh, and Stephen James. I would like to thank everyone in my lab and Berkeley AI lab who discussed research questions together and shared passion and inspiration with one another. Because of their presence radiating with enthusiasm, I continued to pursue my research goals. Although I cannot name every person, I would like to name just a few who brightened my time as a PhD student: Michael, Parsa, Frederik, Pim, Dinesh, Kelvin, Abhishek, Kimin, Dibya, Allan, Vitchyr, Ashvin, Chandan, and JD. Thank you for being great friends. Playing volleyball, tennis, and soccer together was so much fun!

Last but not the least, I would like to thank my parents Sunee Kurutach and Werasak Kurutach for their tireless dedication and struggles gifting me the greatest treasure of life, and their eternal supports. I would like to thank my wife Jaruwan Amtawong who is my dear friend and comrade. Finally, I would like to thank my mentor Daisaku Ikeda who never fails to motivate and encourage me to pursue my unique and noble mission.

Contents

Chapter 1

Introduction

The goal of this thesis is to provide technical contributions to enable robots to perform various tasks in unstructured environments to support and work alongside humans.

The classical approach to planning and control has enabled robots to execute various tasks ranging from manufacturing to dancing or back-flipping, illustrating the capability of our existing hardware in executing most complex and useful behaviors in the real world. However, when certain conditions in the environments slightly change, the robots tend to fail. It has been a grand challenge to program these robots to handle unstructured and diverse environments in the field of both Artificial Intelligence and Robotics.

We have seen recent advancements in deep learning including the rise of ImageNet [44, 218] in computer vision, the success of GPT-3 [27] in natural language processing, and the superhuman performance of AlphaGo in computer games [241, 242]. This suggests a great potential of applying deep learning to handle the unstructured and diverse nature of the real world. Can we apply this to robot learning and what does it look like?

Robot Learning

The most common paradigm of robot learning is considered under Reinforcement Learning (RL) [257, 219]. In this thesis, we assume an agent, an environment, the prior data (including past interactions, or demonstrations), and the set of tasks (which can be specified as reward functions or explicit start and goal configurations). The goal of the agent is to perform well in a test task while minimizing additional data required from the environment. When considering a single task and no prior data, the problem fits under standard RL.

The agent or the policy is typically modeled by neural networks learned by looking at past data and figuring out how to output better actions to perform a little bit better at solving the tasks. In the actor-critic framework, the agent also approximates the value of the future return from its current state and uses it to improve the policy. In the model-based framework [253], instead of approximating the return, we approximate the environment transitions and use it to predict the future for planning or training the policy. In this thesis, we focus on addressing pressing problems in robot learning by modeling the environment using neural networks.

1.1 Contributions

The field of robot learning has made significant progress in recent years, yet we have not seen robots operating around us. There are a few challenges in bringing robot learning into the real world: (1) data efficiency – the samples are expensive to collect especially when the robot is learning; (2) generalization – the robot needs be robust when facing certain changes in the real world; and (3) long-horizon tasks – the robot should be able to handle tasks that are temporally extended or sequential by nature such as cooking.

The key contribution of this thesis is to build a robot that can learn to model the world and use its model for reasoning about tasks and obtaining actions in order to address these challenges. Related work is covered in each chapter.

Data Efficiency

We consider data efficiency in two aspects: (1) the ability to quickly solve tasks and (2) the ability to solve new tasks with the least training additional data. We discuss these aspects in **Chapters 2 and 3** accordingly.

Model-free reinforcement learning (RL) methods are succeeding in a growing number of tasks, aided by recent advances in deep learning [176, 241, 242, 238, 236, 89]. However, they tend to suffer from high sample complexity which hinders their use in real-world domains. Alternatively, model-based reinforcement learning promises to reduce sample complexity, but tends to require careful tuning and, to date, has succeeded mainly in restrictive domains where simple models are sufficient for learning. In **Chapter 2**, we analyze the behavior of vanilla model-based reinforcement learning methods when deep neural networks are used to learn both the model and the policy, and we show that the learned policy tends to exploit regions where insufficient data is available for the model to be learned, causing instability in training. To overcome this issue, we propose to use an ensemble of models to maintain the model uncertainty and regularize the learning process. Our approach reduces the sample complexity compared to model-free deep RL methods on challenging continuous control benchmark tasks by a few hundred fold. This work was published at the International Conference on Learning Representations (ICLR) in 2018 [135].

Since then model-based reinforcement learning (RL) has shown great potential in various control tasks in terms of both sample-efficiency and final performance [135, 34, 183, 91, 90, 233]. However, learning a generalizable dynamics model robust to changes in dynamics remains a challenge since the target transition dynamics follow a multi-modal distribution. In **Chapter 3**, we present a new model-based RL algorithm that learns a multi-headed dynamics model for dynamics generalization. The main idea is updating the most accurate prediction head to specialize each head in certain environments with similar dynamics, i.e., clustering environments. Moreover, we incorporate context learning, which encodes dynamics-specific information from past experiences into the context latent vector, enabling the model to perform online adaptation to unseen environments. Finally, to utilize the specialized prediction heads more effectively, we propose an adaptive planning method, which selects the most accurate prediction head over a recent experience. Our method exhibits superior zero-shot generalization performance across a variety of control tasks, compared

to state-of-the-art RL methods. This work was published at Neural Information Processing Systems (NeurIPS) in 2020 [239].

Generalization

We consider generalization in the ability to generalize to novel tasks such as start and goal configurations, and the ability to generalize to new contexts such as obstacle positions, obstacle shapes, and agent shapes. We study the first in **Chapter 4**, and the latter in **Chapters 5 and 6**.

In recent years, deep generative models have been shown to 'imagine' convincing high-dimensional observations such as images, audio, and even video, learning directly from raw data [82, 124]. In **Chapter 4**, we ask how to imagine *goal-directed visual plans (VP)* – a plausible sequence of observations that transition a dynamical system from its current configuration to a desired goal state, which can later be used as a reference trajectory for control. We focus on systems with high-dimensional observations, such as images, and propose an approach that naturally combines representation learning and planning. Our framework learns a generative model of *sequential observations*, where the generative process is induced by a transition in a low-dimensional *planning model*, and an additional noise. By maximizing the mutual information between the generated observations and the transition in the planning model, we obtain a low-dimensional representation that best explains the causal nature of the data. We structure the planning model to be compatible with efficient planning algorithms, and we propose several such models based on either discrete or continuous states. Finally, to generate a visual plan, we project the current and goal observations onto their respective states in the planning model, plan a trajectory, and then use the generative model to transform the trajectory to a sequence of observations. We demonstrate our method on imagining plausible visual plans of rope manipulation. This work was published at Neural Information Processing Systems (NeurIPS) in 2018 [134].

In **Chapter 5**, we apply the ideas from Chapter 4 to deformable object manipulation directly on a real robot in which classical planning or reinforcement learning alone has not been shown to work. Furthermore, we extend the model to handle contextual changes in the environment such as obstacle positions. We first collect self-supervised interaction data of the robot poking the objects without any goal in mind. The key insight is to separate the problem into visual planning as discussed previously and visual tracking control which is also learn from the data as an inverse dynamics model. Finally, we demonstrate our method outperforms offline RL methods in simulated pushing tasks. This work was published at the Robotics: Science and Systems (RSS), 2019 [272].

Most previous works in visual planning, including the work described in Chapter 5, approached the problem by planning in a learned latent space, resulting in low-quality visual plans and difficulty in training. In **Chapter 6**, we propose a simple VP method that plans directly in image space and displays competitive performance. We build on the semi-parametric topological memory (SPTM) [228] method: image samples are treated as nodes in a graph, the graph connectivity is learned from image sequence data, and planning can be performed using conventional graph search methods. We propose two modifications on SPTM. First, we train an energy-based graph connectivity function using contrastive predictive coding that admits stable training. Second, to allow zero-shot planning in new domains, we learn a conditional VAE model that generates images given a context describing

the domain, and use these *hallucinated* samples for building the connectivity graph and planning. We show that this simple approach significantly outperform state-of-the-art VP methods, in terms of both plan interpretability and success rate when using the plan to guide a trajectory-following controller. Interestingly, our method can pick up non-trivial visual properties of objects, such as their geometry, and account for it in the plans. This work was published at the International Conference on Machine Learning (ICML), 2020 [154].

Long-horizon Tasks

We describe two approaches for addressing long-horizon tasks: (1) hierarchical prediction models and (2) abstract reachability models for high-level planning. We explore the first approach in **Chapter 7**, and the second in **Chapters 8 and 9**.

Prediction is a powerful tool for reinforcement learning: model-based reinforcement learning methods that utilize learned predictive models can utilize these models to plan actions into the future, accomplish new tasks at test-time, and learn efficiently from limited samples. However, prediction errors compound over time, making predictive models difficult to use for planning over long time horizons, especially when directly predicting low-level observations, such as images. In **Chapter 7**, we study how predictive models can predict and plan further into the future by employing a hierarchical prediction method, where a high-level model predicts the outcome of planning to reach sub-goals under a low-level model. The central insight in this work is that predicting the future behavior of a closed-loop planner (the lower-level model) is easier than predicting the outcomes of arbitrary action sequences. This insight allows us to successfully train high-level models to predict much further into the future at a coarser timescale, as it requires solving an easier problem than a standard hierarchical model. We apply our approach to challenging vision-based reinforcement learning tasks, where learning a model is particularly difficult. Our approach significantly improves over state-of-the-art visual model-based reinforcement learning methods on tasks ranging from visual 2D navigation to a multi-cup pouring task. This work was accepted to a workshop at the International Conference on Learning Representations (ICLR), 2020.

Rather than predicting the future, we can build a graph similar to those described **Chapter 6** in order to plan long-horizon tasks. However, most methods are still quite brittle and scale poorly with the size of the environment. In **Chapter 8**, we propose a new data structure that stores states and feasible transitions in a sparse memory. Our method aggregates states according to a novel two-way consistency objective, adapting classic state aggregation criteria to goal-conditioned RL: two states are redundant when they are interchangeable both as goals and as starting states. Theoretically, we prove that merging nodes according to two-way consistency leads to an increase in shortest path lengths that scales only linearly with the merging threshold. Experimentally, we show that our method significantly outperforms current state of the art methods on long horizon, sparse-reward visual navigation tasks. This work was published at Neural Information Processing Systems (NeurIPS) in 2020 [138].

Finally, when presented with multiple objects the size of the search space grows as well as the lengths of the tasks, in **Chapter 9**, we propose an unsupervised learning method that creates discrete, object-factorized, temporally consistent representations from exploratory video data. Our approach

plans a sequence of abstract states for a low-level model-predictive controller to follow. In our experiments, we show that it robustly solves multi-object pushing tasks, and discovers independent object representations and binary factors such as a key-and-door. This work was presented in a workshop at Neural Information Processing Systems (NeurIPS) in 2020.

Part I

Data Efficiency

Chapter 2

Deep Model Based Reinforcement Learning

Deep reinforcement learning has achieved many impressive results in recent years, including learning to play Atari games from raw-pixel inputs [177], mastering the game of Go [241, 242], as well as learning advanced locomotion and manipulation skills from raw sensory inputs [149, 238, 235, 153]. Many of these results were achieved using *model-free* reinforcement learning algorithms, which do not attempt to build a model of the environment. These algorithms are generally applicable, require relatively little tuning, and can easily incorporate powerful function approximators such as deep neural networks. However, they tend to suffer from high sample complexity, especially when such powerful function approximators are used, and hence their applications have been mostly limited to simulated environments. In comparison, model-based reinforcement learning algorithms utilize a learned model of the environment to assist learning. These methods can potentially be much more sample efficient than model-free algorithms, and hence can be applied to real-world tasks where low sample complexity is crucial [42, 149, 269]. However, so far such methods have required very restrictive forms of the learned models, as well as careful tuning for them to be applicable. Although it is a straightforward idea to extend model-based algorithms to deep neural network models, so far there has been comparatively fewer successful applications.

The standard approach for model-based reinforcement learning alternates between model learning and policy optimization. In the model learning stage, samples are collected from interaction with the environment, and supervised learning is used to fit a dynamics model to the observations. In the policy optimization stage, the learned model is used to search for an improved policy. The underlying assumption in this approach, henceforth termed *vanilla* model-based RL, is that with enough data, the learned model will be accurate enough, such that a policy optimized on it will also perform well in the real environment.

Although vanilla model-based RL can work well on low-dimensional tasks with relatively simple dynamics, we find that on more challenging continuous control tasks, performance was highly unstable. The reason is that the policy optimization tends to exploit regions where insufficient data is available to train the model, leading to catastrophic failures. Previous work has pointed out this issue as *model bias*, i.e. [42, 232, 14]. While this issue can be regarded as a form of overfitting, we emphasize that standard countermeasures from the supervised learning literature, such as regularization or cross validation, are not sufficient here – supervised learning can guarantee

generalization to states from the same distribution as the data, but the policy optimization stage steers the optimization exactly towards areas where data is scarce and the model is inaccurate. This problem is severely aggravated when expressive models such as deep neural networks are employed.

To resolve this issue, we propose to use an ensemble of deep neural networks to maintain model uncertainty given the data collected from the environment. During model learning, we differentiate the neural networks by varying their weight initialization and training input sequences. Then, during policy learning, we regularize the policy updates by combining the gradients from the imagined stochastic roll-outs. Each imagined step is uniformly sampled from the ensemble predictions. Using this technique, the policy learns to become robust against various possible scenarios it may encounter in the real environment. To avoid overfitting to this regularized objective, we use the model ensemble for early stopping policy training.

Standard model-based techniques require differentiating through the model over many time steps, a procedure known as backpropagation through time (BPTT). It is well-known in the literature that BPTT can lead to exploding and vanishing gradients [101, 25]. Even when gradient clipping is applied, BPTT can still get stuck in bad local optima. We propose to use likelihood ratio methods instead of BPTT to estimate the gradient, which only make use of the model as a simulator rather than for direct gradient computation. In particular, we use Trust Region Policy Optimization (TRPO) [238], which imposes a trust region constraint on the policy to further stabilize learning.

In this chapter, we propose Model-Ensemble Trust-Region Policy Optimization (ME-TRPO), a model-based algorithm that achieves the same level of performance as state-of-the-art model-free algorithms with $100\times$ reduction in sample complexity. We show that the model ensemble technique is an effective approach to overcome the challenge of model bias in model-based reinforcement learning. We demonstrate that replacing BPTT by TRPO yields significantly more stable learning and much better final performance. Finally, we provide an empirical analysis of vanilla model-based RL using neural networks as function approximators, and identify its flaws when applied to challenging continuous control tasks.

2.1 Related Work

There has been a large body of work on model-based reinforcement learning. They differ by the choice of model parameterization, which is associated with different ways of utilizing the model for policy learning. Interestingly, the most impressive robotic learning applications so far were achieved using the simplest possible model parameterization, namely linear models [16, 1, 148, 279, 149, 131], where the model either operates directly over the raw state, or over a feature representation of the state. Such models are very data efficient, and allows for very efficient policy optimization through techniques from optimal control. However, they only have limited expressiveness, and do not scale well to complicated nonlinear dynamics or high-dimensional state spaces, unless a separate feature learning phase is used [279].

An alternative is to use nonparametric models such as Gaussian Processes (GPs) [209, 127, 42]. Such models can effectively maintain uncertainty over the predictions, and have infinite representation power as long as enough data is available. However, they suffer from the curse of

dimensionality, and so far their applications have been limited to relatively low-dimensional settings. The computational expense of incorporating the uncertainty estimates from GPs into the policy update also imposes an additional challenge.

Deep neural networks have shown great success in scaling up model-free reinforcement learning algorithms to challenging scenarios [177, 241, 238, 235]. However, there has been only limited success in applying them to model-based RL. Although many previous studies have shown promising results on relatively simple domains [192, 231, 111, 76], so far their applications on more challenging domains have either required a combination with model-free techniques [194, 98, 184], or domain-specific policy learning or planning algorithms [145, 4, 202, 150, 67, 185]. In this work, we show that our purely model-based approach improves the sample complexity compared to methods that combine model-based and model-free elements.

Two recent studies have shown promising signs towards a more generally applicable model-based RL algorithm. Depeweg et al. [47] utilize Bayesian neural networks (BNNs) to learn a distribution over dynamics models, and train a policy using gradient-based optimization over a collection of models sampled from this distribution. Mishra et al. [174] learn a latent variable dynamic model over temporally extended segments of the trajectory, and train a policy using gradient-based optimization over the latent space. Both of these approaches have been shown to work on a fixed dataset of samples which are collected before the algorithm starts operating. Hence, their evaluations have been limited to domains where random exploration is sufficient to collect data for model learning. In comparison, our approach utilizes an iterative process of alternatively performing model learning and policy learning, and hence can be applied to more challenging domains. Additionally, our proposed improvements are orthogonal to both approaches, and can be potentially combined to yield even better results.

2.2 Preliminaries

This paper assumes a discrete-time finite-horizon Markov decision process (MDP), defined by $(\mathcal{S}, \mathcal{A}, f, r, \rho_0, T)$, in which $\mathcal{S} \subseteq \mathbb{R}^n$ is the state space, $\mathcal{A} \subseteq \mathbb{R}^m$ the action space, $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ a deterministic transition function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a bounded reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ an initial state distribution, and T the horizon. We denote a stochastic policy $\pi_\theta(a|s)$ as the probability of taking action a at state s. Let $\eta(\theta)$ denote its expected return: $\eta(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T} r(s_t, a_t)]$, where $\tau = (s_0, a_0, \ldots, a_{T-1}, s_T)$ denotes the whole trajectory, $s_0 \sim \rho_0(.)$, $a_t \sim \pi_\theta(.|s_t)$, and $s_{t+1} = f(s_t, a_t)$ for all t. We assume that the reward function is known but the transition function is unknown. Our goal is to find an optimal policy that maximizes the expected return $\eta(\theta)$.

2.3 Vanilla Model-Based Deep Reinforcement Learning

In the most successful methods of model-free reinforcement learning, we iteratively collect data, estimate the gradient of the policy, improve the policy, and then discard the data. Conversely, model-based reinforcement learning makes more extensive use of the data; it uses all the data

collected to train a model of the dynamics of the environment. The trained model can be used as a simulator in which the policy can be trained, and also provides gradient information [254, 42, 47, 253]. In the following section, we describe the vanilla model-based reinforcement learning algorithm (see Algorithm 1). We assume that the model and the policy are represented by neural networks, but the methodology is valid for other types of function approximators.

Model learning

The transition dynamics is modeled with a feed-forward neural network, using the standard practice to train the neural network to predict the change in state (rather than the next state) given a state and an action as inputs. This relieves the neural network from memorizing the input state, especially when the change is small [42, 75, 184]. We denote the function approximator for the next state, which is the sum of the input state and the output of the neural network, as $\hat{f}_\phi$.

The objective of model learning is to find a parameter ϕ that minimizes the L_2 one-step prediction loss[1]:

$$\min_\phi \frac{1}{|\mathcal{D}|} \sum_{(s_t,a_t,s_{t+1})\in\mathcal{D}} \left\| s_{t+1} - \hat{f}_\phi(s_t, a_t) \right\|_2^2, \tag{2.1}$$

where $\mathcal{D}$ is the training dataset that stores the transitions the agent has experienced. We use the Adam optimizer [122] to solve this supervised learning problem. Standard techniques are followed to avoid overfitting and facilitate the learning such as separating a validation dataset to early stop the training, and normalizing the inputs and outputs of the neural network[2]

Policy learning

Given an MDP, $\mathcal{M}$, the goal of reinforcement learning is to maximize the expected sum of rewards. During training, model-based methods maintain an approximate MDP, $\hat{\mathcal{M}}$, where the transition function is given by a parameterized model $\hat{f}_\phi$ learned from data. The policy is then updated with respect to the approximate MDP. Hence, the objective we maximize is

$$\hat{\eta}(\theta;\phi) := \mathbb{E}_{\hat{\tau}}[\sum_{t=0}^{T} r(s_t, a_t)], \tag{2.2}$$

where $\hat{\tau} = (s_0, a_0, ...)$, $s_0 \sim \rho_0(\cdot)$, $a_t \sim \pi_\theta(\cdot|s_t)$, and $s_{t+1} = \hat{f}_\phi(s_t, a_t)$.

We represent the stochastic policy[3] as a conditional multivariate normal distribution with a parametrized mean $\mu_\theta : \mathcal{S} \to \mathcal{A}$ and a parametrized standard deviation $\sigma_\theta : \mathcal{S} \to \mathbb{R}^m$. Using

[1] We found that multi-step prediction loss did not significantly improve the policy learning results.

[2] In BPTT, maintaining large weights can result in exploding gradients; normalization relieves this effect and eases the learning.

[3] Even though for generality we present the stochastic framework of BPTT, this practice is not necessary in our setting. We found that deterministic BPTT suffers less from saturation and more accurately estimate the gradient when using a policy with a small variance or a deterministic policy.

Algorithm 1 Vanilla Model-Based Deep Reinforcement Learning

Initialize a policy π_θ and a model $\hat{f}_\phi$.
Initialize an empty dataset D.
repeat
    Collect samples from the real environment f using π_θ and add them to D.
    Train the model $\hat{f}_\phi$ using D.
    repeat
        Collect fictitious samples from $\hat{f}_\phi$ using π_θ.
        Update the policy using BPTT on the fictitious samples.
        Estimate the performance $\hat{\eta}(\theta; \phi)$.
    until the performance stop improving.
until the policy performs well in real environment f.

the re-parametrization trick [98], we can write down an action sampled from π_θ at state s as $\mu_\theta(s) + \sigma_\theta(s)^T\zeta$, where $\zeta \sim \mathcal{N}(0, I_m)$. Given a trajectory $\hat{\tau}$ sampled using the policy π_θ, we can recover the noise vectors $\{\zeta_0, ..., \zeta_T\}$. Thus, the gradient of the objective $\hat{\eta}(\theta; \phi)$ can simply be estimated by Monte-Carlo methods:

$$\nabla_\theta \hat{\eta} = \mathbb{E}_{s_0 \sim \rho_0(s_0), \zeta_i \sim \mathcal{N}(0, I_m)}[\nabla_\theta \sum_{t=0}^{T} r(s_t, a_t)]. \tag{2.3}$$

This method of gradient computation is called backpropagation through time (BPTT), which can be easily performed using an automatic differentiation library. We apply gradient clipping [198] to deal with exploding gradients, and we use the Adam optimizer [122] for more stable learning. We perform the updates until the policy no longer improves its estimated performance $\hat{\eta}$ over a period of time (controlled by a hyperparameter), and then we repeat the process in the outer loop by using the policy to collect more data with respect to the real model[4]. The whole procedure terminates when the desired performance according to the real model is accomplished.

2.4 Model-Ensemble Trust-Region Policy Optimization

Using the vanilla approach described in Section 2.3, we find that the learned policy often exploits regions where scarce training data is available for the dynamics model. Since we are improving the policy with respect to the approximate MDP instead of the real one, the predictions then can be erroneous to the policy's advantage. This overfitting issue can be partly alleviated by early stopping using validation initial states in a similar manner to supervised learning. However, we found this insufficient, since the performance is still evaluated using the same learned model, which tends to make consistent mistakes. Furthermore, although gradient clipping can usually resolve

[4] In practice, to reduce variance in policy evaluation, the initial states are chosen from the sampled trajectories rather than re-sampled from ρ_0.

exploding gradients, BPTT still suffers from vanishing gradients, which cause the policy to get stuck in bad local optima [25, 198]. These problems are especially aggravated when optimizing over long horizons, which is very common in reinforcement learning problems.

We now present our method, Model-Ensemble Trust-Region Policy Optimization (ME-TRPO). The pseudocode is shown in Algorithm 2. ME-TRPO combines three modifications to the vanilla approach. First, we fit a set of dynamics models $\{f_{\phi_1}, \ldots, f_{\phi_K}\}$ (termed a *model ensemble*) using the same real world data. These models are trained via standard supervised learning, as described in Section 2.3, and they only differ by the initial weights and the order in which mini-batches are sampled. Second, we use Trust Region Policy Optimization (TRPO) to optimize the policy over the model ensemble. Third, we use the model ensemble to monitor the policy's performance on validation data, and stops the current iteration when the policy stops improving. The second and third modifications are described in detail below.

Policy Optimization. To overcome the issues with BPTT, we use likelihood-ratio methods from the model-free RL literature. We evaluated using Vanilla Policy Gradient (VPG) [201], Proximal Policy Optimization (PPO) [236], and Trust Region Policy Optimization (TRPO) [238]. The best results were achieved by TRPO. In order to estimate the gradient, we use the learned models to simulate trajectories as follows: in every step, we randomly choose a model to predict the next state given the current state and action. This avoids the policy from overfitting to any single model during an episode, leading to more stable learning.

Policy Validation. We monitor the policy's performance using the K learned models. Specifically, we compute the ratio of models in which the policy improves:

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{1}[\hat{\eta}(\theta_{new};\phi_k) > \hat{\eta}(\theta_{old};\phi_k)]. \tag{2.4}$$

The current iteration continues as long as this ratio exceeds a certain threshold. In practice, we validate the policy after every 5 gradient updates and we use 70% as the threshold. If the ratio falls below the threshold, a small number of updates is tolerated in case the performance improves, the current iteration is terminated. Then, we repeat the overall process of using the policy to collect more real-world data, optimize the model ensemble, and using the model ensemble to improve the policy. This process continues until the desired performance in the real environment is reached.

The model ensemble serves as effective regularization for policy learning: by using the model ensemble for policy optimization and validation, the policy is forced to perform well over a vast number of possible alternative futures. Even though any of the individual models can still incur model bias, our experiments below suggest that combining these models yields stable and effective policy improvement.

2.5 Experiments

We design the experiments to answer the following questions:

Algorithm 2 Model Ensemble Trust Region Policy Optimization (ME-TRPO)

Initialize a policy π_θ and all models $\hat{f}_{\phi_1}, \hat{f}_{\phi_2}, ..., \hat{f}_{\phi_K}$.
Initialize an empty dataset $\mathcal{D}$.
repeat
  Collect samples from the real system f using π_θ and add them to D.
  Train all models using $\mathcal{D}$.
  repeat ▷ Optimize π_θ using all models.
    Collect fictitious samples from $\{\hat{f}_{\phi_i}\}_{i=1}^{K}$ using π_θ.
    Update the policy using TRPO on the fictitious samples.
    Estimate the performances $\hat{\eta}(\theta; \phi_i)$ for $i = 1, ..., K$.
  until the performances stop improving.
until the policy performs well in real environment f.

1. How does our approach compare against state-of-the-art methods in terms of sample complexity and final performance?

2. What are the failure scenarios of the vanilla algorithm?

3. How does our method overcome these failures?

Our video and code in these experiments are available at `sites.google.com/view/me-trpo` and `github.com/thanard/me-trpo`.

We also provide in the Appendix A.4 an ablation study to characterize the effect of each component of our algorithm.

Environments

To answer these questions, we evaluate our method and various baselines over six standard continuous control benchmark tasks [53, 99] in Mujoco [263]: Swimmer, Snake, Hopper, Ant, Half Cheetah, and Humanoid, shown in Figure 2.1. The details of the tasks can be found in Appendix A.1.

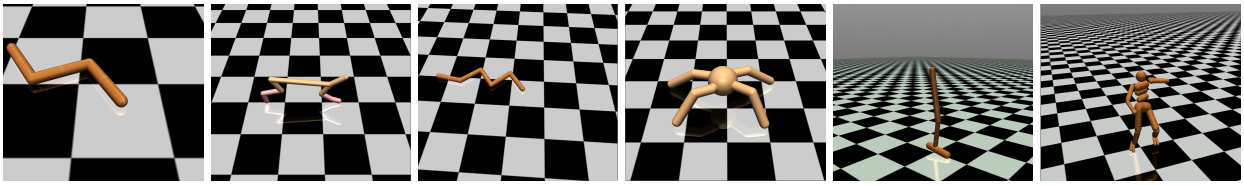


Figure 2.1: Mujoco environments used in our experiments. Form left to right: Swimmer, Half Cheetah, Snake, Ant, Hopper, and Humanoid.

Comparison to state-of-the-art

We compare our method with the following state-of-the-art reinforcement learning algorithms in terms of sample complexity and performance: Trust Region Policy Optimization (TRPO) [238], Proximal Policy Optimization (PPO) [236], Deep Deterministic Policy Gradient (DDPG) [153], and Stochastic Value Gradient (SVG) [98].

The results are shown in Figure 2.2. Prior model-based methods appear to achieve worse performance compared with model-free methods. In addition, we find that model-based methods tend to be difficult to train over long horizons. In particular, SVG(1), not presented in the plots, is very unstable in our experiments. While SVG(∞) is more stable, it fails to achieve the same level of performance as model-free methods. In contrast, our proposed method reaches the same level of performance as model-free approaches with $\approx 100\times$ less data. To the best of our knowledge, it is the first purely model-based approach that can optimize policies over high-dimensional motor-control tasks such as Humanoid. For experiment details please refer to Appendix A.1.

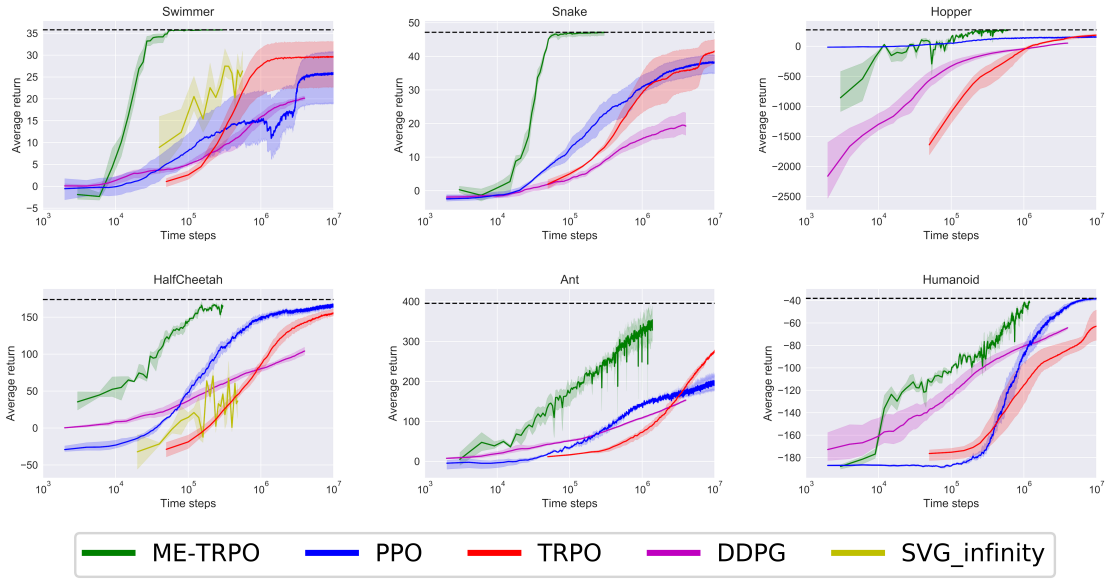


Figure 2.2: Learning curves of our method versus state-of-the-art methods. The horizontal axis, in log-scale, indicates the number of time steps of real world data. The vertical axis denotes the average return. These figures clearly demonstrate that our proposed method significantly outperforms other methods in comparison (best viewed in color).

From vanilla to ME-TRPO

In this section we explain and quantify the failure cases of vanilla model-based reinforcement learning, and how our approach overcomes such failures. We analyze the effect of each of our proposed modifications by studying the learning behavior of replacing BPTT with TRPO in vanilla model-based RL using just a single model, and then the effect of using an ensemble of models.

As discussed above, BPTT suffers from exploding and vanishing gradients, especially when optimizing over long horizons. Furthermore, one of the principal drawbacks of BPTT is the

assumption that the model derivatives should match that of the real dynamics, even though the model has not been explicitly trained to provide accurate gradient information. In Figure 2.3 we demonstrate the effect of using policy gradient methods that make use of a score function estimator, such as VPG and TRPO, while using a single learned model. The results suggest that in comparison with BPTT, policy gradient methods are more stable and lead to much better final performance. By using such model-free algorithms, we require less information from the learned model, which only acts as a simulator. Gradient information through the dynamics model is not needed anymore to optimize the policy.

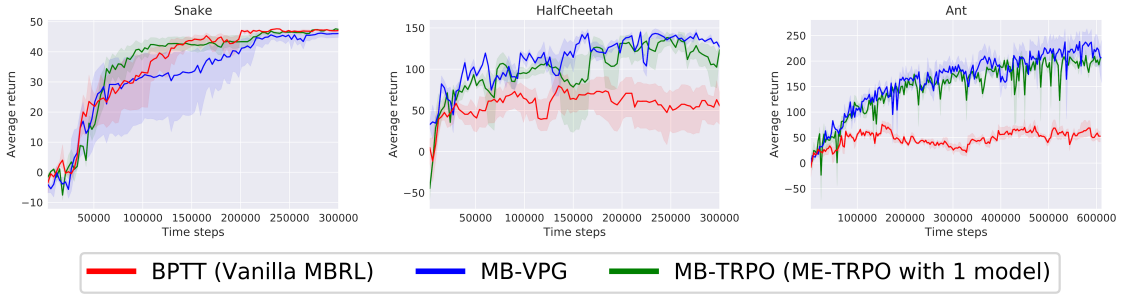


Figure 2.3: Comparison among different policy optimization techniques with one model. Using TRPO for model-based optimization leads to the best policy learning across the different domains (Best viewed in color).

However, while replacing BPTT by TRPO helps optimization, the learned policy can still suffer from model bias. The learning procedure tends to steer the policy towards regions where it has rarely visited, so that the model makes erroneous predictions to its advantage. The estimated performances of the policy often end up with high rewards according to the learned model, and low rewards according to the real one (see Appendix A.2 for further discussion). In Figure 2.4, we analyze the effect of using various numbers of ensemble models for sampling trajectories and validating the policy's performance. The results indicate that as more models are used in the model ensemble, the learning is better regularized and the performance continually improves. The improvement is even more noticeable in more challenging environments like HalfCheetah and Ant, which require more complex dynamics models to be learned, leaving more room for the policy to exploit when model ensemble is not used.

2.6 Discussion

In this chapter, we present a simple and robust model-based reinforcement learning algorithm that is able to learn neural network policies across different challenging domains. We show that our approach significantly reduces the sample complexity compared to state-of-the-art methods while reaching the same level of performance. In comparison, our analyses suggests that vanilla model-based RL tends to suffer from model bias and numerical instability, and fails to learn a good policy. We further evaluate the effect of each key component of our algorithm, showing that both using TRPO and model ensemble are essential for successful applications of deep model-based

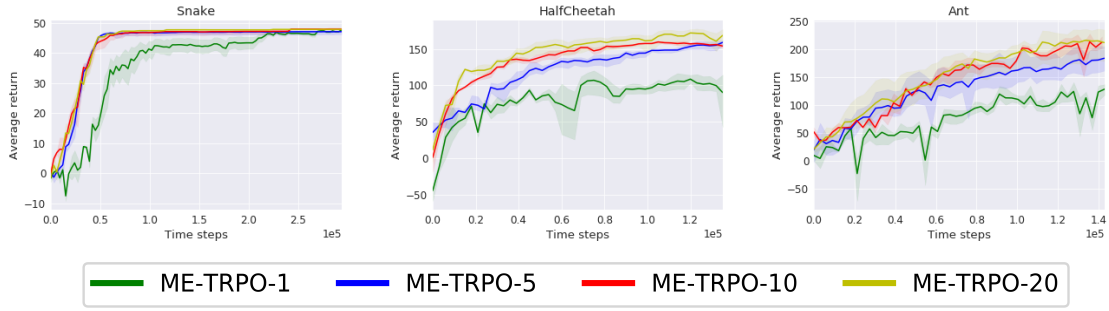


Figure 2.4: Comparison among different number of models that the policy is trained on. TRPO is used for the policy optimization. We illustrate the improvement when using 5, 10 and 20 models over a single model (Best viewed in color).

RL. We also confirm the results of previous work [42, 47, 76] that using model uncertainty is a principled way to reduce model bias. In the next chapter, we will study how model-based RL can efficiently adapt to unseen dynamics without additional training data.

Chapter 3

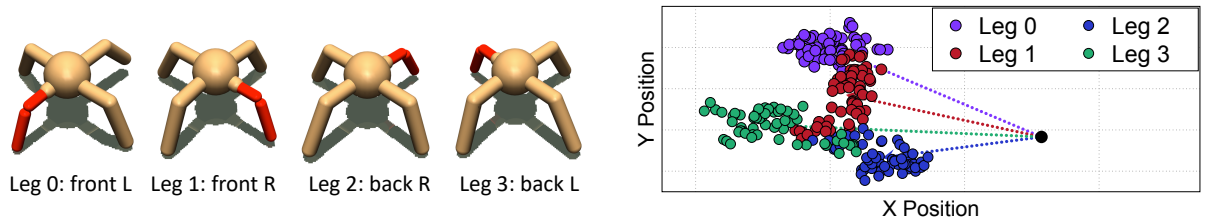
Dynamics Adaptation

Deep reinforcement learning (RL) has exhibited wide success in solving sequential decision-making problems [148, 242, 270]. Early successful deep RL approaches had been mostly model-free, which do not require an explicit model of the environment, but instead directly learn a policy [152, 177, 238]. However, despite the strong asymptotic performance, the applications of model-free RL have largely been limited to simulated domains due to its high sample complexity. For this reason, model-based RL has been gaining considerable attention as a sample-efficient alternative, with an eye towards robotics and other physics domains.

The increased sample-efficiency of model-based RL algorithms is obtained by exploiting the structure of the problem: first the agent learns a predictive model of the environment, and then plans ahead with the learned model [14, 233, 260]. Recently, substantial progress has been made on the sample-efficiency of model-based RL algorithms [36, 43, 145, 148, 149]. However, it has been evidenced that model-based RL algorithms are not robust to changes in the dynamics [143, 183], i.e., dynamics models fail to provide accurate predictions as the transition dynamics of environments change. This makes model-based RL algorithms unreliable to be deployed into real-world environments where partially unspecified dynamics are common; for instance, a deployed robot might not know a priori various features of the terrain it has to navigate.

As a motivating example, we visualize the next states obtained by crippling one of the legs of an ant robot (see Figure 3.1a). Figure 3.1b shows that the target transition dynamics follow a multi-modal distribution, where each mode corresponds to each leg of a robot, even though the original environment has deterministic transition dynamics. This implies that a model-based RL algorithm that can approximate the multi-modal distribution is required to develop a reliable and robust agent against changes in the dynamics. Several algorithms have been proposed to tackle this problem, e.g., learning contextual information to capture local dynamics [143], fine-tuning model parameters for fast adaptation [183]. These algorithms, however, are limited in that they do not explicitly learn dynamics models that can approximate the multi-modal distribution of transition dynamics.

In this chapter, we present a new model-based RL algorithm, coined trajectory-wise multiple choice learning (T-MCL), that can approximate the multi-modal distribution of transition dynamics in an unsupervised manner. To this end, we introduce a novel loss function, trajectory-wise oracle



(a) Ant with crippled legs

(b) Multi-modality in transition dynamics

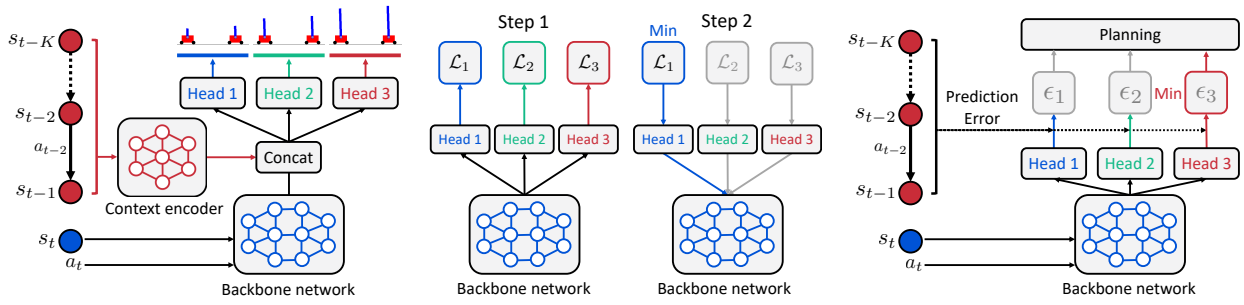
Figure 3.1: (a) Examples from ant robots with crippled legs. (b) We visualize the next (x, y) positions of robots obtained by applying the same action with random noise to robots at the initial start position but with a different crippled leg.

loss, for learning a multi-headed dynamics model where each prediction head specializes in different environments (see Figure 3.2a). By updating the most accurate prediction head over a trajectory segment (see Figure 3.2b), we discover that specialized prediction heads emerge automatically. Namely, our method can effectively cluster environments without any prior knowledge of environments. To further enable the model to perform online adaptation to unseen environments, we also incorporate context learning, which encodes dynamics-specific information from past experiences into the context latent vector and provides it as an additional input to prediction heads (see Figure 3.2a). Finally, to utilize the specialized prediction heads more effectively, we propose adaptive planning that selects actions using the most accurate prediction head over a recent experience, which can be interpreted as finding the nearest cluster to the current environment (see Figure 3.2c).

We demonstrate the effectiveness of T-MCL on various control tasks from OpenAI Gym [26]. For evaluation, we measure the generalization performance of model-based RL agents on unseen (yet related) environments with different transition dynamics. In our experiments, T-MCL exhibits superior generalization performance compared to existing model-based RL methods [34, 143, 183]. For example, compared to CaDM [143], a state-of-the-art model-based RL method for dynamics generalization, our method obtains 3.5x higher average return on the CrippledAnt environment.

3.1 Related work

Model-based reinforcement learning. By learning a forward dynamics model that approximates the transition dynamics of environments, model-based RL attains a superior sample-efficiency. Such a learned dynamics model can be used as a simulator for model-free RL methods [107, 135, 254], providing a prior or additional features to a policy [52, 282], or planning ahead to select actions by predicting the future consequences of actions [14, 145, 260]. A major challenge in model-based RL is to learn accurate dynamics models that can provide correct future predictions. To this end, numerous methods thus have been proposed, including ensembles [34] and latent dynamics models [91, 90, 233]. While these methods have made significant progress even in complex domains [90, 233], dynamics models still struggle to provide accurate predictions on unseen environments [143, 183].



(a) Multi-headed dynamics model (b) Multiple choice learning (c) Adaptive planning

Figure 3.2: Illustrations of our framework. (a) Our dynamics model consists of multiple prediction heads conditioned on a context latent vector. (b) We train our multi-headed dynamics model with the proposed trajectory-wise oracle loss (3.2). (c) For planning, we select the most accurate prediction head over a recent experience.

Multiple choice learning. Multiple choice learning (MCL) [86, 87] is an ensemble method where the objective is to minimize an oracle loss, making at least one ensemble member predict the correct answer. By making the most accurate model optimize the loss, MCL encourages the model to produce multiple outputs of high quality. Even though several optimization algorithms [48, 86] have been proposed for MCL objectives, it is challenging to employ these for learning deep neural networks due to training complexity. To address this problem, Lee at al. [144] proposed a stochastic gradient descent based algorithm. Recently, several methods [142, 180, 262] have been proposed to further improve MCL by tackling the issue of overconfidence problem of neural networks. While most prior works on MCL have focused on supervised learning, our method applies the MCL algorithm to model-based RL.

Dynamics generalization and adaptation in model-based RL. Prior dynamics generalization methods have aimed to either encode inductive biases into the architecture [225] or to learn contextual information that captures the local dynamics [143]. Notably, Lee et al. [143] introduced a context encoder that captures dynamics-specific information of environments, and improved the generalization ability by providing a context latent vector as additional inputs. Our method further improves this method by combining multiple choice learning and context learning.

For dynamics adaptation, several meta-learning based methods have been studied [183, 182, 221]. Recently, Nagabandi et al. [183] proposed a model-based meta-RL method that adapts to recent experiences either by updating model parameters via a small number of gradient updates [65] or by updating hidden representations of a recurrent model [54]. Our method differs from this method, in that we do not fine-tune the model parameters to adapt to new environments at evaluation time.

3.2 Problem setup

We consider the standard RL framework where an agent optimizes a specified reward function through interacting with an environment. Formally, we formulate our problem as a discrete-time

Markov decision process (MDP) [257], which is defined as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0)$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p\left(s'|s,a\right)$ is the transition dynamics, $r\left(s,a\right)$ is the reward function, ρ_0 is the initial state distribution, and $\gamma \in [0,1)$ is the discount factor. The goal of RL is to obtain a policy, mapping from states to actions, that maximizes the expected return defined as the total accumulated reward. We tackle this problem in the context of model-based RL by learning a forward dynamics model f, which approximates the transition dynamics $p\left(s'|s,a\right)$. Then, dynamics model f is used to provide training data for a policy or predict the future consequences of actions for planning.

In order to address the problem of generalization, we further consider the distribution of MDPs, where the transition dynamics $p_c\left(s'|s,a\right)$ varies according to a context c. For instance, a robot agent's transition dynamics may change when some of its parts malfunction due to unexpected damages. Our goal is to learn a generalizable forward dynamics model that is robust to such dynamics changes, i.e., approximating the multi-modal distribution of transition dynamics $p\left(s'|s,a\right) = \int_c p\left(c\right) p_c\left(s'|s,a\right)$. Specifically, given a set of training environments with contexts sampled from $p_{train}(c)$, we aim to learn a forward dynamics model that can produce accurate predictions for both training environments and test environments with unseen (but related) contexts sampled from $p_{test}(c)$.

3.3 Trajectory-wise multiple choice learning

In this section, we propose a trajectory-wise multiple choice learning (T-MCL) that learns a multi-headed dynamics model for dynamics generalization. We first present a trajectory-wise oracle loss for making each prediction head specialize in different environments, and then introduce a context-conditional prediction head to further improve generalization. Finally, we propose an adaptive planning method that generates actions by planning under the most accurate prediction head over a recent experience for planning.

Trajectory-wise oracle loss for multi-headed dynamics model

To approximate the multi-modal distribution of transition dynamics, we introduce a multi-headed dynamics model $\{f\left(s_{t+1}|b(s_t,a_t;\theta);\theta_h^{\texttt{head}}\right)\}_{h=1}^{H}$ that consists of a backbone network b parameterized by θ and H prediction heads parameterized by $\{\theta_h^{\texttt{head}}\}_{h=1}^{H}$ (see Figure 3.2a). To make each prediction head specialize in different environments, we propose a trajectory-wise oracle defined as follows:

$$\mathcal{L}^{\texttt{T-MCL}} = \mathbb{E}_{\tau_{t,M}^{\texttt{F}} \sim \mathcal{B}} \left[\min_{h \in [H]} -\frac{1}{M} \sum_{i=t}^{t+M-1} \log f\left(s_{i+1} \mid b(s_i,a_i;\theta);\theta_h^{\texttt{head}}\right) \right], \tag{3.1}$$

where $[H]$ is the set $\{1, \cdots, H\}$, $\tau_{t,M}^{\texttt{F}} = (s_t, a_t, \cdots, s_{t+M-1}, a_{t+M-1}, s_{t+M})$ denotes a trajectory segment of size M, and $\mathcal{B} = \{\tau_{t,M}^{\texttt{F}}\}$ is the training dataset. The proposed loss is designed to only update the most accurate prediction head over each trajectory segment for specialization (see Figure 3.2b). By considering the accumulated prediction error over trajectory segments,

Algorithm 3 Trajectory-wise MCL (T-MCL).

Initialize parameters of backbone network θ, prediction heads $\{\theta_h^{\texttt{head}}\}_{h=1}^H$, context encoder ϕ.
Initialize dataset $\mathcal{B} \leftarrow \emptyset$.
for each iteration **do**
  Sample $c \sim p_{seen}(c)$. // ENVIRONMENT INTERACTION
  for $t = 1$ **to** TaskHorizon **do**
    Get context latent vector $z_t = g\left(\tau_{t,K}^{\texttt{P}}; \phi\right)$ and select the best prediction head h^* from (3.3)
    Collect $\{(s_t, a_t, s_{t+1}, r_t, \tau_{t,K}^{\texttt{P}})\}$ from environment with transition dynamics p_c using h^*.
    Update $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1}, r_t, \tau_{t,K}^{\texttt{P}})\}$.
  end for
  Initialize $L_{\texttt{tot}} \leftarrow 0$ // DYNAMICS AND CONTEXT LEARNING
  Sample $\{\tau_{t_j,K}^P, \tau_{t_j,M}^F\}_{j=1}^B \sim \mathcal{B}$
  for $j = 1$ **to** B **do**
    for $h = 1$ **to** H **do**
      Compute the loss of the h-th prediction head:

$$L_j^h \leftarrow -\frac{1}{M} \sum_{i=t_j}^{t_j+M-1} \log f\left(s_{i+1} \mid b(s_i, a_i; \theta), g(\tau_{i,K}^{\texttt{P}}; \phi); \theta_h^{\texttt{head}}\right)$$

    end for
    Find $h^* = \arg\min_{h \in [H]} L_j^h$ and update $L_{\texttt{tot}} \leftarrow L_{\texttt{tot}} + L_j^{h^*}$
  end for
  Update $\theta, \phi, \{\theta_h^{\texttt{head}}\}_{h=1}^H \leftarrow \nabla_{\theta,\phi,\{\theta_h^{\texttt{head}}\}_{h=1}^H} L_{\texttt{tot}}$
end for

the proposed oracle loss can assign trajectories from different transition dynamics to different transition heads more distinctively (see Figure 3.4 for supporting experimental results). Namely, our method clusters environments in an unsupervised manner. We also remark that the shared backbone network learns common features across all environments, which provides several advantages, such as improving sample-efficiency and reducing computational costs.

Context-conditional multi-headed dynamics model

To further enable the dynamics model to perform online adaptation to unseen environments, we introduce a context encoder g parameterized by ϕ, which produces a latent vector $g\left(\tau_{t,K}^{\texttt{P}}; \phi\right)$ given K past transitions $(s_{t-K}, a_{t-K}, \cdots, s_{t-1}, a_{t-1})$. This context encoder operates under the assumption that the true context of the underlying MDP can be captured from recent experiences [143, 183, 208, 289]. Using this context encoder, we propose to learn a context-conditional multi-headed dynamics

model optimized by minimizing the following oracle loss:

$$\mathcal{L}^{\texttt{T-MCL}}_{\texttt{context}} = \mathbb{E}_{(\tau^{\texttt{F}}_{t,M}, \tau^{\texttt{P}}_{t,K}) \sim \mathcal{B}} \left[\min_{h \in [H]} -\frac{1}{M} \sum_{i=t}^{t+M-1} \log f\left(s_{i+1} \mid b(s_i, a_i; \theta), g(\tau^{\texttt{P}}_{i,K}; \phi); \theta^{\texttt{head}}_h\right) \right]. \quad (3.2)$$

We remark that the dynamics generalization of T-MCL can be enhanced by incorporating the contextual information into the dynamics model for enabling its online adaptation. To extract more meaningful contextual information, we also utilize various auxiliary prediction losses proposed in [143] (see the supplementary material for more details).

Adaptive planning

Once a multi-headed dynamics model is learned, it can be used for selecting actions by planning. Since the performance of planning depends on the quality of predictions, it is important to select the prediction head specialized in the current environment for planning. To this end, following the idea of [189], we propose an adaptive planning method that selects the most accurate prediction head over a recent experience (see Figure 3.2c). Formally, given N past transitions, we select the prediction head h^* as follows:

$$\arg\min_{h \in [H]} \sum_{i=t-N}^{t-2} \ell\left(s_{i+1}, f\left(b(s_i, a_i), g(\tau^{\texttt{P}}_{i,K}; \phi); \theta^{\texttt{head}}_h\right)\right), \quad (3.3)$$

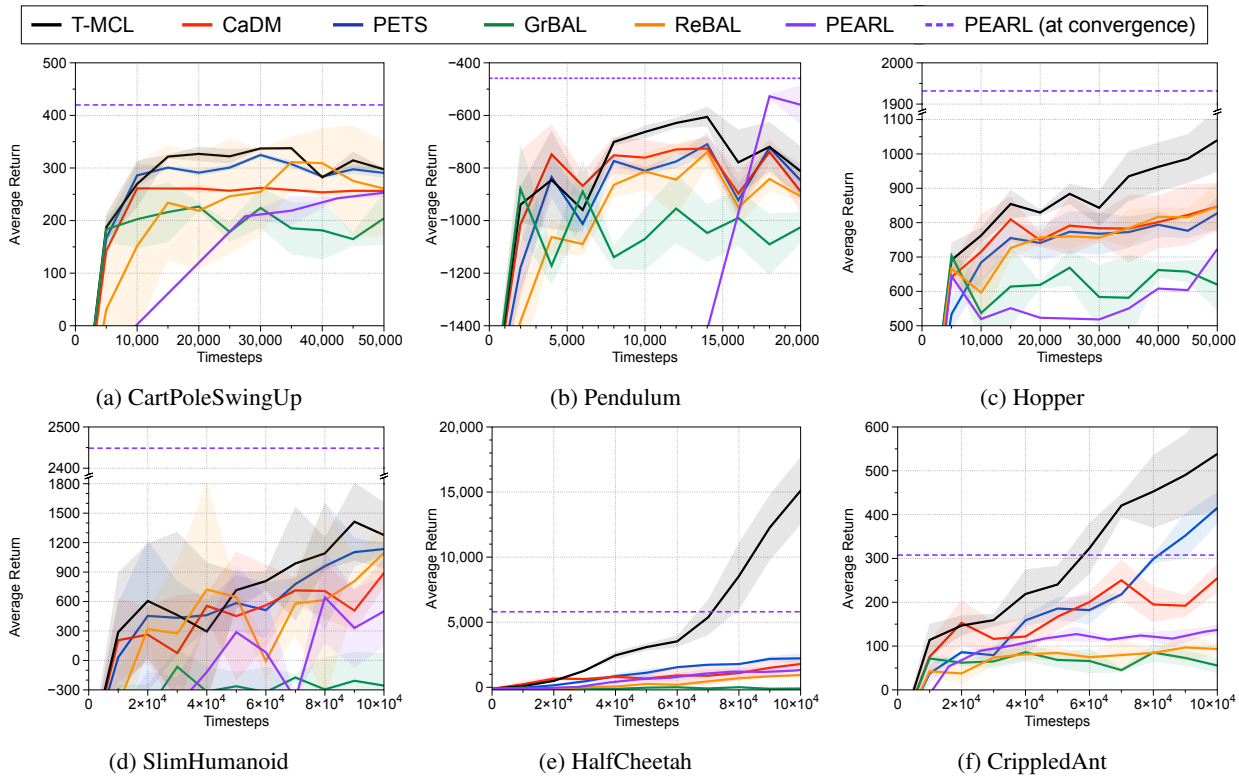
where ℓ is the mean squared error function. One can note that this planning method corresponds to finding the nearest cluster to the current environment. We empirically show that this adaptive planning significantly improves the performance by selecting the prediction head specialized in a specific environment (see Figure 3.6b for supporting experimental results).

3.4 Experiments

In this section, we designed our experiments to answer the following questions:

- How does our method compare to existing model-based RL methods and state-of-the-art model-free meta-RL method (see Figure 3.3)?
- Can prediction heads be specialized for a certain subset of training environments with similar dynamics (see Figure 3.4 and Figure 3.5)?
- Is the multi-headed architecture useful for dynamics generalization of other model-based RL methods (see Figure 3.6a)?
- Does adaptive planning improve generalization performance (see Figure 3.6b)?
- Can T-MCL extract meaningful contextual information from complex environments (see Figure 3.6c and Figure 3.6d)?

Source code and videos are available at `sites.google.com/view/trajectory-mcl`.



(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

Figure 3.3: The average returns of trained dynamics models on unseen environments. Dotted lines indicate performance at convergence. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.

Setups

Environments. We demonstrate the effectiveness of our proposed method on classic control problems (i.e., CartPoleSwingUp and Pendulum) from OpenAI Gym [26] and simulated robotic continuous tasks (i.e., Hopper, SlimHumanoid, HalfCheetah, and CrippledAnt) from MuJoCo physics engine [263]. To evaluate the generalization performance, we designed environments to follow a multi-modal distribution by changing the environment parameters (e.g., length and mass) similar to [197] and [289]. We use the two predefined discrete set of environment parameters for training and test environments, where parameters for test environments are outside the range of training parameters. Then, we learn a dynamics model on environments whose transition dynamics are characterized by the environment parameters randomly sampled before the episode starts. For evaluation, we report the performance of a trained dynamics model on unseen environments whose environment parameters are randomly sampled from the test parameter set. Similar to prior works [34, 183, 275], we assume that the reward function of environments is known, i.e., ground-truth rewards at the predicted states are available for planning. For all our experiments, we report the mean and standard deviation across three runs. We provide more details in the supplementary material.

Planning. We use a model predictive control (MPC) [161] to select actions based on the learned dynamics model. Specifically, we use the cross entropy method (CEM) [41] to optimize action sequences by iteratively re-sampling action sequences near the best performing action sequences from the last iteration.

Implementation details of T-MCL. For all experiments, we use an ensemble of multi-headed dynamics models that are independently optimized with the trajectory-wise oracle loss. We reduce modeling errors by training multiple dynamics models [34]. To construct a trajectory segment $\tau^{\mathrm{F}}_{t,M}$ in equation 3.1, we use M transitions randomly sampled from a trajectory instead of consecutive transitions $(s_t, a_t, \cdots, s_{t+M})$. We empirically found that this stabilizes the training, by breaking the temporal correlations of the training data. We also remark that the same hyperparameters are used for all experiments except Pendulum which has a short task horizon. We provide more details in the supplementary material.

Baselines. To evaluate the performance of our method, we consider following model-based and model-free RL methods:

- Probabilistic ensemble dynamics model (PETS) [34]: an ensemble of probabilistic dynamics models that captures the uncertainty in modeling and planning. PETS employs ensembles of single-headed dynamics models optimized to cover all training environments, while T-MCL employs ensembles of multi-headed dynamics models specialized to a subset of environments.
- Model-based meta-learning methods (ReBAL and GrBAL) [183]: model-based meta-RL methods capable of adapting to a recent trajectory segment, by updating a hidden state with a recurrent model (ReBAL), or by updating model parameters with gradient updates (GrBAL).
- Context-aware dynamics model (CaDM) [143]: a dynamics model conditioned on a context latent vector that captures dynamics-specific information from past experiences. For all experiments, we use the combined version of CaDM and PETS.
- Model-free meta-learning method (PEARL) [208]: a context-conditional policy that conducts adaptation by inferring the context using trajectories from the test environment. A comparison with this method evaluates the benefit of model-based RL methods, e.g., sample-efficiency.

Comparative evaluation on control tasks

Figure 3.3 shows the generalization performances of our method and baseline methods on unseen environments (see the supplementary material for training curve plots). Our method significantly outperforms all model-based RL baselines in all environments. In particular, T-MCL achieves the average return of 19280.1 on HalfCheetah environments while that of PETS is 2223.9. This result demonstrates that our method is more effective for dynamics generalization, compared to the independent ensemble of dynamics models. On the other hand, model-based meta-RL methods

Mass	Head 1	Head 2	Head 3
0.25	95.3	4.7	0.0
0.50	0.0	100.0	0.0
1.50	0.0	0.2	99.8
2.50	0.0	0.0	100.0

(a) CartPoleSwingUp

Length	Head 1	Head 2	Head 3
0.50	100.0	0.0	0.0
0.75	0.0	0.0	100.0
1.0	10.1	89.9	0.0
1.25	0.1	99.9	0.0

(b) Pendulum

Mass	Head 1	Head 2	Head 3
0.25	100.0	0.0	0.0
0.50	73.6	26.4	0.0
1.50	2.0	2.1	95.9
2.50	0.7	86.9	12.4

(c) HalfCheetah

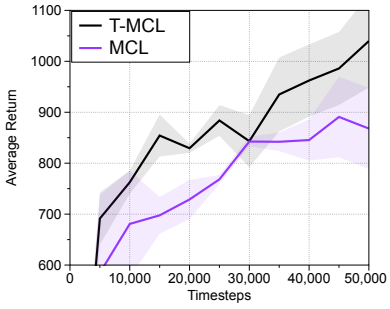
Figure 3.4: Fraction of training trajectories assigned to each prediction head optimized by the proposed trajectory-wise oracle loss (3.2) on (a) CartPoleSwingUp, (b) Pendulum, and (c) HalfCheetah. Number in the (i, j)-th cell of each table denotes the fraction of trajectories with i-th environment parameter, i.e., mass and length, assigned to j-th head of a multi-headed dynamics model.

Mass	Head 1	Head 2	Head 3
0.50	46.3	29.6	24.1
0.75	51.1	28.0	20.9
1.0	50.2	28.3	21.5
1.25	52.9	27.0	20.1
1.50	50.6	27.7	21.8

(a) Multiple choice learning (MCL)

Mass	Head 1	Head 2	Head 3
0.50	83.3	11.0	5.6
0.75	57.9	31.3	10.9
1.0	12.5	63.5	24.0
1.25	5.2	30.5	64.3
1.50	5.3	16.8	77.9

(b) Trajectory-wise MCL (T-MCL)



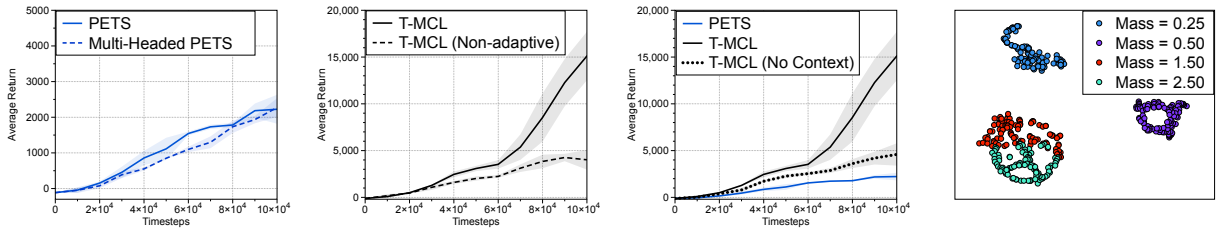
(c) Generalization performance

Figure 3.5: Fraction of training trajectories assigned to each prediction head optimized by (a) MCL and (b) T-MCL on Hopper environments. Number in the (i, j)-th cell of each table denotes the fraction of trajectories with i-th environment parameter, i.e., mass, assigned to j-th head of a multi-headed dynamics model. (c) Generalization performance of dynamics models trained with MCL and T-MCL on unseen Hopper environments.

(ReBAL and GrBAL) do not exhibit significant performance gain over PETS, which shows the difficulty of adapting a dynamics model to unseen environments via meta-learning. We also remark that CaDM does not consistently improve over PETS, due to the difficulty in context learning with a discrete number of training environments. We observed that T-MCL sometimes reaches the performance of PEARL or even outperforms it in terms of both sample-efficiency and asymptotic performance. This result demonstrates the effectiveness of our method for dynamics generalization, especially given that PEARL adapts to test environments by collecting trajectories at evaluation time.

Analysis

Specialization. To investigate the ability of our method to learn specialized prediction heads, we visualize how training trajectories are assigned to each head in Figure 3.4. One can observe that trajectories are distinctively assigned to prediction heads, while trajectories from environments with similar transition dynamics are assigned to the same prediction head. For example, we discover that the transition dynamics of Pendulum with length 1.0 and 1.25 are more similar to each other than Pendulum with other lengths (see the supplementary material for supporting figures), which implies



(a) Multi-head architecture (b) Adaptive planning (c) Context learning (d) t-SNE visualization

Figure 3.6: (a) Generalization performance of PETS and Multi-Headed PETS on unseen HalfCheetah environments. (b) We compare the generalization performance of adaptive planning to non-adaptive planning on unseen HalfCheetah environments. (c) Generalization performance of trained dynamics models on unseen HalfCheetah environments. One can observe that T-MCL still outperforms PETS without context learning, but this results in a significant performance drop. (d) t-SNE visualization of hidden features of context-conditional multi-headed dynamics model on HalfCheetah environments.

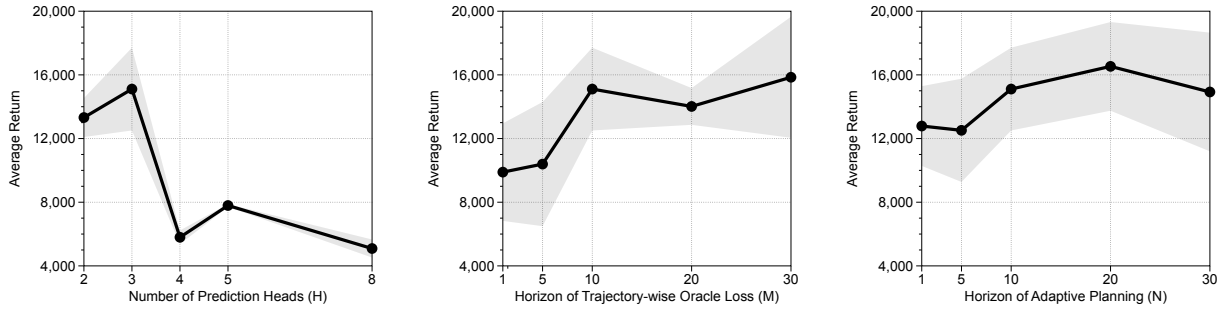
that our method can cluster environments in an unsupervised manner.

Effects of trajectory-wise loss. To further investigate the effectiveness of trajectory-wise oracle loss, we compare our method to MCL, where we consider only a single transition for selecting the model to optimize, i.e., $M = 1$ in equation 3.1. Figure 3.5a and Figure 3.5b show that training trajectories are more distinctively assigned to each head when we use T-MCL, which implies that trajectory-wise loss is indeed important for learning specialized prediction heads. Also, as shown in Figure 3.5c, this leads to superior generalization performance over the dynamics model trained with MCL, showing that learning specialized prediction heads improves the generalization performance.

Effects of multi-headed dynamics model. We also analyze the isolated effect of employing multi-headed architecture on the generalization performance. To this end, we train the multi-headed version of PETS, i.e., ensemble of multi-headed dynamics models without trajectory-wise oracle loss, context learning, and adaptive planning. Figure 3.6a shows that multi-headed PETS does not improve the performance of vanilla PETS on HalfCheetah environments, which demonstrates the importance of training with trajectory-wise oracle loss and adaptively selecting the most accurate prediction head for achieving superior generalization performance of our method.

Effects of adaptive planning. We investigate the importance of selecting the specialized prediction head adaptively. Specifically, we compare the performance of employing the proposed adaptive planning method to the performance of employing non-adaptive planning, i.e., planning with the average predictions of prediction heads. As shown in Figure 3.6b, the gain due to adaptive planning is significant, which confirms that proposed adaptive planning is important.

Effects of context learning. We examine our choice of integrating context learning by comparing the performance of a context-conditional multi-headed dynamics model to the performance of a multi-headed dynamics model. As shown in Figure 3.6c, removing context learning scheme



(a) Prediction heads (H) (b) Horizon of T-MCL (M) (c) Horizon of adaptive planning (N)

Figure 3.7: Performance of dynamics models trained with T-MCL on unseen HalfCheetah environments with varying (a) the number of prediction heads, (b) horizon of trajectory-wise oracle loss, and (c) horizon of adaptive planning.

from the T-MCL results in steep performance degradation, which demonstrates the importance of incorporating contextual information. However, we remark that the T-MCL still outperforms PETS without context learning scheme. Also, we visualize the hidden features of a context-conditional multi-headed dynamics model on HalfCheetah environments using t-SNE [159] in Figure 3.6d. One can observe that features from the environments with different transition dynamics are separated in the embedding space, which implies that our method indeed learns meaningful contextual information.

Effects of hyperparameters. Finally, we investigate how hyperparameters affect the performance of T-MCL. Specifically, we consider three hyperparameters, i.e., $H \in \{2, 3, 4, 5, 8\}$ for the number of prediction heads in (3.2), $M \in \{1, 5, 10, 20, 30\}$ for the horizon of trajectory-wise oracle loss in (3.2), and $N \in \{1, 5, 10, 20, 30\}$ for the horizon of adaptive planning in (3.3). Figure 3.7a shows that $H = 3$ achieves the best performance because three prediction heads are enough to capture the multi-modality of the training environments in our setting. When $H > 3$, the performance decreases because trajectories from similar dynamics are split into multiple heads. Figure 3.7b and Figure 3.7c show that our method is robust to the horizons M, N, and considering more transitions can further improve the performance. We provide results for all environments in the supplementary material.

3.5 Conclusion

In this chapter, we present trajectory-wise multiple choice learning, a new model-based RL algorithm that learns a multi-headed dynamics model for dynamics generalization. Our method consists of three key ingredients: (a) trajectory-wise oracle loss for multi-headed dynamics model, (b) context-conditional multi-headed dynamics model, and (c) adaptive planning. We show that our method can capture the multi-modal nature of environments in an unsupervised manner, and outperform existing model-based RL methods. Overall, we believe our approach would further strengthen the understanding of dynamics generalization and could be useful to other relevant topics such as model-based policy optimization methods [90, 107].

Part II

Generalization

Chapter 4

Bridging Learning and Planning

For future robots to perform general tasks in unstructured environments such as homes or hospitals, they must be able to reason about their domain and plan their actions accordingly. In AI literature, this general problem has been investigated under two main paradigms – automated planning and scheduling [220] (henceforth, AI planning) and reinforcement learning [256] (RL).

Classical work in AI planning has drawn on the remarkable capability of humans to perform long-term reasoning and planning by using abstract representations of the world. For example, humans might think of "cup on table" as a state rather than detailed coordinates or a precise image of such a scene. Interestingly, powerful classical planners exist that can reason very effectively with these kinds of representations, as demonstrated by results in the International Planning Competition [266]. However, such logical representations of the world can be difficult to specify correctly. As an example, consider designing a logical representation for the state of a deformable object such as a rope. Moreover, logical representations that are not grounded *a priori* in real-world observation require a perception module that can identify, for example, exactly when the cup is considered "on the table".

In RL, on the other hand, a task is solved directly through trial and error, guided by a manually provided reward signal. Recent advances in model-free RL (e.g., [177, 147]) have shown remarkable success in learning policies that act directly on high-dimensional observations, such as raw images. Designing a reward function that depends on such observations can be challenging, however, and most recent studies either relied on domains where the reward can be instrumented [177, 147, 215], or required successful demonstrations as guidance [68, 249]. Moreover, since RL is guided by the reward to solve a particular task, it does not automatically generalize to different tasks [259, 118].

In principle, model-based RL can solve the generalization problem by learning a dynamics model and planning with that model. However, applying model-based RL to domains with high-dimensional observations has been challenging [279, 69, 67]. Deep learning approaches to learning dynamics models (e.g., action-conditional video prediction models [194, 4, 67]) tend to get bogged down in pixel-level detail, tend to be computationally expensive, and are far from accurate over longer time scales. Moreover, the representations learned using such approaches are typically unstructured, high-dimensional continuous vectors, which cannot be used in efficient planning algorithms.

In this chapter, we aim to combine the merits of deep learning dynamics models and classical AI planning, and propose a framework for long-term reasoning and planning that is grounded in real-world perception. We present **Causal InfoGAN (CIGAN)**, a method for learning *plannable representations* of dynamical systems with high-dimensional observations such as images. By plannable, we mean representations that are structured in such a way that makes them amenable for efficient search, through AI planning tools. In particular, we focus on discrete and deterministic dynamics models, which can be used with graph search methods, and on continuous models where planning is done by linear interpolation, though our framework can be generalized to other model types.

In our framework, a generative adversarial net (GAN; [83]) is trained to generate sequential observation pairs from the dynamical system. The GAN generator takes as input both unstructured random noise and a structured pair of consecutive states from a low-dimensional, parametrized dynamical system termed the *planning model*. The planning model is meant to capture the features that are *most essential for representing the causal properties* in the data, and are therefore important for planning future outcomes. To learn such a model, we follow the InfoGAN idea [32], and add to the GAN training loss a term that maximizes the mutual information between the observation pairs and the transitions that induced them.

The CIGAN model can be trained using random exploration data from the system. After learning, given an observation of an initial configuration and a goal configuration, it can generate a "walkthrough" sequence of feasible observations that lead from the initial state to the goal. This walkthrough can be later used as a reference signal for a controller to execute the task in the real system. We demonstrate convincing walkthrough generation on synthetic tasks and real image data collected by Nair et al. [185] of a robot randomly poking a rope.

4.1 Preliminaries and Problem Formulation

Let H denote the entropy of a random variable, and I denote the mutual information between two random variables [39].

GAN and InfoGAN: Deep generative models aim to generate samples from the real distribution of observations, P_{data}. In this work we build on the GAN framework [83], which is composed of a generator, $G(z) = o$, mapping a noise input $z \sim P_{\text{noise}}(z)$ to an observation o, and a discriminator, $D(o)$, mapping an observation to the probability that it was sampled from the real data. The GAN training optimizes a game between the generator and discriminator,

$$\min_G \max_D V(G, D) = \min_G \max_D \mathbb{E}_{o \sim P_{\text{data}}} [\log D(o)] + \mathbb{E}_{z \sim P_{\text{noise}}} [\log (1 - D(G(z)))] .$$

One can view the noise vector z in the GAN as a representation of the observation o. In GAN training, however, there is no incentive for this representation to display any structure at all, making it difficult to interpret, or use in a downstream task. The InfoGAN method [32] aims to mitigate this issue. The idea in InfoGAN is to add to the generator input an additional 'state'[1] component

[1] In [32], s is referred to as a *code*. Here we term it as a state, to correspond with our subsequent development of structured GAN input from a dynamical system.

$s \sim P(s)$, and add to the GAN objective a loss that induces maximal mutual information between the generated observation and the state. The InfoGAN objective is given by:

$$\min_G \max_D \quad V(G, D) - \lambda I\left(s; G(z, s)\right), \tag{4.1}$$

where $\lambda > 0$ is a weight parameter, and $V(G, D)$ is the GAN loss above. Intuitively, this objective induces the state to capture the most salient properties of the observation. Optimizing the objective in equation 4.1 directly is difficult without access to the posterior distribution $P(s|o)$, and a variational lower bound was proposed in [32]. Define an auxiliary distribution $Q(s|o)$ to approximate the posterior $P(s|o)$. Then, $I\left(s; G(z, s)\right) \geq \mathbb{E}_{s\sim P(s), o\sim G(z,s)}\left[\log Q(s|o)\right] + H(s)$. Using this bound, the InfoGAN objective equation 4.1 can be optimized using stochastic gradient descent.

Problem Formulation: We consider a fully observable and deterministic dynamical system, $o_{t+1} = f(o_t, u_t)$, where o_t and u_t denote the observation and action at time t, respectively. The function f is assumed to be unknown. We are provided with data $\mathcal{D}$ in the form of N trajectories of observations $\left\{o_1^i, u_1^i \ldots, o_{T_i}^i\right\}_{i\in 1,\ldots,N}$, generated from f, where the actions are generated by an arbitrary exploration policy.[2]

We say that two observations o, o' are *h-reachable* if there exists a sequence of actions that takes the system from o to o' within h steps or less. We consider the problem of generating a *walkthrough* – a sequence of reachable observations along a feasible path between the start and the goal:

Problem 1. ***Walkthrough Planning****: Given $\mathcal{D}$, h, and two observations o_{start}, o_{goal}, generate a sequence of observations $o_{start}, \ldots, o_{goal}$ such that every two consecutive observations in the sequence are h-reachable. If such a sequence does not exist, return $\emptyset$.*

The motivation to solve problem 1 is that it breaks the long horizon planning problem (from o_{start} to o_{goal}) into a sequence of short h-horizon planning problems which can be later solved effectively using other methods such as inverse dynamics or model-free RL [185]. This concept of *temporal abstraction* has been fundamental in AI planning (e.g., [63, 258]). Since we are searching for a sequence of way point observations, the actions are not relevant for our problem, and in the sequel we omit them from the discussion.

4.2 Causal InfoGAN

A natural approach for solving the walkthrough planning problem in Section 4.1 is learning some model of the dynamics f from the data, and searching for a plan within that model. This leads to a trade-off. On the one hand, we want to be expressive, and learn all the transitions possible from every o within a horizon h. When o is a high dimensional image observation, this typically requires mapping the image to an extensive feature space [194, 67]. On the other hand, however, we want to plan efficiently, which generally requires either low dimensional state spaces or well-structured representations. We approach this challenge by proposing *Causal InfoGAN* –

[2] In this work, we do not concern the problem of how to best generate the exploration data.

an expressive generative model with a structured representation that is compatible with planning algorithms. In this section we present the Causal InfoGAN generative model, and in Section 4.3 we explain how to use the model for planning.

Let o and o' denote a pair of sequential observations from the dynamical system f, and let $P_{\text{data}}(o, o')$ denote their probability, as displayed in the data $\mathcal{D}$. We posit that a generative model that can accurately learn $P_{\text{data}}(o, o')$ has to capture the features that are important for representing the *causality* in the data – what next observations o' are reachable from the current observation o. Naturally, such features would be useful later for planning.

We build on the GAN framework [83]. Applied to our setting, a vanilla GAN would be composed of a generator, $o, o' = G(z)$, mapping a noise input $z \sim P_{\text{noise}}(z)$ to an observation pair, and a discriminator, $D(o, o')$, mapping an observation pair to the probability that it was sampled from the real data $\mathcal{D}$ and not from the generator. One can view the noise vector z in such a GAN as a feature vector, containing some representation of the transition to o' from o. The problem, however, is that the structure of this representation is not necessarily easy to decode and use for planning. Therefore, we propose to design a generator with a *structured* input that can be later used for planning. In particular, we propose a GAN generator that is driven by states sampled from a parametrized dynamical system.

Let $\mathcal{M}$ denote a dynamical system with state space S, which we term the set of *abstract-states*, and a parametrized, stochastic transition function $T_{\mathcal{M}}(s'|s)$: $s' \sim T_{\mathcal{M}}(s'|s)$, where $s, s' \in S$ are a pair of consecutive abstract states. We denote by $P_{\mathcal{M}}(s)$ the prior probability of an abstract state s. We emphasize that the abstract state space S can be different from the space of real observations o. For reasons that will become clear later on, we term $\mathcal{M}$ as the *latent planning system*.

We propose to structure the generator as taking in a pair of consecutive abstract states s, s' in addition to the noise vector z. The GAN objective in this case is therefore (cf. Section 4.1):

$$V(G, D) = \mathbb{E}_{o,o' \sim P_{\text{data}}} \left[\log D(o, o')\right] + \mathbb{E}_{z \sim P_{\text{noise}}, s \sim P_{\mathcal{M}}(s), s' \sim T_{\mathcal{M}}(s)} \left[\log \left(1 - D(G(z, s, s'))\right)\right]. \quad (4.2)$$

The idea is that s and s' would represent the abstract features that are important for understanding the causality in the data, while z would model variations that are less informative, such as pixel level details. To learn such representations, we follow InfoGAN [32], and add to the GAN objective a term that maximizes mutual information between the generated pair of observations and the abstract states.

We propose the Causal InfoGAN objective:

$$\min_{\mathcal{M},G} \max_{D} \quad V(G, D) - \lambda I\left(s, s'; o, o'\right), \quad \text{s.t. } o, o' \sim G(z, s, s'); \quad s \sim P_{\mathcal{M}}; \quad s' \sim T_{\mathcal{M}}(s), \quad (4.3)$$

where $\lambda > 0$ is a weight parameter, and $V(G, D)$ is given in equation 4.2. Intuitively, this objective induces the abstract model to capture the most salient possible changes that can be effected on the observation.

Optimizing the objective in equation 4.3 directly is difficult, since we do not have access to the posterior distribution, $P(s, s'|o, o')$, when using an expressive generator function. Following InfoGAN [32], we optimize a variational lower bound of equation 4.3. Define an auxiliary distribution

$Q(s, s'|o, o')$ to approximate the posterior $P(s, s'|o, o')$. We have, following a similar derivation to [32]:

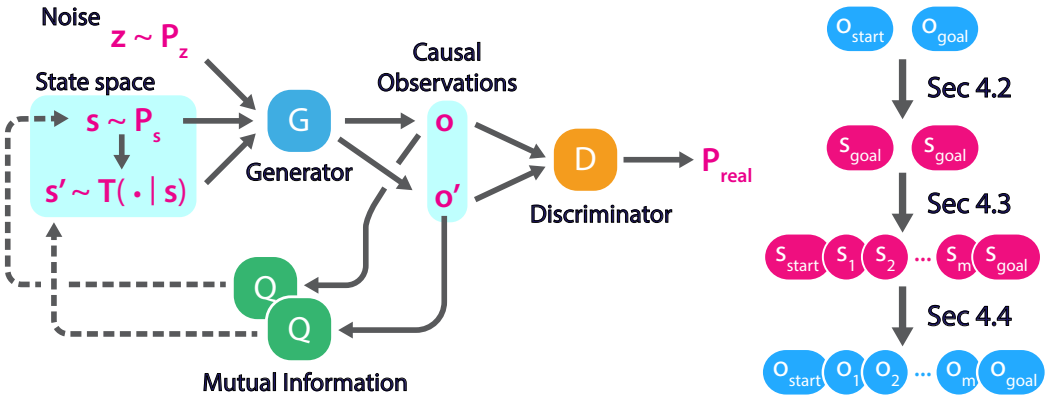
$$I\left((s, s'); G(z, s, s')\right) \geq \mathbb{E}_{\substack{s\sim P_{\mathcal{M}}, s'\sim T_{\mathcal{M}}(s), \\ o,o'\sim G(z,s,s')}} \left[\log Q(s, s'|o, o')\right] + H(s, s') \doteq I_{VLB}(G, Q). \tag{4.4}$$

To encourage the same mapping between s, o and s', o', we propose the disentangled posterior approximation, $Q(s, s'|o, o') = Q(s|o)Q(s'|o')$ (see Appendix B.)

We plug the lower bound equation 4.4 in Eq. equation 4.3 to obtain the following loss function:

$$\min_{G,Q,\mathcal{M}} \max_{D} \quad V(G, D) - \lambda I_{\text{VLB}}(G, Q), \tag{4.5}$$

where $\lambda > 0$ is a constant. The loss in equation 4.5 can be optimized effectively using stochastic gradient descent, and we provide a detailed algorithm in Appendix C.



(a) Causal InfoGAN model (b) Planning paradigm

Figure 4.1: The Causal InfoGAN framework. **(a) Generative model (cf. Section 4.2)**. First, an abstract state s is sampled from a prior $P_{\mathcal{M}}(s)$. Given s, the next state s' is sampled using the transition model $T_{\mathcal{M}}(s'|s)$. The states s, s' are fed, together with a random noise sample z, into the generator which outputs o, o'. The discriminator D maps an observation pair to the probability of the pair being real. Finally, the approximate posterior Q maps from each observation to the distribution of the state it associates with. The causal InfoGAN loss function in Equation equation 4.5 encourages Q to predict each state accurately from each observation. **(b) Planning paradigm (cf. Section 4.3)**. Given start and goal observations, we first map them to abstract states, and then we apply planning algorithms using the model $\mathcal{M}$ to search for a path from s_{start} to s_{goal}. Finally, from the plan in abstract states, we generate back a sequence of observations.

4.3 Planning with Causal InfoGAN models

In this section, we discuss how to use the Causal InfoGAN model for planning goal directed trajectories. We first present our general methodology, and then propose several model configurations for which equation 4.5 can be optimized efficiently, and the latent planning system is compatible with efficient planning algorithms. We then describe how to combine these ideas for solving the walkthrough planning problem in various domains.

General Planning Paradigm

Our general paradigm for goal directed planning is described in Figure 4.1b. We start by training a Causal InfoGAN model from the data, as described in the previous section. Then, we perform the following 3 steps, which are detailed in the rest of this section:

1. Given a pair of observations o_{start}, o_{goal}, we first encode them into a pair of corresponding states s_{start}, s_{goal}. This is described in Section 4.3.
2. Using the transition probabilities in the planning model $\mathcal{M}$, we plan a feasible state trajectory from s_{start} to s_{goal}: $s_{start}, s_1, \ldots, s_m, s_{goal}$. This is described in Section 4.3.
3. Finally, we decode the state trajectory into a corresponding trajectory of observations o_{start}, $o_1, \ldots, o_m$, o_{goal}. This is described in Section 4.3.

The specific method for each step in the planning paradigm can depend on the problem at hand. For example, some systems are naturally described by discrete abstract states, while others are better described by continuous states. In the following, we describe several models and methods that worked well for us, under the general planning paradigm described above. This list is by no means exhaustive. On the contrary, we believe that the Causal InfoGAN framework provides a basis for further investigation of deep generative models that are compatible with planning.

Encoding an Observation to a State

For mapping an observation to a state, we can simply use the disentangled posterior $Q(s|o)$. We found this approach to work well in low-dimensional observation spaces. However, for high-dimensional image observations we found that the learned $Q(s|o)$ was accurate in classifying generated observations (by the generator), but inaccurate for classifying real observations. This is explained by the fact that in Causal InfoGAN, Q is only trained on generated observations, and can overfit to generated images. For images, we therefore opted for a different approach. Following [276], we performed a search over the latent space to find the best latent state mapping $s^*(o)$: $s^*(o) = \arg\min_s \min_{s',z} \|o - G(s, s', z)\|^2$. Another approach, which could scale to complex image observations, is to add to the GAN training an explicit encoding network [50, 290]. In our experiments, the simple search approach worked well and we did not require additional modifications to the GAN training.

Latent Planning Systems

We now present several latent planning systems that are compatible with efficient planning algorithms. Table 4.1 summarizes the different models. In all cases, optimizing the model parameters with respect to the expectation in the loss equation 4.4 is done using the reparametrization trick (following [124] for continuous states, and [106] for discrete states).

Discrete Abstract States – One-Hot Representation. We start from a simple abstract state representation, in which each $s \in S$ is represented as a $N-$dimensional one-hot vector. We

Type	Values s	Prior $P_{\mathcal{M}}(s)$	Transition $T_{\mathcal{M}}(s'\|s)$	Planning algorithms
Discrete – one-hot	$[N]$	$\mathcal{U}\{1,\dots,N\}$	$s' \sim Softmax(s^\top \theta)$	Dijkstra
Discrete – binary	$\{0,1\}^N$	$\mathcal{U}\{0,1\}^N$	See eq. 4.6	Dijkstra
Continuous	$\mathbb{R}^N$	$\mathcal{U}(-1,1)^N$	$s' \sim \mathcal{N}(s, \Sigma_\theta(s))$	Linear interpolation

Table 4.1: Various latent planning systems. In all cases, N is the state dimension. The parameters θ of the transition $T_{\mathcal{M}}$ depending on the state type. In the one-hot case, θ is a matrix in $\mathbb{R}^{N\times N}$. In the binary case, θ denotes parameters in a stochastic neural network; see Eq. equation 4.6. In the continuous case θ represents the parameters of a neural network that controls the variance of the transition.

denote by $\theta \in \mathbb{R}^{N\times N}$ the model parameters, and compute transition probabilities as: $T_{\mathcal{M}}(s'|s) = Softmax(s^\top\theta)$.

Discrete Abstract States – Binary Representation. We present a more expressive abstract state representation using binary states. Binary state representations are common in AI planning, where each binary element is known as a *predicate*, and corresponds to a particular object property being true or false [220]. Using Causal InfoGAN, we learn the predicates *directly from data*.

We propose a parametric transition model that is suitable for binary representations. Let $s \in \{0,1\}^N$ be an $N-$dimensional binary vector, drawn from $P_{\mathcal{M}}(s)$. We generate the next state s' by first drawing a random *action vector* $a \in \{0,1\}^M$ with some probability $P_{\mathcal{M}}(a)$ [3]. Let l_i denote a feed-forward neural network with sigmoid output parametrized by θ that maps the state s and action a to the Bernoulli's parameter of $s'_i|s,a$. Thus, the probability of the next state s' is finally given by:

$$T_{\mathcal{M}}(s'=v|s) = \mathbb{E}_a\left[\prod_i T_{\mathcal{M}}(s'_i = v_i|s,a)\right] = \mathbb{E}_a\left[\prod_i l_i(s,a)^{v_i}(1-l_i(s,a))^{(1-v_i)}\right]. \tag{4.6}$$

We emphasize that there is not necessarily any correspondence between the action vector a and the real actions that generated the observation pairs in the data. The action a is simply a means to induce stochasticity to the state transition network.

For planning with discrete models, we interpret the stochastic transition model $T_{\mathcal{M}}$ as providing the possible state transitions, i.e., for every s' such that $T_{\mathcal{M}}(s'|s) > \epsilon$ there exists a possible transition from s to s'. For planning, we require abstract state representations that are compatible with efficient AI planning algorithms. The one-hot and binary representations above can be directly plugged in to graph-planning algorithms such as Dijkstra's shortest-path algorithm [220].

Continuous Abstract States. For some domains, such as the rope manipulation in our experiments, a continuous abstract state is more suitable. We consider a model where an $s \in S$ is an $N-$dimensional continuous vector. Planning in high-dimensional continuous domains, however, is hard in general.

Here, we propose a simple and effective solution: we will learn a latent planning system such that *linear interpolation between states makes for feasible plans*. To bring about such a model, we consider transition probabilities $T_{\mathcal{M}}(s'|s)$ given as Gaussian perturbations of the state: $s' = s + \delta$, where $\delta \sim \mathcal{N}(0, \Sigma_\theta(s))$ and Σ_θ is a diagonal covariance matrix, represented by a MLP with

[3] See Appendix B.2 Binary States for experimental choices.

parameters θ. The key idea here is that, if only small local transitions are possible in the system, then a linear interpolation between two states s_{start}, s_{goal} has a high probability, and therefore represents a feasible trajectory in the observation space. To encourage such small transitions, we add an L2 norm of the convariance matrix to the loss function equation 4.5: $L_{cont}(\mathcal{M}) = \mathbb{E}_{s\sim P_{\mathcal{M}}} \left\|\Sigma_\theta(s)\right\|_2$. The prior probability $P_{\mathcal{M}}$ for each element of s is uniform in $[-1, 1]$.

Decoding a State Trajectory to an Observation Walkthrough Trajectory

We now discuss how to generate a feasible sequence of observations from the planned state trajectory. Here, as before, we separate the discussion for systems with low-dimensional observations and systems with image observations, as we found that different practices work best for each.

For low-dimensional observations, we structure the GAN generator G to have an observation-conditional form: $o = G_1(z, s, s'), \quad o' = G_2(z, o, s, s')$. Using this generator form, we can sequentially generate observations from a state sequence $s_1, \ldots, s_T$. We first use G_1 to generate o_1 from s_1, s_2, and then, for each $2 \leq t < t$, use G_2 to generate o_{t+1} from s_t, s_{t+1}, and o_t.

For high-dimensional image observations, the sequential generator does not work well, since small errors in the image generation tend to get accumulated when fed back into the generator. We therefore follow a different approach. To generate the i'th observation in the trajectory o_i, we use the generator with the input s_i, s_{i+1}, and a noise z that is fixed throughout the whole trajectory. The generator actually outputs a pair of sequential images, but we discard the second image in the pair.

To further improve the planning result we generate K random trajectories with different random noise z, and select the best trajectory by using a discriminator D to provide a confidence score for each trajectory. In the low-dimensional case, we use the GAN discriminator. In the high-dimensional case, however, we find that the discriminator tends to overfit to the generator. Therefore, we trained an auxiliary discriminator for novelty detection, as described in the Experiment Section 4.5.

4.4 Related Work

Combining deep generative models with structured dynamical systems has been explored in the context of variational autoencoders (VAEs), where the latent space was continuous [35, 110]. Watter et al. [279] used such models for planning, by learning latent linear dynamics, and using a linear quadratic Gaussian control algorithm. Disentangled video prediction [46] separates object content and position, but has not been used for planning. Very recently, Corneil et al. [38] suggested Variational State Tabulation (VaST) – a VAE-based approach for learning latent dynamics over binary state representations, and planning in the latent space using prioritized sweeping to speed up RL. Causal InfoGAN shares several similarities with VaST, such as using Gumbel-Softmax to backprop through transitions of discrete binary states, and leveraging the structure of the binary states for planning. However, VaST is formulated to require the agent actions, and is thus limited to single time step predictions. More generally, our work is developed under the GAN formulation, which, to date, has several benefits over VAEs such as superior quality of image generation [119]. Causal InfoGAN can also be used with continuous abstract states.

The semiparametric topological memory (SPTM) [228] is another recent approach for solving problems such as Problem 1, by planning in a graph where every observation in the data is a node, and connectivity is decided using a learned similarity metric between pairs of observations. SPTM has shown impressive results on image-based navigation. However, Causal InfoGAN's *parametric approach* of learning a compact, model for planning has the potential to scale up to more complex problems, in which the increasing amount of data required would make the nonparametric SPTM approach difficult to apply.

Learning state aggregation and state representation has a long history in RL. Methods such as in [165, 243] exploit the value function for measuring state similarity, and are therefore limited to the task defined by the reward. Methods for general state aggregation have also been proposed, based on spectral clustering [163, 248, 160, 155], and variants of K-means [21]. All these approaches rely in some form on the Euclidean distance as a metric between observation features. As we show in our experiments, the Euclidean distance can be unsuitable even on low-dimensional continuous domains. Recent work in deep RL explored learning goal-conditioned value functions and policies [9, 205], and policies with an explicit planning computation [259, 193, 249]. These approaches require a reward signal for learning (or supervision from an expert [249]). In our work, we do not require a reward signal, and learn a general model of the dynamical system, which is used for goal-directed planning.

Our work is also related to learning models of intuitive physics. Previous work explored feedforward neural networks for predicting outcomes of physical experiments [146], neural networks for modelling relations between objects [281, 226], and prediction based on physics simulators [23, 283]. To the best of our knowledge, these approaches cannot be used for planning. However, related ideas would likely be required for scaling our method to more complex domains, such as manipulating several objects.

In the planning literature, most studies relied on manually designed state representations. In a recent work, Konidaris et al. [129] automatically extracted state representations from raw observations, but relied on a prespecified set of skills for the task. In our work, we automatically extract state representations by learning salient features that describe the causal structure of the data.

4.5 Experiments

In our experiments, we aim to (1) visualize the abstract states and planning in CIGAN; (2) compare CIGAN with recent state-aggregation methods in the literature; (3) show that CIGAN can produce realistic visual plans in a complex dynamical system; and (4) show that CIGAN significantly outperforms baseline methods. We begin our investigation with a set of toy tasks, specifically designed to demonstrate the benefits of CIGAN, where we can also perform an extensive quantitative evaluation. We later present experiments on a real dataset of robotic rope manipulation. Code is available online at `github.com/thanard/causal-infogan`.

Illustrative Experiments

In this section we evaluate CIGAN on a set of 2D navigation problems. These problems abstract away the challenges of learning visual features, allowing an informative comparison on the task of learning causal structure in data, and using it for planning. For details of the training data see Appendix C.4.

Our toy domains involve a particle moving in a 2-dimensional continuous domain with impenetrable obstacles, as depicted in Figure 4.2, and Figure 5. in Appendix C.1. The observations are the (x, y) coordinates of the particle in the plane, and, in the door-key domain, also a binary indicator for holding the key. We generate data trajectories by simulating a random motion of the particle, started from random initial points. We consider the following various geometrical arrangements of the domain, chosen to demonstrate the properties of our method.

1. **Tunnels:** the domain is partitioned into two unconnected rooms (top/bottom), where in each room there is an obstacle, positioned such that transitioning between the left/right quadrants is through a narrow tunnel.

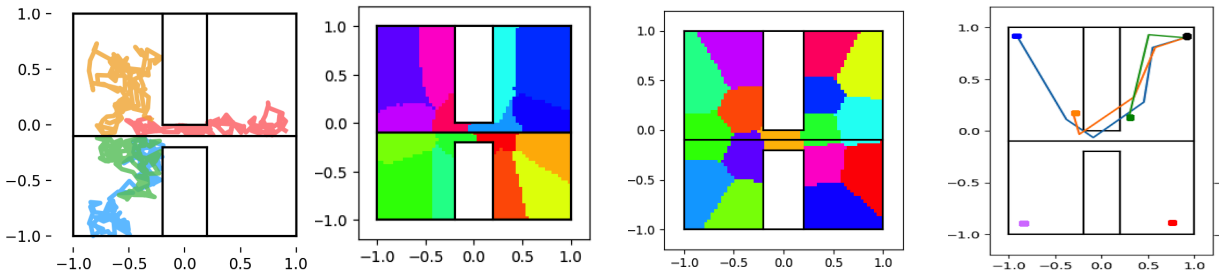
2. **Door-key:** two rooms are connected by a door. The door can be traversed only if the agent holds the key, which is obtained by moving to the red-marked area in the top right corner of the upper room. Holding the key is represented as a binary 0/1 element in the observation.

3. **Rescaled door-key:** Same as door key domain, but the key observation is rescaled to be a small ϵ when the agent is holding the key, and 0 otherwise.

Our domains are designed to distinguish when standard state aggregation methods, which rely on the Euclidean metric, can work well. In the tunnel domain, the Euclidean metric is not informative about the dynamics in the task – two points in different rooms inside the tunnel can be very close in Euclidean distance, but not connected, while points in the same room can be more distant but connected. In the door-key domain, the Euclidean distance is informative if observations with key and without key are very distant in Euclidean space, as in the 0/1 representation (compared to the domain size which is in $[-1, 1]$). In the rescaled door-key, we make the Euclidean distance less informative by changing the key observation to be 0/ϵ.

We compare CIGAN with several recent methods for aggregating observation features into states for planning. Note that in these simple 2D domains, feature extraction is not necessary as the observations are already low dimensional vectors. The simplest baseline is K-means, which relies on the Euclidean distance between observations. In [21], a variant of K-means for temporal data was proposed, using a window of consecutive observations to measure a smoothed Euclidean distance to a cluster centroids. We refer to this method as temporal K-means. In [163], and more recently [248] and [160], spectral clustering (SC) was used to cluster observations. For continuous observations, SC requires a distance function to build a connectivity graph, and previous studies [163, 248, 160] relied on the Euclidean distance, using either nearest neighbor to connect nodes, or exponentiated-distance weighted edges.

In Figure 4.2, we show the CIGAN classification of observations to abstract states, $Q(s|o)$, and compare with the K-means baseline; the other baselines gave qualitatively similar results. Note



(a) (b) (c) (d)

Figure 4.2: 2D particle results on tunnel domain. (a) The domain - top/bottom rooms are not connected. Left/right quadrants are connected through a narrow tunnel. An example of several random walk trajectories are shown. (b) Clustering found by CIGAN. (c) Clustering found by K-means. (d) Example walkthrough trajectories generated by CIGAN, from a point at the top right to five other locations on the map, marked by colored circles. For trajectories that were not found only the target is shown. Note that CIGAN learned clusters that correspond to the possible dynamics of the particle in the task, and was therefore able to generate reasonable planning trajectories.

that CIGAN learned a clustering that is related to the dynamical properties of the domain, while the baselines, which rely on a Euclidean distance, learned clusters that are not informative about the real possible transitions. As a result, CIGAN clearly separates abstract states within each room, while the K-means baseline clusters observations across the wall. This demonstrates the potential of CIGAN to learn meaningful state abstractions without requiring a distance function in observation space. Similarly, the results for door-key domains are shown in Appendix C.1.

To evaluate planning performance, we hand-coded an oracle function that evaluates whether an observation trajectory is feasible or not (e.g., does not cross obstacles, correctly reports $\emptyset$ when a trajectory does not exist). For CIGAN, we ran the planning algorithm described in Section 4.3. For baselines, we calculated cluster transitions from the data, and generated planning trajectories in observation space by using the cluster centroids. We chose algorithm parameters and stopping criteria by measuring the average feasibility score on a validation set of start/goal observations, and report the average feasibility on a held out test set of start/goal observations. Our results in Table 4.2 show that by learning more informative clusters, CIGAN resulted in significantly better planning.

	Tunnels	Door-key	Rescaled door-key
CIGAN	**98%**	98%	**97%**
K-means	12.25%	**100%**	0.0%
Temporal K-means	7.0%	**100%**	0.0%
Spectral clustering	8.75%	60%	20.0%

Table 4.2: Planning results for 2D tasks. Table shows average feasibility of plans (higher is better) generated by the different algorithms. Note that CIGAN significantly outperforms baselines in domains where the Euclidean distance is not informative for planning.

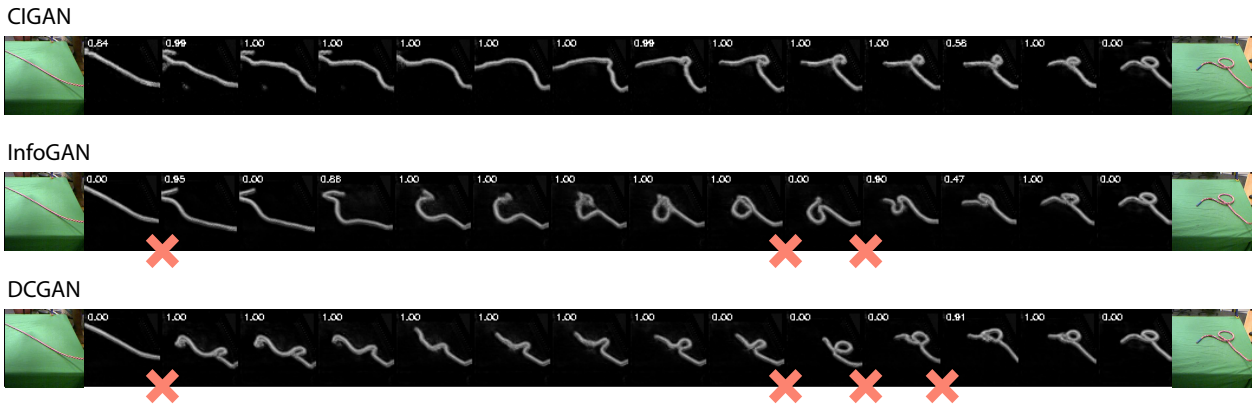


Figure 4.3: Rope walkthroughs generated from CIGAN, InfoGAN, and DCGAN. Red crosses show unfeasible one-step transitions with respect to the data. See more plans in Appendix A.2.

Rope Manipulation

In this section we demonstrate CIGAN on the task of generating realistic walkthroughs of robotic rope manipulation. Then, we show that CIGAN generates significantly better trajectories than those generated by the state-of-the-art generative model baselines both visually and quantitatively.

The rope manipulation dataset [185] contains sequential images of rope manipulated in a self-supervised manner, by a robot randomly choosing a point on the rope and perturbing it slightly. Using these data, the task is to manipulate the rope in a goal-oriented fashion, from one configuration to another, where a goal is represented as an image of the desired rope configuration. In the original study, Nair et al. [185] used the data to learn an inverse dynamics model for manipulating the rope between two images of similar rope configurations. Then, to solve long-horizon planning, Nair et al. required a human to provide the walkthrough sequence of rope poses, and used the learned controller to execute the short-horizon transitions within the plan. In our experiment, we show that CIGAN can be used to generate walkthrough plans *directly from data* for long-horizon tasks, without requiring additional human guidance. We train a CIGAN model on the rope manipulation data of [185]. We pre-processed the data by removing the background, and applying a grayscale transformation. We chose the continuous abstract state representation with the linear interpolation planner as described in Section 4.3. Note that, as described in Section 4.3, the encoding, planning, and decoding methods in this case are not specific to CIGAN, and can be used with a GAN or InfoGAN generative model, allowing a fair comparison with alternative representation learning methods. In Figure 4.3, we generate walkthroughs using CIGAN, InfoGAN, and DCGAN Noticeably, Causal InfoGAN resulted in a smooth latent space where linear interpolation indeed corresponds to plausible trajectories.

To numerically evaluate planning performance we propose a visual fidelity score, inspired by the Inception score for evaluating GANs [224], we train a binary classifier to classify whether two images are sequential in the data or not[4]. For an image pair, the classifier output therefore

[4]The positive data are the pairs of rope images that are 1 step apart and the negative data are randomly chosen pairs

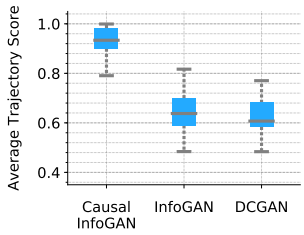


Figure 4.4: Evaluation of walkthrough planning in rope domain. We trained a classifier to predict whether two observations are sequential or not (1=sequential, 0=not sequential), and compare the average classification score for different generative models. Note that CIGAN significantly outperforms the baselines, in alignment with the qualitative results of Figure 4.3.

provides a score between 0 and 1 for the feasibility of the transition. We then compute the *trajectory score* – the average classifier score of image pairs in the trajectory. Note that this classifier is trained independent of the generative models, making for an impartial metric. For each start and goal, we pick the best trajectory score out of 400 samples of the noise variable z.[5] As shown in Figure 4.4, Causal InfoGAN achieved a significantly higher trajectory score averaged over 57 task configurations.

4.6 Conclusion

We presented Causal InfoGAN, a framework for learning deep generative models of sequential data with a structured latent space. By choosing the latent space to be compatible with efficient planning algorithms, we developed a framework capable of generating goal-directed trajectories from high-dimensional dynamical systems.

Our results for generating realistic manipulation plans of rope suggest promising applications in robotics, where designing models and controllers for manipulating deformable objects is challenging.

The binary latent models we explored provide a connection between deep representation learning and classical AI planning, where Causal InfoGAN can be seen as a method for learning object predicates directly from data. In future work we intend to investigate this direction further, and incorporate object-oriented models, which are a fundamental component in classical AI.

In the next chapter, we will further investigate the following questions: (1) how do we retrieve actions to execute the plans?, (2) how do we handle environment changes?, and (3) can this be apply to real robot manipulation problem in which other existing approaches cannot solve?

that are from different runs which are highly likely to be farther than 1 step apart.

[5]This selection process is applied the same way to the DCGAN and InfoGAN baselines.

Chapter 5

Visual Planning and Acting

Many objects that we manipulate every day are deformable or non-rigid. Thus, for future robots to enter environments such as homes and hospitals, non-rigid object manipulation will be essential. Current industrial applications such as wire threading, bin packing, and cloth folding also require such an ability. However, robotics capabilities for the general manipulation of deformable objects are still in their infancy.

The main difficulty in planning the manipulation of deformable objects is that, in contrast with rigid objects, there is no obvious mapping from an observation of the object to a compact representation in which planning can be performed. Thus, traditional task and motion planning approaches, which require manual design of the states and transitions in the problem, are difficult to apply [168, 250]. For example, rope manipulation involves many design aspects: should we represent the shape of rope as a finite set of small segments or as one continuous function? Should length, softness, friction, thickness, etc. be included in the model? How do we infer the system state from the robot's perception system? Different choices can be suitable for different domains, requiring substantial engineering effort.

In recent years, several studies have proposed a data-driven, self-supervised paradigm for robotic manipulation [202, 3, 185, 64, 56]. In this approach, the robot 'plays' with the object using some random manipulation policy (e.g., randomly grasping or poking an object), and collects perceptual data about the interactions with the object. Later, machine learning is used to train a policy that performs the task directly from the perceptual inputs. By relying directly on perceptual data, these approaches overcome the modelling challenges of classical planning approaches, and scale to handle high-dimensional perceptual inputs such as raw images.

In particular, Nair et al. [185] learned an inverse dynamics model for rope manipulation directly from raw image data collected by randomly poking the rope. This controller was used to manipulate the rope into a given shape, whereby a human would first provide a sequence of images – a *visual plan* – that prescribes the desired trajectory of the rope, and then the learned inverse model would compute actions that track the plan (a.k.a. visual servoing). The human demonstration in Nair et al. [185] was essential – performing high-level planning cannot be captured by a reactive inverse model. Indeed, humans' capability of planning long horizon and complex manipulations of general objects has not yet been matched by current AI technology.

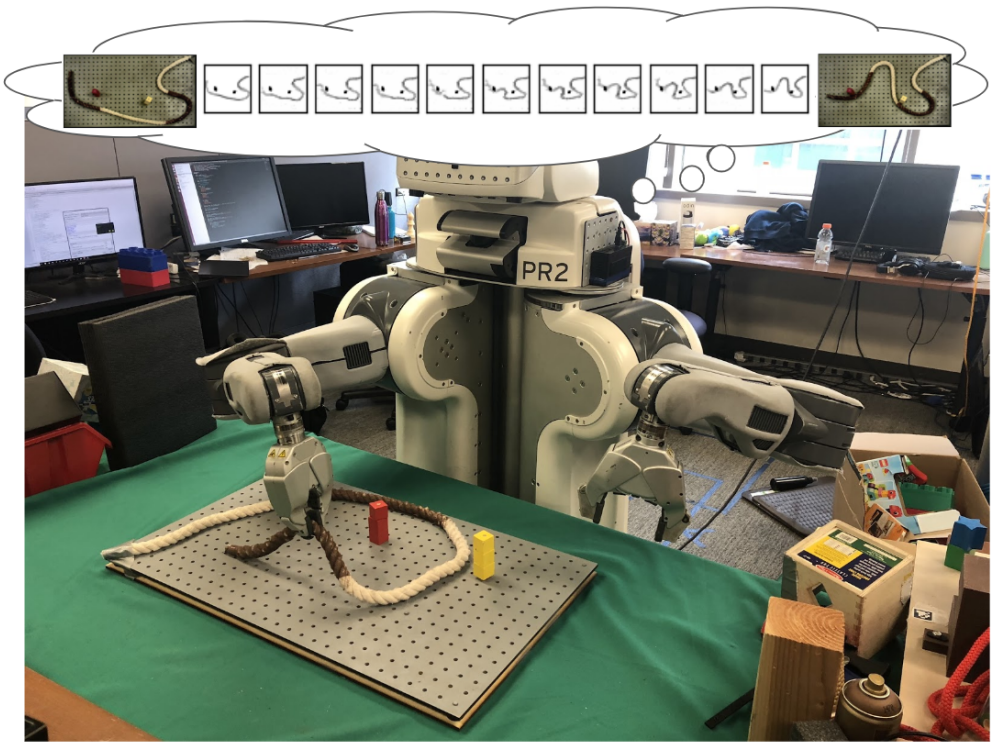


Figure 5.1: Visual planning and acting for rope manipulation. The PR2 robot first collects data through self-supervised random rope manipulation, and learns from this a generative model for possible visual transformations of the rope. Then, given a goal observation for the rope, we *plan* a visual trajectory of a possible manipulation sequence that reaches the goal (shown on top). Finally, visual servoing is used to execute the imagined plan.

In this chapter, we take a step towards closing the gap in complex object manipulation and ask – can we learn from self-supervised data to automatically generate the visual plan and follow it? We term this approach *visual planning and acting (VPA)*, as depicted in Figure 5.1. Concretely, given the current image of the system and some desired goal observation, we would like to generate a sequence of images that manipulate the object to the desired configuration, *without any human guidance*, and then track this imagined plan in practice using a learned inverse model. Such a method would not require the manual guidance of previous approaches, and would also be *safe*, as the imagined plan is visually interpretable, and can be inspected before being executed by the robot.

However, learning visual planning from raw image data has so far been limited to very simple tasks, such as reaching or pushing rigid objects [64, 56]. The fundamental difficulty is that learning an accurate representation of the data requires mapping the image to an extensive feature space, while efficient planning generally requires either low dimensional state spaces or well-structured representations. Current approaches [64, 56] solve this tradeoff by employing very simple planning methods such as random shooting, which do not scale to more complex planning problems.

In this chapter, we propose to learn features that *are compatible* with a strong planning algorithm. At the basis of our approach is the recent Causal InfoGAN (CIGAN) model of Kurutach et al. [133]. In CIGAN, a deep generative model is trained to predict the possible next states of the object, with

a constraint that linear trajectories in the latent state of the model produce feasible observation sequences. Kurutach et al. [133] used a CIGAN model for planning goal-directed trajectories simply by linearly interpolating in the latent space, and then mapping the latent trajectory to observations for generating the visual plan. Building on CIGAN, we propose a method for VPA, where sensory data obtained from self-supervised interaction is used to learn both a CIGAN model for visual planning and an inverse model for tracking a visual plan, as shown in Figure 5.1. After learning, given a goal observation for the system, we first use CIGAN to imagine a sequence of images that transition the system from its current configuration towards the goal. Then, we use the imagined trajectory as a reference for tracking using the inverse model.

We investigate several aspects of the VPA approach for real-world tasks. Our contributions include:

- An extension of the CIGAN model to include contextual input, and imagine plans based on this context (a context can specify, e.g., obstacles in the domain), thereby addressing generalization of VPA to changes in the environment.
- Improvement of the planning algorithm in latent state from interpolation, as suggested in [133], to A* for planning in domains that include obstacles.
- A simulation study showing that separating the control task into visual planning and visual tracking is more sample efficient than model free reinforcement learning methods that learn actions directly from images.
- Application of VPA to real robot rope manipulation tasks, illustrating non-trivial planning and control with deformable objects and demonstrating the interpretability of our approach.

5.1 Related Work

Deformable soft object manipulations have been attempted via classical methods such as motion planning and manipulation planning [168, 74, 121, 109, 222]. These approaches require manual engineering for object models. Previous work has modeled deformable soft objects by hand-engineering representations [132, 271, 179, 164], parametrizing the object shape [170], and using finite element models [102], [24].

Alternatively, there has been recent interest in applying learning-based approaches to robotic manipulation directly from raw image perception. Recent work in model-free reinforcement learning (RL) [176, 128, 152] learns, through trial and error, a policy mapping observations to actions that maximizes reward using deep neural networks. However, specifying reward functions for high dimensional observations such as images can be difficult [10], and the sample efficiency of model free RL can be prohibitive in practice. Because the policy is trained to optimize a predefined reward function, it does not directly generalize to new initial and goal configurations, and requires further interactions with the system. In addition, model-free RL produces black-box policies which are hard to interpret, in contrast with more traditional planning approaches, and our visual planning method in particular, which can predict the trajectory of the robot in advance.

Learning from demonstrations (LfD) guides robots to perform complicated tasks without having to plan from scratch. Schulman et. al. [234] and Mayer et. al. [166] learn a policy that imitates non-rigid object manipulation such as surgical suturing from expert state and action trajectories. One caveat of LfD is that it suffers when generalizing to desired trajectories that deviate from expert demonstrations. Nair et. al. [185] and Kuniyoshi [132] only collect random interactions with the system at training time, and use the data to learn an inverse model. This inverse model is general enough to follow new expert trajectories for new tasks. In our work, we do not require expert demonstrations for new tasks, and show that visual plans can be generated directly from self-supervised data.

Other approaches that learn plannable features for control include Embed to Control (E2C) [278] and related methods based on variational autoencoders [37, 20, 12]. Paxton et. al. [199] learn transitions and an action value function in the latent space, and use that to produce visual plans on simulated domains. To our knowledge, we present the first application of plannable features for real robot experiments.

5.2 Preliminaries and Problem Formulation

In this section we present our problem formulation, and summarize preliminary material.

Problem Formulation

We consider a robot that interacts with the world in a self-supervised manner, and collects sensory data about its interaction. In this work, we do not consider how to collect the data, and assume that the data collection policy visits the 'interesting' configurations of the system. Denote by $\mathcal{D}$ our data, in the form of N trajectories of action-observation pairs, $\{o_1^i, u_1^i, ..., u_{T_i-1}^i, o_{T_i}^i\}_{i \in N}$, where u_j^i is the action that the robot took after observing o_j^i, and led to observation o_{j+1}^i. We assume a deterministic and fully observable system.

After we have collected the data, our goal is to solve a goal-directed planning problem: given the current observation of the system o_{start} and an observation of a desired goal configuration o_{goal}, we want to compute an action selection policy that transitions the system from start to goal.

To solve the problem above, in this work we focus on an approach we term Visual Planning and Acting (VPA). The idea is to decompose the solution into two steps: (1) Visual planning – learning from the data how to *imagine* a goal-directed trajectory of observations that transition the system from start to goal, and (2) Acting – using an inverse model learned from the data on how to take actions that make the system follow the imagined plan.

Visual Planning with Causal InfoGAN

In previous chapter, we describe a method for visual planning based on the CIGAN generative model. Before summarizing CIGAN, we first review two ideas that it builds on – GAN and InfoGAN.

GAN and InfoGAN

GANs [81] are deep generative models that learn to generate samples similar to the data distribution P_{data} by feeding in a random vector $z \sim \mathcal{N}(0, I)$ into a deep neural network generator G. A discriminator neural network D tries to tell apart generated samples from real samples, and the GAN training objective is given by the minimax game: $\min_G \max_D V(G, D) = \min_G \max_D \mathbb{E}_{o\sim P_{data}}[\log D(o)] + \mathbb{E}_z[\log(1 - D(G(z)))]$. The vector z can be interpreted as a latent representation for the generated observation $o = G(z)$. InfoGAN [31] is a method for adding structure to the latent representation. In InfoGAN, the representation is separated into a 'noise' component z and a structured component s. The loss function is modified to maximize the mutual information between s and the generated observation $o = G(z, s)$, which intuitively induces s to capture salient properties of the observation. Let x, y be some random variables. Denoting $H(x) = \mathbb{E}_x[-\log(P(x))]$ as the entropy of x, the mutual information between x and y is defined as $I(x; y) = H(x) - H(x|y) = H(y) - H(y|x)$. The InfoGAN loss function is:

$$\min_G \max_D V(G, D) - \lambda I(s; G(z, s)). \tag{5.1}$$

To optimize this loss in practice, a variational lower bound was proposed in [31]. Let $Q(s|o)$ denote an auxiliary distribution that approximates the posterior $P(s|o)$. Then the lower bound $I(s; G(z, s)) \geq \mathbb{E}_{s\sim P(s),o\sim G(z,s)}[\log Q(s|o)] + H(s)$ can be plugged in equation 5.1 and optimized using the reparametrization trick [31]. Intuitively, the Q function can be understood as a classifier that encodes an observation into its latent representation.

Causal InfoGAN and Plan Generation

In previous chapter, we discuss CIGAN [133] – the extension of InfoGAN for observations from a dynamical system. Consider data that contains trajectories of observations, similar to $\mathcal{D}$ described above in Section 5.2. The CIGAN model learns to generate a pair of *sequential* observations (o, o') that are similar to sequential observations in the data, thereby learning a notion of *causality* in the data. The CIGAN generator input is a pair of latent representations s, s', and a noise vector z: $o, o' = G(s, s', z)$, where similarly to InfoGAN, the intuition is to learn a transition in the latent space $s \to s'$ that captures salient properties of the observation transition $o \to o'$.

In [133], the latent state distributions were $P(s) = \mathcal{N}(0, I), P(s'|s) = \mathcal{N}(s, \sigma(s))$. That is, the next state s' was modelled as a local perturbation of the first state s, where the magnitude of the perturbation $\sigma(s)$ was a learned neural network. The motivation for such dynamics was to structure the latent space to be compatible with a planning algorithm, as described below. The CIGAN loss is similar to InfoGAN, with the additional learning of dynamics in latent space $P(s'|s)$, and the mutual information between the *pairs* of latent states and observations:

$$\begin{aligned} &\min_{P(s'|s),G} \max_D V(G, D) - \lambda I(s, s'; o, o'), \\ &\text{s.t. } o, o' \sim G(z, s, s'), s \sim P(s), s' \sim P(s'|s) \end{aligned} \tag{5.2}$$

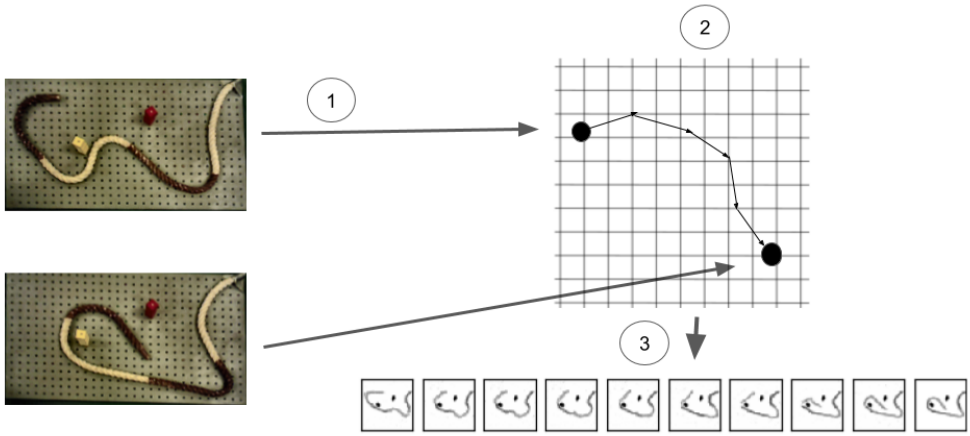


Figure 5.2: Illustration of how the CIGAN model generates a plan. First, start and goal images are encoded to their latent representations (denoted here as points in the plane). Second, search is used to find a sequence of points in the latent space that connect the start to the goal, while obeying the latent space dynamics. Here we illustrate the result of A* search. Third, the plan in latent space is decoded into a sequence of images using the generator, resulting in a visual plan.

To optimize equation 5.2, an InfoGAN lower bound was used, introducing an auxiliary distribution $Q(s, s'|o, o')$ to approximate $P(s, s'|o, o')$. Kurutach et al. proposed to use a disentangled approximation $Q(s, s'|o, o') = Q(s|o)Q(s'|o')$.

A CIGAN model trained on the data $\mathcal{D}$ can be used for visual planning according to the following scheme [133] and visually represented in Figure 5.2:

1. Encoding: given a pair, o_{start}, o_{goal}, find the corresponding s_{start}, s_{goal}.
2. Planning: in the latent space, find a feasible trajectory: $s_{start}, s_1, ..., s_m, s_{goal}$.
3. Decoding: from the latent trajectory generate a feasible observation trajectory $o_{start}, o_1, ..., o_m, o_{goal}$.

For encoding, Kurutach et al. [133] used an optimization based approach, searching for a latent vector that minimizes the absolute pixel difference with the desired observation. For the planning, the key idea in [133] is that due to the local transition structure of $P(s'|s)$, linear interpolation between s_{start} and s_{goal} results in a feasible plan. In this sense, CIGAN learns a representation that is *compatible* with the planning algorithm. For decoding, the CIGAN generator can be used to sequentially produce pairs of observations from the trajectory.

Learning Inverse Dynamics Models

An inverse model M_{IM} maps a pair of sequential observations o, o' to an action that generated them $u = M_{\text{IM}}(o, o')$. This can be cast as a supervised learning problem, by regressing from o_t, o_{t+1} in the data to u_t. Here, we follow the approach of Nair et al. [185], which learned inverse models from image observations using deep convolutional neural networks. Given a reference trajectory in image space $o_1^{ref}, \ldots, o_t^{ref}$, an inverse model can act as a tracking controller (a.k.a. visual servoing [58]) by taking the action $M_{\text{IM}}(o_t, o_{t+1}^{ref})$ at time t.

5.3 Visual Planning and Acting

In this section we present our approach for solving the goal directed planning problem of Section 5.2, which we term Visual Planning and Acting (VPA).

Our approach is model-based, where we first use the data $\mathcal{D}$ to learn both a CIGAN model M_{CIGAN} and an inverse dynamics model M_{IM}. For any two start and goal observations o_{start}, o_{goal}, the CIGAN model M_{CIGAN} can generate a visual plan that transitions the system from start to goal, $o_{start}, o_1, \ldots, o_k, o_{goal}$. Since the CIGAN model is trained to generate feasible pairs of observations (cf. Section 5.2), we are guaranteed that the plan generated by a well-trained CIGAN model will be feasible, in the sense that the robot can actually execute it.

Our VPA method for solving the goal directed planning problem is a combination of planning and replanning using the CIGAN model M_{CIGAN}, and trajectory tracking using the inverse model M_{IM}. The VPA algorithm is given as follows:

1. **Plan:** given a pair, o_{start}, o_{goal}, use the CIGAN model M_{CIGAN} to generate a planned sequence of observations $o_{start}, o_1, ..., o_m, o_{goal}$.

2. **Act:** If the length of the plan m is zero, take an action u to reach the goal $u = M_{\text{IM}}(o_{start}, o_{goal})$, then stop. Otherwise, take an action u to reach the first observation in the plan $u = M_{\text{IM}}(o_{start}, o_1)$ and take a new observation of the current system state o_{new}.

3. **Replan:** update o_{start} to be the current observation o_{new}, and go back to step 1.

VPA effectively uses the inverse model as a feedback controller to follow the imagined CIGAN plan. In practice, we found that the advantage of replanning in our tasks was not significant, and chose to omit it for faster execution times. That is, instead of replanning from the current observation we simply advanced on the original plan by removing the first observation o_1. However other tasks may benefit from full replanning.

We emphasize that while VPA uses planning, it builds on CIGAN, which is completely data-driven, and does not require manually engineering a planning model. The only data required for this is images taken from self-supervised manipulation of an object. Nevertheless, our method enjoys the **interpretability** of model based methods – at every step of our algorithm we have a visual plan of the proposed manipulation. We found that this allows us to reliably evaluate the performance of VPA before performing any robot experiment, significantly reducing time and effort

as well as unpredictability in the robot's actions. We also remark that separating decision making into a high-level trajectory computation step and a low-level action execution is standard in motion planning [139], and has been explored in several recent studies on robotic manipulation [252, 261]. Here, in comparison, the trajectories are in *image space*, and hence can capture complex object features such as deformations and change in appearance. Another benefit of separating observations and actions is the possibility of collecting different data for training M_{CIGAN} and M_{IM}. For example, in rope manipulation, the properties of the rope are largely independent of the robot manipulating it. Thus, we can collect a robot-independent data for training M_{CIGAN} with several different robots, or even a human, as we did in our experiments, and then collect a robot-specific data set for training M_{IM} for a particular robot.[1]

These properties makes our approach suitable for deformable object manipulation, as we demonstrate in our experiments. However, in order to get VPA to work well in practice, we needed to make several fundamental changes to the CIGAN method, as we describe next. We also describe several technical modifications in Appendix D.

Context Conditional CIGAN Model

The CIGAN model in the previous chapter generates observations that are, by definition, similar to the training data. That precludes any *generalization* to problem parameters that are different than those seen during training. However, in practical settings, we would like to generalize our knowledge to change in the environment. For example, in an environment with obstacles, one would like to learn a model that can generalize to different obstacle configurations.

Here we approach this problem by adding to the CIGAN model a *context* input. We assume that the domain can be decomposed into a manipulatable, movable object (e.g., rope), and components which are fixed during manipulation (e.g., obstacles). We propose a modification of the CIGAN architecture that takes as input an observation of the fixed components as a *context vector*. We term this model a Context Conditional CIGAN (C^{3}IGAN). By training on a variety of context vectors, we should hope that the model generalizes to novel contexts.

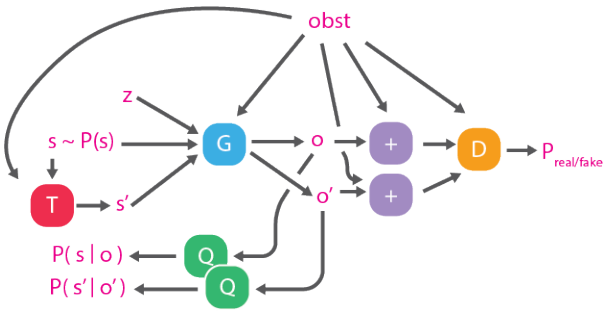


Figure 5.3: C^{3}IGAN Model Architecture

In C^{3}IGAN, as shown in Figure 5.3, the generator takes in as input z, s, s', c, where the context c represents an image of the fixed components in the domain, so in our case, the obstacles (obst). The generated observations are added (pixel-wise) to c before they are passed onto the discriminator. In this way the generator is trained to generate only the movable part in the scene. Thus, the generator is now only in charge of generating the images of the rope, and not the obstacles. By relieving the

[1] In principle, the action can be subsumed in the observation for training a CIGAN model that can plan actions. Due to the benefits mentioned above, we opted for computing actions independently using an inverse model.

generator of the responsibility of generating a fixed backdrop that is fixed throughout a trajectory, it can focus on the nuances of the object whose movement we actually want to control. The generator in this model is also able to generalize to new obstacle domains not seen during training. It is clear how this model could extend to other applications where there is a fixed background to interact with, such as a maze or other physical barriers.

A* Planning in Latent Space

In the previous chapter, we used a simple linear interpolation in latent space as the planning computation, based on the insight that the latent state transitions in CIGAN are Gaussian perturbations, guaranteeing that a small step in any direction in the latent space should result in a feasible transition. In our experiments on rope manipulation, we found that this method did not produce realistic enough of plans, especially in the presence of obstacles, which it tended to go through rather than around. We attribute this to the fact that the actual covariance matrix of the Gaussian probability may be asymmetric such that some directions have extremely low likelihood.

To account for this, we learned a more expressive transition model, $P(s'|s) \sim N(s+\delta(s), \sigma(s))$, where both δ and σ are neural networks. Shifting the mean allows the transition to prefer some direction than the others. We added a loss on the magnitude of δ and σ in order to induce a local transition structure, which complies with the A* heuristic and allows us to perform such planning at test time.

With this new transition model, we propose a different approach for planning in the latent space, which combines sampling and A* directed search. Given any state s, we can sample N possible next states s' from the probability $P(s'|s)$. Thus, we can recursively build a sampled connectivity graph of the possible transitions in latent space. Potentially, we can search this graph for a trajectory that reaches from s_{start} to a state close to s_{goal}. However, in practice, the latent space dimensions are too large to perform a naive search in reasonable time. To solve this problem, similar to the previous chapter, we leverage the *structure* of the latent space, and in particular, the local connectivity structure enforced by the Gaussian transition model. We propose to use the directed search algorithm A* with the Euclidean distance as a heuristic function, utilizing the fact that with local transitions, the Euclidean heuristic is admissible [219].

To improve the precision of our plans, we supplemented the sampling method by pruning unfeasible transitions, using a separately trained classifier, trained on positive examples of perturbed real rope images to bring the distribution closer to that of the generated data, and negative examples of generated rope images. For each sampled pair of states s, s', we generate a corresponding observation transition o, o' from the CIGAN generator, and if the classification score for this pair is lower than a threshold, we prune the transition $s \to s'$ from the connectivity graph.

5.4 Experiments

We designed our experiments to address the following questions:

1. Can our method generate non-trivial visual plans, and is the fidelity of these visual plans high enough to be combined with an inverse model for plan execution?

2. How does VPA compare to alternative methods like batch RL or running the inverse model without a plan?

3. Can VPA leave the simulation and work on a real robot?

We demonstrate our method on three domains. The first is a two-block world in Mujoco [263]. In this domain, we perform a comparison with batch off-policy RL – an alternative method for learning a control policy from data. The second domain contains a movable block with a static obstacle. In this domain, we show the need for planning when the inverse model fails to navigate around the obstacle, while VPA learns to do so. Finally, we deploy the algorithm on a PR2 robot to manipulate a deformable rope around obstacles. Within real world rope manipulation, we explore two similar variations of the domain: one with static obstacles in which we compare our method to that of Nair et al. [185], and the other with dynamic obstacles in which we demonstrate the potential of generalizing to variations in the environment using C^{3}IGAN.

Two-Block Domain

In this domain, the task is to move two rigid blocks on a table to some goal location. Possible actions are moving a block by some small offset in any direction. The table is 1.5 units on each side, and we consider the task a success if the L2 distance between the final and goal states is below 0.5. We collect data by randomly applying actions in the domain, and structure our data to contain 30k observation transitions, where only 2K transitions also have action labels. This corresponds to a setting where collecting possible observation transitions is easier than collecting real robot actions, as described above, and also demonstrated in our real robot experiment.

For VPA, we train a CIGAN model on the full dataset, and an inverse model on the action-labeled data. We use linear interpolation for planning, as in [133], as this domain is simple enough to not require the more complex A*. In Figure 5.5, we show a sample plan generated by CIGAN, and the corresponding trajectory executed by VPA. It is clear that the initial plan is visually interpretable, and resembles the actual trajectory that was executed. Quantitatively, we evaluated VPA on 50 random initial and goal configurations that were not in the data, as shown in Table 5.1.

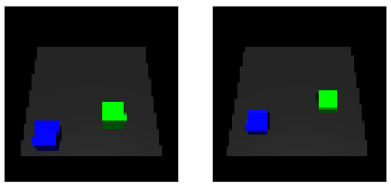


Figure 5.4: For reference, these two images have an L2 distance in state of .3509 from each other

We compare VPA with an alternative data-driven approach based on model-free batch RL[2], namely, fitted Q-iteration [214] (equivalent to a single epoch of DQN [177] with the data as the

[2]For image observations, the state of the art in RL is model free [177, 152, 116], while recent model based approaches are limited to lower dimensional observations [135]. Therefore, we did not consider model based RL in our comparison.

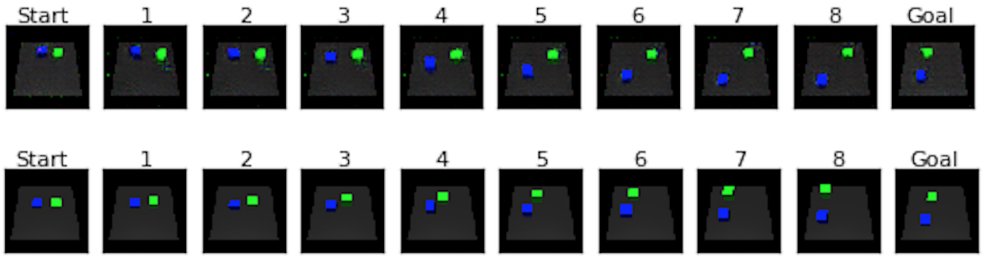


Figure 5.5: Top image: Visual Planning step - imagined plan by Causal InfoGAN. Start and Goal image are both $o_{closest}$ to the actual o_{start} and o_{goal}, which are shown right below them. Bottom image: Execution step - images showing the actual successful results of running entire VPA pipeline on Mujoco

Method	L2 distance	Success Rate
VPA (2k)	**0.335 ±0.121**	**90%**
Batch RL (positions, real r, 2k)	0.657 ±0.701	76%
Batch RL (positions, real r, 30k)	0.675 ±0.739	74%
Batch RL (image, real r, 2k)	1.172 ±0.991	16%
Batch RL (image, real r, 30k)	1.186 ±0.940	42%
Batch RL (image, embedded r, 2k)	1.346 ±0.891	14%
Batch RL (image, embedded r, 30k)	1.445 ±1.096	18%

Table 5.1: The average final L2 distance to goal and the success rate to move two blocks to be within 0.5 radius to the goal when executed on 50 new tasks.

replay buffer). RL requires action labels, so we trained with only the action-labeled part of the data. Since actions are continuous, we used random sampling to find the maximal Q value in the Bellman backup, similar to [116]. For the state space, we embedded the images into a latent space using a variational autoencoder, *trained on all the data*, and the reward was based on distance in latent space, as recently suggested in [187]. Since RL is not expected to generalize, we *retrained the Q network on all the data for each goal in the evaluation*. This is a strong baseline, that makes use of both the action-labeled and unlabeled data, incorporates several recent techniques for image-based RL, and our evaluation forgives the limitations of RL in generalizing to different goals.

However, as stated earlier, RL is known to have difficulties with large state spaces (image), reward specification, and sample efficiency. To demonstrate this, we also run RL with several *artificial benefits*: (1) simple state space – true positions of the blocks, (2) true reward – based on real distance to target, and (3) significantly more data – 30k action-labeled samples.

Our results, reported in Table 5.1 show that, surprisingly, VPA significantly outperforms RL even with the artificial benefits. Only with all benefits added, does RL compare well with VPA. We attribute these results to the fact that in this domain, decomposing control to trajectory planning and tracking control is natural, and VPA exploits this structure. Indeed, the common failure case for RL is pushing the blocks off the table due to an inaccurate Q function. Since VPA never imagines plans where blocks go off the table, our method was resilient to such failures.

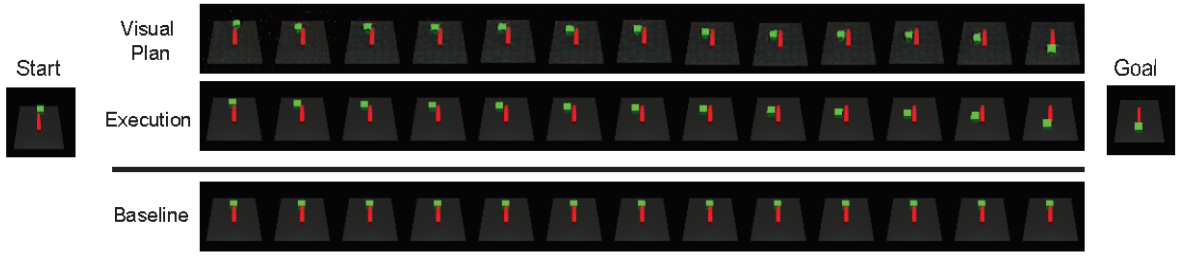


Figure 5.6: Comparison between VPA and an inverse model baseline. The baseline attempts to directly apply the inverse model on the goal, while our method employs the plan generated by CIGAN, shown at top, to navigate from start to goal. Without a plan, the baseline blindly attempts to move the block downwards without accounting for the obstacle in the way. Our model, on the other hand, plans to go around the obstacle, resulting in a successful trajectory.

Method	L2 distance	Success Rate
Baseline	0.459 ±0.433	45%
VPA (hand-selected)	**0.083 ±0.192**	**95%**
VPA (autoselected plan)	0.131 ±0.242	90%

Table 5.2: The average final L2 distance to goal and the success rate to move one block in the block-wall domain to be within 0.5 radius to the goal.

Block-Wall Domain

To further motivate the need for planning, we investigate the efficacy of our model on another simulated domain, now with planning more intuitively necessary to complete the task. In this domain, the agent has to manipulate a green block around a red vertical obstacle. We perform the same VPA method as before, on a new test set of 20 start/goal image pairs.

We compare two variations of our method against the baseline of using only an inverse model, as used in Nair et al. [185]. The first method executes the single best hand-selected plan from many generated by the CIGAN. There is some variability in the quality of generated plans due to the random noise, and some are better than others, so we choose only the best one to execute.

The second method autoselects a generated plan to execute. This plan is selected using a combination of a classifier trained on the dataset and an object detector trained on a simple shape dataset. We describe this more in Appendix D.

Our results in Table 5.2 show that planning with VPA significantly improves upon the inverse model baseline, for both plan selection methods. In Figure 5.6, we show an example where planning is necessary, and the baseline is unable to execute the task while our method is successful.

Real Robot Rope Manipulation Domain

Finally, we bring our method out of simulation and into the real world by conducting experiments with a PR2 robot manipulating a flexible rope that is fixed on one end and can move between two

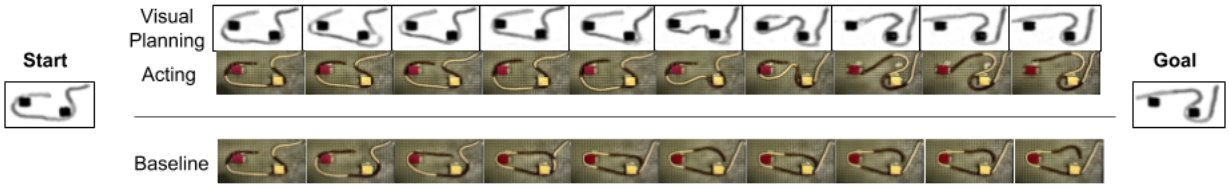


Figure 5.7: Comparison between VPA and an inverse model baseline. In this example, the rope is required to travel around an obstacle to reach the goal. The CIGAN-generated plan is presented in grayscale in the middle. The results after the PR2 robot successfully runs iterations of the M_{IM} with our VPA method by tracking the plan is shown above, and the baseline of only the inverse model, M_{IM} is shown below (similar to the method of [185]). Note that VPA plans to go *around* the obstacle, leading to a successful plan execution, while the inverse model is not capable of such planning, due to its reactive, short-term nature, and therefore cannot complete the task. This example demonstrates the importance of planning for nontrivial manipulation tasks.

obstacles. This domain is inspired by wire threading – an important industrial task that requires complex planning of rigid and non-rigid object interaction.

Static Obstacles

We begin our investigation by comparing our planning based method to a baseline of only using an inverse model without planning, as in the previous block and wall domain. We designed a rope manipulation environment similar to [185], but which also contains fixed obstacles which the rope cannot move through.

For data collection, we followed the approach in [185] for generating random pokes of the rope, and collected 2k samples for observations and actions. To increase the size of our dataset, we collected 10k additional observation samples by manually manipulating the rope (which is much faster to collect). Note, however, that due to the obstacles, our problem is *much* more difficult than in [185]. The images are preprocessed to have one color channel, as demonstrated in Figure 5.8.

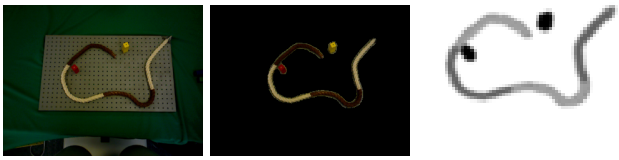


Figure 5.8: Preprocessing procedure. Image 1: original photo as seen by the PR2, Image 2: background removed from image, Image 3: Preprocessed to have one color channel

With the additional constraint of obstacles, we conjecture that the inverse model, which is essentially reactive in its computation, will not suffice to plan movements that involve these obstacles. In contrast, a well trained CIGAN model should generate plans that manipulate the rope around such obstacles. For the inverse model, M_{IM}, we used a CNN similar to [185], outputting a 4-dimensional action: the rope grab and drop location. We found that feeding in to M_{IM} the observation difference $o' - o$ resulted in a more robust controller that generalized well to the generated image sequences. As descvribed in Section 5.3, to reduce computation time we did not replan in the VPA algorithm, and simply ran M_{IM} multiple times for each generated observation, to result in a longer, but more stable execution.

In Figure 5.7, we demonstrate a setting where nontrivial planning is required to solve the task –

going from start to goal requires traveling with the rope around the obstacle. It can be seen that our VPA method *plans* to go around the obstacle, which makes it feasible to solve the task by following the plan with the inverse model M_{IM}. Just using M_{IM} on the goal image, however, does not result in traveling around the obstacle, which leads to a failure in execution. This result demonstrates the inadequacy of purely reactive methods, such as inverse models, for acting in complex domains with more constraints. We further evaluate the planning capability of our method in Figure 5.9, where we demonstrate realistic plans of rope manipulation that obey the physical properties of the rope and obstacles.

Dynamic Obstacles

In this section we demonstrate the potential of C^{3}IGAN in generalizing to unseen environments. To this end, we modified the rope manipulation domain to include *dynamic*, smaller obstacles, which were intermittently moved (manually) while collecting the training data. These changes render this variation harder than the previous one for our vision-based planning method. Our hope is that our model can imagine, plan, and execute rope manipulation in domains with obstacle configurations that were not explicitly seen during training.

In training our C^{3}IGAN model for this domain, the context is an image of the obstacles without any rope, as show in Figure 5.11. As the figure demonstrates, the obstacle embedding is successfully used by the C^{3}IGAN model to generate images that realistically capture the interaction between the rope and obstacles, such as that the rope has to wind around the obstacle, rather than moving through it.

In Figure 5.10 we demonstrate results for VPA on this domain. We present both the imagined C^{3}IGAN plans, and the resulting trajectories when running VPA on the robot.

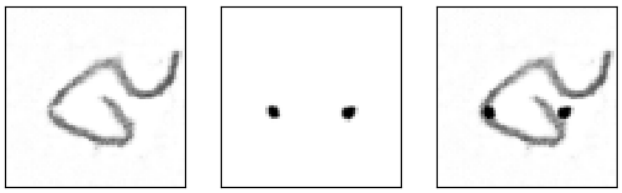


Figure 5.11: Visualization of C^3IGAN results. The model generates an image conditioned on the obstacles, and then performs a pixelwise addition of the left and middle image. Left: generated rope, Middle: obstacles conditioned on, Right: pixelwise addition of images.

In terms of success rate, we qualitatively inspected the plans and found that approximately 15% were visually accurate representations of rope manipulation. The most common failure cases were inaccurate encoding, leading to a misspecified goal image, or the rope breaking and reconnecting somewhere else during the trajectory. We believe that more data would significantly improve these results – our results for the plans with static obstacles, which were trained using the same amount of data yet are significantly easier, were indeed significantly better. From the visually correct plans, the inverse model was able to successfully execute 20%. This is somewhat worse than the results of Nair et. al. [185], which we attribute to the order of magnitude smaller dataset we used, and our additional obstacles. We emphasize that even though our success rates are not high, most failure cases can be caught *by visual inspection*, without running the robot, since our method is readily interpretable. Such interpretability has additional important implications to safety.

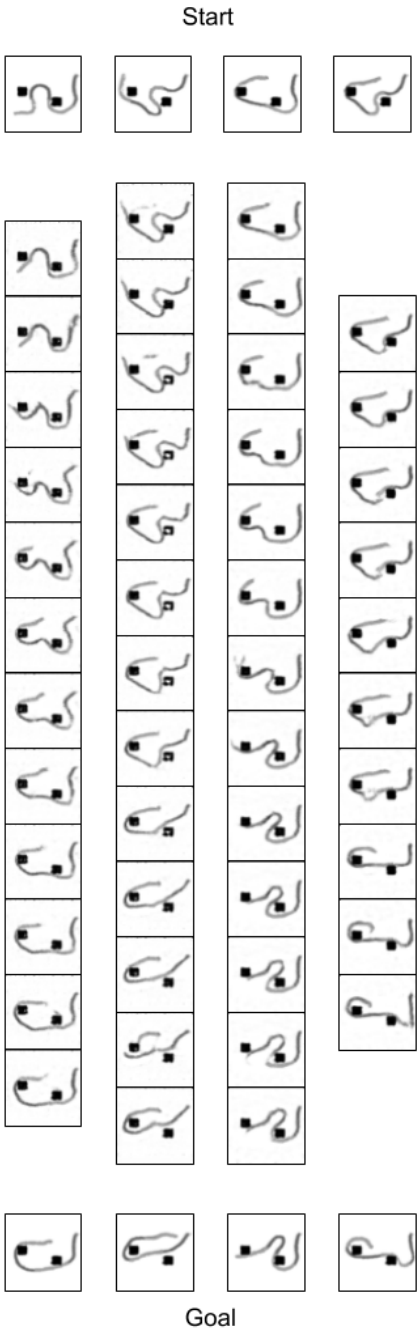


Figure 5.9: Demonstration of CIGAN plans for the rope domain with fixed obstacles. The top row shows the start states and the bottom row shows the goal states of 4 different problems given to the CIGAN. The middle 4 columns show sample generated plans. Note the realistic transitions of the rope around the obstacles, which obey physical properties of the rope such as being stretched when pulled from the end.

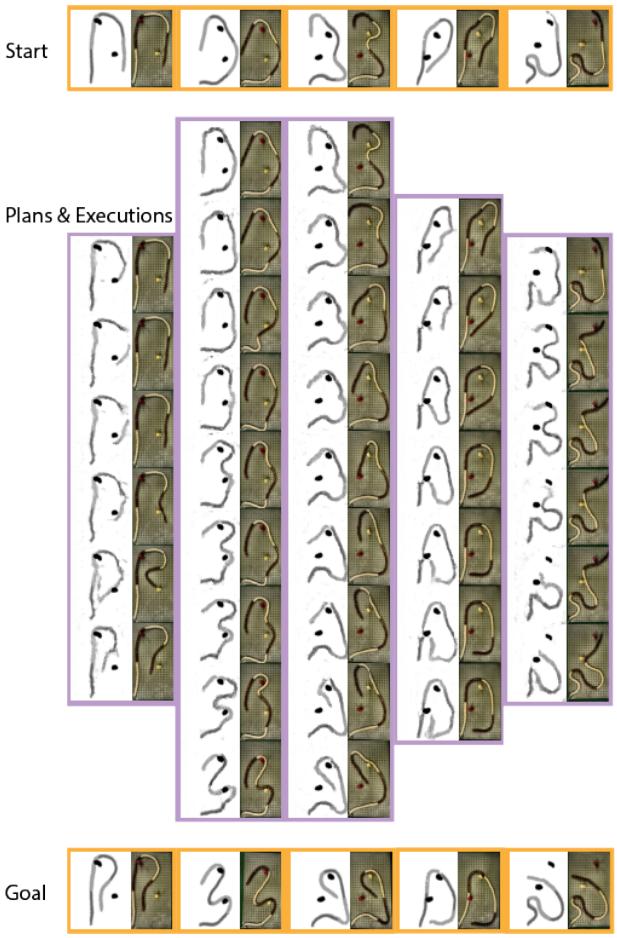


Figure 5.10: 5 examples of VPA executed on the rope domain. The left 4 are successful runs, and the right 1 is where a plan is generated to reach the goal, but the action policy is not strong enough to carry it out. Looking at one column at a time, the top image is the start state and the bottom is the goal state. In the middle, the grayscale images are the visualized plan, and the colored images are the actual results of the rope when we run the inverse model to have the PR2 take actions.

Thus, while further investigation is required to improve the quality of VPA, we see these results as a proof of concept for a promising robotic manipulation paradigm.

5.5 Discussion and Conclusion

We proposed a new data-driven paradigm for robot manipulation by learning a model for planning in image space, and using the imagined plan as a reference for a visual tracking controller. Our method is interpretable, and we showed that it can significantly outperform model-free RL approaches in simulation. We have shown promising results using a PR2 robot in learning to manipulate a deformable rope among obstacles.

Recently, data-driven approaches to robotic manipulation have increased in popularity, and are widely considered to be essential for bringing robots into human centered environments. Most of the work so far has focused on model free RL, which, although capable of learning complex policies, results in opaque controllers that are only verifiable by physical evaluation on the robot. This has raised many issues of safety and interpretability [6], and at present there is no principled method for explainable RL.

We believe that the most important aspect of our approach is its visually interpretable nature. As we have shown, the imagined plans are easy to understand, and when supervised by a human operator, unsafe or undesired plans can be intercepted before the physical execution of the task. Indeed, we have exploited the interpretability of our method for selecting the best performing CIGAN model parameters by visually inspecting the generated plans, before running any robot experiment.

So far we have studied manipulation problems in which the same object is manipulated while the contexts change. In the next chapter, we will investigate further how to take into account changes in object shapes in modeling and planning.

Chapter 6

Context Generalization

We are interested in goal-directed planning problems where the state observations are high-dimensional images, the system dynamics are not known, and only a data set of state transitions is available. In particular, given a starting state observation and a goal state observation, we wish to generate a sequence of actions that transition the system from start to goal. One application for such problems is in self-supervised robot learning, where it is relatively easy to acquire such data by letting the robot explore its environment randomly, and the problem becomes how to process this data for solving various tasks [185, 272, 202, 67].

Given a reward function, deep reinforcement learning (RL) can plan with high-dimensional inputs, and batch off-policy RL algorithms [137] can be applied to the problem above [89, 57, 177, 238]. However, goal-based planning is a sparse reward task, which is known to be difficult for RL [9]. Moreover, RL provides black-box decision policies, which are not interpretable, and can only be evaluated by running them on a robot. Addressing both data-driven modeling and interpretability, the *visual planning* (VP) paradigm seeks to first generate a visual plan – a sequence of images that transition the system from start to goal, which can be understood by a human observer – and only then take actions that follow the plan using visual servoing methods.

Bearing similarity to model-based RL [255], most VP approaches learn a low-dimensional latent variable model for the system dynamics, and plan a state-to-goal sequence by searching in the latent space [134, 11, 57, 91, 188]. There are two shortcomings to this approach: training deep generative models with a structured latent space can be tricky in practice [279, 134], and consequentially, the resulting visual plans are often of low visual fidelity.

In this work, we propose a simple VP method that plans directly in image space. We build on the semi-parametric topological memory (SPTM) method proposed by Savinov et al. [228]. In SPTM, images collected offline are treated as nodes in a graph and represent the possible states of the system. To connect nodes in this graph, an image classifier is trained to predict whether pairs of images were 'close' in the data or not, effectively learning which image transitions are feasible in a small number of steps. The SPTM graph can then be used to generate a visual plan – a sequence of images between a pair of start and goal images – by directly searching the graph. SPTM has several advantages, such as producing highly interpretable visual plans and the ability to plan long-horizon behavior. Here, we ask – is such a simple scheme competitive with VP methods that plan in latent

space?

To answer this question, we need to address a limitation of SPTM compared to VP methods such as visual foresight [67, 57]. Since SPTM builds the visual plan directly from images in the data, when the environment changes – for example, the lighting varies, the camera is slightly moved, or other objects are displaced – SPTM requires *recollecting* images in the new environment; in this sense, SPTM *does not generalize in a zero-shot sense*. To tackle this issue, we assume that the environment is described using some context vector, which can be an image of the domain or any other observation data that contains enough information to extract a plan (see Figure 6.1 top left). We then train a conditional generative model that hallucinates possible states of the domain conditioned on the context vector. Thus, given an unseen context, the generative model hallucinates exploration data without requiring actual exploration. Using the hallucinated images, we can then perform planning in image space.

Additionally, similar to [60], we find that training the graph connectivity classifier as originally proposed by [228] requires extensive manual tuning. We replace the vanilla classifier used in SPTM with an energy-based model that employs a contrastive loss. We show that this alteration drastically improves planning robustness and quality. Finally, for planning, instead of connecting nodes in the graph according to an arbitrary threshold of the connectivity classifier, as in SPTM, we cast the planning as an inference problem, and efficiently search for the shortest path in a graph with weights proportional to the inverse of a proximity score from our energy model. Empirically, we demonstrate that this provides much smoother plans and barely requires any hyperparameter tuning. We term our approach Hallucinative Topological Memory (HTM). A visual overview of our algorithm is presented in Figure 6.1.

We evaluate our method on a set of simulated VP problems that require non-myopic planning, and accounting for non-trivial object properties, such as geometry, in the plans. In contrast with prior work, which only focused on the success of the method in executing a task, here we also measure the *interpretability* of visual planning, through mean opinion scores of features such as image fidelity and feasibility of the image sequence. In both measures, HTM outperforms state-of-the-art data-driven approaches such as visual foresight [57] and the original SPTM. The codebase and videos are at `sites.google.com/view/hallucinativetopologicalmemory`.

6.1 Preliminaries

Context-Conditional Visual Planning and Acting (VPA) Problem. We consider the context-conditional visual planning problem from [134, 272]. Consider deterministic and fully-observable environments $\mathcal{E}_1, ..., \mathcal{E}_N$ that are sampled from an environment distribution $P_{\mathcal{E}}$. Each environment $\mathcal{E}_i$ can be described by a context vector c_i that entirely defines the dynamics $o^i_{t+1} = m(o^i_t, a^i_t|c_i)$, where o^i_t, a^i_t are the observations and actions, respectively, at timestep t from context c_i. For example, in the illustration in Figure 6.1, the context could represent an image of the obstacle positions, which is enough to predict the possible movement of objects in the domain.[1] As is typical in VP problems,

[1] We used such a context image in our experiments. We assume that in practice it is feasible to remove the robot from the domain, making this setting relevant to applications.

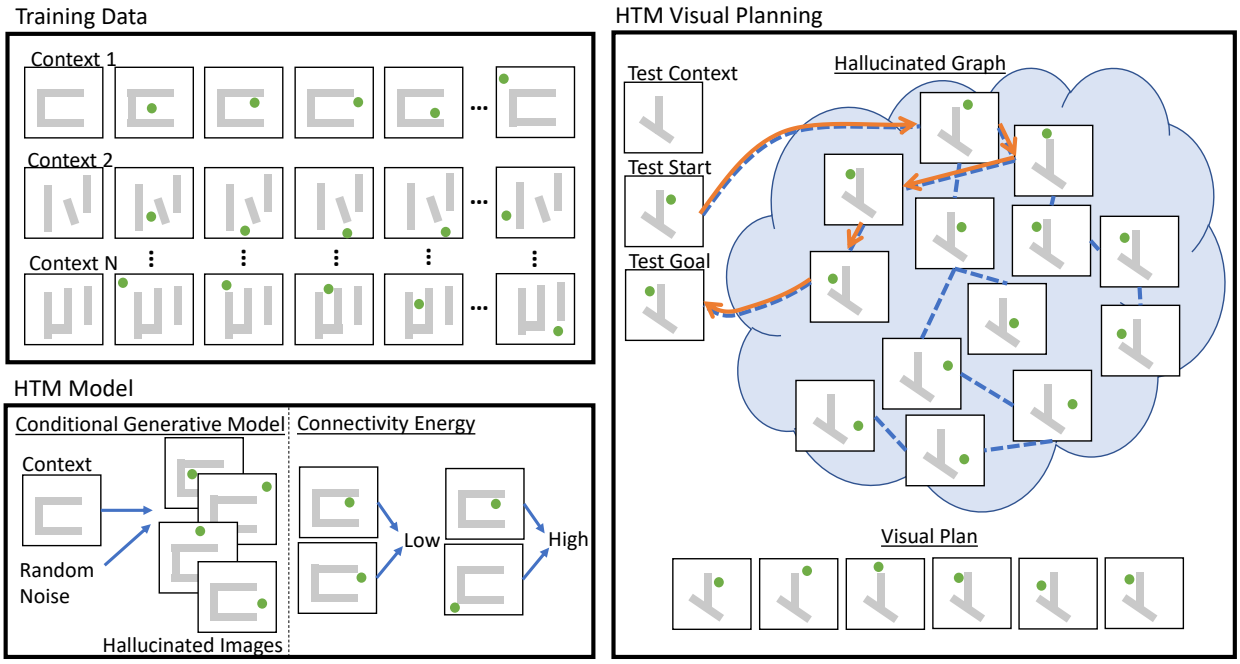


Figure 6.1: HTM illustration. **Top left:** data collection. In this illustration, the task is to move a green object between gray obstacles. Data consists of multiple obstacle configurations (contexts), and images of random movement of the object in each configuration. **Bottom left:** the elements of HTM. A conditional generative model is trained to hallucinate images of the object and obstacles conditioned on the obstacle image context. A connectivity energy model is trained to score pairs of images based on the feasibility of their transition. **Right:** HTM visual planning. Given a new context image and a pair of start and goal images, we first use the conditional generative model to hallucinate possible images of the object and obstacles. Then, a connectivity graph (blue dotted lines) is computed based on the connectivity energy, and we plan for the shortest path from start to goal on this graph (orange solid line). For plan execution, visual servoing is later used to track the image sequence.

we assume our data $\mathcal{D} = \{o_1^i, a_1^i, ..., o_{T_i}^i, c_i\}_{i \in \{1,...,N\}}$ is collected in a self-supervised manner [2], and that in each environment $\mathcal{E}_i$, the observation distribution is defined as $P_o(\cdot|c_i)$.

At test time, we are presented with a new environment, its corresponding context vector c, and a pair of start and goal observations o_{start}, o_{goal}. Our goal is to use the training data to build a planner $\mathcal{K}_h(o_{start}, o_{goal}, c)$ and an h-horizon policy $\pi_h(o_{current}, o_{target})$. The planner's task is to generate a sequence of observations between o_{start} and o_{goal}, in which any two consecutive observations are reachable within h time steps. The policy outputs an action that brings the current image to the target image within h steps which can be used to follow the generated plan. This requires both the planner and the policy to work together in zero-shot.

In this work, we first evaluate the planner and policy separately – the planner by measuring the fidelity of its plans, and the policy, by measuring its success rate in tracking a feasible plan. We

[2]Like Nair et al. [185] and Wang et al. [272], we assume that random actions allow us to sufficiently explore the environment for the tasks we consider. Depending on the task, this may require some engineering of the set of possible actions, e.g., a focused pick-and-place action set in Nair et al. [185]. In our domains, basic movements and rotations sufficed.

then also evaluate the combined planner+policy by measuring the total success rate of the policy applied to the planned trajectories. For simplicity we will omit the subscript h for the planner and the policy.

Semi-Parametric Topological Memory (SPTM) [228] is a visual planning method that can be used to solve a special case of VPA, where there is only a single training environment, $\mathcal{E}$ and no context image. SPTM builds a memory-based planner and an inverse-model controller. At training, a classifier R is trained to map two observation images o_i, o_j to a score $\in [0, 1]$ representing the feasibility of the transition, where images that are $\leq h$ steps apart are labeled positive and images that are $\geq l$ are negative. The policy is trained as an inverse model L, mapping a pair of observation images o_i, o_j to an appropriate action a that transitions the system from o_i to o_j.

Given an unseen environment $\mathcal{E}^*$, new observations are *manually* collected and organized as nodes in a graph G. Edges in the graph connect observations o_i, o_j if $R(o_i, o_j) \geq s_{shortcut}$, where $s_{shortcut}$ is a manually defined threshold. To plan, given start and goal observations o_{start} and o_{goal}, SPTM first uses R to localize, i.e., find the closest nodes in G to o_{start} and o_{goal}. A path is found by running Dijkstra's algorithm, and the method then selects a waypoint o_{w_i} on the path which represents the farthest observation that is still feasible under R. Since both the current localized state o_i and its waypoint o_{w_i} are in the observation space, we can directly apply the inverse model and take the action a_i where $a_i = L(o_i, o_{w_i})$. After localizing to the new observation state reached by a_i, SPTM repeats the process until the node closest to o_{goal} is reached.

Contrastive Predictive Coding (CPC) [196] is a method for learning low-dimensional representations of high-dimensional sequential data. CPC learns both an encoding of the data at every time step, and an energy function for any two observations in different time steps in suggesting their temporal correlation. A non-linear encoder g_{enc} encodes the observation o_t to a latent representation $z_t = g_{enc}(o_t)$. We maximize the mutual information between the latent representation z_t and future observation o_{t+k} with a log-bilinear model[3] $f_k(o_{t+k}, o_t) = \exp(z_{t+k}^T W_k z_t)$. This model is trained to be proportional to the density ratio $p(o_{t+k}|z_t)/p(o_{t+k})$ by the CPC loss function: the cross entropy loss of correctly classifying a positive sample from a set of random observations consisting of 1 positive sample (o_t, o_{t+k}) from the paired data $\mathcal{D}_{pair}$ and $N-1$ negative samples (o_t, o') where o' is sampled separately from the full data $\mathcal{D}_{single}$:

$$I(o_t, o_{t+1}) \geq -\mathbb{E}_{(o_t, o_{t+k}) \sim \mathcal{D}_{pair}} \left[\log \frac{f(z_t, z_{t+1})}{\sum_{\tilde{o} \in \mathcal{D}_{single}} f(\tilde{o}, z_t)} \right].$$

Note that the model f_k is not necessary symmetric, and therefore can capture asymmetric transition in the data. f_k can also be viewed as an inverse energy model whose outputs are high for positive samples and low for negative samples.

[3]The original CPC model has an additional autoregressive memory variable [196]. We drop it in our formulation as our domains are fully observable and do not require memory.

6.2 Hallucinative Topological Memory

By planning directly in image space, and composing the plan from real images (vs. planning in a learned latent space [134]), SPTM is guaranteed to produce high-fidelity visual plans. In addition, SPTM has been shown to solve long-horizon planning problems such as navigation from first-person view [228]. However, *SPTM is not zero-shot*: even a small change to the training environment requires collecting substantial exploration data for building the planning graph. This can be a limitation in practice, especially in robotic domains, as any interaction with the environment requires robot time, and exploring a new environment can be challenging (indeed, Savinov et al. [228] applied manual exploration). In addition, similarly to [60], we found that training the connectivity classifier as proposed by Savinov et al. [228] requires extensive hyperparameter tuning.

In this section, we propose an extension of SPTM to overcome these two challenges by employing three ideas – (1) using a conditional generative model such as CVAE [244] or CGAN [171] to hallucinate samples in a zero-shot setting, (2) using contrastive loss for a more robust score function and planner, and (3) planning based on an approximate maximum likelihood formulation of the shortest path under uniform state distribution. We call this approach Hallucinative Topological Memory (HTM), and next detail each component in our method.

Hallucinating Samples

We propose a zero-shot learning solution for automatically building the planning graph using only a context vector of the new environment. Our idea is that, after seeing many different environments and corresponding states of the system during training, given a new environment we should be able to effectively *hallucinate* possible system states. We can then use these hallucinations in lieu of real samples from the system in order to build the planning graph. To generate images conditioned on a context, we implement a conditional generative model as depicted in Figure 6.1. During training, we learn the conditional distribution $p_\theta(o|c)$. During testing, when prompted with a new context vector c_{test}, we generate samples $\hat{o}_1, ..., \hat{o}_N \sim p_\theta(o|c_{test})$ in replacement of exploration data.

Algorithm

We now describe the HTM algorithm. Given a start observation o_{start}, a goal observation o_{goal} sampled from a potentially new environment $\mathcal{E}^*$, and the context vector c, we propose a 4-step planning algorithm.

1. We hallucinate exploration data $\hat{o}_1, ..., \hat{o}_N$ by sampling from the conditional generative model $p_\theta(\cdot|c)$.

2. We build a fully-connected weighted graph $G(V, E)$ by forming connections between all generated image pairs $\hat{o}_i, \hat{o}_j$ with learned directed edge weight w_{ij}.

3. We find the shortest path using Dijkstra's algorithm on the learned connectivity graph G between the start and goal node.

4. We apply a local policy to follow the visual plan, attempting the next node in our shortest path for h time steps, and replan every fixed number of steps until we reach o_{goal}.

In step 2, the weights should reflect difficulty in transitioning from one state to another using a self-supervised exploration policy. The learned connectivity graph G can be viewed as a topological memory upon which we can use conventional graph planning methods to efficiently perform visual planning. In step 4, for the policy, we train an inverse model which predicts an action given the current observation and a nearby goal observation. In practice, given a transition (o_t, a_t, o_{t+1}), we train a deep convolutional neural network $\pi(o_t, o_{t+1}) = \hat{a}_t$ to minimize the L2 loss between a_t and $\hat{a}_t$ [185, 272].

Learning the Connectivity via Contrastive Loss

A critical component in the SPTM method is the connectivity classifier that decides which image transitions are feasible. False positives may result in impossible short-cuts in the graph, while false negatives can make the plan unnecessarily long. In [228], the classifier was trained discriminatively, using observations in the data that were reached within h steps as positive examples, and more than l steps as negative examples, where h and l are chosen arbitrarily. In practice, this leads to three important problems. First, this method is known to be sensitive to the choice of positive and negative labeling [60]. Second, training data are required to be long, non-cyclic trajectories for a high likelihood of sampling 'true' negative samples. However, self-supervised interaction data often resembles random walks that repeatedly visit a similar state, leading to inconsistent estimates on what constitutes negative data. Third, since the classifier is only trained to predict positively for temporally nearby images and negatively for temporally far away images, its predictions of *medium-distance* images can be arbitrary. This creates both false positives and false negatives, thereby increasing shortcuts and missing edges in the graph.

To solve these problems, we propose to learn a connectivity score using contrastive predictive loss [196]. We initialize a CPC encoder g_{enc} that takes in both observation and context, and a density-ratio model f_k that does not depend on the context. Through optimizing the CPC objective, f_k is trained such that positive pairs, which appear sequentially, have higher score, i.e., lower energy, than negative pairs, which are sampled randomly from the data. Thus, it serves as a proxy for the temporal distance between two observations in the sense that sequential observations should have lower energy, leading to a connectivity score for planning in the next section. Compared to the heuristic classification loss in SPTM, the CPC loss is derived from a clear objective: maximize the mutual information between the latent encodings of current and future observations. In practice, this results in less hyperparameter tuning and a smoother distance manifold in the representation space mitigating the first and the third problems.

To tackle the second problem, instead of only sampling negative data within the same trajectory as an anchor image o_t as done in SPTM, we sample any $\tilde{o}$ that shares the same context c as o_t from the replay buffer. We also find that adding negative data sampled from $p_\theta(.|c)$ can help f_k evaluate more consistently on hallucinated images. Without this trick, we find that the SPTM

classifier suffers from false negatives and fails to train on short, cyclical trajectories collected by self-supervised interaction.

Edge Weight Selection

We would like edge weights to reflect the difficulty in transitioning from one state to another according causality in the data – low weight when the transition is feasible. Based on the connectivity score from the contrastive loss, we proposed two choices of computing the weight w_{ij} from node j to node i: (1) an energy model or an inverse of f_k, i.e., $1/f_k(i,j)$, and (2) a density ratio or an inverse of normalized f_k over outgoing edges from j, i.e., $\sum_{s \in V} f_k(s,j)/f_k(i,j)$. With this heuristic, the shortest path in G tries to predict reachable visual plans. In Appendix, we argue that the shortest path in graph G according to the weights in option 2 leads to maximizing trajectory likelihood bound under uniform data assumption, thus, casting planning as inference.

6.3 Related work

Reinforcement Learning. Most of the study of data-driven planning has been under the model-free RL framework [238, 177, 241]. However, the need to design a reward function, and the fact that the learned policy does not generalize to tasks that are not defined by the specific reward, has motivated the study of model-based approaches. Recent work [115, 105] investigated model-based RL from pixels on Mujoco and Atari domains, but did not study generalization to a new environment. Other work in model-based RL with image-based goals and visual MPC [67, 57] rely on video prediction, and are limited in the planning horizon due to accumulating errors. In comparison, our method does not predict full trajectories but only individual images, mitigating this problem. Our method is orthogonal to and can be combined with visual MPC as a replacement for the inverse model.

Concurrently with our work, Nair and Finn [188] propose a hierarchical visual MPC method that, similarly to our approach, optimizes a sequence of hallucinated images as sub-goals for a visual MPC controller, by searching over the latent space of a CVAE. To compute a search update over a proposed plan, the algorithm evaluates the video prediction score between consecutive subgoals making it more expensive and challenging to optimize as the number of subgoals increases. In practice, it is shown to work with the maximum of 2 subgoals. We also find that it takes approximately to 2 hours to plan a single task comparing to a few seconds in HTM making the algorithm impractical to evaluate without access to a large GPU cluster and especially in a closed-loop setting.

Self-supervised learning. Several studies investigate planning goal directed behavior from data obtained offline, e.g., by self-supervised robot interaction [4, 202]. Nair et al. [185] use an inverse model to reach local sub-goals, but require human demonstrations of long-horizon plans. Wang et al. [272] solve the visual planning problem using a conditional version of Causal InfoGAN [134]. Our work is not tied to specific types of generative models and in the experiments we opted for the CVAE-based approach for stability and robustness.

Classical planning and representation learning. Studies that bridge between classical planning and representation learning include [134, 13, 11, 60]. These works, however, do not consider

zero-shot generalization. While [249] and [206] learn representations that allow goal-directed planning to unseen environments, they require expert training trajectories. Ichter and Pavone [105] also generalize motion planning to new environments, but require a collision checker and valid samples from test environments.

6.4 Experiments

We evaluate HTM on a suite of simulated tasks inspired by robotic manipulation domains. We note that recent work in visual planning (e.g., [134, 272, 57]) focused on real robotic tasks with visual input. While impressive, such results can be difficult to reproduce or compare. For example, it is not clear whether manipulating a rope with the PR2 robot [272] is more or less difficult than manipulating a rigid object among many visual distractors [57]. Our suite of tasks is reproducible, contains clear evaluation metrics, and our code will be made available for evaluating other algorithms in the future.

We consider four domains in varying difficulty using Mujoco simulation [263], as seen in Figure 6.2:

1. **Block wall**: A green block navigates around a static red obstacle, which can vary in position.
2. **Block wall with complex obstacle**: Similar to the above, but here the wall is a 3-link object which can vary in position, joint angles, and length, making the task significantly harder.
3. **Block insertion:** Moving a blue block, which can vary in shape, through an opening.
4. **Robot manipulation:** A simulated Sawyer robot reaching and displacing a block.

With the first three domains, we aim to assess how well HTM can *generalize* to new environments in a zero-shot manner, by varying the position of the obstacle, the shape of the obstacle, and the shape of the object. With the forth domain, we aim to assess whether HTM can plan temporally-extended robotic manipulation.

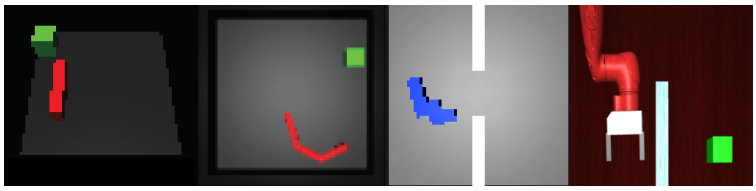


Figure 6.2: Evaluation suite. Block wall, block wall with complex obstacle, block insertion, and robot manipulation domains.

We ask the following questions. First, does HTM improve **visual plan quality** over state-of-the-art VP methods [228, 57]? Second, how does HTM **execution success rate** compare to state-of-the-art VP methods? We discuss our evaluation metrics for these attributes in Section 6.4.

We compare HTM with two state-of-the-art baselines: SPTM [228] and Visual Foresight [57]. When evaluating zero-shot generalization, SPTM requires samples from the new environment. For a fair comparison, we use the same samples generated by the same CVAE as HTM. Thus, in this

case, we only compare between the SPTM classification scores as edge weights in the graph, and our CPC-based scores.[4]

The same low-level controller is also used to follow the plans. The Visual Foresight baseline trains a video prediction model, and then performs model predictive control (MPC), which searches for an optimal action sequence using random shooting. For the random shooting, we used 3 iterations of the cross-entropy method with 200 sample sequences. The MPC acts for 10 steps and then replans, where the planning horizon T is set to 15 as in the original implementation. Experiments with different horizons yielded worse performance. We use the state-of-the-art video predictor as proposed by Lee et al. [141] and the public code provided by the authors. For evaluating trajectories in random shooting, we studied two cost functions that are suitable for our domains: pixel MSE loss and green pixel distance. The pixel MSE loss computes the pixel distance between the predicted observations and the goal image. This provides a sparse signal when the object pixels in the plan can overlap with those of the goal. We also investigate a cost function that uses prior knowledge about the task – the position of the moving green block, which is approximated by calculating the center of mass of the green pixels. As opposed to pixel MSE, the green pixel distance provides a smooth cost function which estimates the normalized distance between the estimated block positions of the predicted observations and the goal image. Note that this assumes additional domain knowledge which HTM does not require.

Evaluation Metrics

We design a set of tests that measure both qualitative and quantitative performance of an algorithm. While the quantitative tests evaluate how successful the algorithm is in solving a task, our qualitative tests provide a measure of *plan interpretability*, which is often desired in practice.

Qualitative evaluation: Visual plans can be inspected by a human to assess their quality. Since human assessment is subjective, we devised a set of questionnaires, and for each domain, we asked 5 participants to visually score 5 randomly generated plans from each model by answering the following questions: (1) *Fidelity*: Does the pixel quality of the images resemble the training data?; (2) *Feasibility*: Is each transition in the generated plan executable by a single action step?; and (3) *Completeness*: Is the goal reachable from the last image in the plan using a single action? Answers were in the range [0,1], where 0 denotes *No* to the proposed question and 1 means *Yes*. The mean opinion scores are reported for each model.

Quantitative In addition to generating visually sensible trajectories, a planning algorithm must also be able to successfully navigate towards a predefined goal. Thus, for each domain, we selected 20 start and goal images, each with an obstacle configuration unseen during training. Success was measured by the ability to get within some L2 distance to the goal in n steps or less, where the distance threshold and n varied on the domain but was held constant across all models. A controller specified by the algorithm executed actions given an imagined trajectory, and replanning occurred every *r* steps. Specific details can be found in the Appendix.

[4] In practice, we found that exponentiating the SPTM classifier score instead of thresholding worked slightly better, without requiring tuning a threshold. We therefore report results using this method.

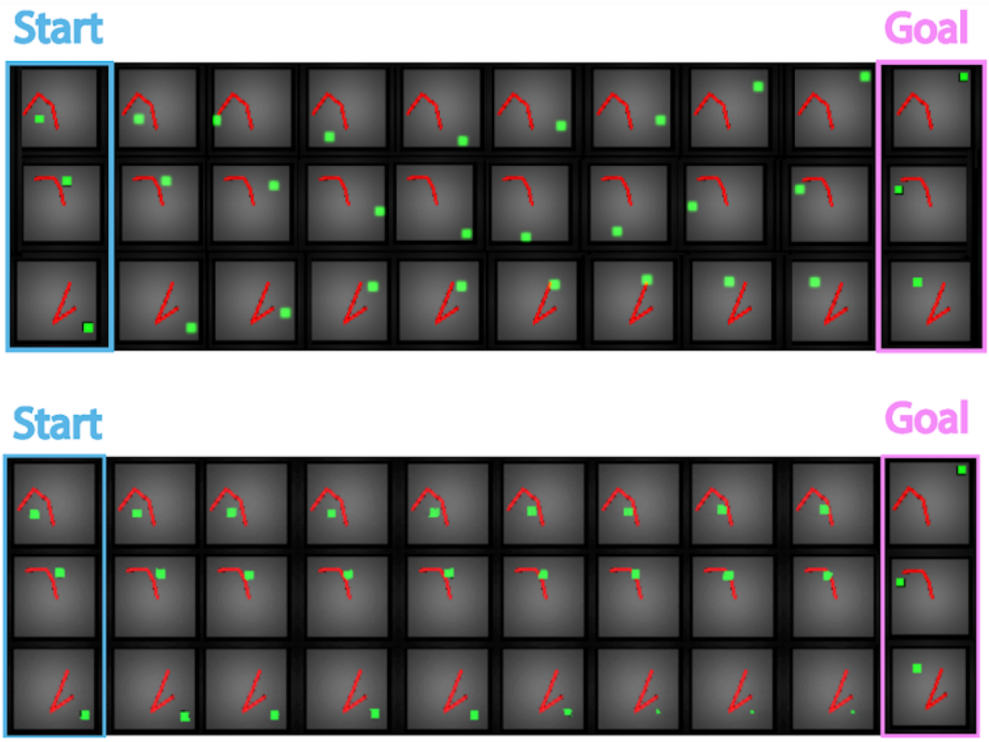


Figure 6.3: HTM plan examples (top 3 rows) and Visual Foresight plan examples (bottom 3 rows). Note Visual Foresight is unable to conduct a long-horizon plan, and thus greedily moves in the direction of the goal state using green pixel distance cost.

Results on Block Domains

As shown in Table 6.1, HTM outperforms all baselines in both qualitative and quantitative measurements across the first two domains. In the simpler block wall domain, Visual Foresight only succeed with the extra domain knowledge of using the green pixel distance. In the complex obstacle domain, Visual Foresight mostly fails to find feasible plans. SPTM, on the other hand, performed poorly on both tasks, showing the importance of our CPC-based edge weights in the graph. **Perhaps the most interesting conclusion from this experiment, however, is that even such visually simple domains, which are simulated, have a single moving object, and do not contain visual distractors or lighting/texture variations, can completely baffle state-of-the-art VP algorithms**. For the complex obstacle domain, we attribute this to the non-trivial geometric information about the obstacle shape that needs to be extracted from the context and accounted for during planning. In comparison, the real-image domains of [57], which contained many distractors, did not require much information about the shape of the objects for planning a successful pushing action.

In regards to perceptual evaluation, Visual Foresight generates realistic transitions, as seen by the high participant scores for feasibility. However, the algorithm is limited in creating a visual plan within the optimal $T = 15$ timesteps consistent with that of [57]. Thus, when confronted with a challenging task of navigating around a concave shape where the number of timesteps required exceeds T, Visual Foresight fails to construct a reliable plan (see Figure 6.3), and thus lacks plan

Algorithms	Domain	Fidelity	Feasibility	Completeness	Success
HTM	1	**0.75 ± .09**	**0.88 ± .14**	**1.00 ± .00**	**95%**
	2	**0.96 ± .03**	**0.96 ± .08**	**0.96 ± .08**	**100%**
SPTM with CVAE	1	0.40 ± .11	0.00 ± .00	1.00 ± .00	55%
	2	0.92 ± .07	0.00 ± .00	1.00 ± .00	30%
Visual Foresight [57] (pixel MSE loss)	1	0.74 ± .08	0.84 ± .16	0.04 ±. 08	25%
	2	0.59 ± .16	0.64 ± .21	0.00 ± .00	0%
Visual Foresight [57] (green pixel distance)	1	0.80 ± .07	0.84 ± .16	0.04 ± .08	90%
	2	0.69 ± .14	0.56 ± .21	0.00 ± .00	35%
Inverse Model	1	-	-	-	20%
	2	-	-	-	25%

Table 6.1: Qualitative and quantitative evaluation for the the block wall (1) and block wall with complex obstacle (2) domains. Qualitative data also displays the 95% confidence interval.

completeness. Conversely, SPTM is able to imagine some trajectory that will reach the goal state. However, as mentioned above and was confirmed in the perceptual scores, SPTM fails to select feasible transitions, such as imagining a trajectory where the block will jump across the wall or split into two blocks. Our approach, on the other hand, received the highest scores of fidelity, feasibility, and completeness. Finally, we show in Figure 6.6 the results of our two proposed improvements to SPTM in isolation. The results clearly show that a classifier using contrastive loss outperforms that which uses Binary Cross Entropy (BCE) loss, and furthermore that the inverse of the score function for edge weighting is more successful than the best tuned version of binary edge weights through thresholding – 0 means no edge connection and 1 means an edge exists.

Algorithms	Difficulty	Success
HTM	**Easy**	**100 %**
	Hard	**70 %**
Visual Foresight	Easy	60 %
	Hard	10 %
Inverse Model	Easy	90 %
	Hard	30 %

Table 6.2: Quantitative evaluation for block insertion domain. Visual Foresight [57] was trained using pixel MSE loss.

Results for Insertion and Manipulation Domains

Unlike previous block domains, we demonstrate the zero-shot generalization ability of our approach by varying the shape and volume of the moving object itself. The challenge in planning is accounting for the orientation of a novel shape when encountering obstacles, and figuring out the best angle at

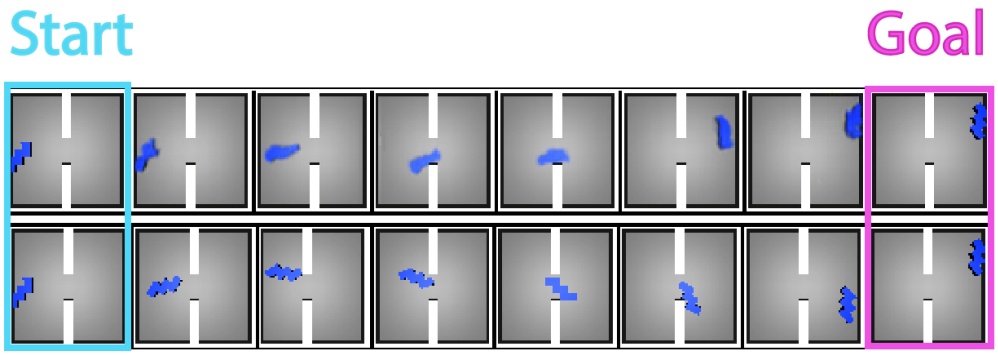


Figure 6.4: HTM plan and execution. The top row demonstrates a generated visual plan on an unseen block configuration, and the bottom displays the execution to follow the plan.

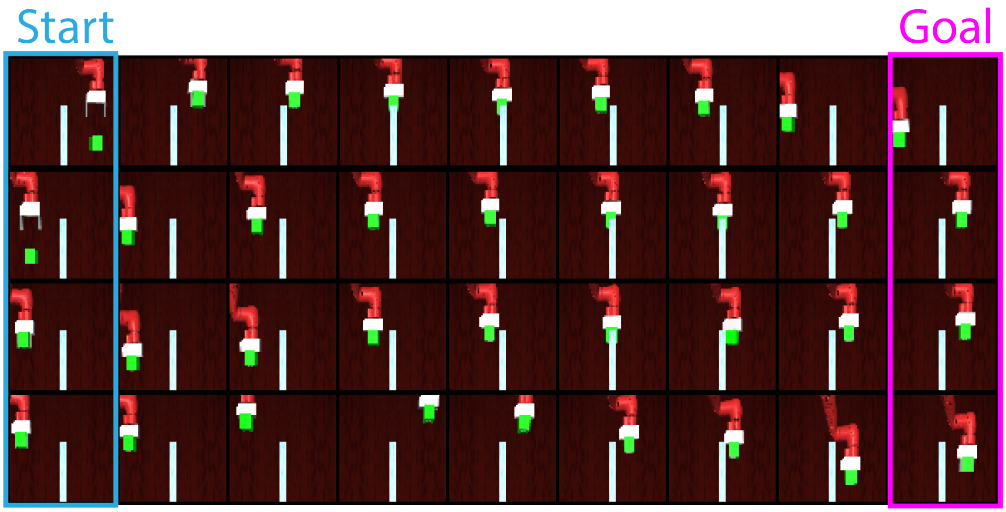


Figure 6.5: HTM plans on real data. The agent plans to grab the green block and/or go around the obstacle with goal directed planning (no reward signal) using dataset $\mathcal{D}$ on unseen starts and goals.

which to approach a narrow passageway. We emphasize that **such geometrical reasoning must be learned from the data, and must generalize to unseen shapes**.

In practice, it might be very difficult to extract a context vector describing the environment every time the environment changes. In this domain, instead, we show that conditioning on a random image in that environmental configuration is sufficient to generate possible image samples in the same configuration, and therefore lead to successful plans. At test time, we can simply use the current image as context.

For testing, we differentiated between 'easy' tasks (ie. block stays on the same side of the wall) and 'hard' tasks (ie. block must pass through the opening). Each task had 10 random start/goal locations, and all configurations were unseen. As seen in Table 6.2, our method is successfully able to tackle these challenges, and an example plan and execution can be found in Figure 6.4. While successful on the majority of the 'easy' tasks, Visual Foresight proved unable to plan the rotations necessary to move the block through the opening, and thus failed on most of the 'hard' tasks.

In addition, we applied HTM to robotic simulation of a Sawyer robot arm as seen in Figure 6.5

in which the robot needs to move a the green block to the desired location around a wall. We collect 45,000 samples of random interaction when the arm holds the green block, and 5,000 samples when the arm moves without the block. Here, we do not have different contexts, but we evaluate on unseen starts and goals. We apply HTM planning ability directly on real images from the replay buffer achieving feasible plans 12 out of 14 test tasks. We find that our visual plans avoid myopic behavior by planning to going around the thin wall, and preferring to grab the block before moving to the goal.

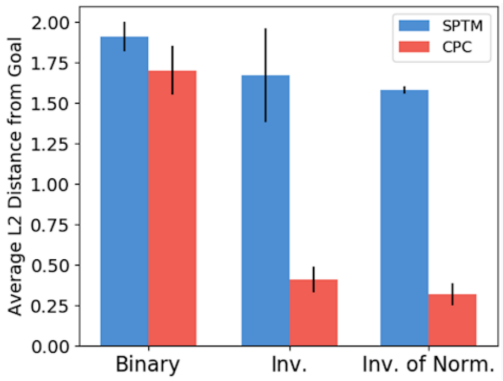


Figure 6.6: Ablation study on weight functions. We show gain in using our proposed score function and weighting function comparing to those proposed in the original SPTM by examining final *average distance* to the goal state for 10 test start/goal pairs on block with complex obstacle domain (the lower the distance, the better). For the score function, we denote our proposed energy model structured with contrastive loss as *CPC* and the classifier as proposed in [228] with BCE loss as *SPTM*. For the edge weighting function, we test the binary thresholding from the original SPTM paper, our proposed inverse of the score function, and our proposed inverse of the normalized score function.

6.5 Discussion

We proposed a simple visual planning method that plans directly in image space, and generalizes in a zero-shot by hallucinating possible images conditioned on a domain context. On a suite of challenging visual planning domains, we find that our method outperforms state-of-the-art methods, and is able to pick up non-trivial geometrical information about objects in the image that is crucial for planning.

Our results further suggest that combining classical planning methods with data-driven perception can be helpful for long-horizon visual planning problems, and takes another step in bridging the gap between learning and planning. In chapter 9, we demonstrate discrete, factorized representation learning and planning in order to efficiently handle more multi-object problem.

Part III

Long-Horizon Tasks

Chapter 7

Hierarchical Model-based Reinforcement Learning

An intuitive understanding of the mechanics of the world provides humans and animals with a powerful tool to rational decision making. For an artificial agent, this understanding can be captured by a predictive model that is used to plan sequences of actions that optimize a desired objective. This is the basis of model-based reinforcement learning (MBRL). Because the model is typically agnostic to the objectives or the environments, MBRL has been demonstrated to generalize to test environments and objectives that are not seen during training. Additionally, learning the physical models has been shown as a way to bring learning to real physical system such as robot manipulation [67]. Despite its promise, when faced with temporally extended or sparse reward tasks, standard model-based algorithms tend to exploit the inaccuracy of the model prediction over time, resulting in poor performance, especially when predicting high-dimensional observations such as images. As a result, prior methods are limited to simple and temporally short planning tasks to best perform only 10-20 lookahead steps [57, 91, 115]. In this chapter, we aim to develop a method that extends the predicting and planning capability of visual MBRL to long-horizon tasks.

One approach that has been explored extensively in the *model-free* reinforcement learning literature to address long-horizon tasks is to employ a hierarchy, where a higher-level policy specifies temporally extended subgoals, and a lower-level policy aims to achieve those subgoals. Can a hierarchical decomposition also enable model-based RL methods to scale to longer horizons? A naïve approach is to build a hierachical model and planning at multiple levels of abstraction. However, if the high-level model must also predict the outcome of action sequences, its prediction problem may not be any easier than the original prediction problem, which may limit performance gains. The key insight of this chapter is to employ the high-level model to predict the outcome of a closed-loop controller.

Inspired by [188], we propose Hierarchical Visual Planning and Control (HVPC). We hypothesize the outcome of a closed-loop controller is easier to predict than the consequence of a series of actions. Building on this insight, HVPC consists of two model-based planning layers, each based on visual MPC [67, 57]. The low-level visual MPC layer aims to take action in the environment to achieve a given subgoal. The high-level visual MPC layer plans high-level actions, which are

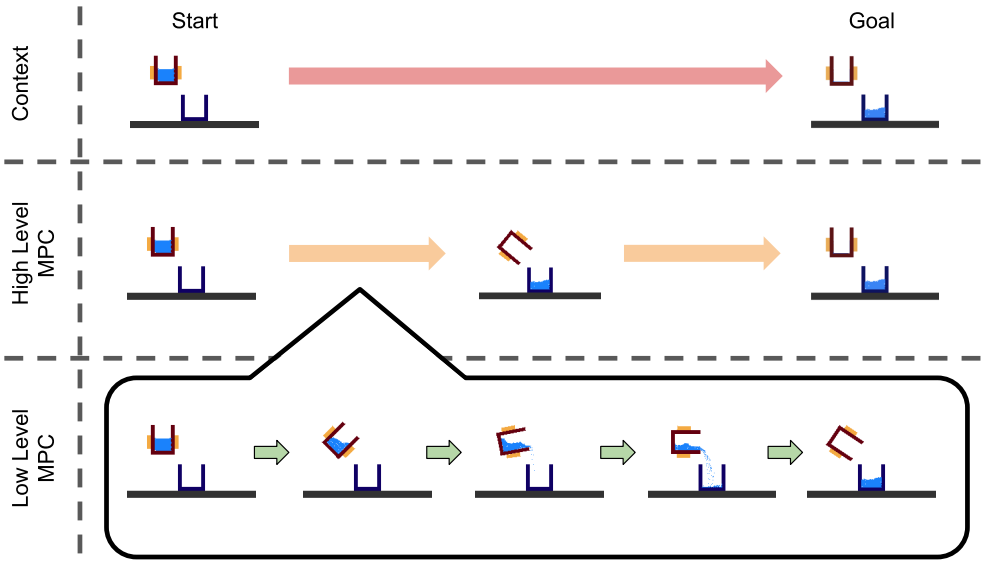


Figure 7.1: Hierarchical Visual Planning and Control. The high-level planner finds the lowest cost predicted trajectory through CEM. Using the first subgoal, the low-level visual MPC performs a closed-loop control to this subgoal. Once that has been completed, the high-level planner may replan to the goal, and the process repeats until the goal has been reached.

subgoal images for the low-level planner (see Figure 7.1). The objective of the high-level predictive model is to predict the actual outcome of the low-level controller given the current and goal state observations. During training, the data is first collected from self-supervised interaction, random motion that perturbs the system, for the low-level video prediction model. The agent then collects more data from the environment by repeatedly attempting a random goal from its current state using the low-level MPC when it predicts to reach the goal; otherwise it samples a new goal. These data are then used to train the high-level video model which acts as a *reachability* model for optimizing a high-level visual plan. At test time, when presented with unseen start and goal images, the high-level planner plans a sequence of subgoals to accomplish the task, and the low-level visual MPC controller executes the first subgoal and replans (see Figure 7.2).

The key contribution in this chapter is a framework for hierarchical planning and control that enables long-horizon reasoning with raw image observations. Our approach learns a two-level hierarchy of predictive models, which operate over varying levels of temporal abstraction, and plans with this hierarchical model for accomplishing challenging tasks. Additionally, our framework is largely self-supervised by the agent collecting the data without a goal in mind or with a goal that is sampled from the past data. On simulated visual block navigation and multi-cup pouring tasks, our approach achieves 1.4 to 2.1 times the success rates compared to Visual Foresight [57] and Hierarchical Visual Foresight [188].

7.1 Related work

In recent years, deep reinforcement learning (RL) has made progress in enabling agents to operate autonomously from raw sensory inputs [177]. However, enabling agents to flexibly perform a variety of long-horizon tasks remains a major challenge. Most existing approaches to long-horizon tasks either learn one narrow task often with considerable reward and data supervision [241, 177, 270], or require significant domain-specific abstractions such as hierarchical RL methods [130, 181, 8, 71]. In this work we instead aim to develop a scalable approach for learning long-horizon tasks that requires minimal human supervision.

Goal-conditional model-free RL methods [112, 190, 187, 186] have attempted to extend model-free RL to learn broader sets of tasks typically done by learning a universal Q-function [230] with off-policy RL algorithms. While promising in handling visual observations, most of the work are still limited to temporally-limited or pushing tasks. Our work instead builds on model-based RL framework which can benefit from non-goal-directed data and data efficiency.

Visual model-based RL has also shown promise in bringing robot learning to real world [67, 57] by learning a predictive model from past interaction data. These methods typically build on video prediction models [66, 15, 141, 194, 91]. While the prediction can be used for planning optimal action sequences, when predicting over many time steps into the future the predicted frames become blurry and less informative. Instead of directly improving the models, we attempt to extend the planning horizon by employing hierarchy in which the high-level model learns the capability of the low-level controller. Our work is orthogonal and can benefit from the improvement in the quality of video models. Recently, Nair and Finn [188] propose to optimize for a sequence of reachable subgoals by minimizing the estimated cost from the video model, and then use the video model for control actions. Inspired by this approach, we propose to search for these subgoals by learning a forward model that predicts a sequence of frames that the low-level model reaches. By explicitly representing the high-level model, HVPC can improve its subgoal planning, incorporate temporal dependencies, and bringing down subgoal optimization time from a few hours to a few seconds. We also show further improvement in success rate in both maze navigation and multi-cup pouring tasks.

Classical planning methods such as task and motion planning [113] have shown impressive results in solving goal-directed temporally-extended tasks. However, these approaches require expert-designed models with low-dimensional states. To extend to high-dimensional inputs, more recent work in visual planning combines classical planning with representation learning. One way to do so is to learn a latent space that is *suitable* for shortest-path search [135, 272, 13], motion planning [104], or optimal control [279]. However, such approaches have not shown for temporally extended planning. Alternatively, one can maintain a graph-based memory [228, 60, 154] for planning a sequence of visual observations bringing the start to the goal and learn a low-level controller for visual servoing. However, because of their nonparametric, static nature, these methods can be limited by scaling, and challenging to generalize to unseen tasks. In practice, these methods also requires extensive manual tuning.

7.2 Problem Formulation and Notation

We define the set of environments $\mathcal{E}$ and the distribution over these environments $\mathbf{P}_{\mathcal{E}}$. Each environment $\xi \in \mathcal{E}$ consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, and a transition probability $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{S}$. Similar to [67], we assume that the agent can collect a diverse dataset by random interactions or designed structured explorations. During training time the agent is allowed to interact with the environments without provided goals. At test time, a goal-conditional cost function $\mathcal{C} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, a discount factor γ, a test environment $\xi_{test} \sim \mathbf{P}_{\mathcal{E}}$ and a goal state $s_g \in \mathcal{S}_{\xi_{test}}$ are provided. Our objective is to find a controller that minimizes the sum of discounted cost $J = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{C}(s_t, s_g)\right]$, where $s_t, s_g \in \mathcal{S}$. An optimal controller under this cost function must result in the agent reaching the goal state s_g. In this work, we emphasize the problems in which the cost function is sparse, i.e., the cost is smaller only when s_t is close to s_g, and, otherwise, is a constant. However, our work still holds with an arbitrary cost function.

In the MBRL framework, optimizing over the infinite horizon is in fact challenging due to accumulated prediction error. In particular, in model predictive control (MPC), a closed-loop controller optimizes for an action sequence that minimizes the sum of discounted cost under a finite planning horizon T and the learned transition function $f_\theta : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{S}$ that regresses the true transition function from past transition data which are pre-collected from self-supervised interaction. Mathematically,

$$a_1^*, ..., a_T^* = \underset{a_1,...,a_T}{\arg\min}\, \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \mathcal{C}(s_t, s_g)\right], \tag{7.1}$$

where $s_t, s_g \in \mathcal{S}$ and $f_\theta(s_t, a_t) = s_{t+1}$. A common approach to solve this optimization problem is cross-entropy method (CEM). CEM randomly samples action sequences, picks the top k lowest cost sequences, and uses them to approximate the action distribution for the next iteration. This controller can be used closed-loop by replanning for a new sequence of actions every few steps in the environment. We call a task *temporally extended* or *long-horizon* if it requires the planning horizon T to be large to be able to approximate the infinite-horizon optimal controller even if the true transition dynamics is given in equation 7.1.

In this chapter, we assume that the agent only perceives image inputs which we refer to using the s notation, and therefore the environment is only partially observed. We learn a predictive model implemented using deep recurrent neural networks on the image space which is referred to as a video prediction model. With images as inputs, we also refer to a class of planning and control algorithms above as Visual MPC [57].

7.3 Hierarchical Visual Planning and Control

While visual MPC has shown promise in its ability to generalize to unseen tasks in various domains such as autonomous vehicles and robot manipulation [40], predicting more than a dozen of frames remains a challenge limiting its applications to simple short-horizon tasks such as object pushing. When these video models predict many time steps into the future, the prediction error tends to

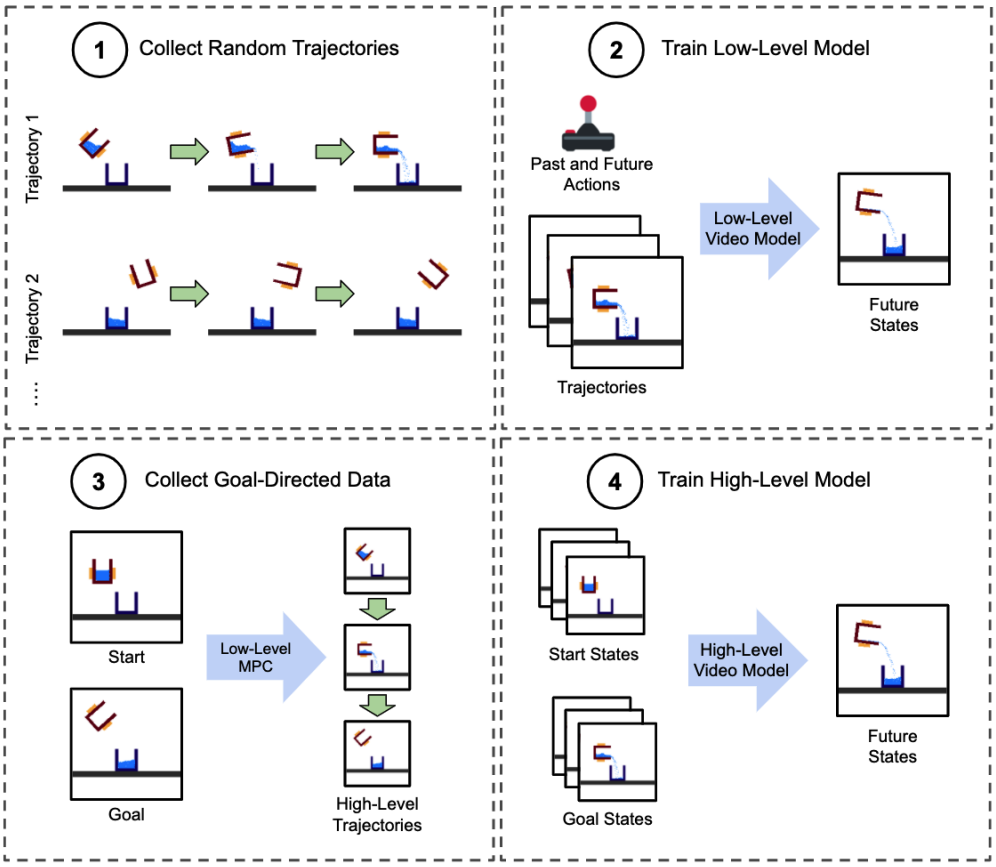


Figure 7.2: Low-level and high-level model training. In the first stage, the agent randomly perturbs the system to collect self-supervised data. These data are then used to train low-level video model to predict future frames from past frames and actions. In goal-directed data collection phrase, the agent uses closed-loop low-level controller to sequentially attempt goals sampled from the goal distribution. This generates trajectories in which actions are proposed goal images and consecutive states are reachable by the controller. Using these data, we train a high-level video model.

accumulate. This inaccuracy causes the prediction to become blurrier over time and hampers the agent's ability to solve tasks . In practice, the most commonly used number of predicted frames is between 10-20 [57]. One stop-gap solution is to learn a temporally-abstracted model that predicts every k frames along with a short-horizon controller that solves k-step planning problem. However, the high-level prediction problem is not necessary any simpler to optimize or more accurate to predict, thus we may not get any gain over regular next-step models. In this section we ask the key question: *how can we learn a useful temporally-abstracted model?*

In this chapter, we hypothesize that predicting *the outcome of the low-level closed-loop controller* is easier than predicting *the outcome of the sequence of actions*. Based on this insight, we propose Hierarchical Visual Planning and Control (HVPC). HVPC employs high-level and low-level models that predict at different timescales in which the high-level model takes in the goal image of the low-level model and predict the outcome of its execution. In the following subsections, we detail our method.

Low-Level Controller

The first component for taking action in the environment is the low-level controller. A careful choice of the low-level controller is required such that it is able to robustly execute various short-horizon tasks. Visual MPC is a suitable choice well because of its robustness and its advantage in terms of flexibly when applied to unseen goals. To train the video model, we first collect a trajectory dataset $\mathcal{D}_{ll}$ by randomly executing actions without any goal in mind. Using this data, we train a low-level video prediction model to predict future frames, given a sequence of future actions. In our experiments, we use Stochastic Variational Video Prediction (SV2P) [15].

High-Level Planner

A high-level planner requires a temporally-abstracted model for reasoning over long-horizon tasks. To train such model, instead of using the data from the low-level training, we propose to collect goal-directed data executed by the low-level controller. These data allow us to exploit more predictable behavior of the closed-loop controller rather than random interactions. More precisely, the agent at state s_t^k first randomly samples a goal image g_t^k from the past data in the same environment or from the state sampler described in section 7.3. Then, the low-level MPC executes actions in the environment toward the sampled goal resulting in the next state s_{t+1}^k; we then repeat this procedure. This trajectory data can then be used for training standard action-conditional video prediction models by viewing goal images as actions.

In the sparse cost settings, randomly attempting goals that are might not be meaningful and can result in highly stochastic behaviors. With this insight, we propose to only attempt the goals when the are within the reach predicted by the low-level video model and stay put otherwise. By only attempting reachable goals, the high-level model can also be seen as a reachability model given the low-level skills. We detail the training procedure in Algorithm 4, and the architecture of the high-level model in the supplementary materials.

Planning and Execution

Instead of optimizing over the long horizon T, the low-level controller chooses a shorter horizon H while still optimizing over the objective in Equation 1. Now, the high-level model learns an H-step transition function $h_\theta : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ that takes in the current state and the goal and outputs the final state after the low-level controller executes for h steps. The high-level model can then be used to optimize for a sequence of subgoals using the following objective:

$$s^*_{T_1}, ..., s^*_{T_N} = \underset{s_{T_1},...,s_{T_N}}{\arg\min} \sum_{i=1}^{N} \gamma^t C(s_{T_i}, s_g), \tag{7.2}$$

where $T_i = iH$, N is the number of subgoals, and $s_{T_{i+1}} = h_\theta(s_{T_i}, \tilde{s})$ for some state $\tilde{s}$. However, optimizing for this objective in image space can be challenging, so we propose to optimize over the latent space which is given by the state sampler, similarly to [188]. We use CEM in the same

manner as done in the low-level controller. Once an optimal sequence of subgoals $s_{T_1}, ..., s_{T_N}$ are found, the low-level model can perform visual servo to the next subgoal.

To sumarize, given a new goal state, HVPC (1) optimizes the objective in Equation 2 for subgoals (high-level visual MPC), (2) runs low-level visual MPC with horizon H to the first subgoal s_{T_1}, and (3) replans until having reached the goal.

State Sampler

As alluded to in previous sections, the state sampler serves as a glue to help the high-level model reason about the low-level ability and for high-level to perform planning at execution time. The key requirements are for the state sampler to have a compact latent embedding for CEM optimization and to be able to sample possible states given another state in the same environment. While there are many ways to implement this, we choose to simply hallucinate a random goal, similar to a conditional generative model such as CVAE [244].

7.4 Experiments

To test visual planning capability, we evaluate HVPC on visual maze 2D navigation environment[1] from the simulation in Nair and Finn [188] which runs on the MuJoCo physics engine [263], and multi-cup pouring domain, which is generated from the Kitchen2D library [277] and simulated using the Pygame engine. In both domains, the observed state corresponds to images. Using these control suites, we test HVPC and other state-of-the-art visual model-based RL algorithms such as Visual Foresight [67, 57] and Hierarchical Visual Foresight [188] on test initial and goal configurations to measure their generalization abilities. For more details on the experimental set-up, see Appendix F.

Maze navigation

In maze navigation, the green block can be put in one of the 3 partitions (left, middle, right). Each partition is separated by a thin wall with a small slit allowing the block to navigate through. At each time step, a 2-dimensional action is taken to move the green block by a small amount generating a new image observation. Inspired by [188], we test our algorithm on easy, medium, and hard tasks in which the start and goal configurations are in the same partition, 1 partition apart, and 2 partition apart accordingly. The size of the maze is 0.3×0.3 and we declare the task *successful* if current position is within 0.05 from the goal which translates to the green block in the current frame has an over lap with that in the goal image. The task reward is sparse with the L2 distance in pixel space. The number of environment steps allowed is 200 for all methods.

We find that HVPC significantly outperforms HVF and VF in hard tasks (see Figure 7.4). Compared to HVF, our method is much more efficient computation time since HVF suffers from

[1] With one small difference in time period (dt) between steps. In our work, the environment has half of the original step size of that in Nair and Finn [188] requiring the horizon needed for planning to be even longer.

Algorithm 4 HVPC Training

input: environment distribution $\mathbf{P}_{\mathcal{E}}$, goal-conditioned cost function $\mathcal{C}$
output: low-level video model f_θ, high-level video model h_θ, and state samplier g_θ
// Initialization
Initialize empty datasets $\mathcal{D}_{ll}, \mathcal{D}_{hl}$, video models f_θ, h_θ, and a conditional generative model g_θ
Sample environments $\mathcal{E}_1, ..., \mathcal{E}_N$ from $\mathbf{P}_{\mathcal{E}}$

// Collect random interaction data without specific goals
for $i = 1$ **to** N **do**
  Collect trajectories $\mathcal{D}_{ll}^i = \{\{(s_t^k, a_t^k, s_{t+1}^k)\}_{t=0}^{T_{ll}-1}\}_{k=0}^{K_{ll}}$ in $\mathcal{E}_i$ by executing random actions
  Add $\mathcal{D}_{ll}^i$ to $\mathcal{D}_{ll}$
end for
Train f_θ to predict future frames on using trajectories in $\mathcal{D}_{hl}$
Train g_θ to predict a random frame from the same environment given the current frame using $\mathcal{D}_{hl}$

// Collect execution data from attempting random goals
for $i = 1$ **to** N **do**
  for $k = 0$ **to** $K_{hl} - 1$ **do**
    Reset to initial state s_0^k in $\mathcal{E}_i$
    for $t = 0$ **to** $T_{hl} - 1$ **do**
      Sample a random goal u_t^k from g_θ or $\mathcal{D}_{ll}^i$
      Optimize for a_1^* in Equation 7.2 under $\mathcal{C}(\cdot, u_t^k)$ using f_θ
      if optimal cost J^* < threshold ϵ **then**
        Execute a_1^* in $\mathcal{E}_i$ resulting in s_{t+1}^k
      else
        $s_{t+1}^k \leftarrow s_t^k$
      end if
      Append $(s_t^k, u_t^k, s_{t+1}^k)$ to $\mathcal{D}_{hl}$
    end for
  end for
end for
Train h_θ to predict future frames using $\mathcal{D}_{hl}$

nested CEM loops. We find that the planning time is reduced from 1-3 hours in HVF to a few seconds in HVPC. As a result, HVPC can benefit further from high-level replanning at execution time because of its fast planning computation. We find that HVPC with high-level replanning can significantly improve the success rate. In hard tasks, we find that more number of subgoals improves the performance. An example of successful plans shown in Figure 7.3. Here the agent plans a sequence of subgoals execute the first subgoal and replan.

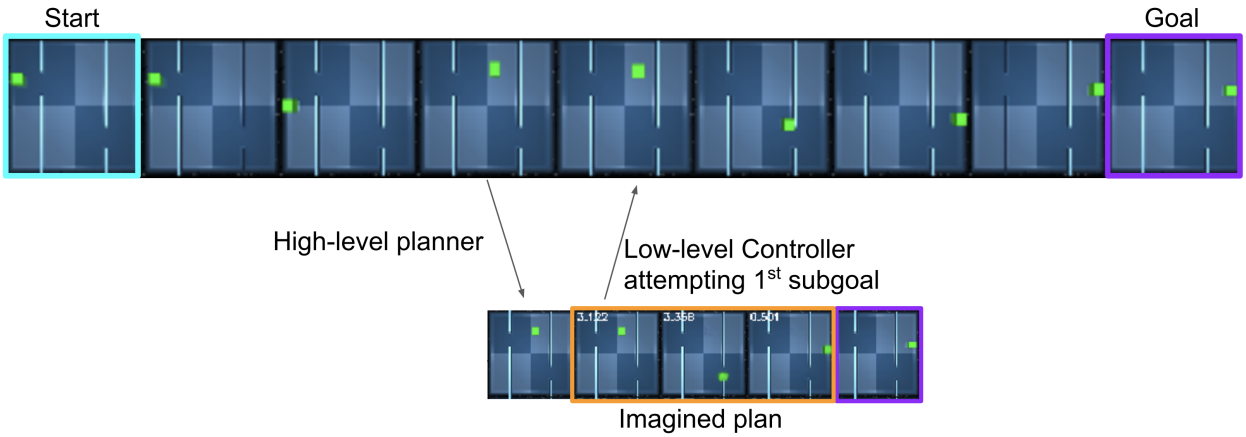


Figure 7.3: HVPC Maze Navigating Execution with Replanning. This illustrates (top row) the low-level model executed trajectory, and the high-level 3-step plan.

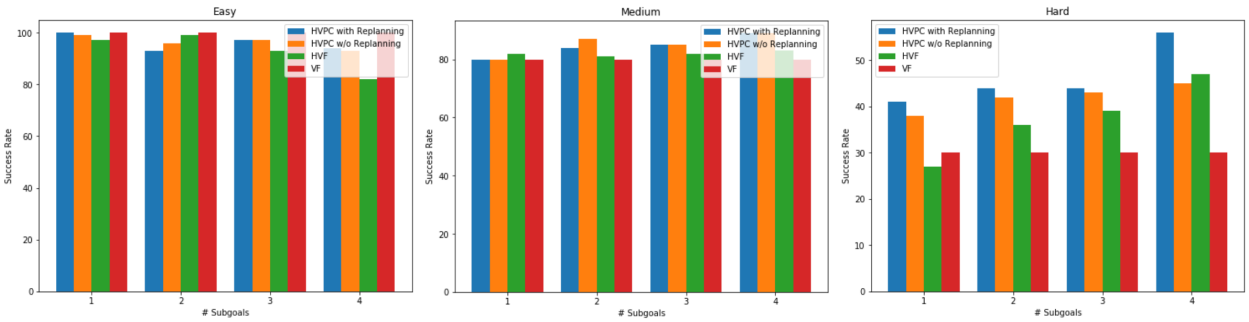


Figure 7.4: HVPC Maze Navigating Execution Success Rate on 100 unseen easy, medium, and hard tasks. We vary the number of subgoals used in each method (VF is not affected). Our method significantly outperforms HVF in hard tasks.

Pouring

In pouring tasks, the agent holds a cup with full amount of water. The agent's objective is to pour water into the cup given the target goal image (see Figure 7.5). The cost function is defined as the blue pixel L2 distance matching the goal image. We note the difficulty of the task that the agent gets signal only when some water is in the goal cup. The task achieved when the water level in the target cup is at least 50%, evaluated using the ground-truth state of the particles (which is not available for any of the algorithms).

Following the procedure in the Maze Navigation domain, we use the same low-level model for VF, HVF, and HVPC. We find that HVF plans suffer from maintaining the temporally consistent water level in each of the standing cup (see Appendix F).

In Table 7.1, we find that HVPC also shows large performance gains over prior methods. Notably, HVPC achieves 80% success and shows a 30% absolute improvement over HVF with one subgoal, and a 32% absolute improvement over visual MPC. We also include qualitative results in Figure 7.5, which shows an example of both a visual plan that HVPC identifies, as well a trajectory that executes that plan using the low-level controller. Finally, in Table 7.2, we compare performance with varying numbers of subgoals. We find that HVPC also improves with more subgoals.

	Pouring
HVPC	**80%**
HVPC-NR	70%
HVF-2SG [188]	30%
HVF-1SG [188]	50%
Visual MPC [57]	48%

Table 7.1: Success rates for all methods for in pouring task. Each algorithm is evaluated on 100 unseen test tasks. We find that HVPC outperforms prior visual planning approaches by a significant margin.

#SG	Water Amount HVPC	HVPC-NR	VF
1	0.55 ± 0.34	0.54 ± 0.37	0.43 ± 0.41
2	0.55 ± 0.22	0.54 ± 0.23	
3	0.60 ± 0.31	$\mathbf{0.66 \pm 0.33}$	
4	$\mathbf{0.65 \pm 0.29}$	0.63 ± 0.32	

Table 7.2: Number of Subgoals (#SG) Ablation. This shows the ratio of water particles ended inside the target cup when varying the number of subgoals. The results are evaluated from the their true states (not available to the algorithm) on 100 pouring test tasks. We also observe that HVPC improves when using more subgoals.

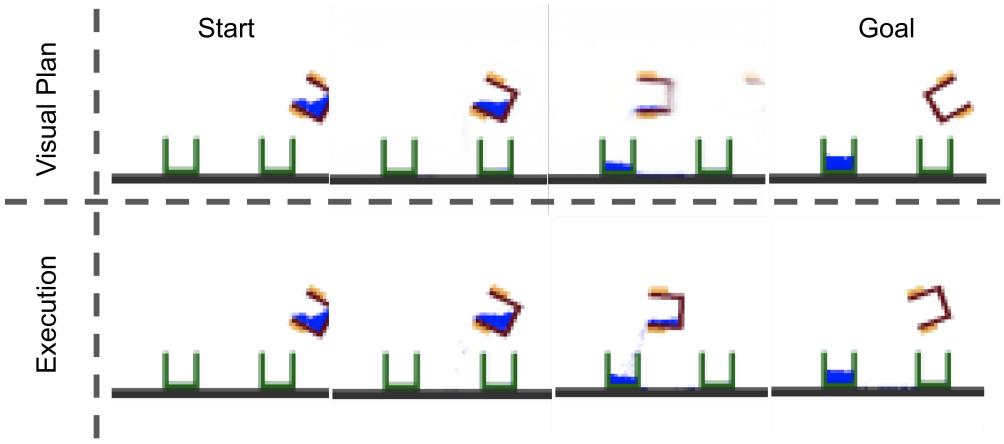


Figure 7.5: HVPC Pouring Execution without Replanning. The high-level model plans 2 subgoals (top row) and the low-level model executes by following the plan (bottom row).

7.5 Conclusion

We proposed a hierarchical, two-staged self-supervised framework that extends the capability of visual model-based reinforcement learning to long-horizon tasks while generalizing to unseen tasks. By proposing reliable subgoals, HVPC outperforms the state-of-the-art model-based reinforcement learning in long-horizon, goal-directed tasks while using much less expensive computation resources. In the next chapter, we aim to further reduce the computation required by asking the question what abstract nodes are important to keep in a graphical memory for long-horizon planning.

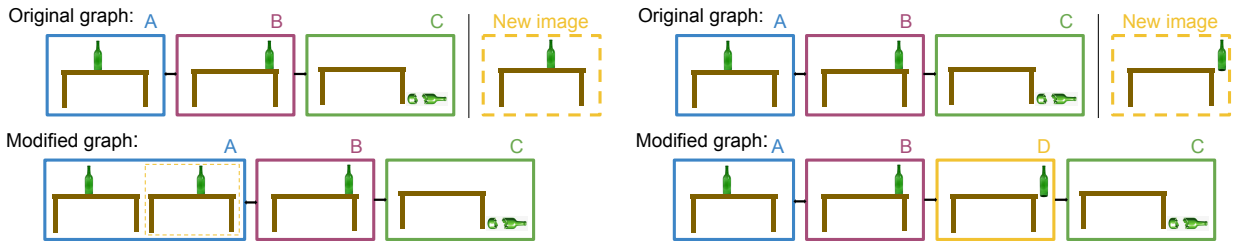
Chapter 8

Sparse Graphical Memory

Learning-driven approaches to control, like imitation learning and reinforcement learning, have been quite successful in both training agents to act from raw, high-dimensional input [177] as well as to reach multiple goals by conditioning on them [9, 187]. However, this success has been limited to short horizon scenarios, and scaling these methods to distant goals remains extremely challenging. On the other hand, classical planning algorithms have enjoyed great success in long-horizon tasks with distant goals by reduction to graph search [94, 140]. For instance, A* was successfully used to control *Shakey the robot* for real-world navigation over five decades ago [51]. Unfortunately, the graph nodes on which these search algorithms operate is abstracted away from raw sensory data via domain-specific priors, and planning over these nodes assumes access to well-defined edges as well as a perfect controller to move between nodes. Hence, these planning methods struggle when applied to agents operating directly from high-dimensional, raw-sensory images [173].

How can we have best of both worlds, *i.e.*, combine the long-horizon ability of classic graph-based planning with the flexibility of modern, parametric, learning-driven control? One way is to build a graph out of an agent's experience in the environment by constructing a node for every state and use a learning-based controller (whether RL or imitation) to move between those nodes. Some recent work has investigated this combination in the context of navigation [227, 59]; however, these graphs grow quadratically in terms of edges and quickly become unscalable beyond small mazes [59]. This strategy either leads to extremely brittle plans because such large graphs contain many errors (infeasible transitions represented by edges), or relies on human demonstrations for bootstrapping [227].

In this chapter, we propose to address challenges in combining the classical and modern paradigms by dynamically sparsifying the graph as the agent collects more experience in the environment to build what we call *Sparse Graphical Memory* (SGM), illustrated in Figure 8.2. In fact, building a sparse memory of key events has long been argued by neuroscientists to be fundamental to animal cognition. The idea of building cognitive topological maps was first demonstrated in rats by seminal work of [264]. The key aspect that makes building and reasoning over these maps feasible in the ever-changing, dynamic real world is the sparse structure enforced by landmark-based embedding [72, 274, 79]. Yet, in artificial agents, automatic discovery of sparse landmark nodes remains a key challenge.



(a) Node merging example (b) Node creation example

Figure 8.1: Examples where two-way consistency (TWC) merges nodes (left) and creates a new node (right). Consider a directed graph with three nodes connected as $A \leftrightarrow B \rightarrow C$. Given a new image in the dashed yellow box, should it be merged into an existing node or a new node be created? (a) On the left, we can merge the new image with node A safely. (b) On the right, the new image D contains a bottle which is about to fall off the table edge. While B is perceptually similar to the new image, the agent cannot move the bottle from D to A, but it can move from B to A. As we cannot transition to neighbors in the same manner, our TWC criterion is not satisfied and a new node is created.

One way to discover a sparse graph structure is to dynamically merge similar nodes. But how does one obtain a similarity measure? This is a subtle but central piece of the puzzle. States that look similar in the observation space may be far apart in the action space, and vice-versa. Consider the example in Figure 8.1, where the graph already contains 3 nodes $\{A, B, C\}$. In 8.1a, similar looking nodes can be merged safely. While in 8.1b, although the new node D is visually similar to B, but merging with B would imply that the bottle can be saved from breaking. Therefore, a purely visual comparison of the scenes cannot serve as a viable metric. We propose to use an asymmetric distance function between nodes and employ *two-way consistency* (TWC) as the similarity measure for merging nodes dynamically. The basic idea is that two nodes are similar if they both can be reached with a similar number of steps from all their neighbors as well as if all their neighbors can be reached from both of them with similar effort. For our conceptual example, it is not possible to go back from the falling-bottle to the standing-bottle, and hence the two-way consistency does not align for scene B and the new state. Despite similar visual appearance, they will not be merged. We derive two-way consistency as an extension of prior Q-function based aggregation criteria to goal-conditioned tasks, and we prove that the sparse graphs that result from TWC preserve (up to an error factor that scales linearly with the merging threshold) the quality of the original dense graphs.

We evaluate the success of our method, SGM, in a variety of navigation environments. First, we observe in Table 8.1 that SGM has a significantly higher success rate than previous methods, *on average increasing the success rate by 2.1x* across the environments tested. As our ablation experiments demonstrate, SGM's success is due in large part to its sparse structure that enables efficient correction of distance metric errors. In addition, we see that the performance gains of SGM hold across a range of environment difficulties from a simple point maze to complex visual environments like ViZDoom and SafetyGym. Finally, compared to prior methods, planning with our proposed sparse memory can lead to nearly an order of magnitude increase in speed (see Appendix G.5).

8.1 Related work

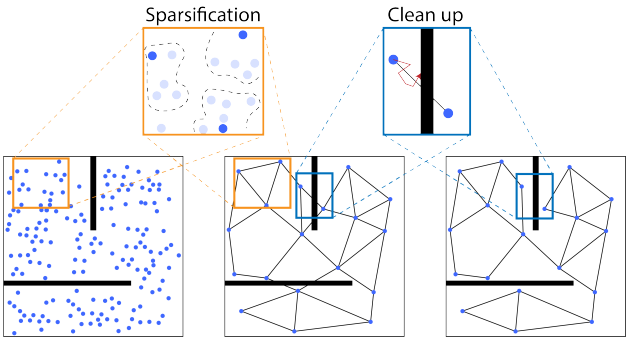


Figure 8.2: Illustration of Sparse Graphical Memory (SGM). New states are either merged with existing graph nodes according to our two-way consistency criterion, or a new node is generated. After graph construction, incorrect edges representing infeasible transitions are corrected.

Planning is a classic problem in artificial intelligence. In the context of robotics, RRTs [140] use sampling to construct a tree for path planning in configuration space, and SLAM jointly localizes the agent and learns a map of the environment for navigation [55, 17]. Given an abstract, graphical representation of an environment, Dijkstra's Algorithm [49] generalizes breadth-first search to efficiently find shortest paths in weighted graphs, and the use of a heuristic function to estimate distances, as done in A* [94], can improve computational efficiency.

Beyond graph-based planning, there are various parametric approaches to planning. Perhaps the most popular planning framework is model predictive control (MPC) [77]. In MPC, a dynamics model, either learned or known, is used to search for paths over future time steps. To search for paths, planners solve an optimization problem that aims to minimize cost or, equivalently, maximize reward. Many such optimization methods exist, including forward shooting, cross-entropy, collocation, and policy methods [217, 93]. The resulting agent can either be open-loop and just follow its initial plan, or it can be closed-loop and replan at each step.

Aside from MPC, a variety of reinforcement learning algorithms, such as policy optimization and Q-learning, learn a policy without an explicit dynamics model [178, 152, 238, 237]. In addition to learning a single policy for a fixed goal, some methods aim to learn hierarchical policies to decompose complex tasks [112, 204, 229], and other methods aim to learn goal-conditioned policies able to reaching arbitrary goals. Parametric in nature, these model-free approaches are highly flexible, but, as does MPC with a learned dynamics model, they struggle to plan over long time horizons due to accumulation of error.

Recent work combines these graph-based and parametric planning approaches by using past observations for graph nodes and a learned distance metric for graph edges. Variations of this approach include Search on the Replay Buffer [59], which makes no attempt to sparsify graph nodes; Semi-Parametric Topological Memory [227], which assumes a demonstration to bootstrap the graph; Mapping State Space Using Landmarks for Universal Goal Reaching [103], which subsamples the policy's past training observations to choose graph nodes; and Composable Planning with Attributes [287], which stores abstracted attributes on each node in the graph. Hallucinative Topological Memory (HTM) [154] uses a contrastive energy model to construct more accurate edges, and [240] use dynamic programming for planning with a learned graph.

In contrast to the methods listed above, the defining feature of our work is a two-way consistency check to induce sparsity. Previous work either stores the entire replay buffer in a graph, limiting scalability as the graph grows quadratically in the number of nodes, or it subsamples the replay buffer without considering graph structure. Moreover, Neural Topological SLAM [29] assumes access to a 360° camera and a pose sensor whereas we do not. In [30], the nodes in the graph are manually specified by humans whereas we automatically abstract nodes from the data. In contrast

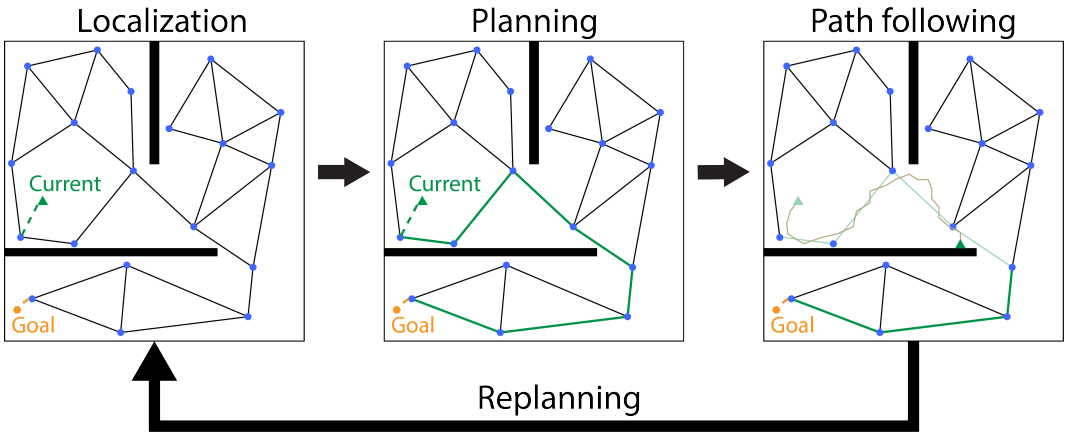


Figure 8.3: Execution using a graphical memory. In localization, the agent finds the closest node using discrepancies in the asymmetric distance function. In planning, the agent uses Dijkstra's algorithm to find the shortest path. (For simplicity, this illustration omits the direction of edges.) In path following, the agent may divergence from the waypoints or experience a transition failure. The agent then needs to correct the memory, relocalize, and replan.

to our method, the work in [169] has no theoretical guarantees, requires trajectories rather than unordered observations, and uses human demonstrations.

Prior work has proposed MDP state aggregation criteria such as bisimulation metrics that compare the dynamics at pairs of states [80, 62] and Utile distiction [167] that compares value functions. Because bisimulation is a strict criterion [151] that would not result in meaningful sparsity in the memory, our proposed two-way consistency criteria adapts approximate value function irrelevance [108] to goal-conditioned settings and high-dimensional observations.

8.2 Preliminaries

We consider long-horizon, goal-conditioned tasks. At test time, an agent is provided with its starting state s_{start} and a goal state s_{goal}, and the agent seeks to reach the goal state via a sequential decision making process. Many visual tasks can be defined by a goal state such as an image of a goal location for navigation. Our task is an instance of undiscounted finite-horizon goal-conditioned RL in which the agent receives a -1 living reward until it reaches a given goal.

To reach distant goals, we use a semi-parametric, hierarchical agent that models feasible transitions with a nonparametric graph (*i.e.* graphical memory) to guide a parametric low-level controller.

Nodes in the graphical memory are prior states encountered by the agent, connected through a learned distance metric. Once the graphical memory is constructed, the agent can plan using a graph search method such as Dijkstra's algorithm [49] and execute plans with a low-level controller learned with reinforcement learning or imitation learning [59, 227]; see Figure 8.3. Therefore, given a final goal, the high-level planner acts as a policy that provides waypoints, nodes along the shortest path, to the low-level controller, and updates the plan as the low-level agent moves between

waypoints:

$$\pi^{hl} \text{ uses } Q^{hl}(s, a = \omega | g) \text{ to select a graph waypoint } \omega \text{ to reach the final goal } g$$
$$\pi^{ll} \text{ uses } Q^{ll}(s, a | \omega) \text{ to take an environment step } a \text{ to reach the waypoint } \omega.$$

As the agent receives an undiscounted -1 living reward, the optimal low-level value function measures the number of steps to reach a waypoint, *i.e.* the distance $d(s, \omega) = -\max_a Q^{ll}(s, a|\omega)$.

The main challenge with the above semi-parametric formalism is the detrimental effect of errors in the graphical memory. Since nodes are connected by an approximate metric, the number of errors in the graph grows as $O(|\mathcal{V}|^2)$ where $|\mathcal{V}|$ is the number of graph nodes. Moreover, in order for graphical memory methods to scale to larger environments, it is infeasible to store every state encountered in the graph as done in prior work [59, 227]. For the above reasons, node sparsity is a desirable feature for any graphical memory method. In this chapter, we seek to answer the following research question: *given a dense graphical memory, what is the optimal algorithm for transforming it into a sparse one?*

8.3 Sparse Graphical Memory

The number of errors in graphical memory can be minimized by removing redundant states, as each state in the memory can introduce several infeasible transitions through its incident edges. Any aggregation algorithm must answer a key question: *when can we aggregate states?*

Sparse Graphical Memory is constructed from a buffer of exploration trajectories by retaining the minimal set of states that fail our *two-way consistency* aggregation criterion. In Sec. 8.3, we introduce this two-way consistency criterion, which extends prior work to the goal-conditioned RL setting. We then prove that as long as a sparsified graph satisfies two-way consistency, plans in the sparsified graph are close to the optimal plan in the original graph. In Sec. 8.3, we design an online algorithm for checking two-way consistency by selecting the approximate minimal set from the replay buffer. Finally, in Sec. 8.3, we outline a cleanup procedure for removing remaining infeasible transitions.

State aggregation via two-way consistency

As a state aggregation criterion, prior work argues for value irrelevance by aggregating states with similar Q-functions [167, 151, 108]. If the optimal Q-functions are the same at two states, the optimal policy will select the same action at either state.

However, in the goal-conditioned setting, we face a challenge: the states in the memory define the action space of the high-level policy, *i.e.* the waypoints selected by the graph planner. Aggregating states changes the action space by removing possible waypoints.

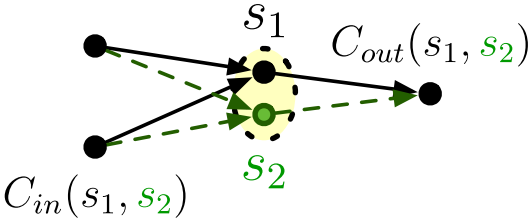


Figure 8.4: SGM uses a two-way consistency criterion to find redundant pairs of states in the replay buffer.

In order to extend value irrelevance to the goal-conditioned setting, we propose two-way consistency (TWC). Under two-way consistency, two states are redundant if they are both interchangeable as goals and interchangeable as starting states according to the goal-conditioned value function.

The graph planner acts as a nonparametric high-level policy, selecting a waypoint ω for the low-level controller as an intermediate goal. Then, the action space is given by candidate waypoints. Formally, using the optimal value function of the high-level policy, states s_1 and s_2 are interchangeable as starting states if

$$\max_{\omega} \left| Q^{hl}(s_1, a = \omega | g) - Q^{hl}(s_2, a = \omega | g) \right| \leq \tau_a. \tag{8.1}$$

Then, as $\tau_a \to 0$, the high-level policy will behave the same in both states. Equation equation 8.1 can be seen as verifying that the sparsification is a τ_a-approximate Q-irrelevant abstraction [108] given a fixed set of waypoints as actions. Further, we say that states s_1 and s_2 are interchangeable as waypoints for the purposes of planning if

$$\max_{s_0} \left| Q^{hl}(s_0, a = s_1 | g) - Q^{hl}(s_0, a = s_2 | g) \right| \leq \tau_a. \tag{8.2}$$

As a consequence, eliminating s_2 from the memory and thus from the action space will not incur a loss in value at any starting state as the policy can select s_1 instead. Together, equation 8.1 and equation 8.2 form the two directions of our two-way consistency criterion in the high-level value space.

While we have motivated two-way consistency in terms of Q^{hl} as $\tau_a \to 0$, it furthermore holds that two-way consistency gives a meaningful error bound for *any asymmetric distance function* $d(\cdot, \cdot)$ satisfying $C_{out}(s_1, s_2) \leq \tau_a$ and $C_{in}(s_1, s_2) \leq \tau_a$ for any finite τ_a where

$$C_{out}(s_1, s_2) := \max_{\omega} |d(s_1, \omega) - d(s_2, \omega)| \text{ and } C_{in}(s_1, s_2) := \max_{s_0} |d(s_0, s_1) - d(s_0, s_2)|. \tag{8.3}$$

The following theorem shows that aggregating states according to two-way consistency of $d(\cdot, \cdot)$ incurs a bounded increase in path length that scales linearly with τ_a. Furthermore, given an error bound on the distance function $d(\cdot, \cdot)$, it provides an error bound on distances in the sparsified graph.

Theorem 1. *Let $\mathcal{G}_{\mathcal{B}}$ be a graph with all states from a replay buffer $\mathcal{B}$ and weights from a distance function $d(\cdot, \cdot)$. Suppose $\mathcal{G}_{TWC}$ is a graph formed by aggregating nodes in $\mathcal{G}_{\mathcal{B}}$ according to two-way consistency C_{out} and $C_{in} \leq \tau_\alpha$. For any shortest path P_{TWC} in G_{TWC}, consider the corresponding shortest path $P_{\mathcal{B}}$ in $G_{\mathcal{B}}$ that connects the same start and goal nodes. Suppose $\mathcal{P}_{\mathcal{B}}$ has k edges. Then:*

(i) The weighted path length of P_{TWC} minus the weighted path length of $P_{\mathcal{B}}$ is no more than $2k\tau_\alpha$.

(ii) Furthermore, if $d(\cdot, \cdot)$ has error at most ϵ, the weighted path length of P_{TWC} is within $k\epsilon + 2k\tau_\alpha$ the true weighted distance along $\mathcal{P}_{\mathcal{B}}$.

Theorem 1 is proved in Appendix G.4. We emphasize that it makes no assumptions on the distance function, but relies on both directions of our criterion. This distance can be derived from

Algorithm 5 `BuildSparseGraph`

Input: replay buffer $\mathcal{B}$, distance function d
Output: sparse graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$
Initialize empty vertex set $\mathcal{V} = \emptyset$
for $\hat{s} \in \mathcal{B}$, each state in the replay buffer, **do**
    if state is novel according to TWC, *i.e.* $C_{in}(s, \hat{s}) \leq \tau_a, C_{out}(s, \hat{s}) \leq \tau_a \; \forall s \in \mathcal{V}$ **then**
        add the state $\hat{s}$ to the graph $\mathcal{G}$:
        $\mathcal{V} = \mathcal{V} \cup \{\hat{s}\}$
        $\mathcal{E} = \mathcal{E} \cup \{(s, \hat{s}) : s \in \mathcal{V}, d(s, \hat{s}) \leq \textsc{MaxDist}\} \cup \{(\hat{s}, s) : s \in \mathcal{V}, d(\hat{s}, s) \leq \textsc{MaxDist}\}$
    end if
end for
assign weights $\mathcal{W}(s_i, s_j) = d(s_i, s_j) \;\; \forall (s_i, s_j) \in \mathcal{E}$
filter transition set $\mathcal{E}$ to k nearest neighbors
return $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$

the value function by negation or learned in a supervised manner, depending on the application. According to Theorem 1, we can aggregate pairs of states that satisfy two-way consistency without leading to significantly longer plans (only a constant increase). Since two-way consistency is approximate, at the same time, it also leads to more sparsity than restrictive criteria like bisimilarity.

In contrast to two-way consistency, a naïve strategy like random subsampling of the replay buffer does not provide a guarantee without further assumptions on the exploration data such as an even coverage of the state space. For example, when randomly subsampling the buffer, a rarely visited bottleneck state will be dropped with the same probability as states in frequently visited regions. As multiple states can be covered by the same state, two-way consistency is a prioritized sparsification of the replay buffer that seeks to cover the environment regardless of sampling density.

Constructing the graph from a replay buffer

State aggregation can be seen as an instance of set cover, a classic subset selection problem. In our instantiation, we select a subset of the states in the replay buffer to store in the agent's memory so that all states in the replay buffer are "covered". Coverage is determined according to the state aggregation criterion, two-way consistency (8.3). Unfortunately, set cover is a combinatorial optimization problem that is NP-hard to solve exactly, or even to approximate to a constant factor [156, 61, 212].

Motivated by the difficulty of the set cover problem, we propose an online, greedy algorithm for replay buffer subset selection. The graphical memory is built via a single pass through a replay buffer of experience according to Algorithm 5, which requires quadratically many evaluations of the TWC criterion. In particular, a state is only recorded if it is novel according to the two-way consistency criterion. Once a state is added to the graph, we create incoming and outgoing edges, and set the edge weight to the distance.

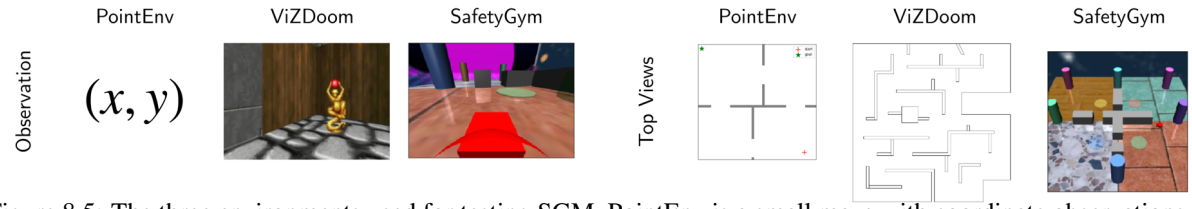


Figure 8.5: The three environments used for testing SGM. PointEnv is a small maze with coordinate observations. We increase its difficulty by thinning the walls. ViZDoom is a large environment, which can take up to 5 minutes and 5k steps to traverse entirely. ViZDoom actions are discrete and observations are first-person camera views. SafetyGym is another large environment with first-person view observations, and supports continuous actions.

Graphical memory cleanup

Although TWC produces a compact graph with substantially fewer errors than the original graphical memory, faulty edges may still remain. If even one infeasible transition between distant states remains as an edge in the graph, the planner will exploit it, and the agent will not be able to carry out the plan [59]. Therefore, eliminating faulty edges is key for robustness.

We bound the number of errors in the memory by filtering edges, limiting nodes to their k nearest successors. After filtration, the worst-case number of untraversable edges grows only *linearly* in the sparsified node count, not quadratically. This is a simple, inexpensive procedure that we experimentally found removes many of the infeasible transitions.

Finally, we propose a novel self-supervised error removal technique – *cleanup*. During test-time execution, a random goal g is sampled and the planner provides a set of waypoints w for the low-level agent. When the low-level agent is unable to reach a consecutive waypoint $w_i \rightarrow w_{i+1}$, it removes the edge between (w_i, w_{i+1}) and re-plans its trajectory. This procedure is efficient for a TWC sparsified graph since $|\mathcal{G}_{\mathcal{TWC}}| \ll |\mathcal{G}_{\mathcal{B}}|$.

8.4 Experimental Setup

We evaluate SGM under two high-level learning frameworks: reinforcement learning (RL), and self-supervised learning (SSL). As a general data structure, SGM can be paired with any learned image features, asymmetric distance metric, or low-level controller. Below, we describe our training procedure in detail.

We benchmark against the two available environments used by the SoRB and SPTM baselines, and an additional visual navigation environment. These range in complexity and are shown in Figure 8.5. The environment descriptions are as follows: **PointEnv**[59] continuous control of a point-mass in a maze used in SoRB. Observations and goals are positional (x, y) coordinates. **ViZDoom**[284] discrete control of an agent in a visual maze environment used in SPTM. Observations and goals are images. **SafetyGym**[211] continuous control of an agent in a visual maze environment. Observations and goals are images, though odometry data is available for observations but not for goals.

Technique	Success Rate	Cleanup Steps	Observation	Env
SoRB	$28.0 \pm 6.3\%$	400k	Proprio	PointEnv
SoRB + SGM	$\mathbf{100.0} \pm \mathbf{.1}\%$	400k	Proprio	PointEnv
SPTM	$39.3 \pm 4.0\%$	-	Visual	ViZDoom
SPTM + SGM	$\mathbf{60.7} \pm \mathbf{4.0}\%$	114k	Visual	ViZDoom
ConSPTM	$68.2 \pm 4.1\%$	1M	Visual	SafetyGym
ConSPTM + SGM	$\mathbf{92.9} \pm \mathbf{1.4}\%$	1M	Visual	SafetyGym

Table 8.1: SGM boosts performance of all existing state-of-the-art semi-parametric graphical methods.

We utilize the PointEnv maze for RL experiments, where SGM is constructed using undiscounted Q-functions with a sparse reward of 0 (goal-reached) and -1 (goal not reached). We increase the difficulty of this environment by thinning the walls in the maze, which exposes errors in the distance metric since two nearby coordinates may be on either side of a maze wall. SoRB also ran visual experiments on the SUNCG houses data set [246], but these environments are no longer public.

To evaluate SGM in image-based environments, we use the ViZDoom navigation environment and pretrained networks from SPTM. In addition, we evaluate navigation in the OpenAI SafetyGym [210]. In both environments, the graph is constructed over *visual first-person view observations* in a large space with obstacles, reused textures, and walls. Such observations pose a real challenge for learning distance metrics, since they are both high-dimensional and perceptually aliased: there are many visually similar images that are temporally far apart. We also implemented state-of-the-art RL algorithms without the graphical planner, such as DDPG [152] and SAC [89] with Hindsight Experience Replay [9], but found that these methods achieved near-zero percent success rates on all three environments, and were only able to reach nearby goals. For this reason, we did not include these baselines in our figures.

8.5 Results

SGM increases robustness of plans: We compare how SGM performs relative to prior neural topological methods in Table 8.1. **SGM sets a new state-of-the-art in terms of success rate** on all three environments tested and, on average, **outperforms prior methods by 2.1x**. In fact, thinning the walls in the PointEnv maze *breaks the policy from the SoRB baseline* because it introduces faulty edges through the walls. SoRB is not robust to these faulty edges and achieves a 28% score even after 400k steps of self-supervised cleanup. SGM, on the other hand, is able to remove faulty edges by merging according to two-way consistency and performing cleanup, robustly achieving a 100% success rate. Similarly, in visual environments, the sparse graph induced by SGM produces significantly more robust plans than the SPTM baselines. While SGM solves the SafetyGym environment with a 96.6% success rate, it navigates to only 60.7% of goals in ViZDoom. Note that this VizDoom success rate is lower than in [227] because [227] uses human demonstrations whereas we do not. We found that the primary source of error in VizDoom was due to perceptual aliasing

Technique	Easy $\leq$ 200 m	Medium $\leq$ 400 m	Hard $\leq$ 600 m	Overall
Random actions	58.0%	21.5%	12.0%	30.5%
Visual controller	75.0%	34.5%	18.5%	42.7%
SPTM, subsampled observations	70.0%	34.0%	14.0%	39.3%
SPTM + SGM + 54K cleanup steps	88.0%	52.0%	**26.0%**	55.3%
SPTM + SGM + 114K cleanup steps	**92.0%**	**64.0%**	**26.0%**	**60.7%**

Table 8.2: SGM improves graph-based success rates across goal difficulties in ViZDoom. More cleanup steps yield better performance because more infeasible transitions are removed from the graph.

where two perceptually similar observations are actually far apart. For further details, see Appendix G.2.

We examine SGM performance with and without cleanup. Figure 8.6 shows that success rapidly increases as the sparse graph is corrected. However, without sparsity, the number of errors is too large to quickly correct. Success rates in Table 8.3 show that **sparsity-enabled cleanup is essential for robust planning** and that **cleanup improves mean performance by 2.4x**.

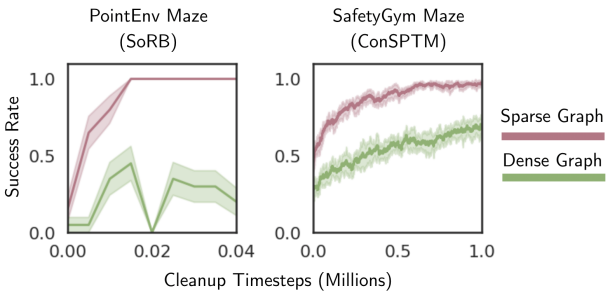


Figure 8.6: Success rate as a function of cleanup steps in PointEnv (FourRooms maze) and SafetyGym. SGM is rapidly corrected while SoRB, because of errors in its dense graph, is infeasible to clean. SPTM can be cleaned, but only slowly.

We also study success as a function of goal difficulty in Table 8.2, where difficulty is determined by the ground-truth distance between the initial agent's position and its goal. Even with cleanup, performance degrades with goal difficulty, which is likely the result of perceptual aliasing causing faulty edges (Appendix G.2).

Two-way consistency is the preferred subsampling strategy: We ablate different subsampling strategies to examine the source of our performance gains. We compare SGM with two-way consistency with variants of SGM with one-way consistency, where connectivity between two nodes is determined by only checking either incoming or outgoing distance function values, as well as a simple random subsampling strategy where a subset of nodes and edges are uniformly removed from the dense graph. Table 8.3 shows success rates for a SafetyGym navigation task before cleanup and after 1M steps of cleanup. Before cleanup, all plans perform poorly with a

Criterion	w/o Cleanup	w/ Cleanup
uniform	$31.2 \pm 3.8\%$	$64.6 \pm 3.4\%$
perceptual	$31.8 \pm 3.6\%$	$77.0 \pm 2.5\%$
incoming	$33.6 \pm 3.1\%$	$84.1 \pm 2.0\%$
outgoing	$28.7 \pm 3.2\%$	$86.8 \pm 2.9\%$
two way	$\mathbf{38.2 \pm 3.8\%}$	$\mathbf{92.9 \pm 1.2\%}$

Table 8.3: Comparing node subsampling strategies in SafetyGym. Two-way consistency improves success rate relative to other criteria.

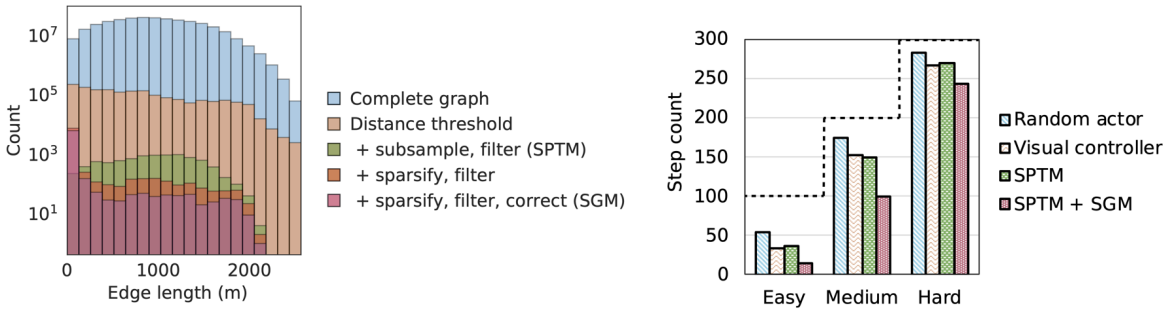


Figure 8.7: (Left) Overlayed histograms of edge lengths for graphical memories in ViZDoom, where long edges are incorrect. Graph quality significantly improves with SGM, which reduces the frequency of long edges. (Right) SGM reaches goals using the fewest environment steps (shortest path length).

maximum 38.2% success rate achieved by two-way SGM. After cleanup the SGM variants significantly outperform a naive uniform subsampling strategy, and subsampling with two-way consistency achieves the highest success rate before and after cleanup.

Investigation of two-way consistent graph quality: We further investigate how individual components of two-way consistent SGM affect final graph and plan quality. In Figure 8.7 (left), we display the total edge count for various edge lengths in the ViZDoom graphical memory after different steps of the SGM algorithm. Since nodes should only be connected locally, long edges are most likely to be faulty, representing infeasible transitions. Although the different components of the SGM algorithm all help reduce errors, **two-way consistency removes the most incorrect edges** from the original dense graph while preserving the most correct edges.

Finally, in Figure 8.7 (right) we display the average number of steps required for different agents to reach their goal in ViZDoom. The average is only taken over successful paths, and the dotted line shows the maximum number of steps allowed. We show that, on average, agents equipped with sparse graphical memory take the fewest steps to reach their goals compared to other methods. This suggests that shorter paths are another reason for improved success relative to other methods.

8.6 Conclusion

In this chapter, we proposed a new data structure: an efficient, sparse graphical memory that allows an agent to consolidate many environment states, model its capability to traverse between states, and correct errors. In a range of difficult visual and coordinate-based navigation environments, we demonstrate significantly higher success rates, shorter rollouts, and faster execution over dense graph baselines and learned controllers. We showed that our novel two-way consistency aggregation criterion is critical to this success, both theoretically and empirically. We hope that this direction of combining classic search-based planning with modern learning techniques will enable efficient approaches to long-horizon sensorimotor control tasks. In particular, we see scaling sparse graphical memory to challenging manipulation tasks as a key outstanding challenge for future work.

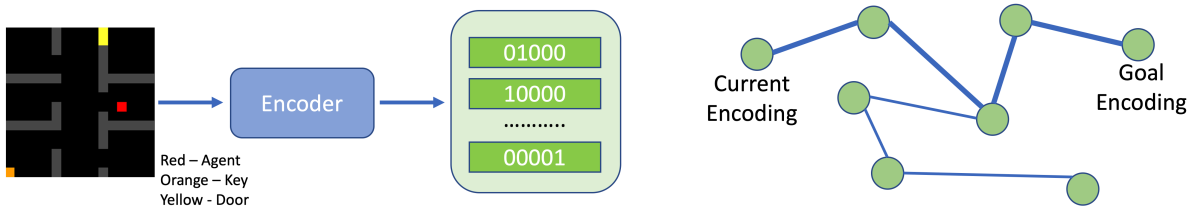
Chapter 9

Discrete Abstraction for Planning

In future, we hope that robots will be able to operate in unstructured environments such as homes and hospitals, and endowed with long-horizon planning ability. Despite successes in deep reinforcement learning (RL) from raw observations, much progress relies on the availability of shaped reward to guide the learning [191, 172]. On the other hand, over past decades, task and motion planning has been shown to solve much longer-horizon goal-directed tasks such as making a cup of coffee from torque control [114, 250, 265, 277]. However, these methods often require pre-specified discrete abstract states, task representations and transition models, e.g., whether the robot is holding a cup and what actions (or perturbations) change such an abstract state. In this paper, we aim to learn discrete representations for high-level abstract planning from video interaction data, combined with a learned short-horizon controller.

Learning discrete representations from unsupervised data for planning is challenging for two reasons. First, the relationship between the optimization objective and the true task objective is not well-defined. Second, optimizing a model with a discrete layer is difficult with standard deep learning techniques. Recent methods approach the first problem using reconstruction or constrastive objectives [279, 7, 88, 134, 92, 247]; however, the learned representations are in continuous latent space which is unstructured and difficult to combine with high-level abstract planning. While other methods show promise in learning discrete representations, they have not been applied to temporally-extended RL tasks [196, 213, 216, 12, 251].

In this work, we propose Discrete Object-factorized Representations for Planning (DORP) – a novel framework for visual planning and control by learning discrete representations and a low-level controller. DORP learns discrete representations from images that change slowly overtime, such as whether or not the agent holds a key or which room the agent is in, along with a low-level predictive model for control. These slow features enable the agent to plan at a low frequency in longer-horizon tasks. More specifically, DORP represents an abstract state as a set of one-hot vectors, and optimizes its encoder by maximizing a mutual information lower bound between the current representations to future observations [196]. In order to train through the discrete layer, we apply the Gumbel-Softmax reparametrization trick [106, 162]. Using abstract states as nodes, we build an approximate feasibility graph based on observed transition data. When provided with new start and goal images, the agent plan the shortest abstract path. Using the next abstraction



(a) Unsupervisedly encode image with one-hot encodings (b) Plan on the discrete encoding space with graph search

Figure 9.1: Learning discrete representions for planning. We approach the long-horizon planning on high dimensional image space by learning a discrete representation and planning on the discrete space. (a) We learn multiple one-hot encodings for each observation fully unsupervisedly with contrastive learning. Each one-hot encoding corresponds to a temporally abstract state of one of the freely moving entity in the observation, such as a room the agent is in or whether a key has been picked up. (b) With the discrete encoding, we build a graph that connects the current encoding to the goal encoding and do a graph search on it to plan efficiently for long horizon tasks.

as waypoint, model-predictive control maximizes the objective that is 1 if it reaches the target abstraction and 0 otherwise with a trained video prediction model. Unlike other subgoal planning work [228, 190, 138, 154], by following abstract waypoints, DORP avoids unneccessary steps to match exact waypoint states.

In a set of experiments, we demonstrate that DORP learns temporally-consistent and object-factorized representations suitable for planning. We show that these representations enable DORP to handle unseen long-horizon tasks more successfully compared to the states-of-the-arts in visual planning. Interestingly, we observe that latent representations show object-level factorization such as key-and-door.

9.1 Preliminaries and Problem Statement

We present background material on unsupervised representation learning and discrete optimization that our method builds on, and our problem formulation.

Contrastive Predictive Coding (CPC) [196] learns low-dimensional representations that are most predictive of the future high-dimensional sequential data. A non-linear encoder $q_\theta : \mathcal{O} \to \mathbb{R}^l$, parametrized by θ, encodes the observation $o_t \in \mathcal{O}$ to a latent l-dimensional vector representation z_t. Let's define a similarity score $f_k(z_t, o_{t+k}) = \exp(z_t \psi q_\theta(o_{t+k}))$ where ψ is a trainable l-by-l similarity matrix and o_{t+k} is a future observation k steps ahead of o_t. Given the *query* observation o_t, we aim to classify the *key* observations – o_{t+k} as positive and other sample $\tilde{o}$ from the dataset as negative. Formally, we optimize the loss function

$$\mathcal{L}_{CPC} = -\mathbb{E}_{o_t, o_{t+k}} \left[\log f_k(z_t, o_{t+k}) - \log \sum_{o_j \in X} f_k(z_t, o_j) \right]$$

with respect to θ and ψ. This also corresponds to maximizing a lowerbound of the mutual information between the latent representation z_t and the future observation o_{t+k}.

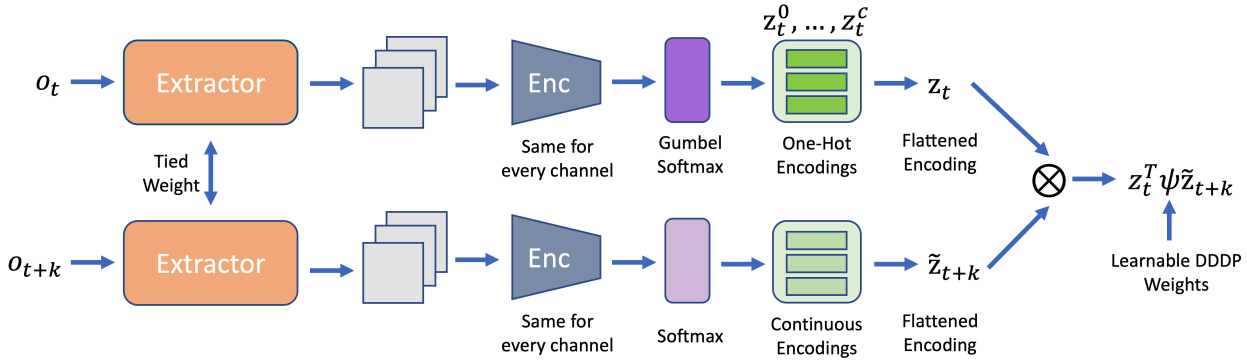


Figure 9.2: Unsupervised discrete code learning. We propose to learn a set of one-hot encodings as the latent representation for each observation. For an anchor observation o_t, its neighbours o_{t+k} for some small k are treated as positives, and random other observations are negatives. We propose a learnable weight matrix to be diagonally dominant and diagonally positive (DDDP). An object extractor architecture is applied to learn to extract objects and a shared encoder is applied per channel. Finally, Gumbel softmax and softmax are used to increase the gradient signal for backpropagation.

Gumbel-Softmax (GS) [106] is a discrete optimization technique to compute the gradient of through samples from a categorical distribution π_i. GS first applies a reparametrization trick to rewrite samples z_i as onehot$(\arg\max_j(g_j + log\pi_i))$ where g_j are i.i.d samples from Gumbel(0, 1) [85]. GS approximates $\arg\max$ with $\text{soft}\max$ making the discrete stochastic layer differentiable, i.e., $z_i = \text{soft}\max((g_i + \log \pi_i)/\tau)$. When the temperature parameter τ approaches 0, the samples converge to the true categorical distribution. Empirically, the temperature τ starts high and is annealed closer to 0 by a certain schedule.

Problem Statement: Goal-Directed Visual Planning and Control

We define an unknown, fully-observable, stochastic dynamical system f which maps input observation $o_t \in \mathcal{O}$ and action $a_t \in \mathcal{A}$ to the next observation o_{t+1}. Under this dynamical system, we assume a simple exploration policy π_{rand} which can collect data that characterizes the dynamics of the system. This is known as self-supervised data or play data [4, 157]. We consider high-dimensional observation such as images.

Our goal is to learn discrete abstraction $z_t \in \mathcal{Z}$ given observation o_t, an abstract feasibility model, and a low-level local controller. At test time, given start o_s and goal o_g observations, we can plan a sequence of abstract states $z_s, z_1, ..., z_g$ where z_s and z_g are the representation of o_s and o_g, and apply the low-level controller to reach these abstract waypoints and finally to o_g.

9.2 Discrete Object-factorized Representation for Planning

Our objective is to derive representation properties such as *object-factorization* and *temporal-consistency* from an unsupervised learning objective to facilitate long-horizon planning. When object representations are factorized, the agent can escape the need of combinatorial data configurations

and can quickly generalize to unseen tasks. Also, the connectivity graph can also be memory-efficiently represented and combined with a more powerful planning algorithm. Another important property for a representation is temporal consistency, meaning any two state observations in the same abstraction should be reachable from one-another by a short sequence of actions. When this property holds in the latent space, a high-level plan can be successfully executed by a low-level controller.

We propose Discrete Object-factorized Representation for Planning (DORP) to tackle goal-directed visual planning and control problem. DORP learns latent discrete representations, a connectivity graph for abstract planning, and a low-level model-predictive controller. We describe each subcomponent in Section 9.2 - 9.2.

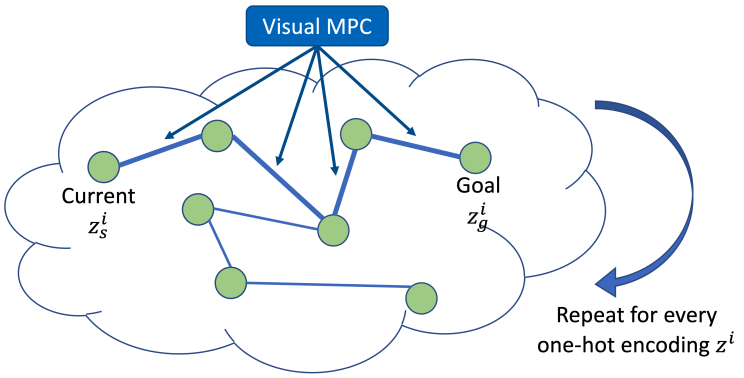


Figure 9.3: DORP Planning. Since the learned representation includes multiple one-hot encodings, we plan for each one-hot encoding one at a time. For each of the one-hot encodings z^i, we build a graph based on that subset of the representation. After finding the shortest path from current state to the goal state, we use MPC to reach the next planned state. See Section 9.2 for more explanations.

Discrete Representations

Despite successes in applying its unsupervisedly learned representations for downstream tasks, CPC is not endowed with an inductive bias for its representations to facilitate planning [196]. Additionally, it is unclear how a discrete representations can be extracted and optimized. Toward this goal, we propose an object-factorized architecture with asymmetric Gumbel softmax for discrete optimization, and a diagonally dominating and diagonally positive similarity matrix to encourage temporal consistency.

We implement our DORP architecture to encourage object-factorization and capture discrete representations. The extractor passes an anchor observation through a convolutional neural network and outputs a c-channel feature map. Each feature map is inputted to a shared encoder followed by a Gumbel softmax. For a positive and negative pair of images, however, we opt for a continuous embedding by swapping Gumbel softmax for softmax. Empirically, without this asymmetry trick, the optimization tends to converge to a poor local optima. Together these one-hots are flattened into long vectors, and their bilinear product is the similarity score between the query-key pair (see Figure 9.2).

We propose an inductive bias in the similarity score function to encourage temporal consistency. Instead of a fully trainable matrix, we parameterize the similarity matrix ψ to be diagonally dominant and diagonally positive (DDDP). This property biases the similarity score to be high when the key embedding $\tilde{z}_{t+k}$ is close to the query embedding z_t and maximized when $\tilde{z}_{t+k} = z_t$. In other words, this incentivizes the representations of positive pairs to share as many one-hots as possible. We

reparameterize the weight matrix as $e^{\psi_0}(-\sigma(\psi_1) + \alpha I_{l\times l})$ where $I_{l\times l}$ is an identity matrix, α is a positive constant, σ is a sigmoid function, ψ_0 is a trainable scalar, and ψ_1 is a trainable l-by-l matrix.

Abstract Planning

A critical component in planning is a connectivity graph that decides whether two discrete representations are connected. One naive solution is to build such graph from data – all observed representations as nodes and all observed transitions as edges. However as the number of independent entities increases, the amount of data required to generate the nodes and edges in order to cover the state space grows exponentially. To solve this, we propose to reduce the planning problem by exploiting factorization (see Figure 9.3).

1. First, embed the current image o_s and goal o_g as $z_s = q_\theta(o_s)$ and $z_g = q_\theta(o_g)$.
2. Pick a random one-hot index i s.t. the current state's one-hot z_s^i differs from the goal's one-hot z_g^i.
3. Build a graph based on how this one-hot transits in the data (ignoring all other one-hots).
4. Plan a path $z_1^i, ..., z_g^i$ from z_s^i to z_g^i. If this does not exists, then we fail the task.
5. Execute the low-level controller following the next target z_k^i, $k = 1, \cdots, g$, while keeping other z^j the same.
6. If it doesn't reach z_k^i, remove the edge from the graph and redo the planning in step 4.
7. If it succeeds, follow the next target until it reaches z_g^i.
8. Repeat step 2 until all the one-hots match.
9. Finally, execute the low-level controller to the goal image o_g.[1]

Our proposed method assumes that each latent one-hot can be manipulated independently to solve the task. This assumption can hold various settings. For example, imagine a task that requires reorganizing furniture in a spacious room, and each objects is represented as a separate one-hot. Here, we can manipulate each object separately, and since there is plenty of empty space in this scenario, we can easily plan around objects to avoid collisions.

This assumption does not always hold; along the same scenario, when the space is more cluttered, objects are more likely to collide, and the proposed fully-factorized planning may fail even though a solution exists (incompleteness). **We extend this planning algorithm by building a graph on a random set of one-hot indices at a time in step 2**. When can choose this set to contain all the one-hots the method is equivalent to the full graph search (complete but expensive). Thus, our extended planning method can trade between completeness and efficiency. We demonstrate this further in our experiments.

[1] An implicit assumption of temporal consistency in the learned discrete representations is required.

Low-level Control

The low-level controller is the key component in order to follow abstract plans and reach the goal image. For this reason, the controller must be flexible in achieving multi-objectives. Model-predictive controller (MPC) achieves this by modeling the dynamics of the world from past data and minimizing with the total predicted cost of horizon T at run-time:

$$a_t^*, ..., a_{t+T-1}^* = \underset{a_t,...,a_{t+T-1}}{\arg\min}\ \mathbb{E}\left[\sum_{i=1}^{T} \gamma^t \hat{c}_{t+i}\right].$$

The predicted cost is computed by applying a known cost function on the predicted outcome of a T-step action sequence from the current observation o_t.

MPC requires the world model and the costs. For the world model, we implement a stochastic video prediction model [15]. In our framework, we define two cost functions for reaching abstract goals and for reaching the final goal observation. The first cost function is defined to be 1 if the current embedding exactly matches the goal embedding (all the one-hots match) and 0 otherwise: $c_{\text{lat}}(o_t, z') = \mathbb{1}[q_\theta(o_t) == z']$. The second cost function is defined by its L2 loss in the observation space to the goal: $c_{\text{obs}}(o_t, o_g) = ||o_t - o_g||$. This cost is versatile for reaching nearby observations particularly to reach the goal observation once it has reached the goal code.

9.3 Related Work

Object Discovery. Recent work has studied unsupervised object segmentations from visual inputs [28, 84], and entity-factorized representations and models for predictions [126]. However, these studies have not been demonstrated to directly solve the task. In this work, we take a step further to evaluate the representations in downstream tasks. While other work [268, 280] shows how the MPC agents benefit from object-factorized models, they require shaped rewards for the tasks. In contrast, we consider a long-horizon task in which the agent only receives its reward when it has reached the goal. A large body of work in computer vision has studied unsupervised video object segmentation [5, 18, 19, 78, 70, 95, 120] and unsupervised object detection [136, 273, 285]. However, semantic segmentation and object detection might not be the correct representation for the end-task. In this work, we learn representations that are more ready to use for planning without explicit segmentation or detection.

World models. Much previous work has applied generative models of the world to visual control tasks [279, 88, 92]. Other work [158, 195, 286] leverages a contrastive objective to learn a latent world model. These methods however require dense reward signals for most tasks. Visual foresight methods [66, 57] demonstrate impressive results on real robots using unshaped reward such as pixel distance to the goal, but it still remains limited to short-horizon object pushing tasks. In our work, we borrow these methods for low-level controllers. Asai and Fukunaga [12] learn discrete abstraction of the system such as an 8-piece puzzle, but do not consider temporal abstraction. Kansky et al. [117] demonstrate that discrete object-factorized representations can be used to learn logic-based transitions. In combination with powerful planning, it improves generalization and data

efficiency. However, it assumes supervised ground-truth labels for representation learning. In this work, we aim to learn this in an unsupervised manner.

Hierarchial RL. Recent work has tried to approach long-horizon visual planning by breaking down tasks using skills [157, 126]. Our approach is orthogonal and may deploy such action abstractions as a low-level controller. Other methods propose to plan subgoals and attempt to follow them either in the latent space [134, 190, 188] or in the visual space [228, 154, 60]. However, such methods require labor-intensive engineering to tune the threshold on when to move on to pursue the next subgoal [60, 154] because the observations or the latent states can never exactly match. Imagine bringing a chair from one office to another. It would be time consuming to match all the positions and orientations of the chair along the way. Rather we should care about rough area the chair has to go through in order to reach the goal. In our work, by learning the discrete codes, our method do not require such threshold as it exactly knows when it has reached the target discrete representation. Additionally, it avoids aiming to match a specific waypoint observation. Instead of planning to the goal, Pertsch et al. [200] predict intermediate images iteratively to construct a subgoal tree. However in order to train such prediction model a long-sequential training data are required. Instead our work can be applied to short-horizon or long-horizon trajectories.

9.4 Experiments

We perform experiments aimed at answering the following questions: (1) Are DORP representations temporally-consistent for high-level planning? (3) Can DORP representations factorize objects that translate or change in appearances over time? (3) How does DORP compare to the visual planning SOTA in solving unseen long-horizon goal-directed tasks?

Environments

We evaluate DORP on two main environments – object-rearrangement and key-room. All the observations are provided as color images of size $16 \times 16 \times 3$.

k-object-rearrangement Multiple (k) 2-by-4 blocks of different colors can each be manipulated independently by an agent. The tasks require the agent to manipulate these objects from a start configuration to a goal configuration. In each step, it can manipulate each block by one unit in one of the 4 directions. The agent collects data by randomly interacting with one of the objects at each timestep in a purely exploratory manner, without any goal in mind. In this environment our trajectory has length one that is the configuration is randomly reset after every step.

key-room Two variations of key-room depend on the number of rooms – key-corridor (6 rooms and a corridor) and key-wall (2 rooms. A key and agent are represented by 1 pixel. A key is placed in a fixed location in a room. If the agent steps on the key a door will be removed allowing the agent to enter a locked room. Each step the agent can move in one of the four directions by 1 unit. The agent collect data by a long random walk – 1500 steps for key-corridor and 1000 steps for key-wall. After each reset the agent will be placed randomly in one of the unlocked rooms. This environment is inspired by MiniGrid [33].

Algorithms	1 object		2 objects		3 objects		5 objects	
	SR	# steps	SR	# steps	SR	# steps	SR	# steps
DORP	**1.00**	$25_{\pm 15}$	**1.00**	$49_{\pm 22}$	**1.00**	$75_{\pm 28}$	**0.87**	$200_{\pm 69}$
– arbitary weight	0.92	$26_{\pm 19}$	0.58	$52_{\pm 16}$	0.26	$52_{\pm 16}$	0.10	$262_{\pm 61}$
– identity weight	**1.00**	$34_{\pm 12}$	0.94	$18_{\pm 7}$	0.48	$83_{\pm 23}$	0.05	224
– full graph	**1.00**	$25_{\pm 15}$	**1.00**	$56_{\pm 17}$	**1.00**	$76_{\pm 26}$	0.00	max
VF-RS	0.83	$19_{\pm 19}$	0.58	$53_{\pm 25}$	0.25	$93_{\pm 24}$	0.00	max
VF-CEM	**1.00**	$12_{\pm 15}$	0.60	$47_{\pm 25}$	0.30	$89_{\pm 27}$	0.10	max

Table 9.1: Comparison. We evaluate DORP in k-object-rearrangement tasks by measuring success rates across 50 sample tasks. DORP is able to successfully solve most of the tasks as we increase the number of objects. We observe that other methods' performances degrade rapidly. We denote 'max' when the agent has reached the limit on the number of steps without reaching the goal in 5-object-rearrangment this limit is 1400.

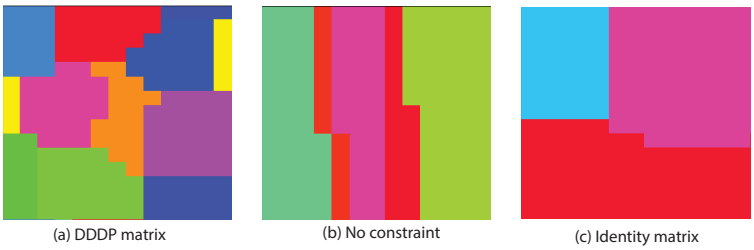
Baselines

We choose the state-of-the-arts visual foresight (VF) [57] implemented using SV2P architecture [15] as our baseline for two reasons. First, VF only requires self-supervised data collection and is applicable to unseen tasks – thus aligning with goal-directed visual planning problems. Second, Visual Foresight is a low-level controller of DORP. By sharing the same video model as baseline, we show a direct improvement of abstract planning for temporally-extended tasks. Two optimization variations of VF – VF with random shooting (VF-RS) and VF with cross-entropy method (VF-CEM) – both share the same video prediction model. Their objective is to minimize its L2 distance from the current image to the goal image. While DORP deploys VF-RS for its low-level control, VF-CEM comparision are provided for an improved baseline. VF-RS randomizes 1000 trajectories and take the full action sequence that achieves the minimum cost. VF-CEM randomizes 1000 samples per iteration. We pick the top 2% to refit a distribution over sequence and repeat for 3 iterations.

To understand DORP representation learning improvement, we evaluate DORP when replacing its DDDP similarity matrix by an arbitrary weight [196] or an identity matrix [97] which have been used extensively in unsupervised representation learning. To understand the benefits of factorized planning, we replace it with the full graph planning with a maximum limit on the number of steps allowed.

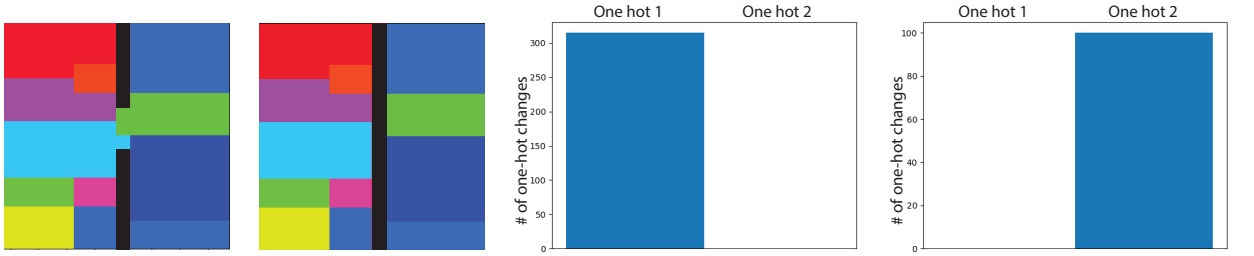
Quantitative Results

Temporal consistency. We investigate DORP representations by color-coding the discrete embeddings on the configuration map of the environments. For visualization simplicity we choose 1-object-rearrangement to visualize the learned code map. In this environment we have a single one-hot code for representing the object. We demonstrate that DORP learns temporally consistent representations in which two states from the same discrete code are connected by a short-horizon controller (see Figure 9.4(a)). Similarly, we observe the same property in key-wall environment (see Figure 9.5).



(a) DDDP matrix (b) No constraint (c) Identity matrix

Figure 9.4: Discrete Embedding Comparison. We visualize the color codes of different object positions in 1-object-rearrangement per similarity matrix type. Each color shows different discrete code. The one-hot embedding has size 16 for all settings. In (a), we find that DORP discrete representation is temporally-consistent, i.e., two states that map to the same embedding are connected by a short sequence of actions. In (b), when using an arbitrary weight matrix the embedding is less temporally consistent. In (c), when using an identity matrix the, the embedding uses only 3 out of 16 available codes



(a) Representation map with key (b) Representation map without key (c) Onehot changes when moving only the agent without key interaction (d) Onehot changes when adding and removing the key on the grid while fixing agent positions

Figure 9.5: Key-Wall Representations. In (a) and (b), we demonstrate the discrete code of the agent at different positions. Each color represents the same code. The grids in black are invalid states (the wall that blocks the agent and separates the two rooms). We demonstrate temporal consistency in the latent space both when the object has the key as not. In (c) and (d), we confirm that two one-hot codes are factorized by observing that only one one-hot changes when removing the key while maintaining the agent position and only the other one-hot changes when the agent moves without interacting with the key.

Object factorization. We evaluate the behavior of the one-hot codes by changing one property of the agent at a time such as positions or whether the agent has a key. In Figure 9.6(c) we observe that by moving one object at a time only at most one one-hot code changes its value k-object-rearrangement. In key-room, we formulate the discrete representation as follows: of the two output one-hot latents, the first is set to a size z_0 and the other is set to size 2. Our aim is for one latent one-hot to encode the agent's abstract state and for the other binary latent to encode the key's abstract state. In Figure 9.5, we demonstrate that the learned abstract representations are factorized as desired by observing (1) the larger one-hot changes when the agent moves with no key interactions and (2) the binary one-hot changes when removing and adding the key while maintaining the agent position.

Weight Ablation. We study the effect of the proposed similarity matrix against commonly-used similarity matrices [196, 97] in unsupervised representation learning. In Figure 9.4, we demonstrate that DORP is able to exploit more available latent code and therefore helping the latent space to be more temporal consistent.

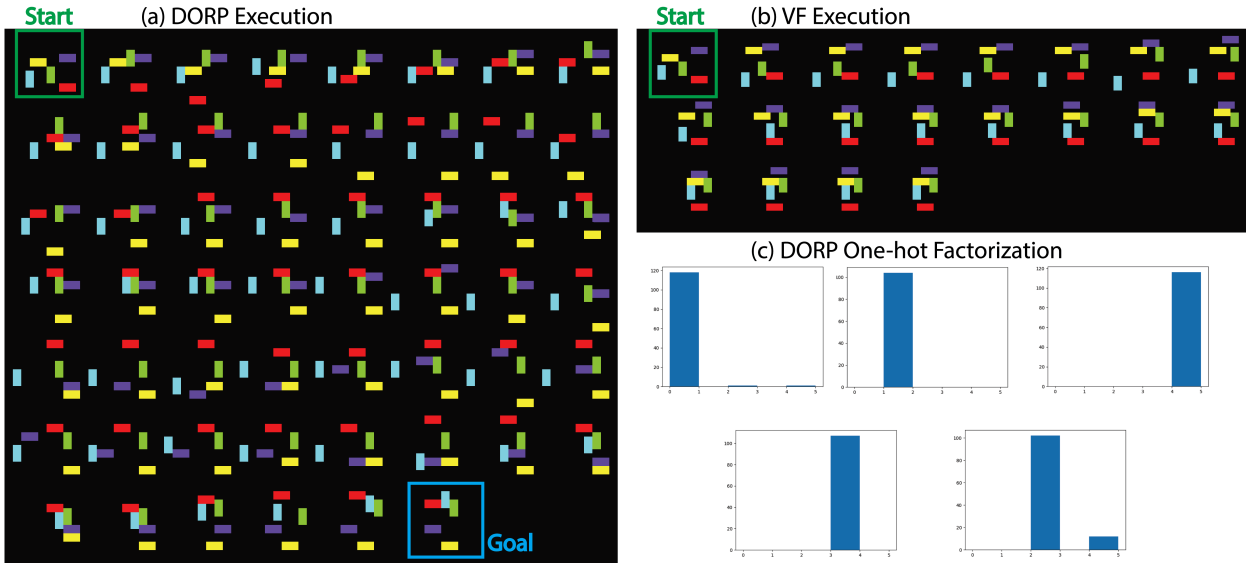


Figure 9.6: DORP in 5-object-rearrangement. In (a), we show a sample image trajectory (read from left-to-right) DORP takes from unseen start and goal images. The images are taken every xxx steps. In (b), when presenting the same task to VF-CEM we find that the objects are stuck in an awkward configuration where 4 blocks (except the purple) are blocking one another to reach their goal positions. Finally, in (c), we visualize the representation factorization by randomly moving one of the five object while maintaining the positions of the others. We plot a histogram of which one-hots have been changed per object. We find that with high probability only the one-hot that corresponds to the moving object is modified.

Quantitative Results

Long-horizon planning. We test the agent by its ability to solve unseen goal-directed tasks. In k-object-rearrangement, we can study the effect of the increase of the length and complexity of the tasks by increasing the number of objects. In Table 9.1, we demonstrate that as the number of objects increases DORP is able to succeed in most of the tasks. While other methods' performances degrade quickly. Note that when faced with 5 objects DORP employs the extended planning version by grouping the 5 one-hots into two groups of 2 and 3 one-hots. By considering more that one object at a time, the agent is able to perform non-myopic planning and achieve 87% success rate. We also evaluate DORP on two variations of key-room. In our evaluations, we find that DORP solves key-wall with a single wall and door successfully in 9 of 10 tasks. When using a ground-truth dynamics model in place of an SV2P model, DORP solves Key-Room successfully in 37 of 40 tasks, and Key-Corridor in 33 of 40 tasks.

9.5 Conclusion

In this chapter, we propose an unsupervised discrete representation learning method for long term planning, which can extract high level abstract states that are suitable for planning in an purely unsupervised manner. We demonstrate the effectiveness of DORP over other methods on challenging

long horizon tasks. We note that our method tractably generate approximate plans when presented with a combinatorial search space as the number of objects increases.

Chapter 10

Conclusion

In this thesis, we developed learning, planning, and acting algorithms in order to bring robots into complex, unstructured environments the real world presents. We underline the application of learned world models for planning and acting to 3 main challenges – data efficiency, generalization, and long-horizon tasks.

For data efficiency, in Chapter 2, we propose a model-based reinforcement learning method that is 100x more data efficient than model-free counterparts by using the key idea of model ensemble to feed training data to the policy and decide to collect more data only when required to. In Chapter 3, we build upon the ensemble idea to allow model-based RL to capture multi-modal dynamics. When presented with new tasks at test time, our method can immediately perform on such tasks without additional training data. We demonstrate that in this zero-shot setting standard model-based RL are on par with the meta-learning model-free counterparts. Our approach however shows significant gain over them. In both chapters, we evaluate our approaches on the standard locomotion benchmark.

For generalization, in Chapter 4, we describe the gap between classical planning and reinforcement learning. We then present a deep generative model that learns the latent representation for planning based on transition data. We demonstrate that it generates feasible visual plans for novel start and goal configurations. In Chapter 5, we propose a context-conditional model in order to handle context changes, e.g., obstacle positions. Given unseen contexts, we show that plans can be generated and executed when combined with inverse models for predicting actions. We demonstrate that a real robot can manipulate a piece of rope around two obstacles, and in simulation benchmark it against offline RL methods. In Chapter 6, instead of generating plans in the latent space, we hallucinate the images and perform visual planning directly in the observation space. To do this, we build a graph and train a connectivity model using a contrastive objective. We demonstrate that such graph-based approach is able to generalize to unseen contexts such as object shapes and obstacle shapes in simulated manipulation tasks for the first time, and improve the robustness over the state-of-the-art methods.

For long-horizon tasks, in Chapter 7, we design a hierarchical model-based RL method in which the high-level planner passes its action as the goal for the low-level controller. We demonstrate the method in temporally-extended block pushing and pouring tasks. In Chapter 8, we propose a

data structure for maintaining a sparse graphical memory and an edge clean-up method for robust long-horizon first-person navigation tasks. In Chapter 9, we develop an unsupervised discrete object-factorized representation learning approach for high-level discrete abstract planning, combined with low-level visual controller for achieving nearby discrete tasks. We demonstrate that our method can solve temporally-extended sequential tasks with a combinatorial configuration space such as multi-object pushing.

While we believe this thesis has led much progress in world models for planning and acting, there are many interesting questions open up to be answered. We would like to offer some future avenues for exploration that can bring us another step closer toward deploying robots widely in the real world:

Date efficiency. While we have demonstrated how model-based RL can overcome model-bias and be much more efficient than model-free counterparts, there is much room to explore how we can improve exploration in model-based RL. For examples, we can use a model ensemble to encourage the policy to explore the state space where the different models disagree, so that more data can be collected to resolve their disagreement and therefore result in a more efficient method. Another enticing direction for future work is to expand the world model such that it can capture not only multi-modal dynamics but also different robot morphologies and domains. Such approach can lead to the ability to extract useful world models from a large-scale robot dataset [40] and quickly be applied to new domains.

Generalization. We demonstrate that our world models can generalize to different start and goal configurations, obstacle positions, obstacle and object shapes, and, in principle, other contexts. These generalization ability require diverse training data that capture most combinations of attributes. One interesting and difficult question is, can we obtain such generalization without such combinatorial data? Can we substitute prior knowledge into the architecture for such purpose? If so, what is the principle?

Long-horizon tasks. We have seen in the final part that training a policy with a hierarchy and abstraction can improve over training a flat, reactive policy. One key question is what is the right way to learn abstractions? How should they change based on the tasks the agent needs to solve? For examples, I do not care about small dusts on my laptop as I type this thesis; however, when my task is to clean the laptop I switch my abstract perception of the world to dust=level details. We may also hypothesize about the possible improvement of building a deeper hierarchy of representations and planning modules.

* * *

Bibliography

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. "Using inaccurate models in reinforcement learning". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 1–8.

[2] Waleed Abdulla. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. `https://github.com/matterport/Mask_RCNN`. 2017.

[3] P. Agrawal et al. "Learning to poke by poking: Experiential learning of intuitive physics". In: 2016.

[4] Pulkit Agrawal et al. "Learning to poke by poking: Experiential learning of intuitive physics". In: *Advances In Neural Information Processing Systems*. 2016.

[5] John Aloimonos, Isaac Weiss, and Amit Bandyopadhyay. "Active vision". In: *International journal of computer vision* 1.4 (1988), pp. 333–356.

[6] Dario Amodei et al. "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565* (2016).

[7] Ankesh Anand et al. "Unsupervised state representation learning in atari". In: *Advances in Neural Information Processing Systems*. 2019, pp. 8769–8782.

[8] Jacob Andreas, Dan Klein, and Sergey Levine. "Modular multitask reinforcement learning with policy sketches". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 166–175.

[9] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[10] K. Arulkumaran et al. "A Brief Survey of Deep Reinforcement Learning". In: *arXiv preprint arXiv:1708.05866* (2017).

[11] Masataro Asai. "Unsupervised Grounding of Plannable First-Order Logic Representation from Images". In: *arXiv preprint arXiv:1902.08093* (2019).

[12] Masataro Asai and Alex Fukunaga. "Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary". In: *AAAI*. 2018.

[13] Masataro Asai and Alex Fukunaga. "Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[14] Christopher G Atkeson and Juan Carlos Santamaria. "A comparison of direct and model-based reinforcement learning". In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. Vol. 4. IEEE. 1997, pp. 3557–3564.

[15] Mohammad Babaeizadeh et al. "Stochastic variational video prediction". In: *arXiv preprint arXiv:1710.11252* (2017).

[16] J Andrew Bagnell and Jeff G Schneider. "Autonomous helicopter control using reinforcement learning policy search methods". In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 2. IEEE. 2001, pp. 1615–1620.

[17] Tim Bailey and Hugh Durrant-Whyte. "Simultaneous localization and mapping (SLAM): Part II". In: *IEEE robotics & automation magazine* 13.3 (2006), pp. 108–117.

[18] Ruzena Bajcsy. "Active perception". In: *Proceedings of the IEEE* 76.8 (1988), pp. 966–1005.

[19] Ruzena Bajcsy, Yiannis Aloimonos, and John K Tsotsos. "Revisiting active perception". In: *Autonomous Robots* 42.2 (2018), pp. 177–196.

[20] Ershad Banijamali et al. "Robust locally-linear controllable embedding". In: *arXiv preprint arXiv:1710.05373* (2017).

[21] Nir Baram, Tom Zahavy, and Shie Mannor. "Spatio-temporal abstractions in reinforcement learning through neural encoding". In: (2016).

[22] Gabriel Barth-Maron et al. "Distributed distributional deterministic policy gradients". In: *arXiv preprint arXiv:1804.08617* (2018).

[23] Peter W Battaglia, Jessica B Hamrick, and Joshua B Tenenbaum. "Simulation as an engine of physical scene understanding". In: *Proceedings of the National Academy of Sciences* 110.45 (2013), pp. 18327–18332.

[24] Matthew P Bell. "Flexible object manipulation". In: (2010).

[25] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[26] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[27] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: (2020). arXiv: 2005.14165 [cs.CL].

[28] Christopher P Burgess et al. "Monet: Unsupervised scene decomposition and representation". In: *arXiv preprint arXiv:1901.11390* (2019).

[29] Devendra Singh Chaplot et al. "Neural Topological SLAM for Visual Navigation". In: *CVPR*. 2020.

[30] Kevin Chen et al. "A Behavioral Approach to Visual Navigation with Graph Localization Networks". In: *Robotics: Science and Systems* (2019).

[31] X. Chen et al. “InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems* (2016), pp. 2172–2180.

[32] Xi Chen et al. “InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2172–2180.

[33] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. *Minimalistic Gridworld Environment for OpenAI Gym*. `https://github.com/maximecb/gym-minigrid`. 2018.

[34] Kurtland Chua et al. “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. In: *Advances in Neural Information Processing Systems*. 2018.

[35] Junyoung Chung et al. “A recurrent latent variable model for sequential data”. In: *Advances in neural information processing systems*. 2015, pp. 2980–2988.

[36] Ignasi Clavera et al. “Model-based reinforcement learning via meta-policy optimization”. In: *Conference on Robot Learning*. 2018.

[37] D. Corneil, W. Gerstner, and J. Brea. “Efficient Model-Based Deep Reinforcement Learning with Variational State Tabulation”. In: *arXiv preprint, arXiv:1802.04325* (2018).

[38] Dane Corneil, Wulfram Gerstner, and Johanni Brea. “Efficient Model-Based Deep Reinforcement Learning with Variational State Tabulation”. In: *arXiv preprint arXiv:1802.04325* (2018).

[39] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[40] Sudeep Dasari et al. “Robonet: Large-scale multi-robot learning”. In: *arXiv preprint arXiv:1910.11215* (2019).

[41] Pieter-Tjerk De Boer et al. “A tutorial on the cross-entropy method”. In: *Annals of operations research* (2005).

[42] Marc Deisenroth and Carl E Rasmussen. “PILCO: A model-based and data-efficient approach to policy search”. In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011, pp. 465–472.

[43] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. *A survey on policy search for robotics*. now publishers, 2013.

[44] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[45] Emily Denton and Rob Fergus. “Stochastic video generation with a learned prior”. In: *arXiv preprint arXiv:1802.07687* (2018).

[46] Emily L Denton and vighnesh Birodkar. “Unsupervised Learning of Disentangled Representations from Video”. In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 4414–4423.

[47] Stefan Depeweg et al. “Learning and policy search in stochastic dynamical systems with bayesian neural networks”. In: *International Conference on Learning Representations (ICLR2017)*. 2017.

[48] Debadeepta Dey et al. “Predicting multiple structured visual interpretations”. In: *International Conference on Computer Vision*. 2015.

[49] E. W. Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[50] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. “Adversarial feature learning”. In: *arXiv preprint arXiv:1605.09782* (2016).

[51] James E Doran and Donald Michie. “Experiments with the graph traverser program”. In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* (1966).

[52] Yilun Du and Karthik Narasimhan. “Task-agnostic dynamics priors for deep reinforcement learning”. In: *International Conference on Machine Learning*. 2019.

[53] Yan Duan et al. “Benchmarking deep reinforcement learning for continuous control”. In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[54] Yan Duan et al. “RL^2: Fast reinforcement learning via slow reinforcement learning”. In: *arXiv preprint arXiv:1611.02779* (2016).

[55] Hugh Durrant-Whyte and Tim Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[56] F. Ebert et al. “Self-Supervised Visual Planning with Temporal Skip Connections”. In: 2017, pp. 344–356.

[57] Frederik Ebert et al. “Visual foresight: Model-based deep reinforcement learning for vision-based robotic control”. In: *arXiv preprint arXiv:1812.00568* (2018).

[58] Bernard Espiau, François Chaumette, and Patrick Rives. “A new approach to visual servoing in robotics”. In: *ieee Transactions on Robotics and Automation* 8.3 (1992), pp. 313–326.

[59] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. “Search on the Replay Buffer: Bridging Planning and Reinforcement Learning”. In: *NeurIPS*. 2019.

[60] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. “Search on the Replay Buffer: Bridging Planning and Reinforcement Learning”. In: *arXiv preprint arXiv:1906.05253* (2019).

[61] Uriel Feige. “A threshold of ln n for approximating set cover”. In: *Journal of the ACM (JACM)* 45.4 (1998), pp. 634–652.

[62] Norm Ferns, Prakash Panangaden, and Doina Precup. “Metrics for Finite Markov Decision Processes.” In: *UAI*. 2004.

[63] Richard E Fikes, Peter E Hart, and Nils J Nilsson. “Learning and executing generalized robot plans”. In: *Readings in Artificial Intelligence*. Elsevier, 1981, pp. 231–249.

[64] C. Finn and S. Levine. “Deep visual foresight for planning robot motion”. In: 2017, pp. 2786–2793.

[65] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *International Conference on Machine Learning*. 2017.

[66] Chelsea Finn, Ian Goodfellow, and Sergey Levine. “Unsupervised learning for physical interaction through video prediction”. In: *Advances In Neural Information Processing Systems*. 2016, pp. 64–72.

[67] Chelsea Finn and Sergey Levine. “Deep visual foresight for planning robot motion”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[68] Chelsea Finn, Sergey Levine, and Pieter Abbeel. “Guided cost learning: Deep inverse optimal control via policy optimization”. In: *International Conference on Machine Learning*. 2016, pp. 49–58.

[69] Chelsea Finn et al. “Deep spatial autoencoders for visuomotor learning”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 512–519.

[70] Paul Fitzpatrick. “First contact: an active vision approach to segmentation”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. Vol. 3. IEEE. 2003, pp. 2161–2166.

[71] Carlos Florensa, Yan Duan, and Pieter Abbeel. “Stochastic neural networks for hierarchical reinforcement learning”. In: *arXiv preprint arXiv:1704.03012* (2017).

[72] Patrick Foo et al. “Do humans integrate routes into a cognitive map? Map-versus landmark-based navigation of novel shortcuts.” In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* (2005).

[73] Meire Fortunato et al. “Noisy networks for exploration”. In: *International Conference on Learning Representations (ICLR2018)*. 2018.

[74] Barbara Frank et al. “Efficient motion planning for manipulation robots in environments with deformable objects”. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 2180–2185.

[75] Justin Fu, Sergey Levine, and Pieter Abbeel. “One-shot learning of manipulation skills with online dynamics adaptation and neural network priors”. In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE. 2016, pp. 4019–4026.

[76] Yarin Gal, Rowan Thomas McAllister, and Carl Edward Rasmussen. “Improving PILCO with Bayesian Neural Network Dynamics Models”. In: *Data-Efficient Machine Learning workshop*. Vol. 951. 2016, p. 2016.

[77] Carlos E Garcia, David M Prett, and Manfred Morari. “Model predictive control: theory and practice—a survey”. In: *Automatica* 25.3 (1989), pp. 335–348.

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

[79] Sabine Gillner and Hanspeter A Mallot. “Navigation and acquisition of spatial knowledge in a virtual maze”. In: *Journal of cognitive neuroscience* (1998).

[80] Robert Givan, Thomas Dean, and Matthew Greig. “Equivalence notions and model minimization in Markov decision processes”. In: *Artificial Intelligence* 147.1-2 (2003), pp. 163–223.

[81] I. Goodfellow et al. “Generative Adversarial Nets”. In: 2014, pp. 2672–2680.

[82] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT Press, 2016.

[83] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[84] Klaus Greff et al. “Multi-object representation learning with iterative variational inference”. In: *arXiv preprint arXiv:1903.00450* (2019).

[85] Emil Julius Gumbel. “Les valeurs extrêmes des distributions statistiques”. In: *Annales de l’institut Henri Poincaré*. Vol. 5. 2. 1935, pp. 115–158.

[86] Abner Guzman-Rivera, Dhruv Batra, and Pushmeet Kohli. “Multiple choice learning: Learning to produce multiple structured outputs”. In: *Advances in Neural Information Processing Systems*. 2012.

[87] Abner Guzman-Rivera et al. “Efficiently enforcing diversity in multi-output structured prediction”. In: *International Conference on Artificial Intelligence and Statistics*. 2014.

[88] David Ha and Jürgen Schmidhuber. “World models”. In: *arXiv preprint arXiv:1803.10122* (2018).

[89] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: 2018.

[90] Danijar Hafner et al. “Dream to Control: Learning Behaviors by Latent Imagination”. In: *International Conference on Learning Representations*. 2020.

[91] Danijar Hafner et al. “Learning latent dynamics for planning from pixels”. In: *International Conference on Machine Learning*. 2019.

[92] Danijar Hafner et al. “Learning latent dynamics for planning from pixels”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2555–2565.

[93] Charles R Hargraves and Stephen W Paris. “Direct trajectory optimization using nonlinear programming and collocation”. In: *Journal of guidance, control, and dynamics* 10.4 (1987), pp. 338–342.

[94] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[95] Karol Hausman et al. “Interactive segmentation of textured and textureless objects”. In: *Handling Uncertainty and Networked Structure in Robot Control*. Springer, 2015, pp. 237–262.

[96] Kaiming He et al. "Mask R-CNN". In: *arxiv preprint arxiv:1703.06870* (2018).

[97] Kaiming He et al. "Momentum contrast for unsupervised visual representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.

[98] Nicolas Heess et al. "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2944–2952.

[99] Christopher Hesse et al. *OpenAI Baselines*. `https://github.com/openai/baselines`. 2017.

[100] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." In: *ICLR* (2017).

[101] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen [in German] Diploma thesis". In: *TU Münich* (1991).

[102] John E Hopcroft, Joseph K Kearney, and Dean B Krafft. "A case study of flexible object manipulation". In: *The International Journal of Robotics Research* 10.1 (1991), pp. 41–50.

[103] Zhiao Huang, Fangchen Liu, and Hao Su. "Mapping state space using landmarks for universal goal reaching". In: *NeurIPS*. 2019.

[104] Brian Ichter, James Harrison, and Marco Pavone. "Learning sampling distributions for robot motion planning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7087–7094.

[105] Brian Ichter and Marco Pavone. "Robot motion planning in learned latent spaces". In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2407–2414.

[106] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax". In: *arXiv preprint arXiv:1611.01144* (2016).

[107] Michael Janner et al. "When to Trust Your Model: Model-Based Policy Optimization". In: *Advances in Neural Information Processing Systems*. 2019.

[108] Nan Jiang. *Notes on State Abstractions*. 2018.

[109] P Jiménez. "Survey on model-based manipulation planning of deformable objects". In: *Robotics and computer-integrated manufacturing* 28.2 (2012), pp. 154–163.

[110] Matthew Johnson et al. "Composing graphical models with neural networks for structured representations and fast inference". In: *NIPS*. 2016, pp. 2946–2954.

[111] Michael I Jordan and David E Rumelhart. "Forward models: Supervised learning with a distal teacher". In: *Cognitive science* 16.3 (1992), pp. 307–354.

[112] Leslie Pack Kaelbling. "Learning to achieve goals". In: *IJCAI*. Citeseer. 1993, pp. 1094–1099.

[113] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Hierarchical planning in the now". In: *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[114] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 1470–1477.

[115] Lukasz Kaiser et al. "Model-based reinforcement learning for atari". In: *International Conference on Learning Representations*. 2020.

[116] Dmitry Kalashnikov et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation". In: *Conference on Robot Learning*. 2018.

[117] Ken Kansky et al. "Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics". In: *arXiv preprint arXiv:1706.04317* (2017).

[118] Ken Kansky et al. "Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics". In: *ICML*. Vol. 70. 2017, pp. 1809–1818.

[119] Tero Karras et al. "Progressive growing of gans for improved quality, stability, and variation". In: *arXiv preprint arXiv:1710.10196* (2017).

[120] Jacqueline Kenney, Thomas Buckley, and Oliver Brock. "Interactive segmentation for manipulation in unstructured environments". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 1377–1382.

[121] Fouad F Khalil and Pierre Payeur. "Dexterous robotic manipulation of deformable objects with multi-sensory feedback-a review". In: *Robot Manipulators Trends and Development*. InTech, 2010.

[122] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[123] Diederik P Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[124] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *International Conference on Learning Representations*. 2014.

[125] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *ICLR*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[126] Thomas Kipf, Elise van der Pol, and Max Welling. "Contrastive learning of structured world models". In: *arXiv preprint arXiv:1911.12247* (2019).

[127] Jonathan Ko et al. "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp". In: *Robotics and Automation, 2007 IEEE International Conference on*. IEEE. 2007, pp. 742–747.

[128] V. Konda and J. N. Tsitsiklis. "Actor-Critic Algorithms". In: 2000.

[129] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "From skills to symbols: Learning symbolic representations for abstract high-level planning". In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 215–289.

[130] Tejas D Kulkarni et al. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems*. 2016, pp. 3675–3683.

[131] Vikash Kumar, Emanuel Todorov, and Sergey Levine. "Optimal control with learned local models: Application to dexterous manipulation". In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 378–383.

[132] Yasuo Kuniyoshi, Masayuki Inaba, and Hirochika Inoue. "Learning by watching: Extracting reusable task knowledge from visual observation of human performance". In: *IEEE transactions on robotics and automation* 10.6 (1994), pp. 799–822.

[133] T. Kurutach et al. "Learning Plannable Representations with Causal InfoGAN". In: *arXiv preprint arXiv:1807.09341* (2018).

[134] Thanard Kurutach et al. "Learning plannable representations with causal infogan". In: *Advances in Neural Information Processing Systems*. 2018, pp. 8733–8744.

[135] Thanard Kurutach et al. "Model-ensemble trust-region policy optimization". In: *International Conference on Learning Representations*. 2018.

[136] Suha Kwak et al. "Unsupervised object discovery and tracking in video collections". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 3173–3181.

[137] Sascha Lange, Thomas Gabel, and Martin Riedmiller. "Batch reinforcement learning". In: *Reinforcement learning*. Springer, 2012, pp. 45–73.

[138] Michael Laskin et al. "Sparse Graphical Memory for Robust Planning". In: *arXiv preprint arXiv:2003.06417* (2020).

[139] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[140] Steven M LaValle. *Rapidly-exploring random trees: A new tool for path planning*. 1998.

[141] Alex X Lee et al. "Stochastic adversarial video prediction". In: *arXiv preprint arXiv:1804.01523* (2018).

[142] Kimin Lee et al. "Confident multiple choice learning". In: *International Conference on Machine Learning*. 2017.

[143] Kimin Lee et al. "Context-aware Dynamics Model for Generalization in Model-Based Reinforcement Learning". In: *International Conference on Machine Learning*. 2020.

[144] Stefan Lee et al. "Stochastic multiple choice learning for training diverse deep ensembles". In: *Advances in Neural Information Processing Systems*. 2016.

[145] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. "DeepMPC: Learning Deep Latent Features for Model Predictive Control." In: *Robotics: Science and Systems*. 2015.

[146] Adam Lerer, Sam Gross, and Rob Fergus. "Learning physical intuition of block towers by example". In: *arXiv preprint arXiv:1603.01312* (2016).

[147] S. Levine et al. "End-to-End Training of Deep Visuomotor Policies". In: *JMLR* 17 (2016).

[148] Sergey Levine and Pieter Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in Neural Information Processing Systems*. 2014, pp. 1071–1079.

[149] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40.

[150] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *arXiv preprint arXiv:1603.02199* (2016).

[151] Lihong Li, Thomas J Walsh, and Michael L Littman. "Towards a Unified Theory of State Abstraction for MDPs." In: *ISAIM*. 2006.

[152] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations*. 2016.

[153] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015). arXiv: 1509.02971.

[154] Kara Liu et al. "Hallucinative Topological Memory for Zero-Shot Visual Planning". In: *arXiv preprint arXiv:2002.12336* (2020).

[155] Miao Liu et al. "The Eigenoption-Critic Framework". In: *arXiv preprint arXiv:1712.04065* (2017).

[156] Carsten Lund and Mihalis Yannakakis. "On the hardness of approximating minimization problems". In: *Journal of the ACM (JACM)* 41.5 (1994), pp. 960–981.

[157] Corey Lynch et al. "Learning latent plans from play". In: *Conference on Robot Learning*. 2020, pp. 1113–1132.

[158] Xiao Ma et al. "Contrastive Variational Model-Based Reinforcement Learning for Complex Observations". In: *arXiv preprint arXiv:2008.02430* (2020).

[159] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[160] Marlos C Machado, Marc G Bellemare, and Michael Bowling. "A laplacian framework for option discovery in reinforcement learning". In: *arXiv preprint arXiv:1703.00956* (2017).

[161] Jan Marian Maciejowski. *Predictive control: with constraints*. Pearson education, 2002.

[162] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. "The concrete distribution: A continuous relaxation of discrete random variables". In: *arXiv preprint arXiv:1611.00712* (2016).

[163] Sridhar Mahadevan and Mauro Maggioni. "Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes". In: *Journal of Machine Learning Research* 8.Oct (2007), pp. 2169–2231.

[164] Jeremy Maitin-Shepard et al. "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 2308–2315.

[165] Shie Mannor et al. "Dynamic abstraction in reinforcement learning via clustering". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 71.

[166] Hermann Mayer et al. "A system for robotic heart surgery that learns to tie knots using recurrent neural networks". In: *Advanced Robotics* 22.13-14 (2008), pp. 1521–1537.

[167] Andrew McCallum. "Overcoming incomplete perception with util distinction memory". In: *Proceedings of the Tenth International Conference on International Conference on Machine Learning*. 1993, pp. 190–196.

[168] Dale McConachie, Mengyao Ruan, and Dmitry Berenson. "Interleaving Planning and Control for Deformable Object Manipulation". In: *International Symposium on Robotics Research (ISRR)*. 2017.

[169] Xiangyun Meng et al. "Scaling local control to large-scale topological navigation". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 672–678.

[170] Stephen Miller et al. "Parametrized shape models for clothing". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 4861–4868.

[171] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014).

[172] Mehdi Mirza et al. "Physically Embedded Planning Problems: New Challenges for Reinforcement Learning". In: *arXiv preprint arXiv:2009.05524* (2020).

[173] Dmytro Mishkin, Alexey Dosovitskiy, and Vladlen Koltun. "Benchmarking Classic and Learned Navigation in Complex 3D Environments". In: *arXiv preprint arXiv:1901.10915* (2019).

[174] Nikhil Mishra, Pieter Abbeel, and Igor Mordatch. "Prediction and Control with Temporal Segment Models". In: *arXiv preprint arXiv:1703.04070* (2017).

[175] Takeru Miyato et al. "Spectral normalization for generative adversarial networks". In: *International Conference on Learning Representations (ICLR)*. 2018.

[176] V. Mnih et al. "Playing Atari with deep reinforcement learning". In: 2013.

[177] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[178] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[179] Takuma Morita et al. "Knot planning from observation". In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 3. IEEE. 2003, pp. 3887–3892.

[180] Jonghwan Mun et al. "Learning to specialize with knowledge distillation for visual question answering". In: *Advances in Neural Information Processing Systems*. 2018.

[181] Ofir Nachum et al. “Data-efficient hierarchical reinforcement learning”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 3303–3313.

[182] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. “Deep online learning via meta-learning: Continual adaptation for model-based RL”. In: *International Conference on Learning Representations*. 2019.

[183] Anusha Nagabandi et al. “Learning to adapt in dynamic, real-world environments through meta-reinforcement learning”. In: *International Conference on Learning Representations*. 2019.

[184] Anusha Nagabandi et al. “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning”. In: *CoRR* abs/1708.02596 (2017).

[185] Ashvin Nair et al. “Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation”. In: *arXiv preprint arXiv:1703.02018* (2017).

[186] Ashvin Nair et al. “Contextual Imagined Goals for Self-Supervised Robotic Learning”. In: *arXiv preprint arXiv:1910.11670* (2019).

[187] Ashvin Nair et al. “Visual Reinforcement Learning with Imagined Goals”. In: *arXiv preprint arXiv:1807.04742* (2018).

[188] Suraj Nair and Chelsea Finn. “Hierarchical Foresight: Self-Supervised Learning of Long-Horizon Tasks via Visual Subgoal Generation”. In: *arXiv preprint arXiv:1909.05829* (2019).

[189] Kumpati S Narendra and Jeyendran Balakrishnan. “Improving transient response of adaptive control systems using multiple models and switching”. In: *IEEE Transactions on Automatic Control*. 1994.

[190] Soroush Nasiriany et al. “Planning with Goal-Conditioned Policies”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 14814–14825.

[191] Andrew Y Ng, Daishi Harada, and Stuart Russell. “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *ICML*. Vol. 99. 1999, pp. 278–287.

[192] Derrick H Nguyen and Bernard Widrow. “Neural networks for self-learning control systems”. In: *IEEE Control systems magazine* 10.3 (1990), pp. 18–23.

[193] Junhyuk Oh, Satinder Singh, and Honglak Lee. “Value prediction network”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6118–6128.

[194] Junhyuk Oh et al. “Action-conditional video prediction using deep networks in atari games”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2863–2871.

[195] Masashi Okada and Tadahiro Taniguchi. “Dreaming: Model-based Reinforcement Learning by Latent Imagination without Reconstruction”. In: *arXiv preprint arXiv:2007.14535* (2020).

[196] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation Learning with Contrastive Predictive Coding”. In: *arXiv preprint arXiv:1807.03748* (2018).

[197] Charles Packer et al. “Assessing generalization in deep reinforcement learning”. In: *arXiv preprint arXiv:1810.12282* (2018).

[198] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International Conference on Machine Learning*. 2013, pp. 1310–1318.

[199] C. Paxton et al. “Visual Robot Task Planning”. In: *arXiv preprint arXiv:1804.00062* (2018).

[200] Karl Pertsch et al. “Long-Horizon Visual Planning with Goal-Conditioned Hierarchical Predictors”. In: *arXiv preprint arXiv:2006.13205* (2020).

[201] J. Peters and S. Schaal. “Policy Gradient Methods for Robotics”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2006, pp. 2219–2225. DOI: 10.1109/IROS.2006.282564.

[202] Lerrel Pinto and Abhinav Gupta. “Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 3406–3413.

[203] Matthias Plappert et al. “Parameter space noise for exploration”. In: *International Conference on Learning Representations (ICLR2018)*. 2018.

[204] Vitchyr Pong et al. “Temporal Difference Models: Model-Free Deep RL for Model-Based Control”. In: *ICLR*. 2018.

[205] Vitchyr Pong et al. “Temporal Difference Models: Model-Free Deep RL for Model-Based Control”. In: *CoRR* abs/1802.09081 (2018). arXiv: 1802.09081. URL: http://arxiv.org/abs/1802.09081.

[206] Ahmed H Qureshi et al. “Motion planning networks”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 2118–2124.

[207] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[208] Kate Rakelly et al. “Efficient off-policy meta-reinforcement learning via probabilistic context variables”. In: *International Conference on Machine Learning*. 2019.

[209] Carl Edward Rasmussen, Malte Kuss, et al. “Gaussian Processes in Reinforcement Learning.” In: *NIPS*. Vol. 4. 2003, p. 1.

[210] Alex Ray, Joshua Achiam, and Dario Amodei. *Benchmarking Safe Exploration in Deep Reinforcement Learning*. 2019.

[211] Alex Ray, Joshua Achiam, and Dario Amodei. “Benchmarking safe exploration in deep reinforcement learning”. In: *arXiv preprint arXiv:1910.01708* (2019).

[212] Ran Raz and Shmuel Safra. “A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP”. In: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 1997, pp. 475–484.

[213] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. "Generating diverse high-fidelity images with vq-vae-2". In: *Advances in Neural Information Processing Systems*. 2019, pp. 14866–14876.

[214] Martin Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning*. Springer. 2005, pp. 317–328.

[215] Martin Riedmiller et al. "Learning by Playing-Solving Sparse Reward Tasks from Scratch". In: *arXiv preprint arXiv:1802.10567* (2018).

[216] Sebastian Risi and Kenneth O Stanley. "Deep neuroevolution of recurrent and discrete world models". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019, pp. 456–462.

[217] Reuven Rubinstein. "The cross-entropy method for combinatorial and continuous optimization". In: *Methodology and computing in applied probability* 1.2 (1999), pp. 127–190.

[218] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[219] S. J. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach (3rd edition)*. Pearson Education, 2010.

[220] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)* Pearson Education, 2010.

[221] Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. "Meta reinforcement learning with latent variable gaussian processes". In: *Conference on Uncertainty in Artificial Intelligence*. 2018.

[222] Mitul Saha and Pekka Isto. "Motion planning for robotic manipulation of deformable linear objects". In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE. 2006, pp. 2478–2484.

[223] Tim Salimans et al. "Improved Techniques for Training GANs". In: *arxiv preprint arxiv:1606.03498* (2016).

[224] Tim Salimans et al. "Improved techniques for training gans". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.

[225] Alvaro Sanchez-Gonzalez et al. "Graph networks as learnable physics engines for inference and control". In: *International Conference on Machine Learning*. 2018.

[226] Adam Santoro et al. "A simple neural network module for relational reasoning". In: *Advances in neural information processing systems*. 2017, pp. 4974–4983.

[227] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. "Semi-parametric Topological Memory for Navigation". In: *ICLR* (2019).

[228] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. "Semi-parametric topological memory for navigation". In: *arXiv preprint arXiv:1803.00653* (2018).

[229] Tom Schaul et al. "Universal Value Function Approximators". In: *ICML*. 2015.

[230] Tom Schaul et al. "Universal value function approximators". In: *International conference on machine learning*. 2015, pp. 1312–1320.

[231] Juergen Schmidhuber and Rudolf Huber. "Learning to generate artificial fovea trajectories for target detection". In: *International Journal of Neural Systems* 2.01n02 (1991), pp. 125–134.

[232] Jeff G Schneider. "Exploiting model uncertainty estimates for safe dynamic control learning". In: *Advances in neural information processing systems*. 1997, pp. 1047–1053.

[233] Julian Schrittwieser et al. "Mastering atari, go, chess and shogi by planning with a learned model". In: *arXiv preprint arXiv:1911.08265* (2019).

[234] John Schulman et al. "A case study of trajectory transfer through non-rigid registration for a simplified suturing scenario". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 4111–4117.

[235] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *International Conference on Learning Representations (ICLR2016)*. 2016.

[236] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* (2017). arXiv: 1707.06347.

[237] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[238] John Schulman et al. "Trust region policy optimization". In: *CoRR, abs/1502.05477* (2015).

[239] Younggyo Seo et al. "Trajectory-wise Multiple Choice Learning for Dynamics Generalization in Reinforcement Learning". In: *arXiv preprint arXiv:2010.13303* (2020).

[240] Wenling Shang et al. "Learning World Graphs to Accelerate Hierarchical Reinforcement Learning". In: *arXiv e-prints*, arXiv:1907.00664 (July 2019), arXiv:1907.00664. arXiv: 1907.00664 [cs.LG].

[241] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[242] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359.

[243] Duncan I Simester, Peng Sun, and John N Tsitsiklis. "Dynamic catalog mailing policies". In: *Management science* 52.5 (2006), pp. 683–696.

[244] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning structured output representation using deep conditional generative models". In: *Advances in neural information processing systems*. 2015, pp. 3483–3491.

[245] Casper Kaae Sønderby et al. "Amortised MAP Inference for Image Super-resolution". In: *arxiv preprint arxiv:1610.04490* (2017).

[246] Shuran Song et al. "Semantic scene completion from a single depth image". In: *CVPR*. 2017.

[247] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. "Curl: Contrastive unsupervised representations for reinforcement learning". In: *arXiv preprint arXiv:2004.04136* (2020).

[248] Aravind Srinivas et al. "Option Discovery in Hierarchical Reinforcement Learning using Spatio-Temporal Clustering". In: *arXiv preprint arXiv:1605.05359* (2016).

[249] Aravind Srinivas et al. "Universal Planning Networks". In: *arXiv preprint arXiv:1804.00645* (2018).

[250] Siddharth Srivastava et al. "Combined task and motion planning through an extensible planner-independent interface layer". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 639–646.

[251] Karl Stratos and Sam Wiseman. "Learning Discrete Structured Representations by Adversarially Maximizing Mutual Information". In: *arXiv preprint arXiv:2004.03991* (2020).

[252] Jaeyong Sung, Seok Hyun Jin, and Ashutosh Saxena. "Robobarista: Object part based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds". In: *Robotics Research*. Springer, 2018, pp. 701–720.

[253] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM SIGART Bulletin* 2.4 (1991), pp. 160–163.

[254] Richard S Sutton. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Proceedings of the seventh international conference on machine learning*. 1990, pp. 216–224.

[255] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.

[256] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[257] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[258] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[259] Aviv Tamar et al. "Value iteration networks". In: *NIPS*. 2016, pp. 2146–2154.

[260] Yuval Tassa, Tom Erez, and Emanuel Todorov. "Synthesis and stabilization of complex behaviors through online trajectory optimization". In: *International Conference on Intelligent Robots and Systems*. 2012.

[261] Garrett Thomas et al. “Learning Robotic Assembly from CAD”. In: *arXiv preprint arXiv:1803.07635* (2018).

[262] Kai Tian et al. “Versatile multiple choice learning and its application to vision computing”. In: *International Conference on Computer Vision*. 2019.

[263] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control.” In: *IROS*. IEEE, 2012, pp. 5026–5033.

[264] Edward C Tolman. “Cognitive maps in rats and men.” In: *Psychological review* (1948).

[265] Marc Toussaint. “Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning.” In: *IJCAI*. 2015, pp. 1930–1936.

[266] Mauro Vallati et al. “The 2014 international planning competition: Progress and trends”. In: *Ai Magazine* 36.3 (2015), pp. 90–98.

[267] Ashish Vaswani et al. “Tensor2Tensor for Neural Machine Translation”. In: *CoRR* abs/1803.07416 (2018). URL: http://arxiv.org/abs/1803.07416.

[268] Rishi Veerapaneni et al. “Entity abstraction in visual model-based reinforcement learning”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 1439–1456.

[269] Arun Venkatraman et al. “Improved Learning of Dynamics Models for Control”. In: *2016 International Symposium on Experimental Robotics*. Ed. by Dana Kulić et al. Cham: Springer International Publishing, 2017, pp. 703–713. ISBN: 978-3-319-50115-4.

[270] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (2019), pp. 350–354.

[271] Hidefumi Wakamatsu, Eiji Arai, and Shinichi Hirai. “Knotting/unknotting manipulation of deformable linear objects”. In: *The International Journal of Robotics Research* 25.4 (2006), pp. 371–395.

[272] Angelina Wang et al. “Learning Robotic Manipulation through Visual Planning and Acting”. In: *arXiv preprint arXiv:1905.04411* (2019).

[273] Ning Wang et al. “Unsupervised deep tracking”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 1308–1317.

[274] Ranxiao Frances Wang and Elizabeth S Spelke. “Human spatial representation: Insights from animals”. In: *Trends in cognitive sciences* (2002).

[275] Tingwu Wang and Jimmy Ba. “Exploring model-based planning with policy networks”. In: *International Conference on Learning Representations*. 2020.

[276] William Wang et al. “Safer Classification by Synthesis”. In: *arXiv preprint arXiv:1711.08534* (2017).

[277] Zi Wang et al. “Active model learning and diverse action sampling for task and motion planning”. In: *International Conference on Intelligent Robots and Systems (IROS)*. 2018. URL: http://lis.csail.mit.edu/pubs/wang-iros18.pdf.

[278] M. Watter et al. “Embed to control: A locally linear latent dynamics model for control from raw images”. In: *Advanced in Neural Information Processing Systems (NIPS)* (2015).

[279] Manuel Watter et al. “Embed to control: A locally linear latent dynamics model for control from raw images”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2746–2754.

[280] Nicholas Watters et al. “Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration”. In: *arXiv preprint arXiv:1905.09275* (2019).

[281] Nicholas Watters et al. “Visual interaction networks”. In: *arXiv preprint arXiv:1706.01433* (2017).

[282] William Whitney et al. “Dynamics-aware Embeddings”. In: *International Conference on Learning Representations*. 2020.

[283] Jiajun Wu et al. “Learning to see physics via visual de-animation”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 152–163.

[284] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. “ViZDoom Competitions: Playing Doom from Pixels”. In: *IEEE Transactions on Games* (2018).

[285] Fanyi Xiao and Yong Jae Lee. “Track and segment: An iterative unsupervised approach for video object proposals”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 933–942.

[286] Wilson Yan et al. “Learning Predictive Representations for Deformable Objects Using Contrastive Estimation”. In: *arXiv preprint arXiv:2003.05436* (2020).

[287] Amy Zhang et al. “Composable planning with attributes”. In: *ICML* (2018).

[288] Shengjia Zhao et al. “Bias and Generalization in Deep Generative Models: An Empirical Study”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 10792–10801.

[289] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. “Environment probing interaction policies”. In: *International Conference on Learning Representations*. 2019.

[290] Jun-Yan Zhu et al. “Toward multimodal image-to-image translation”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 465–476.

Appendix A

Deep Model-based Reinforcement Learning

A.1 Experiment details

Model-Ensemble Trust-Region Policy Optimization

Our algorithm can be broken down into three parts: data collection, model learning, and policy learning. We describe the numerical details for each part below.

Data collection

In each outer iteration, we use the stochastic policy to collect 3000 timesteps of real world data for every environment, except Humanoid in which we collect 6000 timesteps. At the beginning of every roll-out we sample the policy standard deviation randomly from $\mathcal{U}[0.0, 3.0]$, and we keep the value fixed throughout the episode. Furthermore, we perturb the policy's parameters by adding white Gaussian noise with standard deviation proportional to the absolute difference between the current parameters and the previous one [203, 73]. Finally, we split the collected data using a 2-to-1 ratio for training and validation datasets.

Model Learning

We represent the dynamics model with a 2-hidden-layer feed-forward neural network with hidden sizes 1024-1024 and ReLU nonlinearities. We train the model with the Adam optimizer with learning rate 0.001 using a batch size of 1000. The model is trained until the validation loss has not decreased for 25 passes over the entire training dataset (we validate the training every 5 passes).

Policy Learning

We represent the policy with a 2-hidden-layer feed-forward neural network with hidden sizes 32-32 and tanh nonlinearities for all the environments, except Humanoid, in which we use the hidden sizes 100-50-25. The policy is trained with TRPO on the learned models using initial standard deviation

1.0, step size δ_{KL} 0.01, and batch size 50000. If the policy fails the validation for 25 updates (we do the validation every 5 updates), we stop the learning and repeat the overall process.

Environment details

The environments we use are adopted from rllab [53]. The reward functions $r_t(s_t, a_t)$ and optimization horizons are described below:

Environments	Reward functions	Horizon
Swimmer	s_t^{vel} - $0.005\|a_t\|_2^2$	200
Snake	s_t^{vel} - $0.005\|a_t\|_2^2$	200
Hopper	s_t^{vel} - 0.005 $\|a_t\|_2^2$ $-10\max(0.45 - s_t^{height}, 0)$ - $10\sum(\max(s_t - 100, 0))$	100
Half Cheetah	s_t^{vel} - $0.05\|a_t\|_2^2$	100
Ant	s_t^{vel} - $0.005\|a_t\|_2^2$ + 0.05	100
Humanoid	$(s_t^{head} - 1.5)^2$+ $\|a_t\|_2^2$	100

Note that in Hopper we relax the early stopping criterion to a soft constraint in reward function, whereas in Ant we early stop when the center of mass long z-axis is outside [0.2, 1.0] and have a survival reward when alive.

The state in each environment is composed of the joint angles, the joint velocities, and the cartesian position of the center of mass of a part of the simulated robot. We are not using the contact information, which make the environments effectively POMDPs in Half Cheetah, Ant, Hopper and Humanoid. We also eliminate the redundancies in the state space in order to avoid infeasible states in the prediction.

Baselines

In Section 2.5 we compare our method against TRPO, PPO, DDPG, and SVG. For every environment we represent the policy with a feed-forward neural network of the same size, horizon, and discount factor as the ones specified in the Appendix A.1. In the following we provide the hyper-parameters details:

Trust Region Policy Optimization [235]. We used the implementation of [53] with a batch size of 50000, and we train the policies for 1000 iterations. The step size δ_{KL} that we used in all the experiments was of 0.05.

Proximal Policy Optimization [236]. We referred to the implementation of [99]. The policies were trained for 10^7 steps using the default hyper-parameters across all tasks.

Deep Deterministic Policy Gradient [153]. We also use the implementation of [99] using a number epochs of 2000, the rest of the hyper-parameters used were the default ones.

Stochastic Value Gradient [98]. We parametrized the dynamics model as a feed-forward neural network of two hidden layers of 512 units each and ReLU non-linearities. The model was

trained after every episode with the data available in the replay buffer, using the Adam optimizer with a learning rate of 10^{-4}, and batch size of 128. We additionally clipped the gradient we the norm was larger than 10.

A.2 Overfitting

We show that replacing the ensemble with just one model leads to the policy overoptimization. In each outer iteration, we see that at the end of the policy optimization step the estimated performance increases while the real performance is in fact decreasing (see figure A.1).

Figure A.1: Predicted and real performances during the training process using our approach with one model instead of an ensemble in Swimmer. The policy overfits to the dynamics model which degrades the real performance.

A.3 Real-time complexity

We provide wall clock time for the ME-TRPO results from figure 2.2 in the table below:

Environments	Run time (in 1000s)
Swimmer	35.3 ± 3.1
Snake	60.8 ± 4.7
Hopper	183.6 ± 10.2
Half Cheetah	103.7 ± 5.2
Ant	395.2 ± 67.1
Humanoid	362.1 ± 20.5

These experiments were performed on Amazon EC2 using 1 NVIDIA K80 GPU, 4 vCPUs, and 61 GB of memory.

Note that the majority of run time is spent on training model ensemble. However, our algorithm allows this to be simply parallelized across multiple GPUs. This could potentially yield multiple-fold speed-up from our results.

A.4 Ablation study

We further provide a series of ablation experiments to characterize the importance of the two main regularization components of our algorithm: the ensemble validation and the ensemble sampling techniques. In these experiments, we make only one change at a time to ME-TRPO with 5 models.

Ensemble sampling methods

We explore several ways to simulate the trajectories from the model ensemble. At a current state and action, we study the effect of simulating the next step given by: (1) sampling randomly from the different models (step_rand), (2) a normal distribution fitted from the predictions (model_mean_std), (3) the mean of the predictions (model_mean), (4) the median of the predictions (model_med), (5) the prediction of a fixed model over the entire episode (i.e., equivalent to averaging the gradient across all simulations) (eps_rand), and (6) sampling from one model (one_model).

The results in Figure A.2 provide evidence that using the next step as the prediction of a randomly sampled model from our ensemble is the most robust method across environments. In fact, using the median or the mean of the predictions does not prevent overfitting; this effect is shown in the HalfCheetah environment where we see a decrease of the performance in latter iteration of the optimization process. Using the gradient average (5) also provides room for the policy to overfit to one or more models. This supports that having an estimate of the model uncertainty, such as in (1) and (2), is the principled way to avoid overfitting the learned models.

Ensemble validation

Finally, we provide a study of the different ways for validating the policy. We compare the following techniques: (1) using the real performance (i.e., using an oracle) (real), (2) using the average return in the trpo roll-outs (trpo_mean), (3) stopping the policy after 50 gradient updates (no_early_50), (4) or after 5 gradient updates (no_early_5), (5) using one model to predict the performances (one_model), and (6) using an ensemble of models (ensemble). The experiments are designed to use the same number of models and hyper-parameters for the other components of the algorithm.

In Figure A.3 we can see the effectiveness of each approach. It is noteworthy that having an oracle of the real performance is not the best approach. Such validation is over-cautious, and does not give room for exploration resulting in a poor trained dynamics model. Stopping the gradient after a fixed number of updates results in good performance if the right number of updates is set. This burdens the hyper-parameter search with one more hyper-parameter. On the other hand,

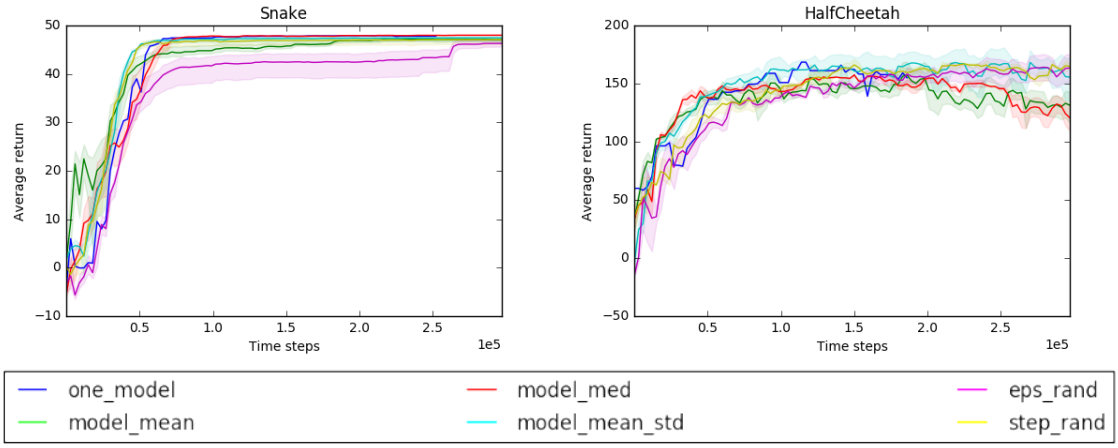


Figure A.2: Comparison among different sampling techniques for simulating roll-outs. By sampling each step from a different model, we prevent overfitting and enhance the learning performance (Best viewed in color).

using the ensemble of models has good performance across environments without adding extra hyper-parameters.

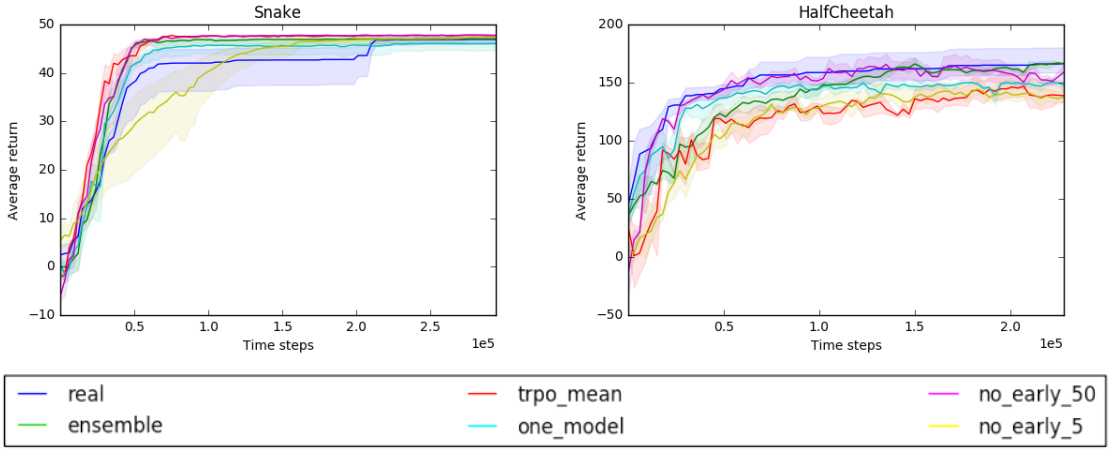


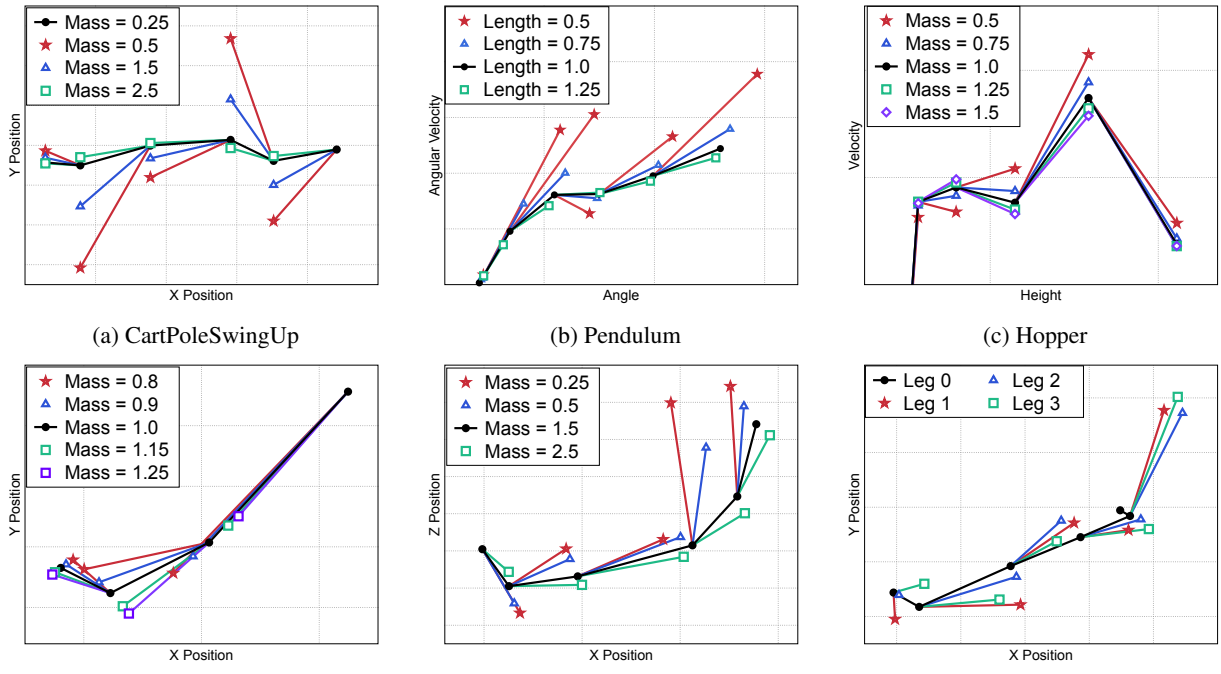
Figure A.3: Comparison among validation techniques. The ensemble of models yields to good performance across environments (Best viewed in color).

Appendix B

Dyanamics Adaptation

B.1 Details on experimental setups

Environments



(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

Figure B.1: Visualization of multi-modal distribution in (a) CartPoleSwingUp, (b) Pendulum, (c) Hopper, (d) SlimHumanoid, (e) HalfCheetah, and (f) CrippledAnt environments. We first collect trajectory from the default environment (black colored transitions in figures) and visualize the next states obtained by applying the same action to the same state with different environment parameters. One can observe that transition dynamics follow multi-modal distributions.

CartPoleSwingUp. For CartPoleSwingUp environments, we use open source implementation of CartPoleSwingUp[1], which is the modified version of original CartPole environments from OpenAI Gym [26]. The objective of CartPoleSwingUp is to swing up the pole by moving a cart and keep the pole upright within 500 time steps. For our experiments, we modified the mass of cart and pole within the set of $\{0.25, 0.5, 1.5, 2.5\}$ and evaluated the generalization performance in unseen environments with a mass of $\{0.1, 0.15, 2.75, 3.0\}$. We visualize the transitions in Figure B.1a.

Pendulum. We use the Pendulum environments from the OpenAI Gym [26]. The objective of Pendulum is to swing up the pole and keep the pole upright within 200 time steps. We modified the length of pendulum within the set of $\{0.5, 0.75, 1.0, 1.25\}$ and evaluated the generalization performance in unseen environments with a length of $\{0.25, 0.375, 1.5, 1.75\}$. We visualize the transitions in Figure B.1b.

Hopper. We use the Hopper environments from MuJoCo physics engine [263]. The objective of Hopper is to move forward as fast as possible while minimizing the action cost within 500 time steps. We modified the mass of a hopper robot within the set of $\{0.5, 0.75, 1.0, 1.25, 1.5\}$ and evaluated the generalization performance in unseen environments with a mass of $\{0.25, 0.375, 1.75, 2.0\}$. We visualize the transitions in Figure B.1c.

SlimHumanoid. We use the modified version of Humanoid environments from MuJoCo physics engine [263][2]. The objective of SlimHumanoid is to move forward as fast as possible while minimizing the action cost within 1000 time steps. We modified the mass of a humanoid robot within the set of $\{0.8, 0.9, 1.0, 1.15, 1.25\}$ and evaluated the generalization performance in unseen environments with a mass of $\{0.6, 0.7, 1.5, 1.6\}$. We visualize the transitions in Figure B.1d.

HalfCheetah. We use the HalfCheetah environments from MuJoCo physics engine [263]. The objective of HalfCheetah is to move forward as fast as possible while minimizing the action cost within 1000 time steps. We modified the mass of a halfcheetah robot within the set of $\{0.25, 0.5, 1.5, 2.5\}$ and evaluated the generalization performance in unseen environments with a mass of $\{0.1, 0.15, 2.75, 3.0\}$. We visualize the transitions in Figure B.1e.

CrippledAnt. We use the modified version of Ant environments from MuJoCo physics engine [263][3]. The objective of CrippledAnt is to move forward as fast as possible while minimizing the action cost within 1000 time steps. We randomly crippled one of three legs in a ant robot and evaluated the generalization performance by crippling remaining one leg of the ant robot. We visualize the transitions in Figure B.1f.

[1]We use implementation available at `https://github.com/angelolovatto/gym-cartpole-swingup`

[2]We use implementation available at `https://github.com/WilsonWangTHU/mbbl`

[3]We use implementation available at `https://github.com/iclavera/learning_to_adapt`

Training

We train dynamics models for 10 iterations for all experiments. Each iteration consists of data collection and updating model parameters. First, we collect 10 trajectories with MPC controller from environments with varying environment parameters. For planning via MPC, we use the cross entropy method with 200 candidate actions and 5 iterations to optimize action sequences with horizon 30. We also use $N = 10$ for the number of past transitions in adaptive planning (3.3). Then we update the model parameters with Adam optimizer of learning rate 0.001. To effectively increase the prediction ability of each prediction head before specialized prediction heads emerge, we train all prediction heads independently on all environments for initial 3 iterations, instead of training each prediction head seperately from the first iteration. Except for Pendulum environments, we update model parameters for 50 epochs. For Pendulum environments, we update model parameters for 5 epochs due to the short horizon length of the task.

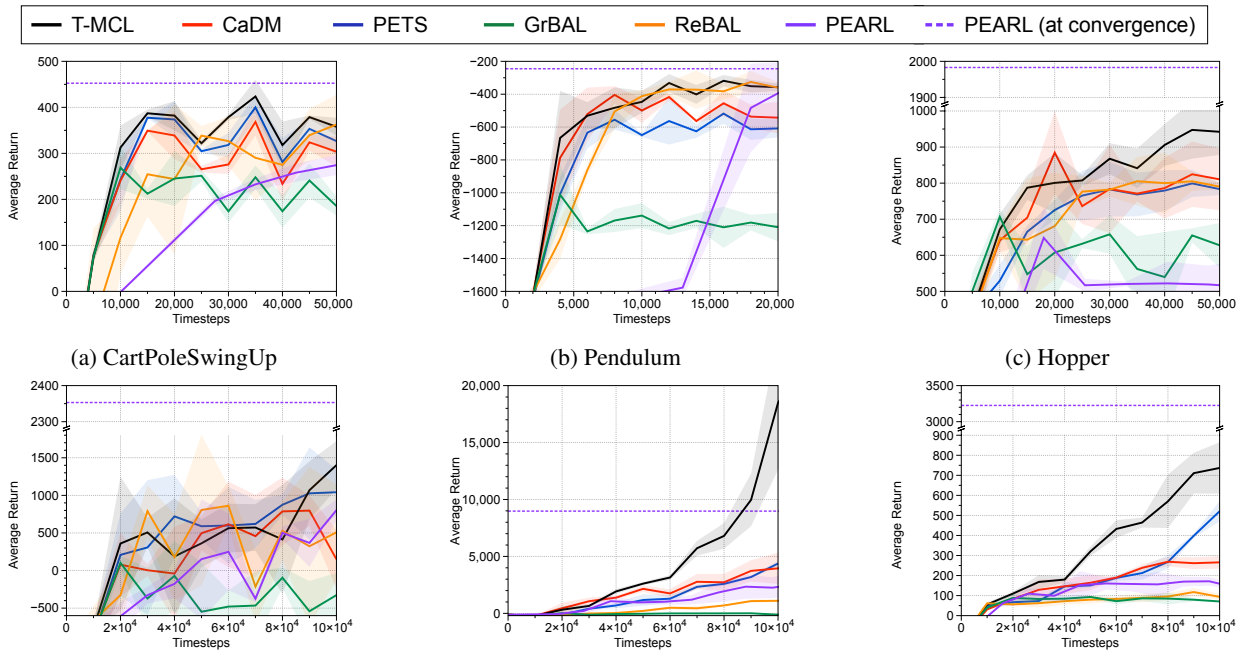
Architecture

Following [34], our backbone network is modeled as multi-layer perceptrons (MLPs) with 4 hidden layers of 200 units each and Swish activations. Each prediction head is modeled as Gaussian, in which the mean and variance are parameterized by a single linear layer which takes the output vector of the backbone network as an input. We use $H = 3$ for the number of prediction heads and $M = 10$ for the size of trajectory segment in (3.2). We remark that hyperparameters H and M are selected based on trajectory assignments, i.e., how distinctively trajectories are assigned to each prediction head. We use an ensemble of 5 multi-headed dynamics models that are independently trained on entire training environments and 20 particles for trajectory sampling. Note that we do not use bootstrap models. For context-conditional dynamics model, following [143], a context encoder is modeled as MLPs with 3 hidden layers that produce a 10-dimensional vector. Then, this context vector is concatenated with the output vector of a backbone network.

Auxiliary prediction losses for context learning

Here, we provide a more detailed explanation of how we implemented various prediction losses proposed in [143]. These losses are proposed to force context latent vector to be useful for predicting both (a) forward dynamics and (b) backward dynamics. Specifically, we compute the proposed forward and backward prediction losses by substituting log-likelihood loss for proposed trajectory-wise oracle loss (3.2). In order to further remove the computational cost of optimization, we first compute the assignments, i.e., h^* for each transition, through the entire dataset and optimize the forward and backward prediction losses with mini-batches.

B.2 Learning curves

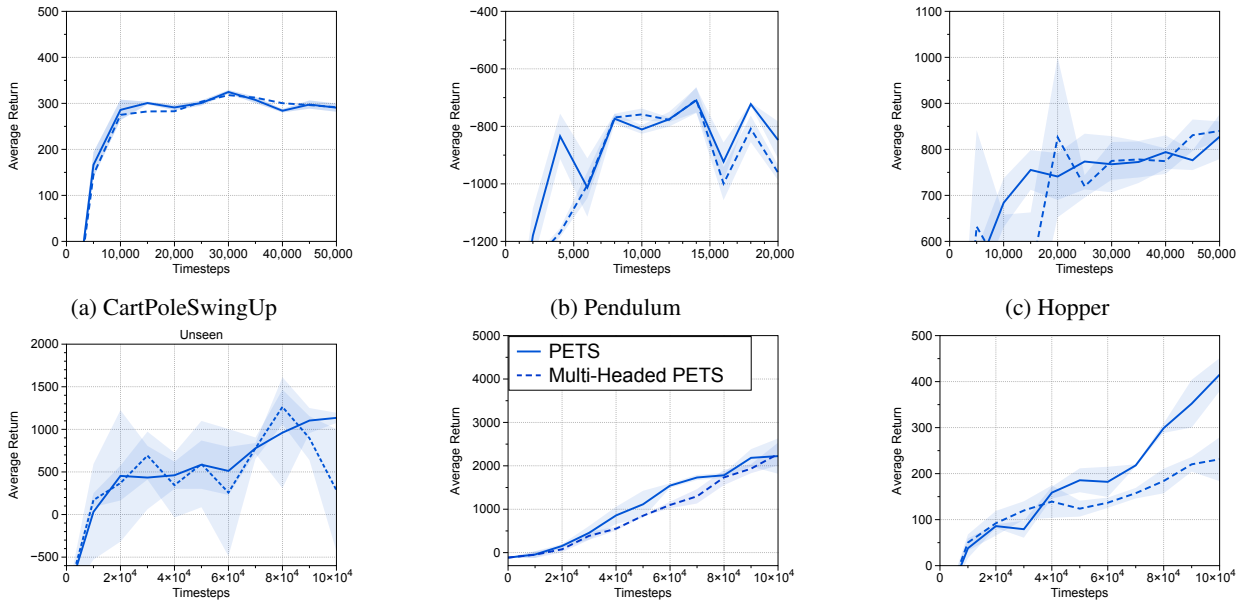


(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

Figure B.2: The average returns of trained dynamics models on training environments. Dotted lines indicate performance at convergence. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.

B.3 Effects of multi-headed dynamics model

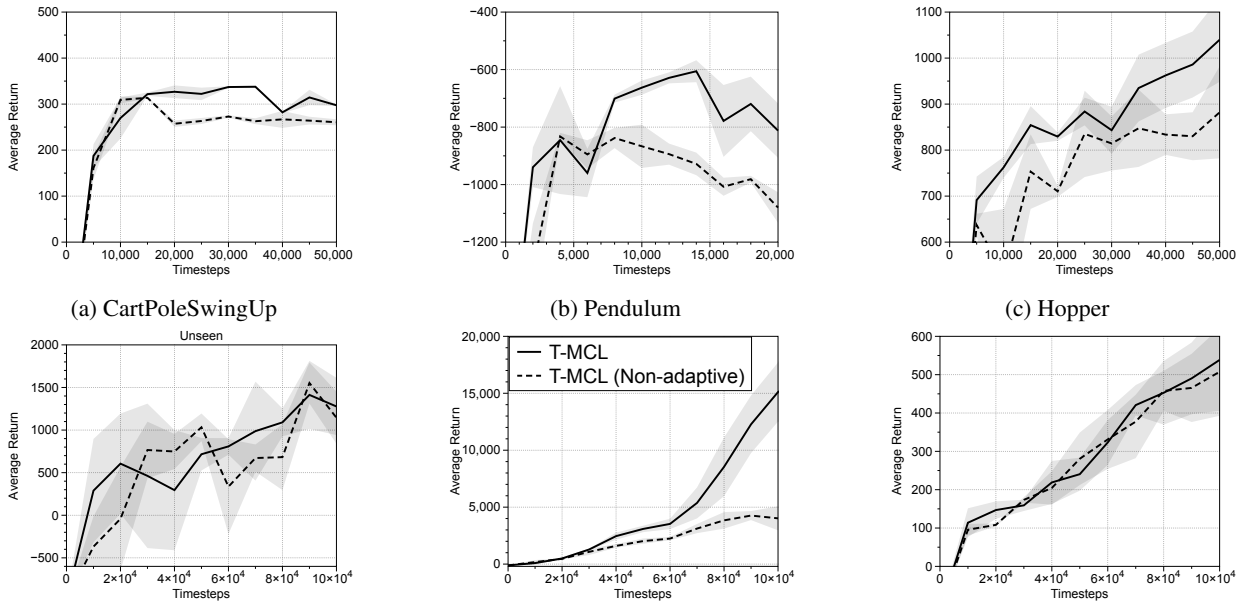


(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

Figure B.3: Generalization performance of PETS and Multi-Headed PETS on unseen (a) CartPoleSwingUp, (b) Pendulum, (c) Hopper, (d) SlimHumanoid, (e) HalfCheetah, and (f) CrippledAnt environments. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.

B.4 Effects of adaptive planning

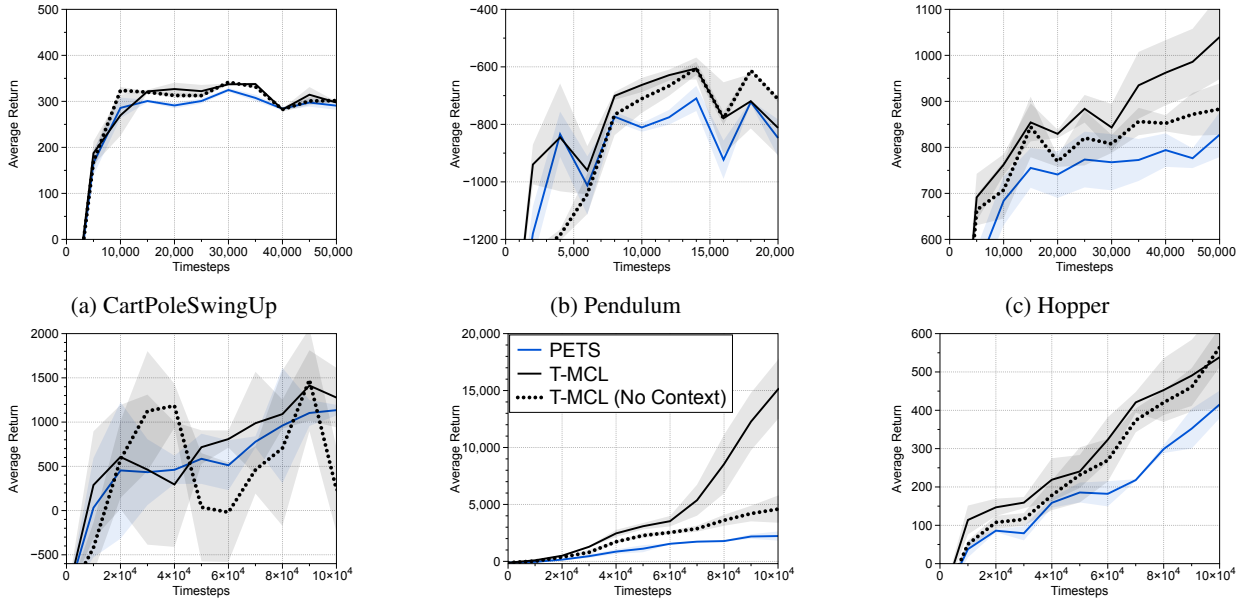


(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

Figure B.4: Generalization performance of employing adaptive planning and non-adaptive planning on unseen (a) CartPoleSwingUp, (b) Pendulum, (c) Hopper, (d) SlimHumanoid, (e) HalfCheetah, and (f) CrippledAnt environments. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.

B.5 Effects of context learning

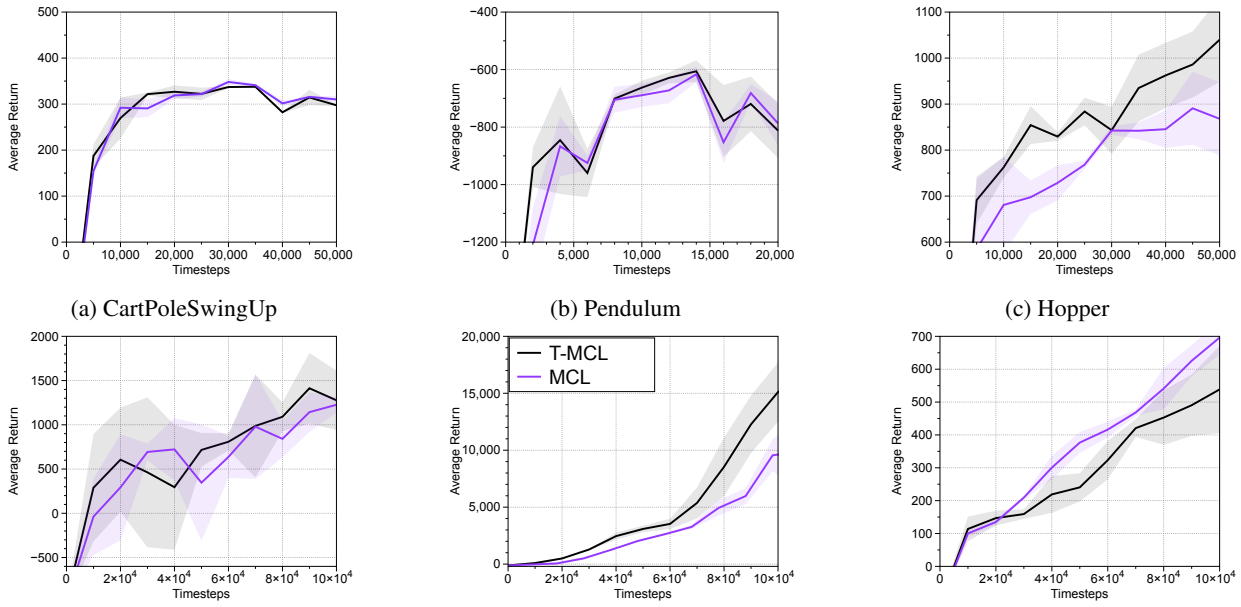


(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

Figure B.5: Generalization performance of trained dynamics models on unseen (a) CartPoleSwingUp, (b) Pendulum, (c) Hopper, (d) SlimHumanoid, (e) HalfCheetah, and (f) CrippledAnt environments. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.

B.6 Effects of trajectory-wise loss



(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

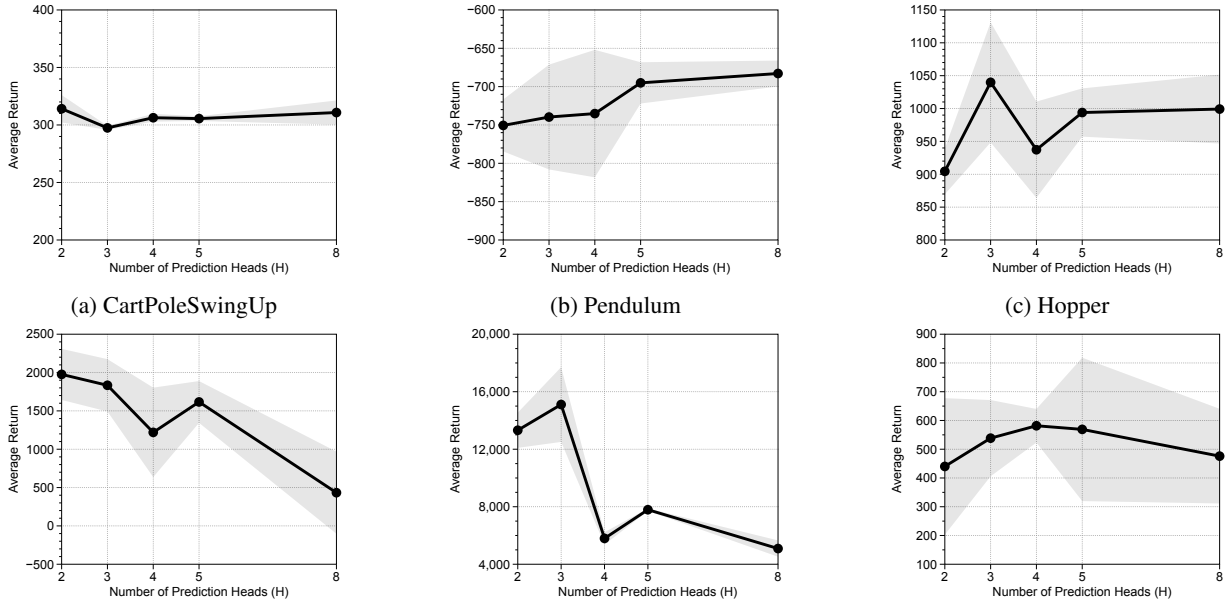
Figure B.6: Generalization performance of dynamics models trained with MCL and T-MCL on unseen (a) CartPoleSwingUp, (b) Pendulum, (c) Hopper, (d) SlimHumanoid, (e) HalfCheetah, and (f) CrippledAnt environments. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.

B.7 Effects of hyperparameters

Number of prediction heads

Horizon of trajectory-wise oracle loss

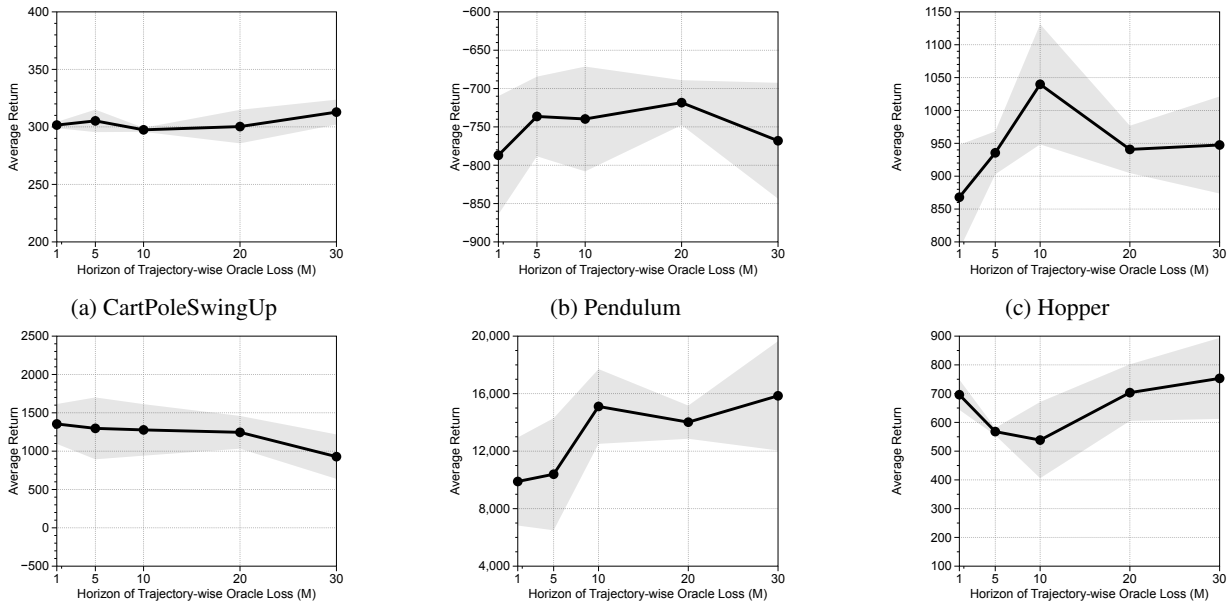
Horizon of adaptive planning



(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

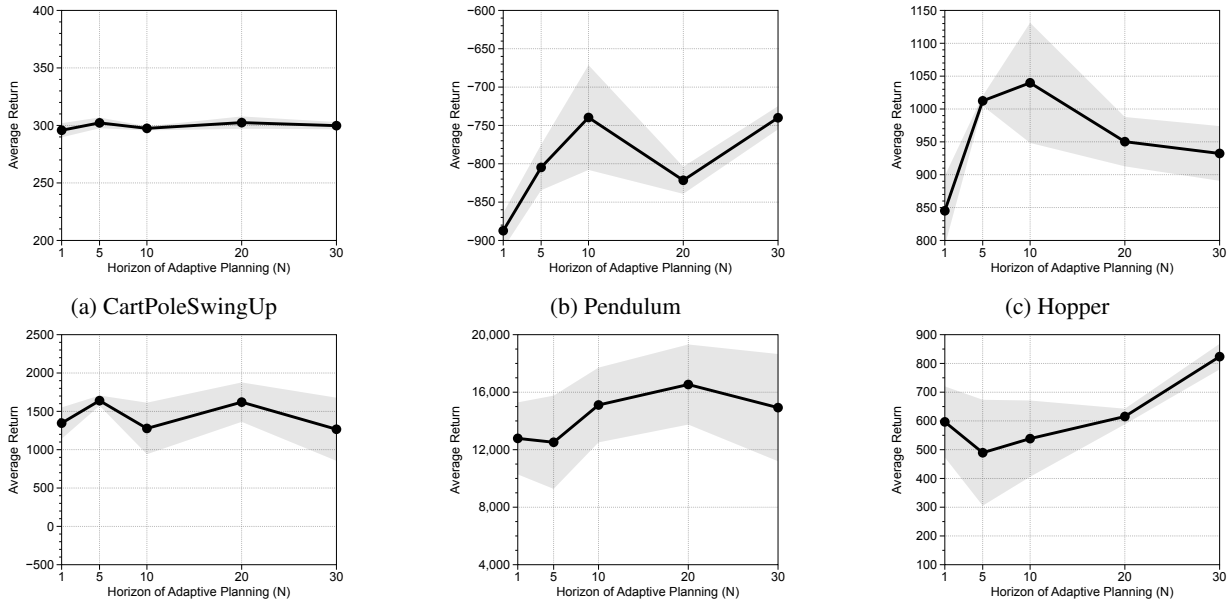
Figure B.7: Generalization performance of dynamics models trained with MCL on unseen (a) CartPoleSwingUp, (b) Pendulum, (c) Hopper, (d) SlimHumanoid, (e) HalfCheetah, and (f) CrippledAnt environments with varying number of prediction heads. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.



(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

Figure B.8: Generalization performance of dynamics models trained with MCL on unseen (a) CartPoleSwingUp, (b) Pendulum, (c) Hopper, (d) SlimHumanoid, (e) HalfCheetah, and (f) CrippledAnt environments with varying horizon of trajectory-wise oracle loss. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.



(a) CartPoleSwingUp (b) Pendulum (c) Hopper

(d) SlimHumanoid (e) HalfCheetah (f) CrippledAnt

Figure B.9: Generalization performance of dynamics models trained with MCL on unseen (a) CartPoleSwingUp, (b) Pendulum, (c) Hopper, (d) SlimHumanoid, (e) HalfCheetah, and (f) CrippledAnt environments with varying horizon of adaptive planning. The solid lines and shaded regions represent mean and standard deviation, respectively, across three runs.

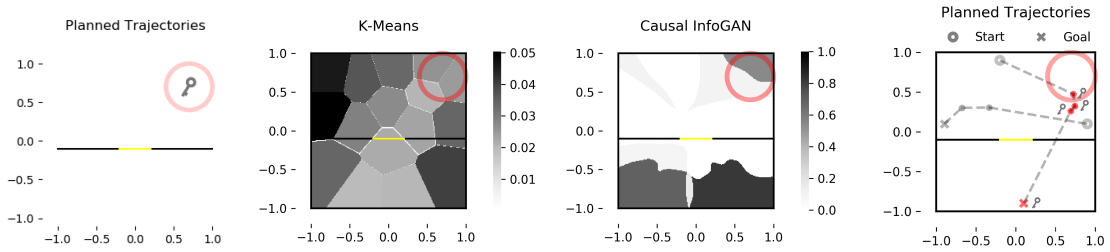
Appendix C

Bridging Learning and Planning

C.1 Additional results

Door-key Results

In Figure C.1 we show similar results for the door-key domain. When the key observation was scaled to $0/\epsilon$, standard clustering methods did not separate states with the key and without key to different clusters. CIGAN, on the other hand, learned a binary predicate for holding the key, and learned that obtaining the key happens in the correct position.

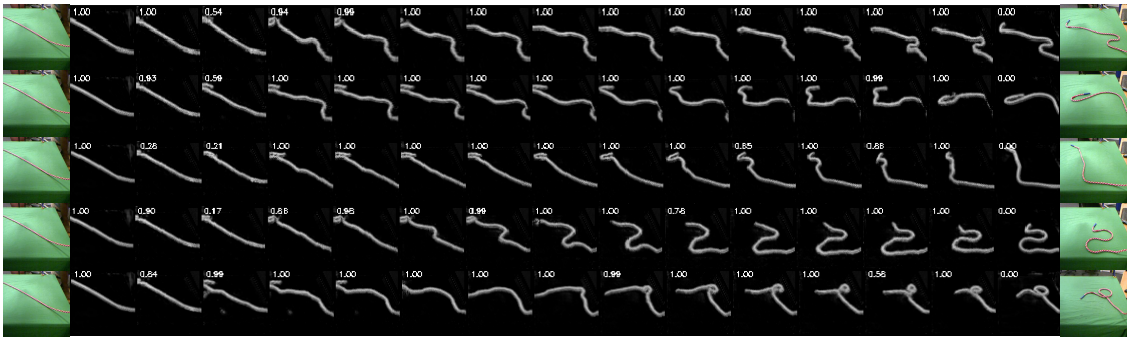


(a) (b) (c) (d)

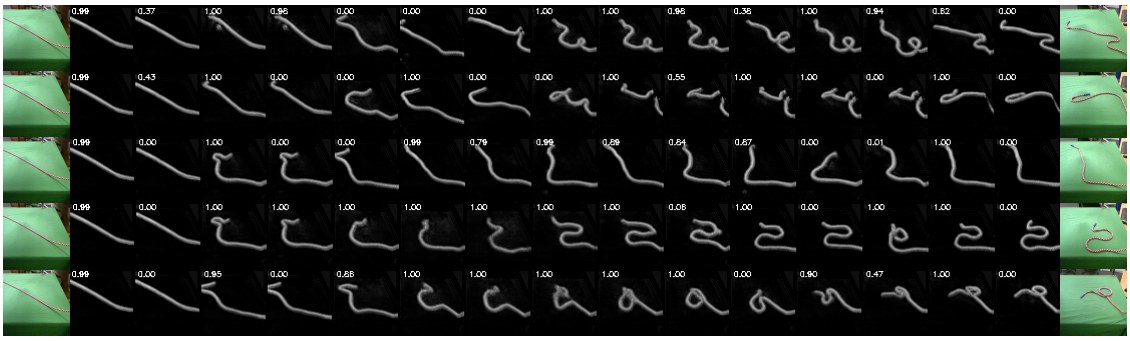
Figure C.1: 2D particle results on the ϵ-key domain where the key dimension is scaled down to 0.1. (a) **The key domain**: The rooms are separated in-between by a wall with a door (yellow). The door only opens when the agent has the key, which can be obtained if the agent is within the area indicated by the red circle on the upper right corner. (b) From no-key to has key, **k-means**: Value indicates the probability for the agent to transition from a state not having the key to a state having the key at each (x, y) location. This transition should only occur near the key region (indicated by the red ring). In this case, K-means fails to learn the separation between having and not having the key, and generated high transition probability over the entire domain. (c) The same figure as (b), generated by **Causal InfoGAN**. On the top right corner where the key is located, the GAN correctly learns that it can transition from having no key to having the key. Bottom blots appear where the posterior sees no data. (d) Causal-InfoGAN planned walkthrough trajectories, showing how the agent acquires the key to cross the door. When the goal is in the top room, the agent goes directly towards the goal without making a detour for the key region.

Rope Results

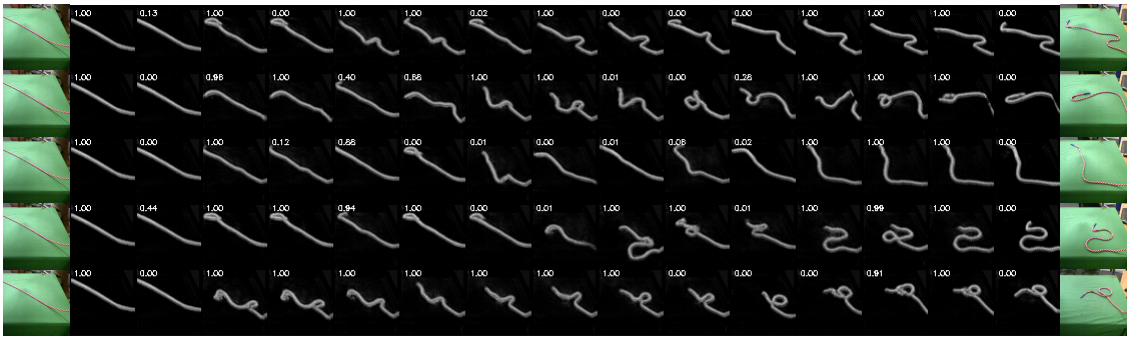
In Figure 6, we provide additional planning results generated by Causal InfoGAN.



(a) Causal InfoGAN



(b) InfoGAN



(c) DCGAN

Figure C.2: Results for rope manipulation data. We compare planning using Causal InfoGAN (top), InfoGAN (middle), and DCGAN (bottom) by interpolation in the latent space, for several rope manipulation goals starting from the same initial configuration. Each plot shows 5 planning instances, from left (starting observation) to the right (goal observation). For each instance, the shown trajectory is picked using the highest trajectory score. The training loss in Causal InfoGAN led to a latent model that most accurately represents possible changes to the rope, compared to the other two baselines.

C.2 Detailed description of our approach

Disentangled Posterior Approximation

We now note a subtle point. The mutual information in equation 4.3 is not sensitive to the order of the code words of the random variables s and s'.[1] This points to a potential caveat in the optimization objective equation 4.3: we would like the random variable for the next abstract state s' to have the *same meaning* as the random variable for the abstract state s. That would allow us to roll-out a sequence of changes to the abstract state, by applying the transition operator $T_{\mathcal{M}}$ sequentially, and effectively plan in the abstract model $\mathcal{M}$. However, the loss function equation 4.3 does not automatically induce s' to have the same meaning as s, since We solve this problem by proposing the *disentangled posterior approximation*, $Q(s, s'|o, o') = Q_1(s|o)Q_2(s'|o')$, and choose $Q_1 = Q_2 \doteq Q$. This effectively induces a generator for which $P(s|o) = P(s'|o')$.[2]

Binary States

In this work, we chose $P_{\mathcal{M}}(s)$ and $P_{\mathcal{M}}(a)$ to be fixed distributions, where each binary element was independent, with a Bernoulli(0.5) distribution. In this case, the marginalization can be calculated in closed form. It is also possible to extend this model to a parametric distribution for $P_{\mathcal{M}}(s)$ and $P_{\mathcal{M}}(a)$, and marginalize using sampling.

C.3 Algorithm

Given the training data $\mathcal{D} = \{(o, o')|\ o \text{ and } o' \text{ are sequential observations.}\}$, Causal infoGAN learns a generative model that structures the latent space in a way that is useful for planning. We provide the algorithm details below:

Let $\theta_D, \theta_G, \theta_Q, \theta_T$ denote the parameters of neural networks $D, G, Q, T_{\mathcal{M}}$, respectively.
For a minibatch of m samples $\{(o_i, o'_i)\}_{i=1}^m$ from $\mathcal{D}$, we:

- Generate m fake samples
  - Sample abstract states $s_1, \ldots, s_m$, where $s_i \sim P_{\mathcal{M}}$
  - Sample next states $s'_1, \ldots, s'_m$, where $s'_i \sim T_{\mathcal{M}}(s'_i|s^i)$
  - Sample noise $z_1, \ldots, z_m$, where $z_i \sim P_{\text{noise}}$
  - Generate fake observations $\hat{o}_1, \hat{o}'_1, \ldots, \hat{o}_m, \hat{o}'_m$, where $\hat{o}_i, \hat{o}'_i = G(z_i, s_i, s'_i)$.

[1] This is a general property of the entropy of a random variable, which only depends on the probability distribution and not on the variable values. In our case, for example, one can apply a permutation to the transition operator $T_{\mathcal{M}}$, and an inverse of that permutation to the generator's s' input. Such a permutation would change the meaning of s', without changing the mutual information term nor the distribution of generated observations.

[2] Note that in a system where the state is fully observable, the posterior is disentangled by definition, therefore in such cases the bound is tight.

- Update the discriminator by descending its stochastic gradient

$$\nabla_{\theta_D}\left(-\frac{1}{m}\sum_{i=1}^{m}\left[\log D(o_i,o_i')+\log(1-D(\hat{o}_i,\hat{o}_i'))\right]\right)$$

- Update the generator and transition model by descending its stochastic gradient

$$\nabla_{\theta_G,\theta_T}\left(-\frac{1}{m}\sum_{i=1}^{m}\left[\log D(\hat{o}_i,\hat{o}_i')\right]\right),$$

  where the gradient θ_T is backpropagated using the reparametrization trick of Gumbel-softmax [106].

- Update posterior, generator, and transition model in the direction of maximal mutual information

$$\nabla_{\theta_Q,\theta_G,\theta_T}\left(\frac{1}{m}\sum_{i=1}^{m}\left[\log P_{\mathcal{M}}(s_i)-\log Q(s_i|\hat{o}_i)+\log T_{\mathcal{M}}(s_i'|s_i)-\log Q(s_i'|\hat{o}_i')\right]\right),$$

  where the gradient θ_T is backpropagated using the reparametrization trick of Gumbel-softmax [106].

- (*For continuous states with linear interpolation planning*) Update transition model to ensure small local transitions in the state space generate plausible observations.

$$\nabla_{\theta_T}\left(\frac{1}{m}\sum_{i=1}^{m}\left\|\Sigma_{\theta_T}(s_i)\right\|_2\right),$$

  where Σ_{θ_T} is part of $T_{\mathcal{M}}$ (see Section 4.3).

- (*Optional*) Update transition model to minimize a self-consistency loss (see details below)

$$\nabla_{\theta_T}\left(\frac{1}{m}\sum_{i=1}^{m}\left[-\log T_{\mathcal{M}}(s^*(o_i')|s^*(o_i))\right]\right),$$

  where $s^*(o)=\arg\max_s Q(s|o)$.

The self-consistency loss is added to further strengthen the relationship between transitions in the latent planning system and the real observations. We maximize the likelihood of observed transitions in the predicted states from real transitions. Namely, let $s^*(o)=\arg\max_s Q(s|o)$ denote the most likely state encoding for an observation, then the self-consistency loss is given by,

$$L_{sc}(\mathcal{M})=\mathbb{E}_{o,o'\sim P_{\text{data}}}\left[-\log T_{\mathcal{M}}(s^*(o')|s^*(o))\right].$$

This loss guides T to be consistent with Q which stabilizes the training. We found this loss to help in stabilizing training for low-dimensional observations. We did not find that adding this loss is beneficial in the high-dimensional case, since in that case, while Q provides meaningful state estimation on fake observations, it tends to overfit to the generated samples, and does not predict reliable states on real observations.

C.4 Experiment details

2D Navigation Experiment

The model parameters we used for the toy domains is as follows:

In the key domain, we used a 4-dimensional space for the latent state[3]. Actions are sampled from a 3 dimensional space whereas the noise z is 4 dimensional. The loss for the generator, the posterior and the transition consistency are weighted equally with a learning rate of 10^{-4}, whereas the learning rate for the discriminator is five times larger (5×10^{-4}). We found that the transition consistency loss important in the stability of models using binary representations. The same hyperparameters are used for both the key domain and the ϵ-key domain. In the latter $\epsilon = 0.1$.

In the tunnel domain we also used a 4-dimensional latent state (a 3-dimensional latent state gave similar results). Actions are sampled from a 3-dimensional space and noise is 4 dimensional. The learning rates are identical to those of the key domain.

To generate the training samples in the tunnel domain, the random walk had a characteristic length scale of 0.05. The rooms are from -1 to 1 in both width and height. The meridian is placed slightly off the middle, at $y = -0.1$. We bias the starting point in the particle trajectories around the choke in the middle, so that the sample trajectories have substantial probability is crossing from one room to the other.

In the key domain, we used a a characteristic length scale of 0.3. This much larger step size is needed because the particle needs to cover the top room, make it to the key zone (to obtain the key), and carry the key to the door to cross to the bottom room in a single trajectory.

In the tunnel domain, we chose the horizon k to be uniform in $5 - 9$. That is, we sample observations o, o' that are k-timesteps apart in the data when training the model. In the key domain, since the characteristic length scale is larger, we chose k to be smaller, uniform in $1 - 4$.

In the key domain, we represent the possession of the key by a single number in the binary set $\{0, 1\}$. Incidentally, it was necessary to inject Gaussian noise to this key dimension during training. Otherwise the generator is required to learn a singularity around 0 and 1, making it numerically highly unlikely. We varied the normalized standard deviation of this Gaussian noise (w.r.t. ϵ). Larger noise (≤ 0.2) produces more stable training, but too much noise can cause blurriness in the cluster boundaries. Overall the scale of this Gaussian noise doesn't substantially impact the representation that is learned.

The models are identical between the key domain and the tunnel domain. Both the generator, the discriminator and the binary posterior are two layer perceptrons with two 100 dimensional hidden layers. The transition function also has two hidden layers, with 10 neurons each.

For the represetantion of the latent space, we use a binary representation of the states as described in section 4.3. The generator uses a sequential architecture as described in Section 4.3, but the two outputs of the generator are trained with only 1 timestep in between with no autoregression on the autoregressive sub module.

[3]We also experimented with 3-5 dimensional latent spaces which gave similar results. Smaller latent space tend to have less expressive power and subject to generator collapse. Beyond 5 however the benefit is marginal.

Rope Experiment

We use Adam [122] optimizer with the learning rate of 0.0002 for both discriminator and generator losses. The generator loss is the sum of three losses described in the Appendix C.3. We use coefficients 1 for the main generator loss, and 0.1 for both the mutual information and the transition loss. We deploy standard DCGAN architectures [207] for the discriminator D and the generator G. The posterior estimator Q has the same architecture as D with the change of the last CNN layer to output 128 channels and the addition of another layer of batchnorm, leaky ReLU and conv layer to the dimension of code. The details are described in table C.1.

In DCGAN and infoGAN baselines, the size of latent code (or abstract state) and the noise is 7 and 2 respectively. In Causal InfoGAN, the generator takes in two abstract states at the same time so the size of noise is doubled to 4. However, we found that the result is quite robust to the dimension sizes.

discriminator D / posterior estimator Q	generator G
Input 64 x 64 grayscale images (2 for D and 1 for Q)	Input a vector in $\mathbf{R}^7 \times \mathbf{R}^7 \times \mathbf{R}^4$
4 x 4 conv. 64 lReLU, stride 2 , batchnorm	4 x4 upconv. 512 lReLU, stride 2 , batchnorm
4 x 4 conv. 128 lReLU, stride 2 , batchnorm	4 x4 upconv. 256 lReLU, stride 2, batchnorm
4 x 4 conv. 256 lReLU, stride 2 , batchnorm	4 x4 upconv. 128 lReLU, stride 2, batchnorm
4 x 4 conv. 512 lReLU, stride 2 , batchnorm	4 x4 upconv. 64 lReLU, stride 2, batchnorm
4 x 4 conv. 1 for D and 128-batchnorm-lReLU-7 for Q	4 x4 upconv. 2 Tanh (1 channel for each image)

Table C.1: The architectures for generating rope images. The discriminator takes in two grayscale images, and outputs the probability of the pair being real. The posterior shares the same architecture with D except the first and the last layer. It takes in one image, and outputs the mean and the variance of its predicted state. The generator takes in the current and the next abstract states (dim 7), and the noise (dim 4). It outputs the current and the next observations. The leaky coefficient is 0.2.

The latent planning system uses a uniform prior between $[-1, 1]$ and a Gaussian transition with zero mean and state-dependent variance. The variance is diagonal and parametrized by a two-layer feed forward neural network of size 64 with ReLU nonlinearity. The last layer is exponentiated to output a positive value for the variance.

For training set, we use sequential observation pairs with 1 step apart from the rope dataset by Nair et al. [185].

Causal Classifier

We trained a binary classifier to function as an evaluator for whether an observation transition is feasible or not, given the data. We use the classifier for two tasks: (1) To post-select transitions (in the observation space) during planning, and (2) to evaluate the score of a walkthrough trajectory.

During training, the classifier takes in a pair of images and output a binary classification of whether this image pair appears sequentially related. The training dataset consists of positive image pairs that are 1 timestep apart, and negative pairs that are randomly sampled from different rope

manipulation runs. To avoid overfitting to the background in the rope dataset and learning a trivial solution where the classifier uses the background to distinguishi different runs, we preprocess the rope data using the background subtraction pipeline mentioned above.

The training accuracy converges to 100% on the training set, and 98% on a held-out test set.

To validate that this classifier actually learns to tell if the transition between two images is feasible or not, we evaluate it on images that are k steps apart where the largest k is the length of an rope experiment. Despite the classifier never seeing samples that are more than 1 step apart, it learns to predict 0 probability for image pairs with large k. The prediction is well-behaved – As we increase k from 1 to the length of the run, the binary output smoothly and monotonically decreases from 1 to 0.

The model architecture used is a convolutional neural network with the following architecture. The two input images are concatenated channel-wise, fed together into the classifier. The optimization is done with the Adam optimizer with a learning rate of 10^{-3}. These hyperparameters are not tuned since the performance of the classifier is sufficient.

Appendix D

Visual Planning and Acting

Training Details

In the CIGAN training, we used the same architecture for all domains from [133], but doubled the number of filters at each layer of the generator, discriminator, and posterior model (Q) when we moved domains from the blocks to the rope, because models with more parameters were needed to capture the nuances for the more complex domain. Additionally, we used GAN training techniques of spectral normalization [175], instance noise [245], and label smoothing [223]. We also got rid of the random noise, z, from the final models that were run on the rope, because we found that the added noise would end up influencing the transitions too much, and they were not actually needed to provide diversity to the generations, as the latent codes were already sufficient for that. Both the discriminator and generator were trained in unison with the Adam optimizer [123].

In order to improve the latent space learned by the model, we gradually increase the information weight λ by .01 starting from .01 every 5 epochs. We already knew that by increasing the weight on the information term we could improve the learned representations and encode more meaning about transitions into them. However, if we increased the weight on this term any higher than the .1 we used on the blocks, the generation quality and fidelity became too poor to be able to follow and execute a control policy on. However, once the generation quality is at a certain level, there would be no reason for the generator to start generating images of worse quality, even if its loss function did change (by a small amount, so as not to disrupt the fickle stability of a GAN). Thus, gradually increasing the information weight had an improvement on both the quality of images generated by the CIGAN, as well as the causal meaning captured by the latent space. This was shown by the improvement in plans generated from this model, as they went through obstacles far less, and had better image quality.

Plan Autoselection in Block-Wall Domain

We want to be able to autoselect the plans that seem feasible and allow the inverse model to follow them. To autoselect the best generated plans, we use a weighted combination of a transition classifier and an object detector. The classifier is trained using image pairs that are within 1-step

apart as positive labels, and random image pairs as negative labels. In general, generative models like GAN and VAE suffer from consistently generating the same number of objects in the scene [288]. Similarly, we find that CIGAN models sometimes mistakenly produce plans with extraneous blocks. Thus, we use the Mask-RCNN as an object detector [2, 96] to restrict this behavior. The Mask-RCNN is trained to detect randomly generated shapes which are agnostic to this domain. The object detector gives a binary value which is 1 if the object detector finds only one object (which is realistic), and 0 otherwise.

We give an image pair by a score s such that $s(o, o_{next}) = (\beta + c(o, o_{next}))^2 + \alpha f(o, o_{next})$ where c is the classifier score and f is the the object detector score. For our purposes, we found the most promising results when $\alpha = 3.0, \beta = 1.0$.

A* Search Details

In the rope domain, we performed A* search in the latent space using the Euclidean distance as the cost function and the heuristic. The graph search is done by expanding the node in the computation stac with the minimum cost-so-far plus λ times cost-to-go, evaluated using the heuristic. Note that λ is the relaxation term that we set at 1.4. The relaxation term controls the trade off between the close-to-optimal path and the computation time. The larger the lambda the greedier the search becomes. To expand a node, we find neighbors by doing Monte Carlo sampling of the transition function to find 50 neighbors. Finally, to check whether we have reached the goal, we threshold any distance below 1.5 in the latent space as sufficient.

Network Architectures

In Tables D.2 and D.1, we outline the architectures used for the CIGAN and inverse control models.

Table D.1: Architecture of CIGAN used for rope domain

Discriminator D	Generator G
Input 2 64 x 64 grayscale images	Input a latent vector in $\mathbb{R}^{10}$
4 x 4 conv. 128 lReLU, stride 2, batchnorm	4 x 4 upconv. 1024 lReLU, stride 2, batchnorm
4 x 4 conv. 256 lReLU, stride 2, batchnorm	4 x 4 upconv. 512 lReLU, stride 2, batchnorm
4 x 4 conv. 512 lReLU, stride 2, batchnorm	4 x 4 upconv. 256 lReLU, stride 2, batchnorm
4 x 4 conv. 1024 lReLU, stride 2, batchnorm	4 x 4 upconv. 128 lReLU, stride 2, batchnorm
4 x 4 conv. 1	4 x 4 upconv. 2 Tanh

Table D.2: Architecture of Inverse Model

Input 1 64 x 64 image of pixelwise difference	
4 x 4 conv. 64 lReLU, stride 2, dropout (.5)	
4 x 4 conv. 128 lReLU, stride 2, dropout (.5)	
4 x 4 conv. 256 lReLU, stride 2, dropout (.5)	
4 x 4 conv. 512 lReLU, stride 2, dropout (.5)	
4 x 4 conv. 40, Sigmoid	
40 dimensional linear, Tanh	
2 dimensional Linear, ReLu	
	42 dimensional linear, Tanh
	2 dimensional linear, ReLu

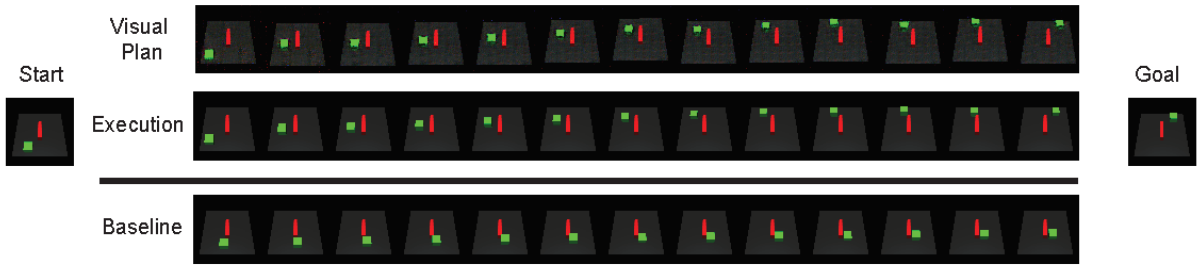


Figure D.1: Another example of a start/goal test image where planning via VPA is essential in reaching our desired state. Our baseline, on bottom, is unable to navigate past the bottom right corner, while the plan generated by CIGAN finds a path around the obstacle rather than trying to go through.

Datasets

Block wall dataset available here:
`https://drive.google.com/drive/folders/16L3Bir66Y31O0khBKAntsxoxq_94U1T_?usp=sharing`

Rope datasets available here:
`https://drive.google.com/drive/folders/1zUQWAzxzN5-WvMc3u9C9GwWCrFb6UjVK?usp=sharing`

Additional Block-Wall Results

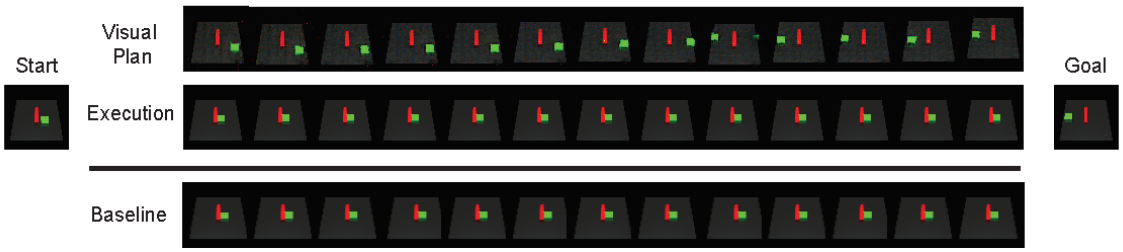


Figure D.2: Example of failure case. While CIGAN succeeded 90% of the time in our autoselected evaluation, there were a few cases that our approach failed to find a viable plan, and thus we were unable to successfully reach the goal state during execution.

Appendix E

Context Generalization

E.1 Discriminative Models: Classifier vs. Energy model

In this section, we assume the dataset as described in VPA, $\mathcal{D} = \{o_1^i, ..., o_{T_i}^i\}_{i=1}^n$. There are two ways of learning a model to distinguish the positive from the negative transitions.

Classifier: As noted above, SPTM first trains a classifier which distinguishes between an image pair that is within h steps apart, and the images that are far apart using random sampling. The classifier is used to localize the current image and find possible next images for planning. In essence, the classifier contains the encoder g_θ that embeds the observation x and the the score function f that takes the embedding of each image and output the logit for a sigmoid function. The binary cross entropy loss of the classifier $L_{SPTM}(\theta, \psi; \mathcal{D})$ is

$$\begin{aligned} &= -\sum_{(z_t, z_{t+k})\sim\mathcal{D}} (\log \frac{f_\psi(z, z_{t+k})}{1 + f_\psi(z_t, z_{t+k})} \\ &\qquad + \log \frac{1}{1 + f_\psi(z_t, z_t^-)}) \\ &= -\sum_{(z_t, z_{t+1})\sim\mathcal{D}} \log \left[\frac{f_\psi(z_t, z_{t+k})}{f_\psi(z_t, z_{t+k}) + \alpha_\psi^t} \right] \end{aligned}$$

where $\alpha_\psi^t = 1 + f_\psi(z_t, z_t^-) + f_\psi(z_t, z_{t+k}) f_\psi(z_t, z_t^-)$, and z_t^- is a random sample from $\mathcal{D}$.

Energy model: Another form of discriminating the the positive transition out of negative transitions is through an energy model. Oord et al. [196] learn the embeddings of the current states that are predictive of the future states. Let g be an encoder of the input x and $z = g_\theta(x)$ be the embedding. The loss function can be described as a cross entropy loss of predicting the correct sample from $N+1$ samples which contain 1 positive sample and N negative samples $L_{CPC}(\theta, \psi; \mathcal{D})$ is

$$= -\sum_{(z_t, z_{t+k})\sim\mathcal{D}} \log \left[\frac{f_\psi(z_t, z_{t+k})}{f_\psi(z_t, z_{t+k}) + \sum_{i=1}^N f_\psi(z_t, z_t^{i-})} \right]$$

where $f_\psi(u, v) = \exp(u^T \psi v)$ and $z_t^{1-}, ..., z_t^{N-}$ are the random samples from $\mathcal{D}$.

Note that when the number of negative samples is 1 the loss function resembles the SPTM.

E.2 Mutual Information (MI)

This quantity measures how much knowing one variable reduces the uncertainty of the other variable. More precisely, the mutual information between two random variables X and Y can be described as

$$\begin{aligned} I(X, Y) = H(X) - H(X|Y) &= H(Y) - H(Y|X) \\ &= \mathbb{E}_{X,Y}\left[\frac{p_{X,Y}}{p_X p_Y}\right]. \end{aligned}$$

E.3 Planning as Inference

After training the CPC objective to convergence, we have $f_k(o_{t+k}, o_t) \propto p(o_{t+k}|o_t)/p(o_{t+k})$ [196]. To estimate $p(o_{t+k}|o_t)/p(o_{t+k})$, we compute the normalizing factor $\sum_{o' \in V} f_k(o', o_t)$ for each o_t by averaging over all nodes in the graph. Therefore, our non-negative weight from o_t to o_{t+k} is defined as $\omega(o_t, o_{t+k}) = \sum_{o' \in V} f_k(o', o_t)/f_k(o_{t+k}, o_t) \approx p(o_{t+k})/p(o_{t+k}|o_t)$.

A shortest-path planning algorithm finds $T, o_0, ..., o_T$ that minimizes $\sum_{t=0}^{T-1} \omega(o_t, o_{t+1})$ such that $o_0 = o_{start}, o_T = o_{goal}$. By Jensen's inequality and the Markovian property of $o_0, ..., o_T$ we have that, $\log \frac{1}{T} \sum_{t=0}^{T-1} \omega(o_t, o_{t+1}) \geq \frac{1}{T} \sum_{t=0}^{T-1} \log \omega(o_t, o_{t+1}) = \frac{1}{T} \sum_{t=0}^{T-1} (\log p(o_{t+1}) - \log p(o_{t+1}|o_t)) = \frac{1}{T} \sum_{t=1}^{T-1} p(o_t) - \log p(o_1, ..., o_{T-1}|o_0 = o_{start}, o_T = o_{goal})$. Thus, since $p(o_t)$ is fixed by uniform asssumption, the shortest path algorithm with proposed weight ω maximizes a lower bound on the trajectory likelihood given the start and goal states. In practice, this leads to a more stable planning approach and yields more feasible plans.

E.4 Block Insertion Domain

In this domain, we kept the obstacle constant and varied the agent itself. In particular, we uniformly chose from 4 to 10 units, with 6 as the holdout, and then randomly placed those units such that they resembled a contiguous shape. When applying an action, we applied a vertical and horizontal force to the middle block, and also a rotation force on the first and last unit laid down, leading to a total action space of four. As our context vector, we randomly chose any image from all trajectories with that same context, as seen in Figure E.1. During testing time, we randomly generated shapes from 3, 6, and 11 units. The L2 threshold distance for success was thus the total L2 distance for all units divided by the number of units.

E.5 Additional Results and Hyperparameters

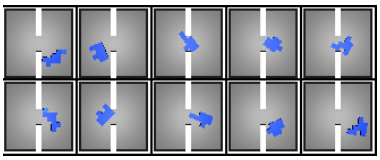


Figure E.1: Example of observations (top) and contexts (bottom) of block insertion domain.

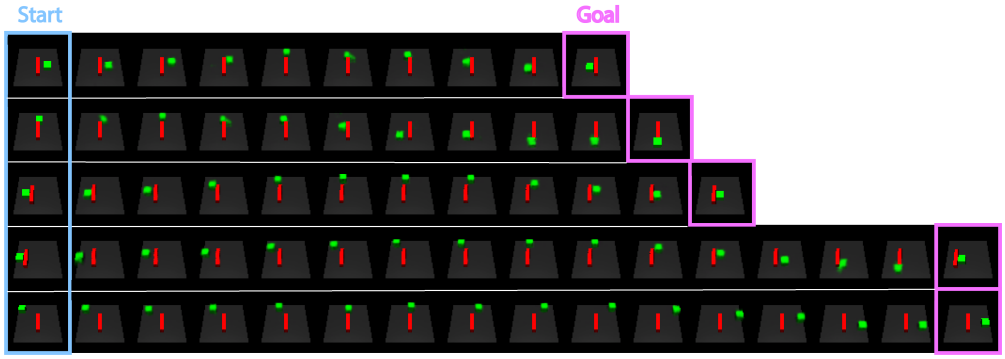


Figure E.2: HTM plan examples on the block wall domain. The hallucination allows the planner to imagine how to go around the wall even though it has not seen the context before.

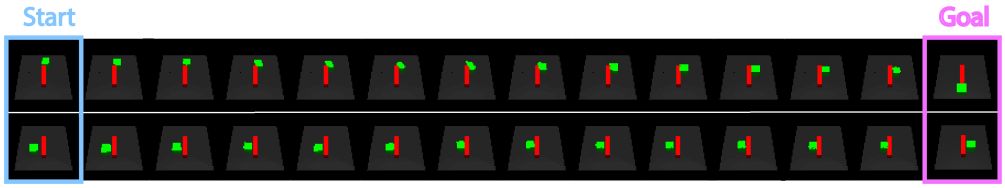


Figure E.3: Visual Foresight plan examples on the block wall domain. The plans do not completely show the trajectory to the goal.

Table E.1: Data parameters.

	Domain 1	Domain 2	Domain 3	Domain 4
no. contexts	150	400	360	1
initializations per context	50	30	20	1000
trajectory length	20	100	50	50
action space	$[-.05, .05]^2$	$[-.1, .1]^2$	$[-.05, .05]^4$	$[-1, 1]^2$
table size	2.8x2.8	2.8x2.8	.8x.8	.9x.7

Table E.2: Planning hyperparameters.

	Domain 1	Domain 2	Domain 3
no. of samples from CVAE	300	500	300
L2 threshold for success (for each unit)	.5	.75	.1
n (timesteps to get to goal)	500	400	400
r (timesteps until replanning)	200	80	80

Appendix F

Hierarchical Model-based Reinforcement Learning

Architectures

Low-level video models. For the low-level architecture we use the default SV2P hyperparameters the original paper [15] and the code is open sourced on Tensor2Tensor [267].

High-level video models. In Maze Navigation experiments, we use the same SV2P architecture as that in the low-level model. Similar to the low-level actions, the high-level action dimension is 2, here we use a filter to compute the center of mass of the green pixels of the block for high-level actions. In Pouring, we allow the high-level model to process action images directly. Using the same encoder as state inputs, we concatenate the action and state embeddings, both of which have size 128. For the high-level architecture we use the default SVG hyperparameters from the orginal paper [45] and the code is also open sourced on Tensor2Tensor [267]. The prior network is learned

CVAE models. The CVAE encoder consists of a series of CNN layers (number of channels: $32 \rightarrow 64 \rightarrow 128 \rightarrow 256$, kernel sizes: $4 \rightarrow 4 \rightarrow 4 \rightarrow 4$, strides: $2 \rightarrow 2 \rightarrow 2 \rightarrow 2$) with batch normalization and ReLU after each layer accordingly. The context frames are passed through the same encoder. Both outputs are concatenated and passed to the two final FF layers of hidden sizes 512 that maps to the latent dimension of size 10 – 5 dimensions for the inference means and 5 dimensions for the inference variances. The decoder is the transpose of the encoder passing the embedding of size 5 through another 2-layer FF neural networks and the transposed CNN layers. The prior network is 2-layer FF neural network on top of the context embeddings to output the means and the variances. The learning rate is 0.001 with Adam Optimizer [244].

Experiment Details

In this section, we detail the environments and the data collection in maze and pouring environments. All the experiments are run on NVIDIA P100 or NVIDIA V100 GPUs.

Environments: In maze navigation, actions represent forces exerted along x,y coordinates for 0.025 seconds. During training, we assume that the agent can be reset to a random location in the

maze. In pouring, actions represent the distances the cup is moved along x-axis and the angles the cup rotates.

Random Data Collection: In block environment, the agent collect the data of 10,000 episodes of length 20 by sampling a random action from range $[-3, 3]$. The wall is reset every 50 episodes. In pouring, at each step, the agent perturbs the cup along x-axis with probability 0.7 and rotate the cup wit probability 0.3 in a random direction. The agent collects 100,000 episodes of length 5 with the level of water (half or full in each cup while maintaining the total number of water particles) and the positions of all cups reset every 50 episodes.

Goal-directed Data Collection: In block environment, the agent collects 1,000 episodes. In each episode, the agent is reset and repeatedly attempts a new goal sampled from the reset for 50 times. In Pouring, the agent collects 20,000 episodes. In each episode, the agent is reset to a new configuration. Then, the agent samples a new goal from the environment reset to attempt and repeat for 4 times.

Hyperparameters

For all experiments in both Maze Navigation and Pouring, the same low-level video prediction model is applied to plan for 15 steps ahead with action repeats every 3 steps – effectively 5 unique actions each of dimension 2 –. The CEM optimizer samples 200 actions (initially from a uniform distribution within the action range $[-3, 3]$), select the top 40 best samples, fit a Gaussian distribution, and repeat for 3 iterations. After resulting in the best sequence actions, the low-level controller executes the first 5 steps and replans.

We report all the results under green-pixel L2 distance cost to goal $C(s_t, s_g)$, using a green pixel detector to approximate the center of mass of the object. The cost has hyperparameter β to weigh the cost of the final state: $\frac{1}{T}\sum_{t=1}^{T} C(s_t, s_g) + \beta * C(s_T, s_g)$.

HVPC is tested on different numbers of high-level planning horizon (or subgoals) and cost weight β as shown in Table F.1 evaluated over 500 random tasks. The low-level controller attempts the first high-level subgoal for 20 steps (or until reaching the subgoal) and then replans to a new high-level subgoal for 10 times (or until reaching the goal). In total, HVPC takes at most 200 steps.

Subgoals	$\beta = 0$	$\beta = 2$	$\beta = 4$
1	52.8	54	49.8
2	70.25	**87.6**	**86.2**
3	**80.4**	82.2	67.5

Table F.1: HVPC success rate (in percent) in Maze domain.

HVPC without high-level replanning aims to make this comparable to HVF [188] on different numbers of high-level planning horizon (or subgoals) and cost weight β as shown in Table F.2 evaluated over 400 random tasks. The low-level controller attempts the high-level subgoals sequentially for 20 steps each and finally attempts the goal. In total, HVPC takes at most 80 steps (with 3 subgoals).

Subgoals	$\beta = 0$	$\beta = 2$	$\beta = 4$
1	40.33	37.5	37.25
2	45.25	60.5	60.5
3	**55.75**	**68.75**	**64.25**

Table F.2: HVPC without high-level replanning success rate (in percent) in Maze domain.

HVF [188] compares the results using different number of subgoals and cost weight β with planning horizon 5 (used in the original paper) and 15 evaluated over 100 random tasks in Table F.3 and Table F.4. HVF pursues the each subgoal sequentially for 50 steps and move to the next one. Finally, it will pursue the goal using the number of steps left from 200 steps allowed. When the number of subgoals is zero, the algorithm is equivalent to Visual Foresight [57].

Subgoals	$\beta = 0$	$\beta = 2$	$\beta = 4$
0	31	25	33
1	28	**32**	35
2	**37**	31	**37**

Table F.3: HVF success rates (in percent) in Maze domain. Planning horizon 5.

Subgoals	$\beta = 0$	$\beta = 2$	$\beta = 4$
0	41	52	44
1	41	**58**	**60**
2	**46**	52	55

Table F.4: HVF success rates (in percent) in Maze domain. Planning horizon 15.

These results demonstrate that HVPC achieves significantly higher success rates (87.6%) on the long-horizon tasks than the state-of-the-art baselines. When replanning is not allowed, the the success rates (68.75%) decrease; however, they still surpass those of HVF (60%) and VF (41%).

Additional Results

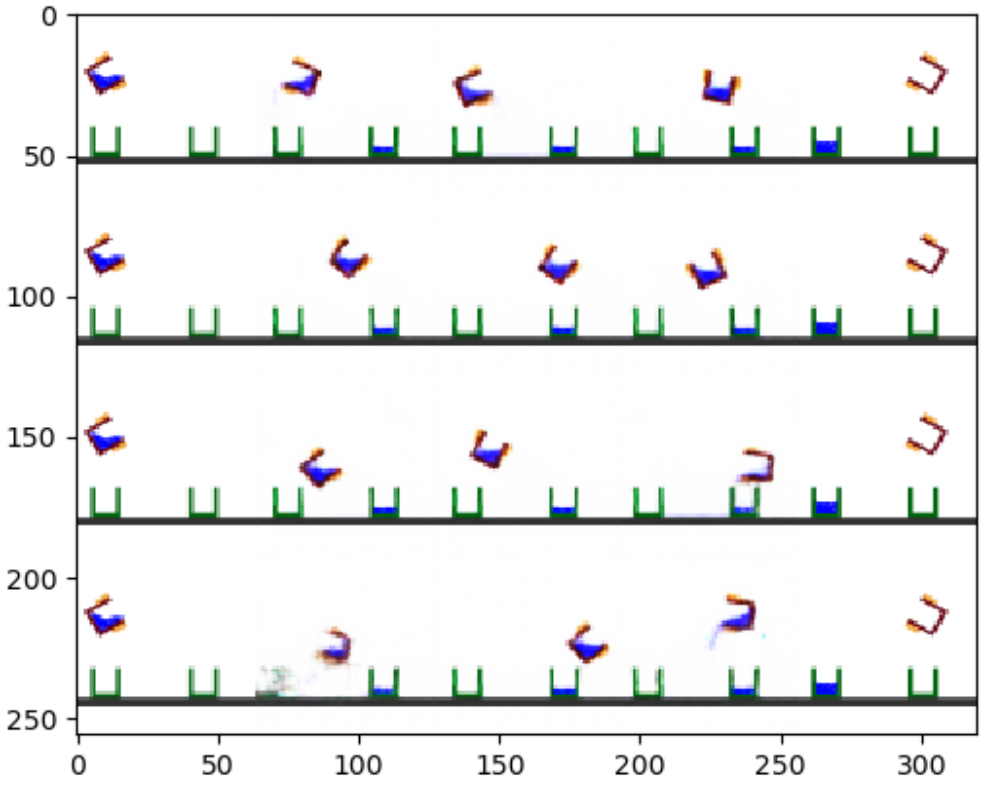


Figure F.1: HVF Baseline Sample Plans. The top two rows are the plans with the highest scores and the bottom two rows are the plans with the lowest scores.

Appendix G

Sparse Graphical Memory

G.1 Environments & Hyperparameters

Perceptual consistency baseline and fast-path

Two-way consistency (8.1-8.2) involves a maximization over the replay buffer, which can be costly. We use a pairwise, symmetric *perceptual consistency* criterion as a fast test of similarity in latent space to accelerate graph construction. A new state $\hat{s}$ is only added to the memory if it fails both perceptual and two-way consistency with each state already in the memory, in which case we consider it novel and useful to retain for planning. This allows us to skip the TWC check for substantially different states.

We say that $\hat{s}$ is perceptually consistent with a previously recorded state $s \in \mathcal{V}$ if

$$||\phi(s) - \phi(\hat{s})||_2 \leq \tau_p, \tag{G.1}$$

where $||\phi(s) - \phi(\hat{s})||_2$ measures the visual similarity of states that the agent encounters through the l_2 distance between embeddings of each state. In proprioceptive tasks, the identity function is used for the embedding $\phi(\cdot)$. However, in visual navigation, nearby locations can correspond to images that are significantly different in pixel space, such as when an agent rotates [227]. To mitigate this problem, for high-dimensional images, $\phi(\cdot)$ is a learned embedding network such as a β-VAE [124, 100] for SafetyGym or a subnetwork of the distance function for ViZDoom.

Table 8.3 shows that two-way consistency significantly outperforms perceptual consistency on its own, so the TWC criterion is still important. Theorem 1 still holds when combining the strategies as equation G.1 can only make aggregation more conservative.

Low-level Controller

For the RL experiments, we use actor-critic methods to train a low-level controller (the actor) and corresponding distance metric (the critic) simultaneously. In particular, we use distributional RL and D3PG, a variant of deep deterministic policy gradient [152, 22]. For experiments on SafetyGym, we use a proprioceptive state-based controller for both SGM and the dense baseline. For the SSL

experiments in ViZDoom, we use the trained, behavior cloned visual controller from SPTM. The controller is trained to predict actions from a dataset of random rollouts, where goals are given by achieved visual states.

Environments

In our experiments, we tune SGM hyperparameters to maintain graph connectivity and achieve a desired sparsity level. Thresholds are environment-specific due to different scaling of the distance function. Success rate is only evaluated after selecting parameters.

PointEnv: PointEnv is maze environment introduced in [59] where the observation space is proprioceptive. We run all SoRB experiments in this environment. The episodic return is undiscounted $\gamma = 1$, and the reward is an indicator function: $r = 0$ if the agent reaches its goal and $r = -1$ otherwise. The distance to goals can thus be approximated as $d = |Q(s, a)|$. We approximate a distributional Q function, which serves as a critic, with a neural network that first processes the observation with a 256-unit, fully-connected layer, that then merges this processed observation with the action, and that then passes the observation-action combination through another 256-unit, fully-connected layer. For an actor, we use a fully-connected network that has two layers of 256 units each. Throughout, we use ReLU activations and train with an Adam optimizer [125] with a step size of 0.0003. To evaluate distances, we use an ensemble of three such distributional Q functions, and we pessimistically aggregate across the ensemble. For SGM, we set MAXDIST $= 10$, $\tau_p = 0.05$, $\tau_a = 5$, $k = 5$, MAXSTEPS $= 30$, and ACTINGCUTOFF $= 1$ for the localization threshold. For SoRB, we set MAXDIST $= 6$, $k = 5$ and MAXSTEPS $= 18$ due to a higher density of states.

ViZDoom: For our ViZDoom visual maze navigation experiments, we use the large training maze environment of [227]. Following [227], the distance metric is a binary classifer trained with a Siamese network using a ResNet-18 architecture. The convolutional encoder embeds image observations into a 512 dimensional latent vector. Two image embeddings are then concatenated and passed through a 4 layer dense network with ReLU activations, 512 hidden units, and a binary cross entropy objective where $y = 1$ if the two embeddings are temporally close and $y = 0$ otherwise. An Adam optimizer [125] with a step size of 0.0001 is used for gradient updates to finetune the pretrained network of [227]. For the controller, we use the pretrained network of [227] with no finetuning.

For graph creation, we collect a replay buffer of 100 episodes of random actions, each consisting of 200 steps (*i.e.* 20,100 total images) and add each image sequentially to SGM to simulate an online data collection process. For both SPTM and SPTM with SGM, we set MAXDIST $= 2$, $k = 5$, ACTINGCUTOFF $= 5.75$, and MAXSTEPS $= 10$. For SGM, $\tau_p = 20$ and $\tau_a = 2$. To make the SPTM baseline tractable, we randomly subsample the replay buffer to 2,087 states (the same size as our sparsified vertex set), as edge creation is $O(|\mathcal{V}|^2)$ and start/goal node localization is $O(|\mathcal{V}|)$. The baseline graph has $2,087$ nodes and $18,921$ edges. While we can evaluate the baseline with a graph consisting of all $20,100$ states, this is a dense oracle that has $20,100$ nodes and $1,734,524$ edges and takes hours to construct. The oracle achieves $75\%, 55\%$, and 35% success rates at easy, medium and hard goal difficulties ($55.0\% \pm 6.4\%$ overall).

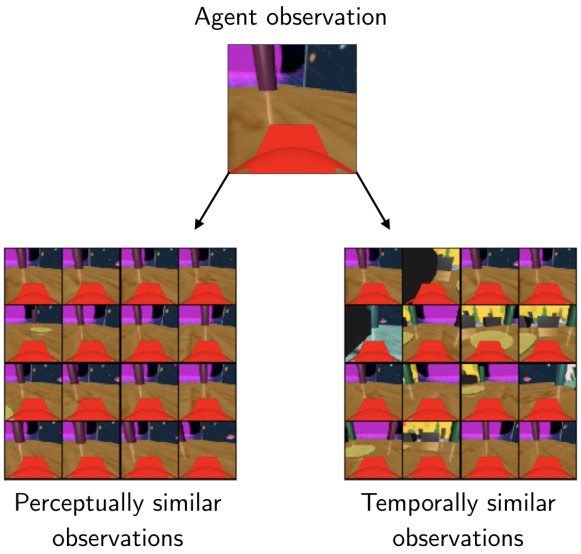


Figure G.1: Observations passing perceptual (feature difference threshold) and two-way consistency criteria for SafetyGym using a β-VAE for perceptual features and a contrastive distance for two-way consistency. The two-way consistent observations are, unsurprisingly, more diverse in orientation than perceptually nearby ones. Although the majority of two-way consistent observations are correct *i.e.* nearby in space, there is one false positive (blue floor).

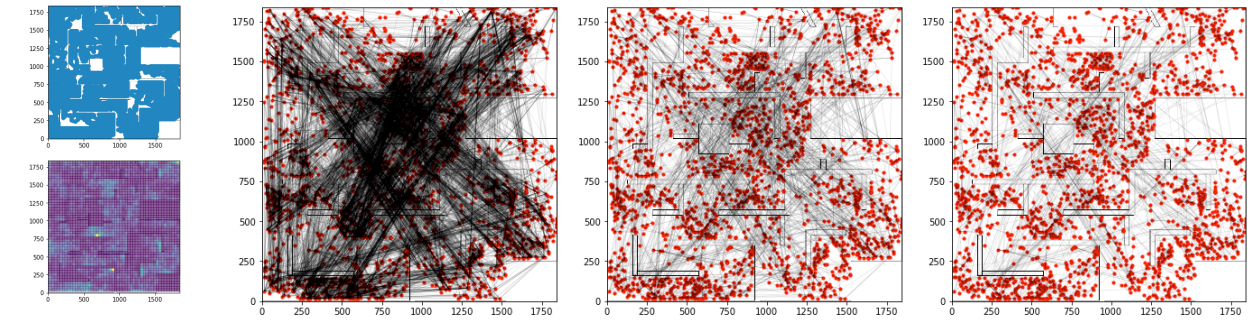


Figure G.2: Construction of Sparse Graphical Memory in the ViZDoom environment. We add nodes from a source replay buffer that unevenly covers the environment (left), creating a sparsified memory. k-nearest neighbor edge filtration limits the number of errors, which are further corrected via cleanup. The image memory in SGM much more evenly covers the environment, even though no coordinate information is used during graph construction.

To increase the robustness of edge creation under perceptual aliasing, we aggregate the distance over temporal windows in the random exploration sequence. For states $s_t^{(i)}$ in episode i and $s_{t'}^{(j)}$ in episode j, we set the distance to the maximum pairwise distance between states $s_{t-2}^{(i)}, s_{t-1}^{(i)}, s_t^{(i)}, s_{t+1}^{(i)}, s_{t+2}^{(i)}$ and states $s_{t'-2}^{(j)}, s_{t'-1}^{(j)}, s_{t'}^{(j)}, s_{t'+1}^{(j)}, s_{t'+2}^{(j)}$, aggregating over up to 25 pairs. In contrast, SPTM aggregated with the median and compared only 5 pairs. Our aggregation is more pessimistic as our replay buffer is created with random exploration that suffers from extensive perceptual aliasing rather than a human demonstrator that mostly stays in the center of hallways. Figure G.2 shows the graph construction process.

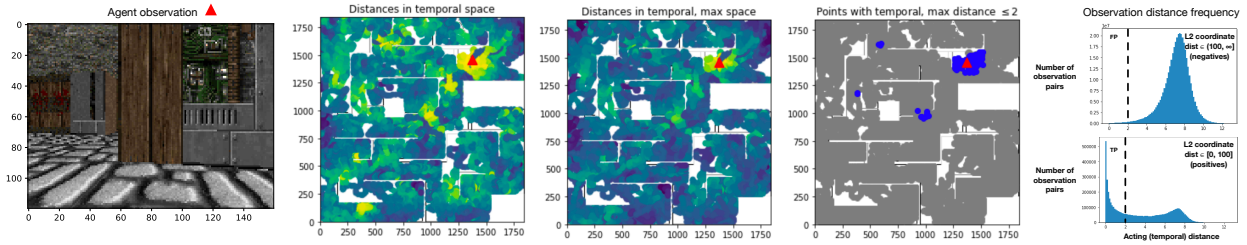


Figure G.3: A visualization of the fine-tuned SPTM distance metric in the ViZDoom. The agent sees the reference image on left. The second image shows the acting distance between the reference image and states previously seen throughout the maze according to $d(s_{\text{agent}}, \cdot)$. A coordinate is colored yellow if the associated state is close to the reference in acting distance, while green and blue coordinates are distant with respect to the reference state. Aggregating the distance pessimistically across temporal windows (third image) reduces false positives that are distant in coordinate space but close in acting distance space. In the fourth image, we threshold the aggregated distance according to τ_a. While most states passing the threshold are near the agent, some are distant. These are false positives. In the rightmost figure, we show histograms of the aggregated acting distance for negative, distant pairs of states (top) and close, positive pairs (bottom), totaling 404M pairs.

SafetyGym: In the SafetyGym environment we employ a contrastive objective to discriminate between observations from the same window of a rollout and random samples from the replay buffer. The contrastive objective is a multiclass cross entropy over logits defined by a bilinear inner product of the form $f(z_t, z_{t'}) = z_t^T W z_{t'}$, where W is a parameter matrix, and the distance scores are probabilities $d = \exp(-f(z_t, z_{t'}))$. To embed the observations, which are 64×64 rgb images, we use a 3 layer convolutional network with ReLU activations with a dense layer followed by a LayerNorm to flatten the output to a latent dimension of 50 units. We then train the square matrix W to optimize the ccontrastive energy function. As before, we use Adam [125] with a step size of 0.0001 for optimization.

The β-VAE maximum likelihood generative model for perceptual features has an identical architecture to the distance metric but without the square matrix W. Each image is transformed into an embedding, which is stored in the node of the graph. When a new image is seen, to isolate visually similar neighbors, we compute the L2 distance between the latent embedding of the image and all other nodes in the graph. Since computing the L2 distance is a simple vector operation, it is much more computationally efficient than querying the distance function, which requires a forward pass through a neural network, $O(|\mathcal{V}|)$ times at each timestep.

Perceptually consistent and two-way consistent observations relative to an agent's current observation are shown in Figure G.1. For constructing both dense and sparse graphs, we use MAXSTEPS $= 9$, $k = 6$, $\tau_p = 6$, and $\tau_a = 5$.

G.2 Perceptual Aliasing with Learned Distance

A common issue with learning distances from images is perceptual aliasing, which occurs when two images are visually similar but were collected far apart in the environment. We examine a heatmap of learned distances in ViZDoom in Figure G.3. Although most observations with small distance are correctly clustered around the agent's location, there are several clusters of false

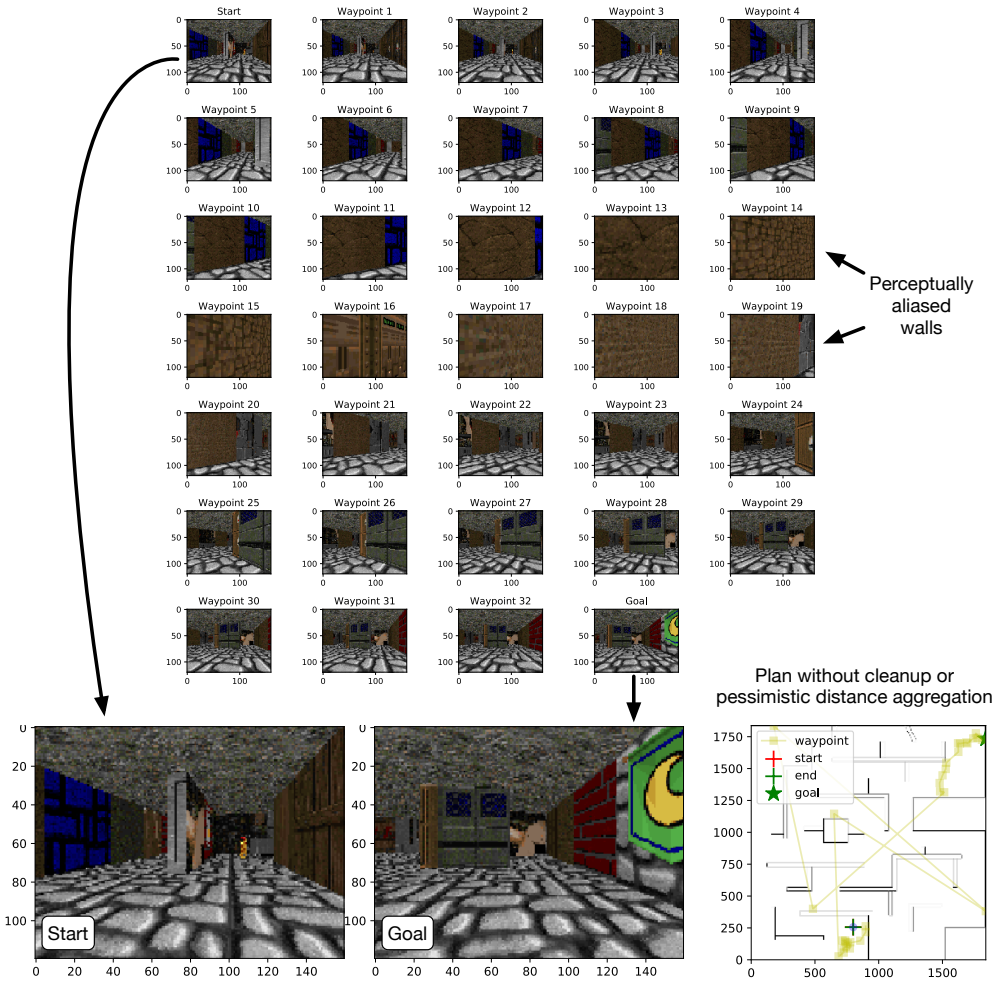


Figure G.4: Failure mode in ViZDoom planning when cleanup and pessimistic distance aggregation are not used. While waypoints in the plan between start and goal states are closely grouped for much of the path, the planner exploits the perceptual aliasing of walls in the environment as a shortcut through the environment. Pessimistic aggregation of the distance metric can help with the issue, but does not fully resolve the problem. By stepping through the environment during cleanup, we can remove the remaining untraversable edges.

positive observations throughout the map due to perceptual aliasing between parts of the maze. Perceptual aliasing results in wormhole connections throughout the graph where two distant nodes are connected by an edge, which creates an attractor for path planning. We show an example path planned by an agent that is corrupted by perceptual aliasing in Figure G.4. In its plan, the agent draws a connection between two visually identical but distant walls, which corrupts its entire plan to reach the goal.

False positives can be reduced further by aggregating the distance pessimistically across consecutively collected observations. However, doing so does not eliminate them altogether. The presence of false positives further supports the argument for sparsity. With sparsity and cleanup, it is possible to remove the majority of incorrect edges to yield robust plans.

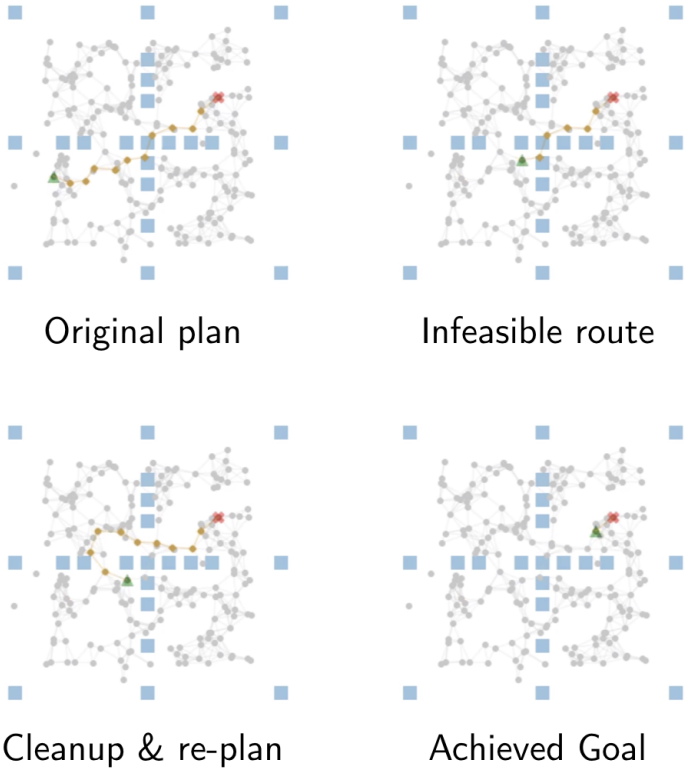


Figure G.5: Evaluation of SGM in SafetyGym. This figure is a top-down view abstraction of the SafetyGym environment made to cleanly represent the sparse graph. The actual environment is more visually complex and the agent sees first-person view images.

G.3 Re-planning with a Sparse Graph

Figure G.5 shows an example of an evaluation rollout, which includes a cleanup step when the agent encounters an unreachable waypoint. The agent creates an initial plan and moves along the proposed waypoints until it encounters an obstacle. Unable to pass the wall (represented by the blue blocks), the agent removes the edge between two nodes across the wall and re-plans. Its second plan has no obstacles and it is therefore able to reach its goal.

G.4 Proof Bounding the Cost Gap of Two-way Consistency

Theorem 1. *Let $\mathcal{G}_{\mathcal{B}}$ be a graph with all states from a replay buffer $\mathcal{B}$ and weights from a distance function $d(\cdot, \cdot)$. Suppose $\mathcal{G}_{TWC}$ is a graph formed by aggregating nodes in $\mathcal{G}_{\mathcal{B}}$ according to two-way consistency C_{out} and $C_{in} \leq \tau_\alpha$. For any shortest path P_{TWC} in G_{TWC}, consider the corresponding shortest path $P_{\mathcal{B}}$ in $G_{\mathcal{B}}$ that connects the same start and goal nodes. Suppose $\mathcal{P}_{\mathcal{B}}$ has k edges. Then:*

(i) The weighted path length of P_{TWC} minus the weighted path length of $P_{\mathcal{B}}$ is no more than $2k\tau_\alpha$.

(ii) Furthermore, if $d(\cdot,\cdot)$ has error at most ϵ, the weighted path length of P_{TWC} is within $k\epsilon+2k\tau_\alpha$ the true weighted distance along $\mathcal{P}_\mathcal{B}$.

Proof. Before proceeding with the proof, we establish more formal notation about the statement of the theorem.

We denote the replay buffer graph $\mathcal{G}_\mathcal{B} = (\mathcal{V}_\mathcal{B}, \mathcal{E}_\mathcal{B}, \mathcal{W}_\mathcal{B})$. Aggregating nodes according to TWC leads to the subgraph $\mathcal{G}_{TWC} = (\mathcal{V}_{TWC}, \mathcal{E}_{TWC}, \mathcal{W}_{TWC})$ where all $V_\mathcal{B} \backslash V_{TWC}$ satisfy two-way consistency at threshold τ_a with some node in V_{TWC}.

For any start and goal node $s_1, s_{k+1} \in V_{TWC}$, consider the shortest path that connects them in $G_\mathcal{B}$: $P_\mathcal{B} = (s_1, s_2, \ldots, s_{k+1})$. Suppose the edge weights of $P_\mathcal{B}$ are $(w_1, w_2, \ldots, w_k)$, given by the distance function according to $w_1 = d(s_1, s_2)$, etc.

Proof of (i): We will show that there exists a corresponding shortest path in $\mathcal{G}_{TWC}$ given by $P_{TWC} = (s_1, s'_2, \ldots, s'_k, s_{k+1})$ with edge weights (distances) $(w'_1, w'_2, \ldots, w'_k)$ where the total weight of P' is no more than $2k\tau_a$ the total weight of $P_\mathcal{B}$, *i.e.*, $\sum_i w'_i \leq 2k\tau_a + \sum_i w_i$.

We proceed by construction by first specifying s'_i for $i \in \{2, 3, \ldots, k\}$. If $s_i \in \mathcal{V}_{TWC}$, let $s'_i = s_i$. Otherwise, let s'_i be any node in $\mathcal{V}_{TWC}$ satisfying two-way consistency with s_i.

Given this choice of P_{TWC}, we will show for arbitrary $j \in \{1, 2, \ldots, k\}$ that

$$w'_j = d(s'_j, s'_{j+1}) \leq 2\tau_a + d(s_j, s_{j+1}) = 2\tau_a + w_j.$$

By outgoing two-way consistency between s_j and s'_j (or because $s_j = s'_j$), we have

$$|d(s_j, s_{j+1}) - d(s'_j, s_{j+1})| \leq \tau_a.$$

Similarly, by incoming two-way consistency between s_{j+1} and s'_{j+1} (or because $s_{j+1} = s'_{j+1}$),

$$|d(s'_j, s_{j+1}) - d(s'_j, s'_{j+1})| \leq \tau_a.$$

By the triangle inequality, $|d(s_j, s_{j+1}) - d(s'_j, s'_{j+1})| \leq 2\tau_a$, *i.e.*, $w'_j \leq w_j + 2\tau_a$. Finally, we note that the result (i) follows by summing the bound $w'_j \leq 2\tau_a + w_j$ over all indices j.

Proof of (ii): Furthermore, assume that the edge weights of $P_\mathcal{B}$ given by $(w_1, w_2, \ldots, w_k)$ all have error at most ϵ. That is to say, if $(w^*_1, w^*_2, \ldots, w^*_k)$ denote the true distances along the path $P_\mathcal{B}$, assume $|w_i - w^*_i| \leq \epsilon$ for all $i \in \{1, 2, \ldots, k\}$. We will show that the shortest weighted path length in the aggregated graph $\mathcal{G}_{TWC}$ differs from the true weighted length of the shortest path in $\mathcal{G}_\mathcal{B}$ by at most $k\epsilon + 2k\tau_\alpha$, *i.e.*, $|\sum_i w'_i - \sum_i w^*_i| \leq k\epsilon + 2k\tau_\alpha$.

In the proof of (i), we showed $\sum_i w'_i \leq 2k\tau_a + \sum_i w_i$. By the additional assumption of part (ii), we have $w_i \leq w^*_i + \epsilon$ for all $i \in \{1, 2, \ldots, k\}$. Chaining these two inequalities yields $\sum_i w'_i \leq k\epsilon + 2k\tau_a + \sum_i w^*_i$.

Similarly, the proof of (i) shows $|d(s_j, s_{j+1}) - d(s'_j, s'_{j+1})| \leq 2\tau_a$ for arbitrary $j \in \{1, 2, \ldots, k\}$. Hence, $w_j \leq w'_j + 2\tau_\alpha$. Rearranging and summing this bound yields $-2k\tau_\alpha + \sum_j w_j \leq \sum_j w'_j$. Moreover, as for part (ii) we assume $|w_i - w^*_i| \leq \epsilon$ for all $i \in \{1, 2, \ldots, k\}$, we also have $w^*_j - \epsilon \leq w_j$. Summing and chaining inequalities yields $-k\epsilon - 2k\tau_\alpha + \sum_i w^*_i \leq \sum_i w'_i$.

Together, the results of the previous two paragraphs yield what is desired: $|\sum_i w'_i - \sum_i w^*_i| \leq k\epsilon + 2k\tau_\alpha$. □

G.5 Planning speed of SGM

Table G.1 shows the time required to take a single action including graph planning on a dense graph constructed using SoRB and with a sparse graph in the PointEnv environment. Sparse Graphical Memory significantly accelerates action selection.

Method	Time to Take Action (s)
SoRB	0.550 ± 0.220
SGM (ours)	$\mathbf{0.077 \pm 0.004}$

Table G.1: The average and standard-deviation wall-clock time for taking an action with SGM (our method) and SoRB (previous state-of-the-art) in PointEnv.

G.6 Reproducibility checklist

Models & Algorithms Section 8.4 and Section G.1 describe our model architectures, which follow past work in the two previously studied planning environments (PointEnv and ViZDoom). We analyze the complexity of graph construction with our online approximation in Section 8.3 and of edge creation and agent localization in Section G.1.

Theoretical Claims We state and prove Theorem 1, which (i) bounds the increase in plan cost when aggregating states via two-way consistency and (ii) given an existing error bound on the distance function, bounds the additional error induced by aggregating states via two-way consistency. The theorem is stated informally in the main paper and precisely with a proof in Section G.4.

Datasets Environments and experience collection are described in Section G.1.

Code Code and documentation are included on the project website: `https://mishalaskin.github.io/sgm/`.

Experimental Results We describe the setup of our experiments in Section 8.4 and Section G.1, including hyperparameter selection and specification. Runtimes are described, including in Section G.5.