Technische Universität München
TUM School of Computation, Information and Technology

CONTROL WITH SPATIAL WORLD MODELS

BARIŞ KAYALIBAY

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz		Prof. Dr. Daniel Cremers
Prüfer der Dissertation:	1.	Prof. Dr. Hans-Joachim Bungartz
	2.	Prof. Dr. Georg Groh
	3.	Prof. Dr. Wolfram Burgard

Die Dissertation wurde am 17.11.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 18.06.2024 angenommen.

ABSTRACT

Decision making is among the hardest unsolved problems that Machine Learning has been applied to. We have methods for solving complex decision making problems given enough budget for interacting with the world and collecting data. But solving decision making problems with few data and partial observations is generally beyond our reach. World models are believed to be capable of tackling both the problem of scarce data and that of partial observations by leveraging Bayesian state estimation and model-based planning.

Indeed, world models have been used to solve some of the hardest control problems found in the literature. Still, knowledge about how these models work is incomplete. We first identify a ubiquitous pain-point in model-based reinforcement learning, building on recent work which has shown that learning state-space models from data can fail under certain conditions. We propose a simple fix and show that the resulting model works better than the standard found in the literature.

Next, we show that world models can understand spatial environments, which is a highly diverse and challenging setting. Given the right inductive biases, we can learn models of everyday places such as office buildings or factory floors, which can then be used for decision making through simple model-predictive control (MPC).

Finally, we identify a weakness of MPC itself and propose an improved version. Our method, which we call filter-aware MPC is able to reason about how the controls picked by the controller affect the state estimate which is used for planning. By reasoning about this relationship, filter-aware MPC is able to avoid state estimation failures which plague regular MPC.

ZUSAMMENFASSUNG

Das Treffen von Entscheidungen ist eines der schwierigsten ungelösten Probleme, auf die Maschinelles Lernen angewendet wird. Aktuelle Verfahren sind in der Lage, komplexe Regelungsprobleme zu lösen, solange genügend Daten vorhanden sind. Schwierigkeiten treten auf, sobald diese Verfahren auf Probleme eingesetzt werden, wo ein Teil der Information nicht beobachtet werden oder wenn nur wenig Daten verfügbar sind. Weltmodelle (englisch *World Models*) versuchen, das Problem der Datenmangel und das Problem der teilweisen Beobachtbarkeit zu lösen, indem sie Bayessche Inferenz und model-basiertes Planen einsetzen.

In der Tat lassen sich einige der schwierigsten Regelungsprobleme in der Literatur durch Weltmodelle lösen. Wissen über diese Modelle ist jedoch mangelhaft. In dieser Arbeit wird zunächst ein weit verbreitetes Problem im modellbasierten Bestärkenden Lernen (englisch *Reinforcement Learning*) identifiziert und eine einfache Lösung vorgeschlagen. Das resultierende Modell funktioniert besser als das Standardverfahren aus der Literatur auf unseren Testproblemen.

Anschließend wird gezeigt, dass auch räumliche Umgebungen wie Wohnungen oder Fabrikhallen durch Weltmodelle simuliert werden können, wenn die Architektur des Modells der Struktur der zugrunde liegenden Daten angepasst wird. Das resultierende Weltmodell ermöglicht das Einsetzen modellprädiktiver Regelung (englisch *Model-Predictive Control*) auf Regelungsprobleme räumlicher Art.

Zuletzt wird ein Schwachpunkt modellprädiktiver Regelung identifiziert und eine Verbesserung vorgeschlagen. Unsere Methode zieht, im Gegensatz zu normaler modellprädiktiver Regelung, die Auswirkungen der Entscheidungen des Regelungsalgorithmus auf zukünftige Messungen mit ein. Dadurch können sowohl Kosten minimiert werden, als auch sicher getstellt werden, dass der Zustand des Reglers und des Systems akkurat bestimmt werden können.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

Kayalibay, Baris, Atanas Mirchev, Ahmed Agha, Patrick van der Smagt, and Justin Bayer (2023). *Filter-Aware Model-Predictive Control*. DOI: 10.48550/ARXIV.2201.10335. URL: https://arxiv.org/abs/2201.10335.

Kayalibay, Baris, Atanas Mirchev, Patrick van der Smagt, and Justin Bayer (2021). "Less Suboptimal Learning and Control in Variational POMDPs." In: *Self-Supervision for Reinforcement Learning Workshop - ICLR 2021*. URL: https://openreview.net/forum?id=oe4q7ZiXwkL.

Kayalibay, Baris, Atanas Mirchev, Patrick van der Smagt, and Justin Bayer (2022). *Tracking and Planning with Spatial World Models*. DOI: 10.48550/ARXIV.2201.10335. URL: https://arxiv.org/abs/2201.10335.

Figures and tables which have previously appeared in these are marked as such. Passages of text which are taken directly from these publications or have been modified minimally appear in grey text, exemplified in this sentence. Such passages are found in sections 3.1, 6.5 and 6.6 and appendices B and C.

The author has contributed in a second-author capacity to the following publication, which presents a method that is foundational to the algorithms presented in this work.

Mirchev, Atanas, Baris Kayalibay, Patrick van der Smagt, and Justin Bayer (2021). "Variational State-Space Models for Localisation and Dense 3D Mapping in 6 DoF." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=XAS3uKeFWj.

The following publications took place before the author started working on his dissertation, and are therefore outside the contributions of this thesis, but are covered in the text due to their status as precursors to this work:

Kayalibay, Baris, Atanas Mirchev, Maximilian Soelch, Patrick van der Smagt, and Justin Bayer (2018). *Navigation and planning in latent maps*. URL: http://reinforcement-learning.ml/papers/pgmrl2018_kayalibay.pdf.

Mirchev, Atanas, Baris Kayalibay, Maximilian Soelch, Patrick van der Smagt, and Justin Bayer (2019). *Approximate Bayesian inference in spatial environments*. arXiv: 1805.07206 [stat.ML].

ACKNOWLEDGEMENTS

It is safe to say that we do not thank people enough. I will take the opportunity given to me by thesis-writing tradition to say thank you to some people here.

Thank you, Patrick, for letting me write a Bachelor's thesis with you all those years ago. Working with you, I have time and again felt like I was doing something that I did not yet know how to do—and that was the best way to learn. Thank you for being the kind of teacher that trusts people like that.

Justin, thank you for creating an environment where I never felt scared to be **exactly** as stupid as I am. From the first day you treated me as a peer, and that has meant the world to me.

Atanas, I have learned more from hanging out around you than I ever thought would be possible. Thank you.

Ole, thank you for working at a Volkswagen factory with me for three weeks.

Felix, thank you for our bouldering sessions together.

And thank you to many others that I had the pleasure to work with at the lab: Adnan, Ahmed, Alex, Alexej, Ankur, Begüm, Botond, Djalel, Eileen, Elie, Eva, Fahad, Felix Frank, Grady, Gülce, James, Karolina, Laura, Liesbeth, Marvin, Max Karl, Max Sölch, Michelle, Nutan, Philip, Richard, Salem, Tai, Xingyuan and Ziqing. Thanks everyone!

Finally, thank you to Prof. Dr. Hans-Joachim Bungartz for your kindness and generosity.

Arkadaşlarım Ali, Can, Dağhan, Didem, Ezgi, Ozan, Ömer, Salih ve Zeynep: hayatımda olduğunuz için teşekkürler.

Anne, baba, sizi çok seviyorum.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ANS Active Neural SLAM

CMP Cognitive Mapping and Planning

DVBF-LM Deep Variational Bayes Filters with Latent Maps

DVRL Deep Variational Reinforcement Learning

EKF Extended Kalman Filter

ELBO Evidence Lower-Bound

GTM-SM Generative Temporal Models with Spatial Memory

ICP Iterative Closest Point

KL Kullback-Leibler

MDP Markov Decision Process

MLE Maximum Likelihood Estimation

MLP Multi-Layer Perceptron

MPC Model-Predictive Control

NeRF Neural Radiance Field

NM Neural Map

POMDP Partially-Observable Markov Decision Process

RL Reinforcement Learning

RNN Recurrent Neural Network

SLAM Simultaneous Localisation and Mapping

SDF Signed Distance Function

SLAC Stochastic Latent Actor-Critic

SPL Success Weighted by Inverse Path Length

SSM State-Space Model

UKF Unscented Kalman Filter

VAE Variational Auto-Encoder

VIN Value Iteration Networks

VRM Variational Recurrent Model

Part I

INTRODUCTION AND BACKGROUND

1 INTRODUCTION

This thesis deals with *spatial reasoning*. To explain what spatial reasoning means, we can use an example that has been honed in discussions with colleagues and friends over the course of a couple of years.

Consider the problem of finding a common household object such as a bottle of milk in a previously unseen house. The task clearly requires exploration, because we are not yet familiar with the house. It is, however, not blind exploration. We can rule out a room before fully exploring it as soon as we realise it is a bathroom or a bedroom, which are usually not where bottles of milk are stored. In fact, in a first stage of the problem, we are probably not searching for the bottle at all, but instead trying to find the kitchen, which is where we expect most households to keep their food.

This example can be formalised as a Partially-Observable Markov Decision Process (POMDP). POMDPs are one of the most diverse objects in the mathematical landscape of modern control and reinforcement learning. In practice, this means that different types of POMDPs are difficult for different reasons. Spatial reasoning contains a particularly interesting set of POMDPs, where an autonomous agent has to actively and deliberately seek out information about the space it is operating in, while also balancing its exploration with a specific task it is trying to solve.

A long line of work has dealt with spatial reasoning tasks such as exploration and navigation with the language of POMDPs (Thrun, Burgard, and Fox, 2005). This approach naturally leads to treating the environment and the agent's state over time as random variables. Based on our observations, we try to find the posterior distribution over the agent's state and the environment and use those distributions to inform planning. A limitation of this approach is that modelling the dependency between agent states, the environment and observations becomes more difficult as we consider high-dimensional observations such as images.

On the other hand, image-based observations are easily handled by modern Simultaneous Localisation and Mapping (SLAM) systems (Engel, Koltun, and Cremers, 2016; Mur-Artal, Montiel, and Tardós, 2015). These methods isolate the problem of estimating the environment and the agent's location and orientation from downstream control and focus on the SLAM-backend as a single component that can be improved independently of any control task. Indeed, a SLAM backend itself often contains many components that can be isolated from the rest and improved upon independently.

Another line of work is summarised by the nascent umbrella term *world models* (Ha and Schmidhuber, 2018), and the more established paradigm of model-based control. These methods seek to build models, which capture the most relevant aspects of a system: the dynam-

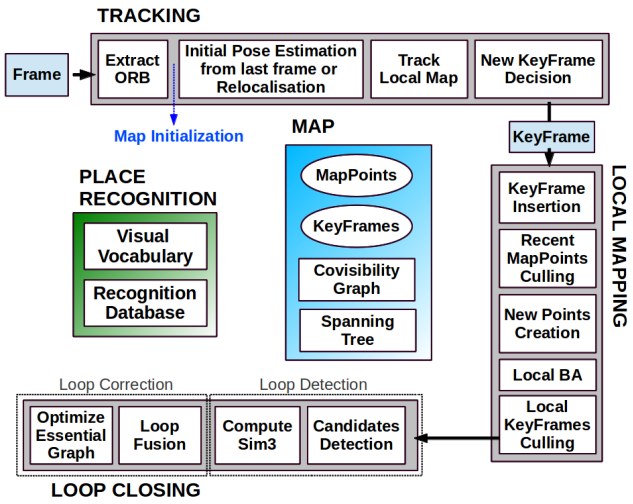

Figure 1: Overview of ORB-SLAM.
This figure is taken from (Mur-Artal, Montiel, and Tardós, 2015).

ics of the system state over time, the emission process which explains how observations are generated and the cost function, which specifies a task we wish to solve. With a model of the dynamics, emission and cost, we can use a host of well-established algorithms for control and state estimation. The model itself, which is typically called a *world model*, can be learned from observations alone, by posing the task as an approximate inference problem over the unobserved system state.

In contrast to more traditional methods where the variables of interest are interpretable physical quantities such as occupancy, landmark positions and agent's location and orientation, world models often work with an abstract internal representation of the problem. Owing partly to these abstract representations and partly to the usage of neural networks, which have historically been difficult to analyse and are often referred to as black-boxes, there are many unknowns regarding world models. We start this thesis by examining one such unknown in chapter 3. Next, in chapter 4 we chronicle the emergence of a series of world models tailored to spatial environments, culminating in our own work in chapter 5.

The tasks that we focus on for most of this thesis, namely localisation, mapping and navigation are regarded as solved problems by many (Cadena et al., 2016). We do not argue otherwise. Today, there exist SLAM systems that are mature enough that it is possible to take an off-the-shelf solution and apply it to a navigation task in a new environment.

Our ambition is to tackle these problems with the perspective of model-based control and within the framework of POMDPs. This does not necessarily give rise to algorithms that outperform existing solutions. However, viewing the pipeline of modelling, state estimation and control as a whole allows us to create systems with new capabilities, as we show in chapter 6, where we present a controller that can avoid taking actions which make it difficult to estimate the state of the agent.

In contrast, controllers that rely on state estimation are often designed such that the controller does not reason about state estimation at all. This is typically out of practicality: reasoning about the impact of control on state estimation is rarely feasible. We present a computationally tractable method that allows taking future state estimation into account during planning. Our goal is to bridge the gap between control and state estimation.

Similarly, we strive to bridge the gap between world models, traditional robotics and modern SLAM systems. We model the environment and agent states as random variables and model the dependency between these and camera observations. While research on world models is often demonstrated on simplified systems, we prioritise real-time capability and applicability in the real world. At the same time, we focus on the model and state estimator as components of a controller and examine how the system works as a whole.

2

BACKGROUND

For the sake of achieving a common vocabulary, this chapter gives a concise overview over the fundamental concepts that are used in the main work. Later chapters then contain a discussion of the relevant pieces of related work so as to keep the discussion between the state-of-the-art and methods newly introduced by our work close to each other.

Specifically, in chapter 3 we examine the literature of model-based policy search in partially-observed environments. Chapter 4 then discusses the history of world models for spatial environments, where we see various ways of specialising generic world models to tasks like navigation, mapping and exploration. This chapter leads into our own work on the topic. Finally, chapter 6 offers an overview of the literature on belief planning and contrasts the method presented in this thesis against existing methods.

2.1 ON NOTATION

Most of the math in this thesis involves vectors $\mathbf{x} \in \mathbb{R}^d$ and sequences of vectors $\mathbf{x}_{1:T} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$. Often, these vectors are samples from random variables $\mathbf{x} \sim \mathbf{X}$. For random variables, we use the same notation $p(\cdot)$ for probability density functions and probability mass functions (when discrete random variables are involved) and write cumulative distribution functions as:

$$p(\mathbf{x} \leqslant \mathbf{X}).$$

Thus, $p(\mathbf{x})$ for a vector $\mathbf{x} \in \mathbb{R}^d$, which is a sample from a random variable $\mathbf{x} \sim \mathbf{X}$ denotes the probability density function:

$$p(\mathbf{x}) = (\partial / \partial \mathbf{x}) p(\mathbf{X} \leqslant \mathbf{x}).$$

We write $\mathbf{x} \sim p(\mathbf{x})$ to signify that $\mathbf{x}$ is sampled according to the density $p(\mathbf{x})$. For conditional random variables, we use lower-caps letters:

$$\mathbf{z} \mid \mathbf{x}.$$

We often parametrise density functions as $p_\theta(\mathbf{x} \mid \mathcal{C})$, where θ are learned parameters which map some set of conditions $\mathcal{C} = \mathbf{z}_1, \ldots, \mathbf{z}_N$ to the parameters of a probability density function. The most common example is a neural network or Multi-Layer Perceptron (MLP) which takes $(\mathbf{z}_1, \ldots, \mathbf{z}_N)$ as input and produces the mean and standard deviation of Gaussian distribution. In this case we write:

$$\begin{aligned} p_\theta(\mathbf{x} \mid \mathcal{C}) &= \mathcal{N}(\mathbf{x} \mid [\mu, \sigma] = \mathrm{MLP}(\mathcal{C})), \\ &= \mathcal{N}(\mathbf{x} \mid \mathrm{MLP}(\mathcal{C})). \end{aligned}$$

Using μ and σ to denote the mean and standard deviation. We sometimes use the notions of function and probability interchangeably. As an example, if we have an emission probability $p(\mathbf{x} \mid \mathbf{z})$, we use the term *emission function* to mean a function (possibly parametrised by weights θ) that takes $\mathbf{z}$ and produces the parameters of the density $p(\mathbf{x} \mid \mathbf{z})$, which would be μ and σ in the case of a Gaussian density.

2.2 MARKOV DECISION PROCESSES

We are interested in problems where the state of a system, denoted by $\mathbf{z}_t$ changes over time under the influence of a control input $\mathbf{u}_t$ according to stochastic dynamics. The state and control produce a cost signal $\mathbf{c}_t$, which defines a task that should be solved. This setup can be summaries through the following:

$$\begin{aligned} \mathbf{z}_1 &\sim p_1(\mathbf{z}_1), \\ \mathbf{z}_{t+1} &\sim p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t), \\ \mathbf{c}_t &\sim p(\mathbf{c}_t \mid \mathbf{z}_t, \mathbf{u}_t), \\ \mathbf{u}_t &= \pi(\mathbf{z}_t). \end{aligned} \tag{1}$$

Here, $p_1(\mathbf{z})$ denotes an initial state distribution and π is a policy function which maps the state to a control. This is a mathematical formalism known as a Markov Decision Process (MDP) (Bellman, 1957). In this thesis we are mainly interested in infinite-horizon MDPs. That is, given a starting state $\mathbf{z}_1$, we receive an expectation over an infinite sum of costs:

$$J^\pi(\mathbf{z}) = \mathbb{E}\Big[\sum_{t=1}^{\infty} \beta^{t-1} \mathbf{c}_t \mid \mathbf{z}_1 = \mathbf{z}\Big]. \tag{2}$$

Here, β is a discount factor with $0 < \beta < 1$, which ensures that the sum is finite, as long as $|\mathbf{c}_t| < \infty$ for all t. Note that $\beta = 1$ is allowed for some classes of problems, an example of which are *stochastic shortest path problems*, which are—among other things—one way of formalising a navigation problem.

We call $J^\pi(\mathbf{z})$ the *cost-to-go* of $\mathbf{z}$ under the policy π. The goal of optimal control is to find a policy which minimises the cost-to-go of the initial state under the initial state distribution:

$$\pi^* = \arg\min_\pi \mathbb{E}_{\mathbf{z}_1}[J^\pi(\mathbf{z}_1)].$$

If the states and controls belong to discrete sets $\mathcal{Z}$ and $\mathcal{U}$, the optimal policy can be found through a dynamic programming algorithm called *value iteration*.

Value iteration starts from an arbitrarily initialised cost-to-go function J^0 and converges to the cost-to-go function of the optimal policy $J^\star$ by repeatedly applying the Bellman operator:

$$J^k = \min_{\mathbf{u}} \mathbf{C}_\mathbf{u} + \beta T_\mathbf{u} J^{k-1}, \quad \text{for } k \geqslant 1,$$

where we introduce the vector $\mathbf{C}_\mathbf{u} \in \mathbb{R}^{|\mathcal{Z}|}$ and matrix $T_\mathbf{u} \in \mathbb{R}^{|\mathcal{Z}| \times |\mathcal{Z}|}$, which are the vector of costs for executing the control $\mathbf{u}$ from each

state and the transition probability matrix for the same. Note that for discrete state and control spaces, the cost-to-go function is also a vector $J^k \in \mathbb{R}^{|\mathcal{Z}|}$. As $k \to \infty$, the cost-to-go converges to the cost-to-go of the optimal cost function, that is, $\lim_{k\to\infty} J^k = J^\star$. The optimal policy can then be retrieved as:

$$\pi^\star = \arg\min_{\mathbf{u}} \mathbf{C}_\mathbf{u} + \beta \mathsf{T}_\mathbf{u} J^\star.$$

For continuous state and control spaces, the value iteration algorithm is not applicable. A common choice here is to discretise the state and control spaces and apply value iteration in the discretised space (Bertsekas, 2005).

2.3 MODEL-PREDICTIVE CONTROL

The term Model-Predictive Control (MPC) is used to describe slightly different algorithms in different fields. Other terms related to MPC are *limited-lookahead control*, *receding horizon control* and *rollout* (Bertsekas, 2005). The central idea of all of these is to simplify the optimal control problem by separating it into a short planning horizon and a future term, represented by some terminal cost function $\hat{J}(\cdot)$.

In this thesis, MPC is associated with any algorithm which solves the following minimisation problem:

$$\begin{aligned} &\arg\min_{\mathbf{u}_{1:T-1}} \mathcal{L}_{\text{mpc}}(\mathbf{u}_{1:T-1}, \mathbf{z}_1), \\ &\mathcal{L}_{\text{mpc}}(\mathbf{u}_{1:T-1}, \mathbf{z}_1) = \mathbb{E}_{p(\mathbf{z}_{2:T}|\mathbf{z}_1, \mathbf{u}_{1:T-1})}\left[\beta^{T-1}\hat{J}(\mathbf{z}_T) + \sum_{t=1}^{T-1} \beta^{t-1}\mathbf{c}_t\right]. \end{aligned} \tag{3}$$

Starting from the current system state $\mathbf{z}_1$. The terminal cost function, $\hat{J}(\mathbf{z})$ is usually an approximation of the cost-to-go under an optimal policy J^*. The purpose of the terminal cost is to reduce the bias introduced by limiting the optimisation to a horizon of T steps.

A key point in MPC is that the control sequence which minimises eq. (3), $\mathbf{u}^\star_{1:T-1}$ is not used in its entirety. Only the first control is used, after which we plan a new sequence, setting $\mathbf{z}_1 := \mathbf{z}_2$.

A typical application of MPC to a continuous control problem might discretise the state and control space to use value iteration and obtain an approximate cost-to-go $\hat{J}(\mathbf{z})$. This can then be used as a terminal cost in MPC. This strategy will form the basis of our approach to the navigation task later on.

Many optimisation algorithms can be used to solve eq. (3). Here we present two methods. Both approaches rely on approximating the expectation via Monte Carlo sampling. For that, we take K sample rollouts:

$$(\mathbf{z}^k_{2:T}, \mathbf{c}^k_{1:T-1})_{k=1}^K \sim p(\mathbf{z}_{2:T}, \mathbf{c}_{1:T} \mid \mathbf{z}_1, \mathbf{u}_{1:T-1}),$$

and approximate the objective as:

$$\begin{aligned}\mathcal{L}_{\text{mpc}}(\mathbf{u}_{1:T-1}, \mathbf{z}_1) &\approx \tilde{\mathcal{L}}_{\text{mpc}}(\mathbf{u}_{1:T-1}, \mathbf{z}_1) \\ &= \sum_{k=1}^{K} \left[\beta^{T-1} \hat{J}(\mathbf{z}_T^k) + \sum_{t=1}^{T-1} \beta^{t-1} \mathbf{c}_t^k \right]. \end{aligned} \tag{4}$$

If the transition and cost functions are differentiable, we can use gradient descent to solve the minimisation problem:

$$\mathbf{u}_{1:T-1} \leftarrow \mathbf{u}_{1:T-1} - \alpha \nabla \tilde{\mathcal{L}}_{\text{mpc}}(\mathbf{u}_{1:T-1}, \mathbf{z}_1).$$

The gradient descent rule is used either until convergence or for a fixed number of iterations.

When the transition or cost function is non-differentiable, we have to resort to gradient-free optimisation algorithms, the simplest of which is random search. Here we take a set of candidate plans $\mathbf{u}_{1:T-1}^{l}{}_{l=1}^{L}$ and take the plan which minimises the objective:

$$\mathbf{u}_{1:T-1}^{\star} = \arg\min_{l} \tilde{\mathcal{L}}_{\text{mpc}}(\mathbf{u}_{1:T-1}^{l}, \mathbf{z}_1).$$

More complex methods for gradient-free optimisation usually involve an iterative procedure where the set of candidates is gradually refined (Hansen and Ostermeier, 1996; Mania, Guy, and Recht, 2018; Rubinstein, 1997; Williams et al., 2016).

2.4 PARTIALLY-OBSERVABLE MDPS

The main focus of this thesis are problems where the state cannot be observed directly, but is revealed through partial observations $\mathbf{x}_t$, which are produced by a stochastic emission process (Åström, 1965). Here, we extend and modify eq. (1) by including the observed variables $\mathbf{x}_t$ and changing the structure of the policy π:

$$\begin{aligned}\mathbf{x}_t &\sim p(\mathbf{x}_t \mid \mathbf{z}_t), \\ \mathbf{u}_t &= \pi(\mathbf{x}_{1:t}). \end{aligned} \tag{5}$$

Note that the policy is now no longer looking at $\mathbf{z}_t$, but at the history of observations $\mathbf{x}_{1:t}$. In general, the optimal policy of a partially-observable Markov decision process (POMDP) is a function of the observation history $\mathbf{I}_t = \mathbf{x}_{1:t}$ or a sufficient statistic thereof. The interaction history can be summarised by the posterior distribution over the system state given previous observations:

$$p(\mathbf{z}_t \mid \mathbf{x}_{1:t}).$$

This distribution is often called the *belief*.

While we were easily able to characterise the optimal policy for an MDP, doing so is much more challenging for POMDPs. Any algorithm that computes the optimal policy or optimal cost-to-go for a POMDP has to work with either the interaction history $\mathbf{I}_t$ or a sufficient statistic of it, the belief being one example. The interaction

history is an unwieldy object because its size changes over time. Sufficient statistics like the belief do not exhibit this problem. In fact, we can use the belief to reduce any POMDP to an MDP.

For doing so, we introduce the variable $\mathbf{b}_t$, which induces a density over the state space:

$$p_{\mathbf{b}_t}(\mathbf{z}) = p(\mathbf{z}_t \mid \mathbf{x}_{1:t}).$$

The initial belief $\mathbf{b}_0$ corresponds to the initial state distribution $p_1(\mathbf{z})$. The belief after receiving the first observation is then:

$$p_{\mathbf{b}_1}(\mathbf{z}) = \frac{p(\mathbf{x}_1 \mid \mathbf{z}) p_1(\mathbf{z})}{\int p(\mathbf{x}_1 \mid \mathbf{z}') p_1(\mathbf{z}') d\mathbf{z}'}.$$

Given a current belief $\mathbf{b}_t$, a control $\mathbf{u}_t$ and a new observation $\mathbf{x}_{t+1}$, the new belief follows from chaining the transition and emission functions:

$$\begin{aligned} p_{\mathbf{b}_{t+1}}(\mathbf{z}) &= p(\mathbf{z} \mid \mathbf{b}_t, \mathbf{u}_t, \mathbf{x}_{t+1}), \\ &= \frac{1}{Z} p(\mathbf{x}_{t+1} \mid \mathbf{z}) \int p_{\mathbf{b}_t}(\mathbf{z}_t) p(\mathbf{z} \mid \mathbf{z}_t, \mathbf{u}_t) d\mathbf{z}_{t+1}, \end{aligned} \tag{6}$$

with normalising constant:

$$Z = \int p(\mathbf{x}_{t+1} \mid \mathbf{z}') \int p_{\mathbf{b}_t}(\mathbf{z}_t) p(\mathbf{z}' \mid \mathbf{z}_t, \mathbf{u}_t) d\mathbf{z}_{t+1} d\mathbf{z}'.$$

We refer to eq. (6) as the belief update, and use the shorthand $\mathbf{b}_{t+1} = b(\mathbf{b}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$ to denote the transition into the new belief state.

Any POMDP can be reduced to an MDP where the state is the belief state $\mathbf{b}_t$ and the transition and cost functions are defined as:

$$\begin{aligned} p(\mathbf{b}_{t+1} \mid \mathbf{b}_t, \mathbf{u}_t) = \int & \mathbb{I}[\mathbf{b}_{t+1} = b(\mathbf{b}_t, \mathbf{u}_t, \mathbf{x}_{t+1})] \\ & p(\mathbf{x}_{t+1} \mid \mathbf{z}_{t+1}) \\ & p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t) \\ & p_{\mathbf{b}_t}(\mathbf{z}_t) d\mathbf{x}_{t+1} d\mathbf{z}_{t+1} d\mathbf{z}_t, \\ p(\mathbf{c}_t \mid \mathbf{b}_t, \mathbf{u}_t) = \int & p(\mathbf{c}_t \mid \mathbf{z}_t) p_{\mathbf{b}_t}(\mathbf{z}_t) d\mathbf{z}_t. \end{aligned} \tag{7}$$

Thus, the cost function of this MDP integrates the cost function of the original MDP over the belief, while the transition function does so for both the transition and emission functions, as well as the belief update function.

It slowly becomes clear why solving POMDPs is much harder than solving MDPs. Even though the belief state representation is handier than the interaction history, due to its finite length, it presents a new problem. The belief is a continuous variable, even when the state and control are discrete. We therefore cannot use a dynamic programming approach naively even for fully discrete systems.[1]

1 It is indeed possible to use dynamic programming to solve some POMDPs by using different representations of the belief space and we will give a deeper overview of POMDP-solvers in chapter 6. Nevertheless it is true that solving POMDPs is much more computationally demanding than solving MDPs.

Moreover, suboptimal but efficient approaches like MPC, which rely on simulating the system for some amount of steps are also often out of the question. That is because simulating the transition of a POMDP is much more expensive than doing so for an MDP. Simulating a POMDP requires repeatedly applying eq. (7), which requires evaluating both the emission function and the state estimator. For many systems, these are much more expensive to compute than the transition itself, certainly whenever high-dimensional observations are involved. In addition to the computational cost, including the state estimator inside the optimisation objective may present further limitations. For instance, many state estimators are not differentiable, which would immediately rule out gradient-based optimisation.

2.5 MPC WITH STATE ESTIMATION

The challenging nature of POMDPs inspires many suboptimal strategies. The simplest of these is to separate state estimation and control completely. Here, the controller takes the current belief and picks controls under the assumption that no further observations will be taken.

At this point we make a distinction between state estimates and the true posterior distribution over the state. For the latter, we continue to use $\mathbf{b}_t$. For the former, we introduce the variable $\mathbf{h}_t$, which is the internal state of the state estimator. We refer to $\mathbf{h}_t$ as a *carry*. Similar to how $\mathbf{b}_t$ induces a probability over states, the carry is also associated with a distribution over the state space, which we denote $q(\mathbf{z}_t \mid \mathbf{h}_t)$. Finally, the carry is also subject to an update function $\mathbf{h}_{t+1} = h(\mathbf{h}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$.

To make the concept of carry more concrete, we can look at a couple of examples for popular state estimators. Perhaps the most ubiquitous of all state estimators is the Kalman filter, which maintains a Gaussian state estimate. The carry of the Kalman filter is simply the mean and covariance matrix of the current state estimate. The update function $h(\mathbf{h}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$ is then given by the Kalman update. Similarly, for a particle filter, the carry would be the set of particles and their weights. For a Recurrent Neural Network (RNN), the carry would be the RNN state.

Model-predictive control with a state estimator boils down to extending eq. (4) with one more expectation, which goes over the state estimate:

$$\mathbb{E}_{q(\mathbf{z}_1 \mid \mathbf{h}_1)}\left[\tilde{\mathcal{L}}_{\text{mpc}}(\mathbf{u}_{1:T-1}, \mathbf{z}_1)\right].$$

In practice, this can similarly be approximated via Monte Carlo estimation:

$$\mathbb{E}_{q(\mathbf{z}_1 \mid \mathbf{h}_1)}\left[\tilde{\mathcal{L}}_{\text{mpc}}(\mathbf{u}_{1:T-1}, \mathbf{z}_1)\right] \approx \sum_{m=1}^{M} \tilde{\mathcal{L}}_{\text{mpc}}(\mathbf{u}_{1:T_1}, \mathbf{z}_1^m),$$

with $\mathbf{z}_1^1, \ldots, \mathbf{z}_1^M \sim q(\mathbf{z}_1 \mid \mathbf{h}_1)$.

2.6 VARIATIONAL INFERENCE

A ubiquitous strategy in machine learning is to model prior terms and likelihood terms parametrically:

$$\mathbf{z} \sim p_\theta(\mathbf{z}) \quad \mathbf{x} \mid \mathbf{z} \sim p_\theta(\mathbf{x} \mid \mathbf{z}).$$

We refer to such models as *generative models*. A generative model describes how a set of observed variables $\mathbf{x}$ are created from a set of unobserved values $\mathbf{z}$. Generative models are interesting to this thesis for two reasons. First, they can be used as *world models*, which allow us to simulate a dynamical system by learning an approximation of it from sequences of observation. Second, they allow us to reason about unobserved quantities of the system, denoted by $\mathbf{z}$, which are relevant for decision-making.

The two use cases bring forth two challenges. The first is that we would like to learn θ, the parameters of the system. The standard approach for doing so is Maximum Likelihood Estimation (MLE). Unfortunately, MLE is not straightforward in a model with unobserved variables. Optimising the likelihood of the observed variables would require us to marginalise over the unobserved ones:

$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z}.$$

The second challenge is that we need to estimate the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$. Posterior distributions are easy to characterise using Bayes' theorem:

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{\int p(\mathbf{x} \mid \mathbf{z}') p(\mathbf{z}') \mathrm{d}\mathbf{z}'}.$$

Here we find another marginalisation, that is, another integral.

Both the challenge of learning model parameters and that of obtaining the posterior are only tractable for simple cases such as systems where observations and latent states are jointly Gaussian-distributed or discrete systems with a tractable size. For more powerful models, for instance those which feature non-linear relationships between the state and the observation, we resort to approximate Bayesian inference methods. Most notable to this thesis is *variational inference*.

Variational inference (Jordan et al., 1999) introduces a new concept, that of the variational approximate posterior distribution, denoted $q_\phi(\mathbf{z} \mid \mathbf{x})$, with ϕ denoting learnable parameters. The approximate posterior distribution has a tractable form such as a Gaussian distribution parametrised by learned mean and standard deviation parameters. A more sophisticated model such as a Variational Auto-Encoder (VAE) (Kingma and Welling, 2022) lets these mean and standard deviation parameters be predicted by an inference network, which takes the observation as input:

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z} \mid \mathrm{MLP}(\mathbf{x})).$$

The approximate posterior allows us to simultaneously learn model parameters and approximate the posterior $\mathbf{z} \mid \mathbf{x}$ by way of a simple trick:

$$\begin{aligned}\log p_\theta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\ &= \log \int \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} q_\phi(\mathbf{z} \mid \mathbf{x}) d\mathbf{z} \\ &\geqslant \int \Big(\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} \Big) q_\phi(\mathbf{z} \mid \mathbf{x}) d\mathbf{z} = \mathcal{L}_{\text{elbo}}.\end{aligned} \tag{8}$$

The inequality follows from Jensen's inequality. The final term is called the Evidence Lower-Bound (ELBO), evidence being another name for $p_\theta(\mathbf{x})$ or $\log p_\theta(\mathbf{x})$. For our purposes, it is useful to rewrite the ELBO as:

$$\mathcal{L}_{\text{elbo}} = \int \Big(\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} \Big) q_\phi(\mathbf{z} \mid \mathbf{x}) d\mathbf{z} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \Big[\log p_\theta(\mathbf{x} \mid \mathbf{z}) \Big]}_{\text{reconstruction}} - \underbrace{\text{KL} \left[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z}) \right]}_{\text{prior}}.$$

This form of the ELBO breaks down into a *reconstruction term* and a *prior term*. For the reconstruction term, we maximise the likelihood of observations under the approximate posterior. For the prior term, we minimise the Kullback-Leibler (KL) divergence of the approximate posterior from the prior.

By simultaneously optimising the model parameters θ and the variational parameters ϕ, we can address both previously mentioned challenges. Here, an important fact is that the inequality in eq. (8) is tight if and only if the approximate posterior distribution equals the true posterior, in the sense that:

$$\text{KL} \left[q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x}) \right] = 0.$$

2.7 LEARNING POMDPS

A central goal of model-based Reinforcement Learning (RL) is to learn an approximate model of a POMDP from a set of sequences of interaction data:

$$\mathcal{D} = (\mathbf{x}^i_{1:T}, \mathbf{u}^i_{1:T-1}, \mathbf{c}^i_{1:T-1})_{i=1}^N.$$

Here, we replace each functional dependency by a parametric model such as a neural network and learn the parameters, jointly referred to as θ. One complicating factor are the unknown hidden states $\mathbf{z}_{1:T}$. The predominant method here is to resort to variational inference, assisted by an inference network $q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})$, which produces an approximation to the posterior $\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}$.

Using the inference network, we can optimise a lower-bound to the likelihood of the data:

$$\begin{aligned}\log p_\theta(\mathbf{x}_{1:T}, \mathbf{c}_{1:T-1}) \geqslant -\mathcal{L}_{\text{elbo}} = \mathbb{E}\, q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})\Big[& \\ & \log p_\theta(\mathbf{x}_{1:T}, \mathbf{c}_{1:T} \mid \mathbf{z}_{1:T}) + \\ & \log p_\theta(\mathbf{z}_{1:T}) - \\ & \log q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})\Big]. \end{aligned} \tag{9}$$

To further break down the ELBO, we need to look at the factorisation of the distributions over time:

$$p_\theta(\mathbf{z}_{1:T}) = p_\theta(\mathbf{z}_1) \prod_{t=2}^{T} p_\theta(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}),$$

$$p_\theta(\mathbf{x}_{1:T} \mid \mathbf{z}_{1:T}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}_t \mid \mathbf{z}_t).$$

The factorisation of the inference network is a modelling decision which varies in the literature. Here, we present a method that closely follows the work of Bayer et al. (2021), though there are many implementations of this framework (Becker-Ehmck et al., 2020; Hafner et al., 2019; Karl et al., 2017). The inference network is partitioned into an initial inference network $q_\phi(\mathbf{z}_1 \mid \mathbf{x}_{1:T})$ and a recurrent inference network $q_\phi(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t, \mathbf{x}_{t+1:T})$. The factorisation of the approximate posterior is then:

$$q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}) = q_\phi(\mathbf{z}_1 \mid \mathbf{x}_{1:T}) \prod_{i=2}^{T} q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}, \mathbf{x}_{t:T}).$$

With these factorisations, we can decompose the negative ELBO into time-indexed terms:

$$\begin{aligned}\mathcal{L}_{\text{elbo}} = \mathbb{E}\, q_\phi(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T})\Big[& \\ & \mathrm{KL}\left[q_\phi(\mathbf{z}_1 \mid \mathbf{x}_{1:T}) \parallel p_\theta(\mathbf{z}_1)\right] \\ & + \sum_{t=2}^{T} \mathrm{KL}\left[q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}, \mathbf{x}_t) \parallel p_\theta(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1})\right] \\ & - \sum_{t=1}^{T} \log p_\theta(\mathbf{x}_t \mid \mathbf{z}_t) \\ & - \sum_{t=1}^{T} \log p_\theta(\mathbf{c}_t \mid \mathbf{z}_t)\Big]. \end{aligned} \tag{10}$$

In the implementation of Bayer et al. (2021), the initial and recurrent inference parts use an RNN extracting a sequence of features $\mathbf{f}_{1:T}$ from the observations. The features are defined as:

$$\mathbf{f}_t = \mathrm{RNN}(\mathbf{x}_{t:T}).$$

The approximate posterior over the initial step is then predicted by a neural network, which produces the mean and standard deviation of a Gaussian distribution given the feature for the first time step:

$$q_\phi(\mathbf{z}_1 \mid \mathbf{x}_{1:T}) = \mathcal{N}\big(\mathbf{z}_1 \mid [\mu_{\mathbf{z}_1}, \sigma_{\mathbf{z}_1}] = \mathrm{MLP}_a(\mathbf{f}_1)\big).$$

The prior over the first time step $p_\theta(\mathbf{z}_1)$ is a RealNVP normalising flow (Dinh, Sohl-Dickstein, and Bengio, 2017). The prior and approximate posterior state recurrence both use neural networks to predict the parameters of a Gaussian distribution:

$$p_\theta(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}\big(\mathbf{z}_{t+1} \mid [\mu_{\mathbf{z}_{t+1}}, \sigma_{\mathbf{z}_{t+1}}] = \mathrm{MLP}_b(\mathbf{z}_t, \mathbf{u}_t)\big),$$
$$q_\phi(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t, \mathbf{x}_{t+1:T}) = \mathcal{N}\big(\mathbf{z}_{t+1} \mid [\mu_{\mathbf{z}_{t+1}}, \sigma_{\mathbf{z}_{t+1}}] = \mathrm{MLP}_c(\mathbf{z}_t, \mathbf{u}_t, \mathbf{f}_{t+1})\big).$$

Finally, the emission and cost distributions also use the same paradigm:

$$p_\theta(\mathbf{x}_t \mid \mathbf{z}_t) = \mathcal{N}\big(\mathbf{x}_t \mid [\mu_{\mathbf{x}_t}, \sigma_{\mathbf{x}_t}] = \mathrm{MLP}_d(\mathbf{z}_t)\big)$$
$$p_\theta(\mathbf{c}_t \mid \mathbf{z}_t) = \mathcal{N}\big(\mathbf{c}_t \mid [\mu_{\mathbf{c}_t}, \sigma_{\mathbf{c}_t}] = \mathrm{MLP}_e(\mathbf{z}_t)\big).$$

2.8 WORLD MODELS AND POLICY LEARNING

A learned model of a POMDP comprises the initial state distribution, state transition, emission and cost:

$$\mathbf{z}_1 \sim p_\theta(\mathbf{z}_1) \qquad \mathbf{z}_{t+1} \sim p_\theta(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t)$$
$$\mathbf{x}_t \sim p_\theta(\mathbf{x}_t \mid \mathbf{z}_t) \qquad \mathbf{c}_t \sim p_\theta(\mathbf{c}_t \mid \mathbf{z}_t, \mathbf{u}_t).$$

The model, which is typically called a *world model* can be used to train a parametric policy π through model simulations. The possibilities for doing so are as wide as those for model-free RL methods. Here we will discuss gradient-based optimisation by differentiating through the model, as this is the method most relevant to this thesis. Notable implementations of this framework can be found in the work of Hafner et al. (2020b) and Becker-Ehmck et al. (2020).

For a finite-horizon problem with a discounted cost, policy search with model parameters θ boils down to:

$$\arg\min_\pi \mathbb{E}_{\mathbf{z}_1 \sim p_\theta(\mathbf{z}_1), \mathbf{c}_i \sim p_\theta(\mathbf{c}_i \mid \mathbf{z}_1, \pi)} \Big[\sum_{i=1}^{T} \beta^{i-1} \mathbf{c}_i \Big]. \tag{11}$$

Where the controls are picked by a policy network looking at the current state $\mathbf{u}_t = \pi(\mathbf{z}_t)$. If the horizon T is short enough, we can optimise eq. (11) through gradient descent by sampling rollouts from the model with ancestral sampling. The expectation is then typically approximated with a batch of initial states and a single Monte Carlo sample over the future. The states $\mathbf{z}_t$ are usually modelled as Gaussians, allowing differentiation with the reparametrisation trick (Kingma and Welling, 2022). Discrete states have also been used with straight-through gradient estimation (Bengio, Léonard, and Courville, 2013; Hafner et al., 2020a).

For the infinite-horizon discounted case, we can truncate the horizon by introducing a critic J_ψ with parameters ψ:

$$\arg\min_\pi \mathbb{E}_{\mathbf{z}_1 \sim p_\theta(\mathbf{z}_1), \mathbf{c}_i \sim p_\theta(\mathbf{c}_i \mid \mathbf{z}_1, \pi)} \Big[\beta^{T-1} J_\psi(\mathbf{z}_T) + \sum_{i=1}^{T-1} \beta^{i-1} \mathbf{c}_i \Big]. \tag{12}$$

The critic should be an accurate approximation of the cost-to-go under the policy π:

$$J_\psi(\mathbf{z}) \approx \mathbb{E}_{\mathbf{c}_i, \mathbf{u}_i \sim p_\theta(\mathbf{c}_i|\mathbf{z}_1=\mathbf{z}, \pi)} \Big[\sum_{i=1}^{\infty} \beta^{i-1} \mathbf{c}_i \Big].$$

In practice this is done by training policy and critic networks jointly. The policy network is trained by gradient descent on eq. (12), where the rollout is calculated through ancestral sampling through the model and by evaluating the policy on sampled states. The critic network is trained through supervised learning on the Bellman error:

$$\arg\min_{J_\psi} \mathbb{E}_{\mathbf{z}_1 \sim p_\theta(\mathbf{z}_1)} \Big[\| J_\psi(\mathbf{z}_1) - \tilde{J}(\mathbf{z}_1) \|_2^2 \Big],$$

with:

$$\tilde{J}(\mathbf{z}_1) = \mathbb{E}_{\mathbf{c}_i \sim p_\theta(\mathbf{c}_i|\mathbf{z}_1, \pi)} \Big[\beta^{T-1} J_\psi(\mathbf{z}_T) + \sum_{i=1}^{T-1} \beta^{i-1} \mathbf{c}_i \Big]. \tag{13}$$

Model-based RL is a notoriously brittle process with no convergence guarantees. As such, practical implementations contain many tricks. We list the ones that are relevant for our work.

TEMPORAL DIFFERENCE OBJECTIVES The objectives in eq. (12) and eq. (13) are replaced by a temporal difference objective (Sutton, 1988), which is a weighted sum of k-step objectives for $k = 1 \dots T$:

$$J^\lambda(\mathbf{z}_1) = (1-\lambda) \sum_{k=1}^{T} \lambda^{k-1} J_k(\mathbf{z}_1), \quad J_k(\mathbf{z}) = \sum_{t=1}^{k} \beta^{t-1} \mathbf{c}_t + \beta^k J_\psi(\mathbf{z}_k).$$

This specific temporal difference objective is referred to as TD(λ). The goal is to balance accuracy and efficiency. Rolling out the model for longer horizons allows for more complex behaviours, though at the cost of lower accuracy. Under stochastic dynamics, even if the model is perfect, the total cost of a 10-step simulation will be noisier than that of a 3-step one. By averaging shorter-horizon estimates with longer ones, TD(λ) seeks to strike the best of both worlds.

INITIAL STATE DISTRIBUTION Instead of sampling from the prior over the initial state $p_\theta(\mathbf{z}_1)$, it is common to maintain a replay buffer of inferred states, that is, samples from $q(\mathbf{z}_t^N)$, and use these as the starting point of model rollouts. This has two benefits. The first is that inferred states are generally more accurate than samples from the learned prior, since the learned prior has to accommodate the entire data distribution, and is therefore solving a harder optimisation problem. At the same time, the learned prior is only approximating the distribution over the first time step, whereas we would ideally want the policy to gain experience from all time steps in an episode. Doing by starting from the first step would only be possible if we use long training rollouts, but that is both computationally expensive and suffers from modelling errors building up over time. Thus, it is more efficient to sample from inferred states over all time steps and use relatively short training rollouts.

TARGET NETWORKS The training of the critic is self-supervised, as the targets are computed by re-using the current critic. To stabilise this procedure, it is common to maintain two copies of the critic. One is updated continuously, while the other is only used for computing targets and is either kept frozen for a set number of time steps, or updated via a parameter-averaging scheme. Here, a common choice is Polyak-averaging:

$$\psi_{t+1}^{\text{target}} = \eta\psi_t^{\text{target}} + (1-\eta)\psi,$$

where ψ^{target} are the parameters of the target critic and ψ those of the critic that is updated continuously.

2.9 DISCUSSION ON MODEL-BASED POLICY LEARNING FOR POMDPS

The approach outlined in section 2.8 is similar to the MPC and state estimator combination from section 2.5. Both separate control and state estimation. In the model-based RL strategy, we use an inference network to estimate the state of a learned POMDP, which is fed to a policy that is trained to be optimal on the learned POMDP. If we assume that the learned POMDP is an exact replica of the original POMDP from which the training data came, then we can safely say that the policy cannot reason about state estimation or how future observations might change its belief. On the other hand, in most scenarios this is unlikely to be the case.

In practice, learned POMDPs have abstract state spaces which differ from the original state space of the training data, since only the observations and costs are enforced to match those of the original system. In the end, the state space of a learned POMDP can learn belief-state-like representations, which would allow the policy to approximate belief planning. Unfortunately, as is often the case with neural networks, we cannot characterise the conditions where such representations would be learned.

Though the community's confidence grows as world models are successfully applied to policy learning in harder settings, empirical success in benchmark problems does not always produce knowledge about when and why a method can be expected to work well. That is to say, the evaluation of methodologies by benchmarks has to be supplemented with another line of work where hypotheses are formulated about how a method works, which can then be verified or falsified experimentally.

There is a need for both more theoretical and empirical work dissecting model-based policy search. The next chapter addresses one source of uncertainty regarding learning models of partially-observable systems and using them to train policies. We start from a theoretical result, which suggests that inference networks are often designed wrong in world models. We then design experiments to verify this in the context of reinforcement learning.

Part II

MAIN WORK

3

LESS SUBOPTIMAL LEARNING AND CONTROL IN VARIATIONAL POMDPS

Model-based control has achieved enormous success over the recent years. Using deep neural networks to parametrise sequential latent-variable models that enable purely simulation-based policy search, researchers have solved increasingly difficult control problems (Becker-Ehmck et al., 2020; Hafner et al., 2019, 2020a, 2023; Wu et al., 2022). Yet, in usual deep learning fashion, our ability to solve problems using our models does not reflect our knowledge about how these models work. This became clear when a paper by Bayer et al. (2021) revealed that many of our deep variational state-space models are implemented in a way that is subtly wrong. They introduced the concept of the *conditioning gap*, where a neural network trying to approximate a posterior distribution does not have access to all of variables that the posterior distribution depends on.

In the first chapter of our main work, we will investigate the effect of the conditioning gap on policy search with deep variational state-space models. We will make use of simple systems to test a simple hypothesis: that the conditioning gap can prevent learning a generative model that is faithful to the true system, which in turn prevents successful policy search. To that end, we will rely on large hyperparameter searches over two families of models, where one family conditions its inference network on all relevant parameters, while the other is more aligned with the practicality-oriented design choices that are common in the literature and ignores some of the relevant variables. The results will reveal a systematic under-performance on the side of the latter.

3.1 THE CONDITIONING GAP

The conditioning gap arises when an inference network is not conditioned on all of the variables that the posterior distribution it is trying to approximate depends on. Let $\mathbf{x}$ and $\mathbf{z}$ be two random variables, where we are interested in approximating the posterior $p(\mathbf{z} \mid \mathbf{x})$. Using the same notation as Bayer et al. (2021), we partition the observation $\mathbf{x}$ into variables that are inputs to the inference network C and variables that are not provided to it $\bar{C}$, as in $\mathbf{x} = (C, \bar{C})$.

What Bayer et al. (2021) have shown is that as long as $p(\mathbf{z} \mid C) \neq p(\mathbf{z} \mid C, \bar{C})$, two statements are true:

- The optimal amortised variational posterior matches neither $p(\mathbf{z} \mid C)$ nor $p(\mathbf{z} \mid C, \bar{C})$.
- The ELBO-optimal generative model p under the optimal amortised variational posterior does not match the maximum-likelihood model.

How is this relevant for us? As observed by Bayer et al. (2021), many implementations of deep variational state-space models partition the observations in this fashion when approximating the posterior over the system state at some time step given a sequence of past, present and future observations. Turning to a State-Space Model (SSM) with states $\mathbf{z}_{1:T}$, observations $\mathbf{x}_{1:T}$ and controls $\mathbf{u}_{1:T-1}$, we are typically interested in approximating the posterior $\mathbf{z}_t \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T}$ using a neural network.

In section 2.7, we outlined an implementation of deep variational state-space models where the approximate posterior is informed about all past and future observations. This was done by letting an RNN process the observations $\mathbf{x}_{1:T}$ in reverse order and creating feature vectors $\mathbf{f}_{1:T}$ with:

$$\mathbf{f}_t = \text{RNN}(\mathbf{x}_{t:T}).$$

The final approximate posterior was then the output of a neural network which takes the previous state and control and the current feature vector:

$$q(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid [\mu, \sigma] = \text{MLP}(\mathbf{z}_{t-1}, \mathbf{u}_{t-1}, \mathbf{f}_t)).$$

Interestingly, the work of Bayer et al. (2021) is rather atypical in this respect. Most publications on the topic feature inference networks that only use the observations up to the current time step, that is, $\mathbf{x}_{1:t}$. As examples, we will look at two of the most prominent deep variational state-space models, and how the approximate posterior is conditioned in each.

DREAMER The Dreamer algorithm has undergone many iterations in the past years (Hafner et al., 2019, 2020a,b, 2023). At its core, the approach closely follows the description of model learning and policy search we provided in section 2.7 and section 2.8. The main difference is that the state $\mathbf{z}$ contains a deterministic part and a stochastic part. The deterministic part, which we denote with $\mathbf{h}_t$ here, uses the following update rule:

$$\mathbf{h}_{t+1} = \text{GRU}(\mathbf{h}_t, \mathbf{z}_t, \mathbf{u}_t),$$

where GRU signifies that a Gated Recurrent Unit is used for the update (Cho et al., 2014). The stochastic state, $\mathbf{z}_t$ is given by:

$$p(\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_t \mid [\mu, \sigma] = \text{MLP}(\mathbf{h}_t)).$$

The deterministic state naturally does not require an approximate posterior distribution. The approximate posterior for the stochastic state uses the current deterministic state and the current observation:

$$q(\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_t \mid [\mu, \sigma] = \text{MLP}(\mathbf{h}_t, \mathbf{x}_t)).$$

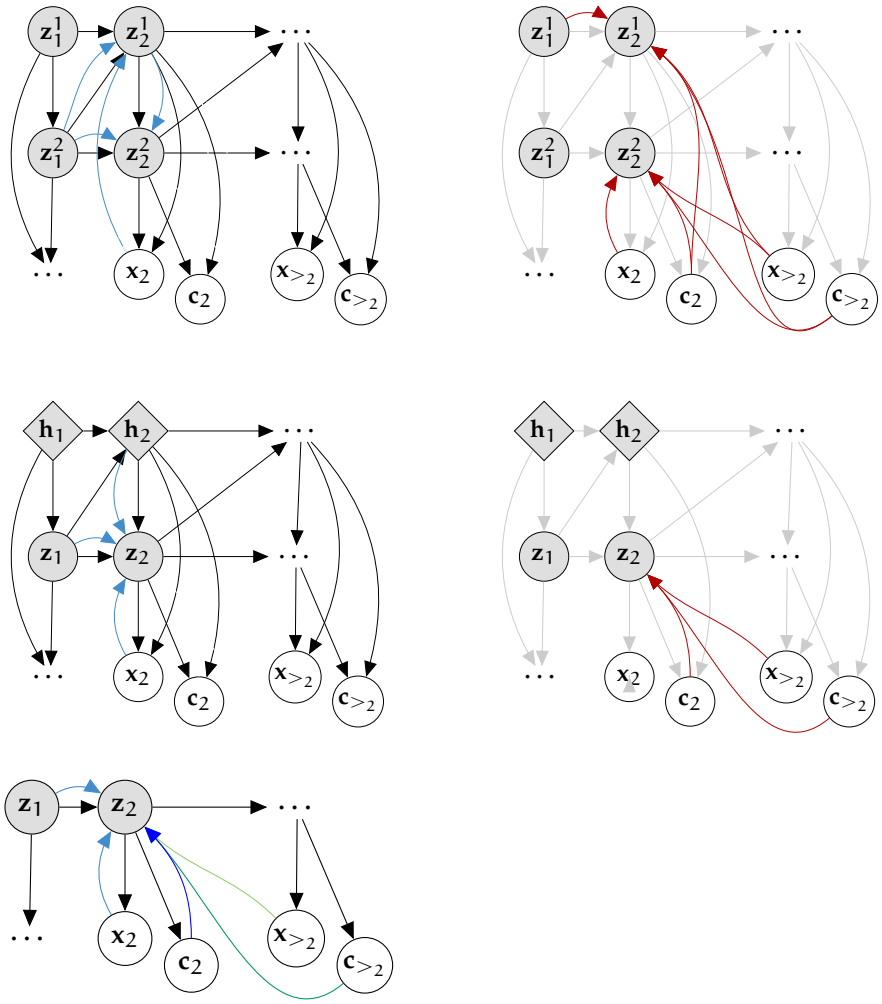

Figure 2: Graphical models of SLAC (top row), Dreamer (middle row) and the approach used in this chapter (bottom row). For SLAC and Dreamer, the left column shows the generative model with black arrows and the inference model with light blue arrows. The right column shows the missing edges in the inference model with red arrows. We only show the inference model for the second time step for clarity. For our method, we use different coloured arrows to show different kinds of conditioning. filter-no-cost only uses light blue arrows. filter-yes-cost uses dark blue arrows on top of light blue ones. smoother-no-cost uses light green arrows on top of all blue ones. smoother-yes-cost uses dark green arrows on top of the others.

STOCHASIC LATENT ACTOR CRITIC Stochastic Latent Actor-Critic (SLAC) by Lee et al. (2019) differs from both Dreamer and the approach from section 2.8 in that it does not train a policy with model-based simulations. Instead, the policy is trained using the soft actor-critic approach by (Haarnoja et al., 2018), where the latent state inferred by the inference network is used as a feature representation for the critic network. Similarly to Dreamer, SLAC features two sets of latent states, which we denote $\mathbf{z}_t^1$ and $\mathbf{z}_t^2$. Unlike Dreamer, both sets of variables are stochastic, and their evolution over time is modelled as:

$$\begin{aligned}
p(\mathbf{z}_1^1) &= \mathcal{N}(0, I), \\
p(\mathbf{z}_1^2 \mid \mathbf{z}_1^1) &= \mathcal{N}(\mathbf{z}_1^2 \mid \mathrm{MLP}(\mathbf{z}_1^1)), \\
p(\mathbf{z}_{t+1}^1 \mid \mathbf{z}_t^2) &= \mathcal{N}(\mathbf{z}_t^1 \mid \mathrm{MLP}(\mathbf{z}_t^2, \mathbf{u}_t)), \\
p(\mathbf{z}_{t+1}^2 \mid \mathbf{z}_t^2, \mathbf{z}_{t+1}^1) &= \mathcal{N}(\mathbf{z}_t^1 \mid \mathrm{MLP}(\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, \mathbf{u}_t)).
\end{aligned}$$

The approximate posterior is structured as:

$$\begin{aligned}
q(\mathbf{z}_1^1 \mid \mathbf{x}_1) &= \mathcal{N}(\mathbf{z}_1^1 \mid \mathrm{MLP}(\mathbf{x}_1)), \\
q(\mathbf{z}_1^2 \mid \mathbf{z}_1^1) &= p(\mathbf{z}_1^2 \mid \mathbf{z}_1^1), \\
q(\mathbf{z}_{t+1}^1 \mid \mathbf{x}_{t+1}, \mathbf{z}_t^2, \mathbf{u}_t) &= \mathcal{N}(\mathbf{z}_{t+1}^1 \mid \mathrm{MLP}(\mathbf{x}_{t+1}, \mathbf{z}_t^2, \mathbf{u}_t)), \\
q(\mathbf{z}_{t+1}^2 \mid \mathbf{z}_t^2, \mathbf{z}_{t+1}^1) &= p(\mathbf{z}_{t+1}^2 \mid \mathbf{z}_t^2, \mathbf{z}_{t+1}^1).
\end{aligned}$$

Note that aside from the omission of future observations, Lee et al. (2019) also decide to re-use parts of the generative model in the approximate posterior. This decision could lead to suboptimal posterior inference in itself, but does not fall under the conditioning gap, as it is not a result of missing inputs to a learned component.

Why is it, that these implementations decide to omit the future observations $\mathbf{x}_{t+1:T}$? The reason is linked to how these methods approach policy search in partially-observable problems. The learned deep variational state-space model allows reducing the policy search problem into one where the policy is trained using the system state $\mathbf{z}$. The policy then does not need to learn how to extract useful information about the system state from the observations, as it already has access to the system state.

The catch is that we now need to continuously estimate the state from observations, as the policy needs it as an input. It is convenient to simply re-use the inference network from the model-learning part to estimate the state during deployment as well. Now it becomes clear why most approaches only condition the inference network on information that is available up to the current time step: the future is unknown during deployment.

Aside from future observations, there is one more place where the conditioning gap comes into play. That is the treatment of the cost signal. When learning a state-space model for control, we have a fourth set of variables, the cost signal $\mathbf{c}_{1:T-1}$. The SSM needs to model these as well, since we need them for policy search. All approaches we are aware of treat the cost as another variable that appears in the reconstruction loss portion of the ELBO, but one that is not passed to

the inference network, with the exception of the work by Zhao et al. (2021), who are interested in a meta-RL use-case, where the cost signal is informative about which task the agent needs to solve. Thus, they use the cost to condition the approximate posterior, in addition to the current observation and previous state.

3.2 PROBING THE CONDITIONING GAP FOR POLICY SEARCH

To investigate the effect of the conditioning gap on policy search, we will adopt the standard model-based RL framework outlined in section 2.7 and section 2.8 and compare two modelling questions:

- Conditioning the inference network on $\mathbf{x}_{1:t}$ vs $\mathbf{x}_{1:T}$.
- Using the costs $\mathbf{c}_{1:T-1}$ as an input to the inference network or not.

Both of these questions are only relevant in the model-learning stage. The rest of the pipeline is identical in all cases. The first question is resolved by the setup of the inference network. In either case, the inference network consists of two parts. The first is an RNN, which processes the stream of observations to produce a sequence of features $\mathbf{f}_{1:T} = \text{RNN}(\mathbf{x}_{1:T})$. The second is an MLP, which takes the previous state and control as well as the current feature to predict the distribution parameters of the current state:

$$q(\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_t \mid [\mu_{\mathbf{z}_t}, \sigma_{\mathbf{z}_t}] = \text{MLP}(\mathbf{z}_{t-1}, \mathbf{u}_{t-1}, \mathbf{f}_t)).$$

To condition the inference network on $\mathbf{x}_{1:t}$, we let the RNN producing $\mathbf{f}_{1:T}$ process the observations $\mathbf{x}_{1:T}$ going forward in time. Thus the feature map $\mathbf{f}_t$ has only seen the observations $\mathbf{x}_{1:t}$. To let the inference network see the full sequence, $\mathbf{x}_{1:T}$, we reverse the sequence of observations so that the RNN moves backward in time. In other words, the feature map $\mathbf{f}_t$ has access to the observations $\mathbf{x}_{t:T}$. Looking at earlier observations is not necessary, as the random variable $\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{x}_{t:T}$ is independent of $\mathbf{x}_{1:t-1}$. Put differently, the previous state tells the MLP everything it would need to know about past observations. To give the inference network access to the cost, we simply concatenate these to the observations before they are provided to the RNN.

Figure 2 shows the four types of conditioning we examine with colour-coding. These are:

- filter-no-cost, which uses $\mathbf{x}_{1:t}$,
- filter-yes-cost, which uses $\mathbf{x}_{1:t}$ and $\mathbf{c}_{1:t}$,
- smoother-no-cost, which uses $\mathbf{x}_{t:T}$,
- smoother-yes-cost, which uses $\mathbf{x}_{t:T}$ and $\mathbf{c}_{t:T}$.

We use an online learning procedure where we alternate between steps of model-learning, policy search and environment interaction.

Environment interactions produce new data, which is accumulated over the course of training. Model-learning uses all previously collected samples. The procedure is similar to the Simple algorithm by Kaiser et al. (2020).

There are two differences between our setup and the one described in section 2.8. First, we omit the critic network. Learning critics is a difficult process in itself and complicates the already complicated procedure of model-based RL even further. We pick simple systems that can be solved without a critic in order to simplify the comparison between methods. In mathematical terms, omitting the critic means setting $J_\psi(\mathbf{z}_T) := 0$ in eq. (12).

The second difference is related to what the policy expects as input. Because future observations $\mathbf{x}_{t+1:T}$ are not available at deployment, we cannot generally re-use the inference network for state estimation. Therefore we learn a recurrent policy which processes the stream of observations directly, instead of a policy that expects access to a state estimate. This means the policy has to learn its own internal state estimator, and that training the policy also involves generating observations from the learned emission model. In other words it is a decision which introduces quite a bit of overhead, but we make it to probe the conditioning gap on its own, regardless of the practicality of training in this way. Note that it is not uncommon to structure policies in this way, even where models are involved (Lee et al., 2019), though it is not a common choice when training on model-generated rollouts.

3.3 EXPERIMENTS

We experiment with simple settings where the problem of the conditioning gap has different levels of relevance. Our first setup is *Dark Room*, which features a 2D point-agent in a square room with four walls. The agent can measure its distance to each wall up to a certain maximum range. This means that the agent is blind in a square area at the centre of the room. Confoundingly, its task is to stay within a circle that is also placed at the centre of the room. This setup is visualised in fig. 3, where we also compare a policy that was learned by full-conditioning with a policy with partial-conditioning. Only the fully-conditioned model can be use to learn a policy that can solve the task.

Our second test bed is the *Mountain Hike* task, which was introduced by Igl et al. (2018). We use the medium noise setting. The third and final task we consider is a version of the classic pendulum swing-up problem where the mass of the pendulum is picked randomly with uniform probability from $[0.8, 1.2]$ in each rollout. We refer to this task as *Meta Pendulum*.

The simplicity of our environments allows us to do large-scale hyper-parameter searches for each and compare methods based on populations of hyper-parameters. We test 300 hyper-parameter configurations for each method and task, where the only differences be-

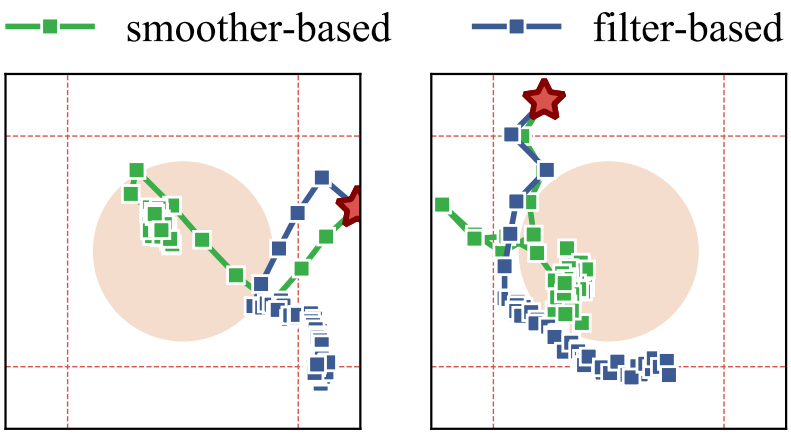

Figure 3: Two rollouts from dark room. Red dashed lines indicate the maximum range of the distance sensor. Pink circle indicates the goal zone. The star shows the initial location of the agent. The smoother model is learned with full conditioning, while the filter-based one uses partial conditioning with an inference network that only looks at past and present observations, as done in most of the literature. The fully conditioned model can be used to learn a policy that manages to reach and remain in the circle, while the partially conditioned model is not good enough to give rise to this behaviour.

This figure has previously appeared in (Kayalibay et al., 2021).

tween the methods are given by the two questions from section 3.2. For each task and method, we pick a number of top-performing hyper-parameters and use these to compare different methods. "Top-performing" means we identify a level of cost that we would define as solving the task and pick all hyper-parameter configurations which are equal to or better than this level. For a fair comparison, we pick the same number of configurations for each method, which amounts to picking the top 50 in Dark Room and Mountain Hike, and the top 70 in Meta Pendulum.

Another benefit of our simple task settings is that we are able to learn near-optimal policies for each task using gradient descent on the true system. We use these optimal policies to calculate the regret of the policies that are obtained from model-based RL. The regret is given by the difference between the total cost of a policy and the total cost of the (near-)optimal one.

The results of our evaluation are shown in fig. 4 and fig. 5. The former shows cumulative regret curves, while the latter shows cumulative distribution functions of the total cost. The biggest difference is in the Dark Room task, where there is a clear gap between fully-conditioned and partially-conditioned methods. In other tasks, the difference can be much more subtle, but fully-conditioned models are never worse than partially-conditioned ones.

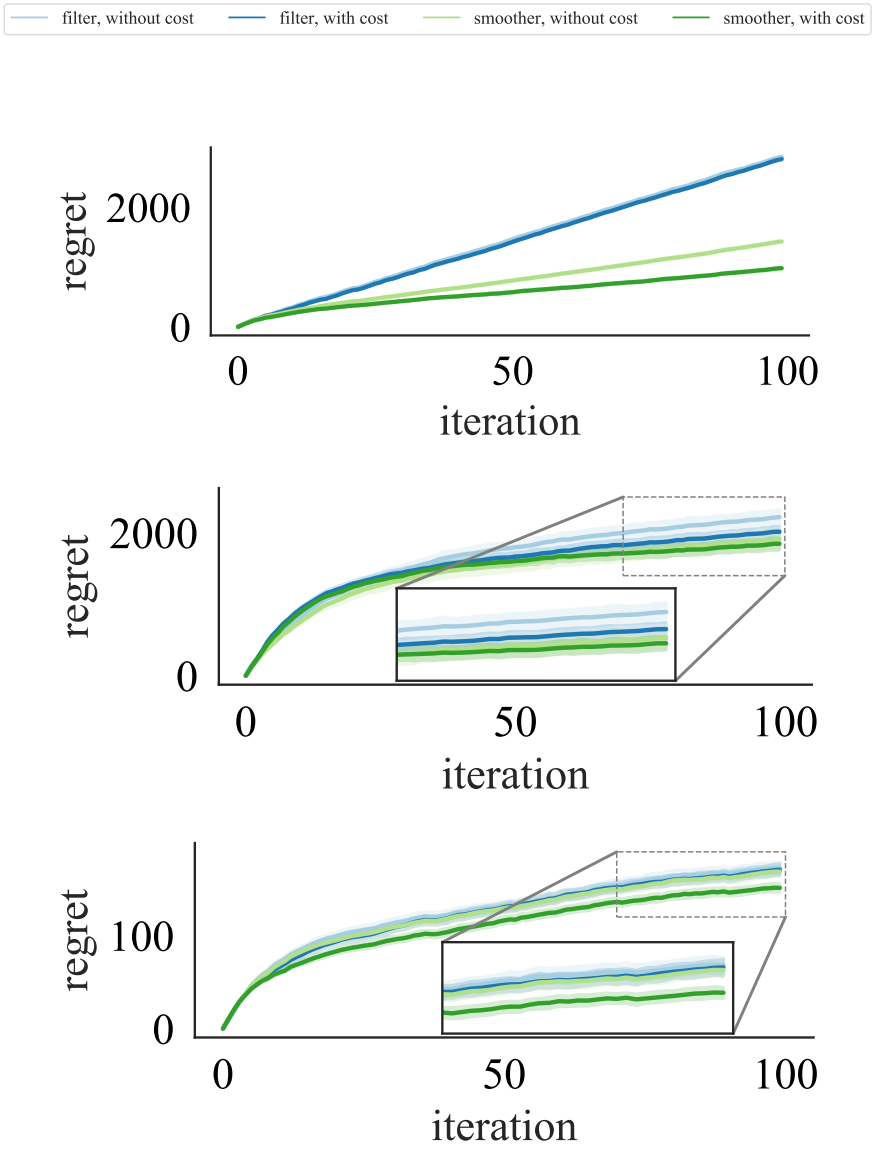

Figure 4: Cumulative regret curves for top-performing hyper-parameters of each method. From the top: Dark Room, Mountain Hike, Meta Pendulum. Filter models only look at $\mathbf{x}_{1:t}$, while smoothers look at $\mathbf{x}_{1:T}$ also. "with cost" vs "without cost" indicates whether the cost is used as an input to the inference network. "filter, without cost" is the standard model that is used in most of the literature. "smoother, with cost" is the only fully conditioned method.
This figure has previously appeared in (Kayalibay et al., 2021).

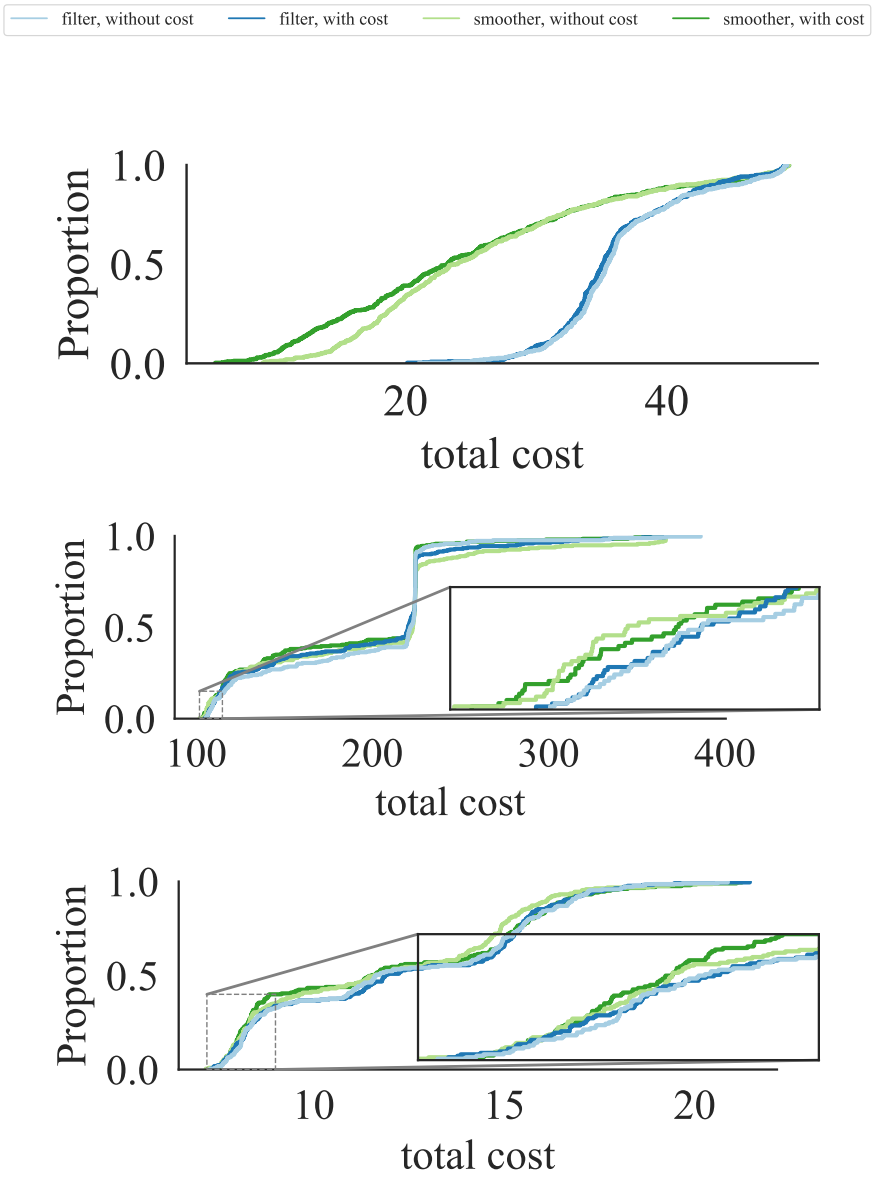

Figure 5: Cumulative distribution functions for the total cost of top-performing hyper-parameters of each method. From the top: Dark Room, Mountain Hike, Meta Pendulum. Filter models only look at $\mathbf{x}_{1:t}$, while smoothers also look at $\mathbf{x}_{1:T}$. "with cost" vs "without cost" indicates whether the cost is used as an input to the inference network. "filter, without cost" is the standard model that is used in most of the literature. "smoother, with cost" is the only fully conditioned method.

This figure has previously appeared in (Kayalibay et al., 2021).

3.4 DISCUSSION

Our findings suggest that model-based control in POMDPs is not entirely well-understood. Many of the strongest methods in the literature contain a pitfall. On the other hand, this does not prevent them from solving difficult tasks. In our own evaluation, we find that the effect of the conditioning gap can be quite negligible for some settings. Looking at the range of tasks used to test state-of-the-art model-based methods for POMDPs, we see that most of the benchmarks focus on problems with weaker forms of partial observability. In these tasks it is often possible to reach near-perfect state information by concatenating observations from a couple of adjacent time steps (Srinivas, Laskin, and Abbeel, 2020).

By contrast, earlier literature on POMDPs features much more interesting problems. As an example, consider the *Heaven and Hell* task by Thrun (1999). In this problem, an 2D agent in a square room must reach either the bottom right or the top left corner of the room, which is decided randomly at the beginning of an episode. The agent does not know which corner is the goal, and it can only observe this information while it is in the top right corner. Thus, to solve this task, the agent first go to the top right corner, and then move to whichever corner is revealed to be goal. Without any form of reward-shaping to encourage the agent to inform itself about the goal zone, Heaven and Hell is a difficult POMDP because it requires a complex form of credit assignment, in which the agent must learn that observing the location of the goal resolves its uncertainty about the cost function, and allows a lower expected total future cost.

Consider solving Heaven and Hell with model-based RL, given access to the true model. This is an easier version of the RL setup used in this chapter, as we already know the true model and do not need to learn a deep variational state-space model. If we try to learn a policy that operates in state-space, as done in many works (Becker-Ehmck et al., 2020; Hafner et al., 2020a), we will not be able to solve the problem. This is because the state space of Heaven and Hell contains the information about where the goal is. Thus, the policy will expect to know the location of the goal, which is normally unknown during execution, until we move to the top right corner. On the other hand, the policy has no reason to learn to do so, because it already has this information during training.

It could be argued that this is always the case with POMDPs, that we are missing some information about the state, which a policy trained in state space would assume to have. The difference between Heaven and Hell and many other POMDPs is that reaching that information takes deliberate, long-term acting on the side of the policy. Compare that to an image-based Mujoco task, where simply observing a couple of successive images already resolves the uncertainty.

We bring up Heaven and Hell because we find it much more intimately linked to the problems we are interested in in this thesis, that is, problems about spatial reasoning. Doing anything in a spatial environment requires a deliberate effort to learn the layout of the

environment. At the same time, spatial reasoning features a twofold state estimation problem, where the agent has to learn its environment while trying to estimate its own location and orientation in that environment.

We believe the conditioning gap and its effect on policy learning represent the tip of the iceberg of potential pitfalls for model-based RL in a spatial setting. That is, until we try to bridge the gap with strong priors. The next chapter will review a series of models for spatial reasoning with increasing levels of inductive biases.

4

MODELS FOR SPATIAL REASONING

The models we have seen so far take a task-agnostic approach to control. Both the model used to describe the world and the policy used to take actions are plain neural networks that make no assumptions about the setting we are working in. Indeed, the only inductive biases in these models are the ones stemming from mathematical formalisms like POMDPs, dynamic programming and SSMs. The models themselves are close to a pure implementation of these mathematical structures, with neural networks to replace building blocks such as transition and emission functions.

This pure approach follows from the design goal of *generality*. These methods reach outstanding performance in a wide range of problems without being tailored to any one setting. What this generality sacrifices is *efficiency*.

The generalist methods of the previous chapter require a long training phase where the behaviour of the agent is arbitrary. In physical systems this means we need to carefully create a safe environment where the agent can learn, and then manually intervene and reset. The process can be time-consuming and risky, as the learning phase can create dangerous situations. At the same time, successful methods for model-based RL on POMDPs typically require an enormous amount of training data. In many tasks we do not have rich datasets and environment interactions can take a long time. Such situations limit the applicability of generalist models.

In this chapter we will turn towards models that eschew generality in seeking efficiency. We will review a series of methods where neural networks are delegated to smaller tasks and feed into classical algorithms which incorporate inductive biases about geometry and planning. Over the past years, these approaches have made steady progress towards a harsh requirement, which lies at the heart of this chapter. Namely that a mobile robot is placed in a previously unseen environment, and has to reach a goal coordinate without any preliminary training phase.

ON NOTATION. We apply the notation from the previous chapter to spatial reasoning. As before, $\mathbf{x}$ denotes an observation, which, in the spatial setting, is typically an RGB-D image or a list of depth readings. The latent state $\mathbf{z}$, which was previously treated abstractly, now corresponds to the agent's pose, comprised of a location and an orientation. Control signals are denoted $\mathbf{u}$ as before.

In addition to these letters, we will sometimes use $\mathbf{m}$, which we refer to as a *chart*. The chart $\mathbf{m}$ captures local properties of an environment, which are sufficient to explain the current observation $\mathbf{x}$. For an RGB-D image, we can imagine the chart as describing only what is within view of the camera. The concept is defined loosely, but we

find it useful as it emerges in different forms in all of the works that are discussed in this chapter.

Finally we will use $\mathcal{M}$ to denote the map, which captures all properties of the environment that are relevant for explaining the observations. Unlike $\mathbf{x}, \mathbf{u}, \mathbf{z}$ and $\mathbf{m}$, which are associated with a time index t (as in $\mathbf{x}_t$), the map $\mathcal{M}$ may be a time-invariant variable, depending on the setting.

4.1 GENERATIVE MODELS

An early contribution that is very much in spirit with the modern works that will be discussed in this chapter is by Murphy (1999). Their Bayesian grid maps work with discrete observations $\mathbf{x} \in \mathcal{X}$, states $\mathbf{z} \in \mathcal{Z}$ and controls $\mathbf{u} \in \mathcal{U}$. The map $\mathcal{M}$ is dynamic (i.e. time-dependent) and consists of a finite set of cells that correspond one-to-one to the state space, which we reflect by associating each grid cell with a state subscript: $\mathbf{m}_\mathbf{z}$ for each $\mathbf{z} \in \mathcal{Z}$. Each grid cell contains an observation variable $\mathbf{m}_\mathbf{z} = \mathbf{x}$. Given a map and a state, the emission model is defined as:

$$\Pr(\mathbf{x} \mid \mathcal{M}, \mathbf{z}) = \begin{cases} \theta & \mathbf{m}_\mathbf{z} = \mathbf{x} \\ 1 - \theta & \text{otherwise} \end{cases}.$$

The prior distribution over the map is a product over the grid cells: $\Pr(\mathcal{M}) = \prod_{\mathbf{z} \in \mathcal{Z}} \Pr(\mathbf{m}_\mathbf{z})$. The dynamics of the state and map over time are modeled with simple transition functions. Inference over the map and the agent state is done via a Rao-Blackwellized Particle Filter.

We omit a more thorough description of the work. The details presented above are interesting in how closely they resemble much later works. In fact, the other works discussed in this chapter can be seen as extensions of Murphy's work to the continuous case.

4.1.1 GENERATIVE TEMPORAL MODELS WITH SPATIAL MEMORY

In Murphy's work the map contents correspond one-to-one to the state space. An extension to continuous spaces requires a new way of associating parts of the map with parts of the state space. The *differentiable neural dictionary*, first proposed by Pritzel et al. (2017) and later applied by Fraccaro et al. (2018) to a spatial environment, provides a simple method. The map is once again a collection of a finite set of cells. Each cell comprises an agent state $\mathbf{z}$ and distribution parameters for a chart $\mathbf{m} \sim \mathcal{N}(\mathbf{m} \mid \mu_\mathbf{z}, \sigma_\mathbf{z})$. The association between a new state $\mathbf{z}'$ and map content is done via nearest neighbours lookup:

$$p(\mathbf{m}' \mid \mathbf{z}') = \sum_{(\mathbf{z}, \mu_\mathbf{z}, \sigma_\mathbf{z}) \in \mathbf{N}(\mathbf{z}')} \omega(\mathbf{z}, \mathbf{z}') \mathcal{N}(\mathbf{m}' \mid \mu_\mathbf{z}, \sigma_\mathbf{z}). \tag{14}$$

Here, $\mathbf{N}(\mathbf{z}')$ is a set containing the k nearest neighbours of $\mathbf{z}'$ in the map and their respective charts and $\omega(\mathbf{z}, \mathbf{z}') \in [0, 1]$ is a mixing coefficient which inversely depends on the distance between $\mathbf{z}$ and $\mathbf{z}'$.

Fraccaro et al. have named their family of models Generative Temporal Models with Spatial Memory (GTM-SM), owing to the indexing of map cells via spatial relationships.

GTM-SM uses a non-parametric map representation that grows over time, which translates itself into a factorisation assumption where each chart $\mathbf{m}_t$ depends on all the charts before it $\mathbf{m}_{<t}$:

$$p(\mathbf{m}_{1:T}, \mathbf{z}_{1:T}, \mathbf{x}_{1:T} \mid \mathbf{u}_{1:T-1}) = p(\mathbf{z}_1)p(\mathbf{m}_1) \prod_{i=2}^{T} p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1})p(\mathbf{m}_t \mid \mathbf{z}_t, \mathbf{x}_{<t}) \prod_{i=1}^{T} p(\mathbf{x}_i \mid \mathbf{m}_i).$$

Note that the chart prior depends on all previous observations: $p(\mathbf{m}_t \mid \mathbf{z}_t, \mathbf{x}_{<t})$. This is because the chart distribution parameters from eq. (14) are predicted by an encoder network looking at the observations. Charts are transformed into observations using a density network:

$$p(\mathbf{x} \mid \mathbf{m}) = \mathcal{N}(\mathbf{x} \mid \mu, \sigma), \text{ with } [\mu, \sigma] = \text{MLP}(\mathbf{m}).$$

The state dynamics $p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t)$ are modeled with parametric models that contain some inductive biases about the problem. Fraccaro et al. (2018) propose different transition functions, for the different settings they work on. As an example, let's consider their dynamics model for a 2D planar robot navigation scenario:

$$p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_{t+1} \mid f(\mathbf{z}_t, \mathbf{u}_t), \sigma_T),$$
$$f(\mathbf{z}_t, \mathbf{u}_t) = \mathbf{z}_t + \mathbf{R}(\mathbf{z}_t)\mathbf{M}\mathbf{u}_t.$$

Here, the state comprises a scalar yaw orientation and xy-coordinates $\mathbf{z} = (\alpha, x, y) \in \mathbb{R}^3$. The control consists of a turning angle and a 2D velocity vector $\mathbf{u} = (\dot{\alpha}, \dot{x}, \dot{y}) \in \mathbb{R}^3$. $\mathbf{R}(\cdot) \in \mathbb{R}^{3\times3}$ is defined as:

$$\mathbf{R}(\mathbf{z} = (\alpha, x, y)) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and $\mathbf{M} \in \mathbb{R}^{3\times3}$ is a learned parameter. These dynamics amount to a unicycle model with a learned control transformation $\mathbf{M}$.

As before with Bayesian grid maps, learning the map requires joint inference over the agent states $\mathbf{z}_{1:T}$ and the charts $\mathbf{m}_{1:T}$ given a stream of observations and control commands $(\mathbf{x}_{1:T}, \mathbf{u}_{1:T-1})$. For that, Fraccaro et al. (2018) use a variational family which factorises as:

$$q(\mathbf{z}_{1:T}, \mathbf{m}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = p(\mathbf{z}_1) \prod_{i=2}^{T} p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}) \prod_{i=1}^{T} q(\mathbf{m}_t \mid \mathbf{x}_t).$$

Note that the variational approximate posterior over the states simply re-uses the transition model. The variational approximation over

the charts $q(\mathbf{m}_t \mid \mathbf{x}_t)$ relies on a recognition network in the style of variational auto-encoders.

Under these assumptions, we obtain a conditional ELBO:

$$\begin{aligned}\log p(\mathbf{z}_{\tau+1:T}, \mathbf{m}_{\tau+1:T} \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T}, \mathbf{z}_1) \\ = \sum_{t=\tau+1}^{T} \mathbb{E}_{q(\mathbf{m}_t \mid \mathbf{x}_t)}\big[\log p(\mathbf{x}_t \mid \mathbf{m}_t)\big] \\ - \mathbb{E}_{p(\mathbf{z}_t)}\big[\mathbf{KL}\big[q(\mathbf{m}_t \mid \mathbf{x}_t) \| p(\mathbf{m}_t \mid \mathbf{z}_{\leqslant t}, \mathbf{x}_{<t})\big]\big],\end{aligned}$$

where $p(\mathbf{z}_t)$ is the marginal distribution over $\mathbf{z}_t$, which marginalises the dynamics noise. All expectations are approximated by ancestral sampling. The chart prior $p(\mathbf{m}_t \mid \mathbf{z}_{\leqslant t}, \mathbf{x}_{<t})$ depends on all previous states and observations and the current state, as it is defined as a mixture over the charts of the k nearest neighbours:

$$p(\mathbf{m}_t \mid \mathbf{z}_{\leqslant t}, \mathbf{x}_{<t}) = \sum_{(\mathbf{z},\mathbf{x}) \in \mathbf{N}(\mathbf{z}_t)} \omega(\mathbf{z}, \mathbf{z}_t) q(\mathbf{m}_t \mid \mathbf{x}).$$

The strength of the prior in modelling the appearance of a scene from a previously unvisited state depends on the accuracy of the state estimates $\mathbf{z}_{1:T}$. This poses a challenge since these estimates entirely depend on the transition dynamics $p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t)$, which contain parameters that also need to be estimated, such as the matrix $\mathbf{M}$ from the previously discussed unicycle dynamics. Fraccaro et al. mitigate this issue by using the true velocity of the agent $\mathbf{z}_{t+1} - \mathbf{z}_t$ as a regression target during model training, allowing any parameters in the transition to receive direct supervision.

It is useful to review the design choices in GTM-SM, which serve to tune the method towards spatial reasoning:

- Spatially organised map.
- Transition models using common planar robot dynamics (i.e. the unicycle model).
- Using supervised learning to anchor parameters in the transition model.

GTM-SM is already a large step away from the task-agnostic approaches in the previous chapter. Still, the method requires a separate training phase, which does not fully align with our central design tenet.

4.1.2 DEEP VARIATIONAL BAYES FILTERS WITH LATENT MAPS

GTM-SM defines a map over a continuous state space using an expandable, non-parametric memory. Deep Variational Bayes Filters with Latent Maps (DVBF-LM) (Mirchev et al., 2018) remain closer to the Bayesian Grid Maps of Murphy et al. and use bilinear interpolation to continuously index the map between the individual grid cells.

Specifically, the map is a 3D tensor $\mathcal{M} \in \mathbb{R}^{w \times l \times d}$, where the first two dimensions correspond to the width and length of a rectangular

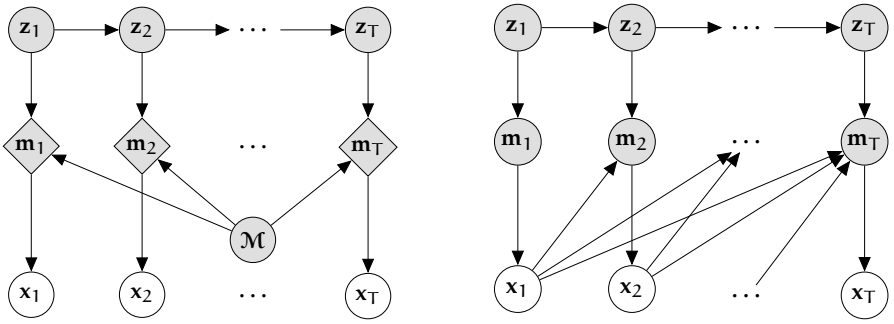

Figure 6: The graphical models of DVBF-LM (left) and GTM-SM (right).

2D area, and the final dimension corresponds to a set of d features. The map spans over a section of 2D space defined by the extremal x- and y-coordinates $x_{min}, y_{min}, x_{max}, y_{max}$. In other words we discretise a certain coordinate range with a grid, and then associate each grid location $\mathbf{z}_{ij}$ with a feature vector $\mathbf{m}_{ij} \in \mathbb{R}^d$. In between the grid cells, the chart for a state $\mathbf{z}$ is obtained by bilinear interpolation:

$$\textbf{attention}(\mathbf{z}) = \sum_{i,j \in \mathbf{N_4}(\mathbf{z})} \omega(\mathbf{z}, \mathbf{z}_{ij}) \mathbf{m}_{ij}.$$

Here, $\mathbf{N_4}(\mathbf{z})$ is the set containing the grid indices for the four grid cells surrounding the state $\mathbf{z}$ and $\omega(\mathbf{z}, \mathbf{z}_{ij})$ returns the bilinear interpolation weight for a neighbour. The grid structure and interpolation simply ignore the orientation component of the state.

The mapping from charts $\mathbf{m}$ to observations $\mathbf{x}$ differs depending on the setting. When observations are depth readings, the emission model is defined as:

$$g(\mathbf{z}, \mathbf{m}) = r(\hat{\mathbf{x}} = \text{MLP}_{\text{emit}}(\mathbf{m}), \mathbf{z}),$$

where the function $r(\hat{\mathbf{x}}, \mathbf{z})$ takes a preliminary depth reading reconstruction $\hat{\mathbf{x}}$ and rotates it depending on the orientation component of $\mathbf{z}$. The goal is to reduce the workload of the neural network MLP_{emit}. The network only has to learn a mapping from charts $\mathbf{m}$ to depth readings given an orientation of $0°$. This reading is then rotated using the actual orientation of the agent to match the correct agent-centric coordinate frame. The rotation operation uses linear interpolation in the angle space.

If the observations are images, the emission model is defined similarly, with slight differences. First, the chart $\mathbf{m}$ is reshaped to match the width and length components of an image: $\hat{\mathbf{m}} \in \mathbb{R}^{n \times m}$. The rotation now happens before the network and also includes a cropping operation which only selects the part of $\hat{\mathbf{m}}$ that corresponds to the current view. Each row of $\hat{\mathbf{m}}$ is rotated and cropped independently, where rotation relies on linear interpolation as before. We can imagine this as carving out one view from a panoramic image.

As for the probabilistic treatment of the model, the prior distribution over the map is a standard Normal distribution $\mathcal{N}(\mathbf{o}, \sigma\mathbf{I})$. Given a state $\mathbf{z}$ and a map $\mathcal{M}$, the respective chart is distributed as a point mass over the bilinear interpolation result: $p(\mathbf{m} \mid \mathbf{z}, \mathcal{M}) \propto \mathbb{I}[\mathbf{m} =$

attention$(\mathbf{z})$]. Finally, the emission resulting from a chart and a state is a Gaussian distribution with homoscedastic noise, i.e. $p(\mathbf{x} \mid \mathbf{z}, \mathbf{m}) = \mathcal{N}(\mathbf{x} \mid g(\mathbf{z}, \mathbf{m}), \sigma_E)$.

Similarly to GTM-SM, the transition model is a learned component, in this case an MLP:

$$p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{z}_{t+1} \mid \mathrm{MLP}_{\mathrm{trans}}(\mathbf{z}_t, \mathbf{u}_t), \sigma_T).$$

This causes the same danger as in GTM-SM, that joint estimation of the map, states and transition parameters might fail. As a remedy, the transition is learned offline before learning the map and estimating the states.

Combining the map, the emission and the transition, the joint probability factorises as:

$$p(\mathbf{z}_{2:T}, \mathbf{x}_{1:T}, \mathcal{M} \mid \mathbf{z}_1, \mathbf{u}_{1:T-1}) = p(\mathcal{M}) \prod_{i=2}^{T} p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}) \prod_{i=1}^{T} p(\mathbf{x}_t \mid \mathbf{z}_t, \mathcal{M}).$$

Note that we assume the initial state $\mathbf{z}_1$ is known, which in most cases is the same as assuming we are only interested in motion relative to the initial state. The charts $\mathbf{m}_{1:T}$ are omitted above as they are determined by the map and the states.

We estimate the map and the states using variational approximate distributions. The approximate posterior for the map is a Gaussian distribution factorising over the map cells:

$$q(\mathcal{M}) = \prod_{ijk} \mathcal{N}(\mathcal{M}_{ijk} \mid \mu_{ijk}, \sigma_{ijk}).$$

The states are estimated via a bootstrap particle filter:

$$\begin{aligned} q(\mathbf{z}_t) &= \sum_k \nu_t^k \mathbb{I}[\mathbf{z}_t = \mathbf{z}_t^k], \\ \nu_t^k &= \frac{\tilde{\nu}_t^k}{\sum_i \tilde{\nu}_t^i}, \\ \tilde{\nu}_t^k &= \nu_{t-1}^k p(\mathbf{x}_t \mid \mathbf{z}_t^k, \mathcal{M}_t). \end{aligned}$$

The factorisation of the full approximate posterior is then:

$$q(\mathbf{z}_{2:T}, \mathcal{M}) = q(\mathcal{M}) \prod_{i=2}^{T} q(\mathbf{z}_t).$$

We estimate the map and learn the emission network $\mathrm{MLP}_{\mathrm{emit}}$ by optimising the ELBO with gradient descent:

$$\mathcal{L}(\mathcal{M}, \mathrm{MLP}_{\mathrm{emit}}) = -\sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{z}_{2:T}, \mathcal{M})} \left[\log p(\mathbf{x}_t \mid \mathbf{z}_{1:T}, \mathcal{M}) \right] + \mathbf{KL}\left[q(\mathcal{M}) \| p(\mathcal{M}) \right].$$

Note that with the off-loading of the transition parameters to a pre-training step, the only learned parameters in the model are the map cells and the emission network, which are both learned by gradient descent, while the states are estimated with particle filtering. These decisions are crucial for meeting the design requirement that the algorithm should be applicable in a previously unseen environment[1].

To review the most important design choices in DVBF-LM:

- Spatially organised map. The map covers an area of 2D space.
- Geometric priors in the emission. Orientations are handled with an engineered solution.
- Transition model is learned beforehand.
- State- and map estimation relies on particle filtering and gradient-based optimisation.
- The emission network and the map are the only learned parameters.

Together, these decisions allow DVBF-LM to do SLAM in a previously unseen environment. Still, DVBF-LM has one key short-coming: its map structure is redundant. Consider DVBF-LM would model an empty room with four walls. The map cells capture panorama-like views from different locations. If the agent stands at one place and does a 360°-rotation, it has seen everything there is to see. Yet, this information is only recorded at the four grid cells surrounding the current location. As soon as the agent moves away from these cells it will have to re-record the same information again. Worse, the agent cannot rely on the old cells for localising itself, as they are now outside the reach of its attention mechanism.

4.2 DETERMINISTIC MODELS

In the following we will review a series of deterministic models oriented at navigation. These models explicitly have different goals from our line of work, as they assume access to a set of training environments and try to maximise performance on another set of held-out scenes. Still, it is interesting to inspect these models for the wide variety of geometric priors and inductive biases they feature.

4.2.1 NEURAL MAP

Proposed by Parisotto and Salakhutdinov (2017), Neural Map (NM) is a model for control in spatial environments. The method features a 2D feature grid $\mathcal{M}_t \in \mathbb{R}^{w \times l \times d}$ which changes over time. Agent states $\mathbf{z}$, which are assumed to be measurable during operation, are associated with grid cells using a mapping function $\psi : \mathbb{R}^2 \to \{0, \ldots, w\} \times$

1 The transition could be modelled in a way that depends on the map, e.g. to model collisions. DVBF-LM uses an obstacle-agnostic transition model and instead relies on the particle filter to accurately estimate the state.

$\{0, \ldots, l\}$. Only the location component of the state informs the mapping. We abbreviate the map cell at the current location as $\mathcal{M}_{\mathbf{z}_t} := \mathcal{M}_t[\psi(\mathbf{z}_t)]$. As the agent moves around in the environment and collects observations, the map is read from and written to using a mix of neural components and engineered attention mechanisms. The agent's observation-planning-acting loop consists of a set of operations.

READ. The map is summarised in to a feature vector $\mathbf{r}_t \in \mathbb{R}^f$ using a convolutional neural network $\text{CNN}_{\text{read}}(\mathcal{M}_t)$.

ATTENTION. An attention mechanism is applied to the map to produce a context vector $\mathbf{c}_t \in \mathbb{R}^d$. Here, the state $\mathbf{z}_t$ and feature vector $\mathbf{r}_t$ are converted into an embedding vector $\mathbf{q}_t \in \mathbb{R}^d$, of which the dot product is taken with every grid cell $\mathcal{M}_t^{ij}$ to produce attention weights ω_{ij}. These weights are normalised and used to produce a convex combination of all of the map cells, which is the context vector $\mathbf{c}_t$. In other terms:

$$\begin{aligned}
\mathbf{q}_t &= \mathbf{W}_{\text{embed}} \begin{bmatrix} \mathbf{z}_t \\ \mathbf{r}_t \end{bmatrix}, \\
\omega_{ij} &= \langle \mathbf{q}_t, \mathcal{M}_t^{ij} \rangle, \\
\tilde{\omega}_{ij} &= \frac{\exp(\omega_{ij})}{\sum_{nm} \exp(\omega_{nm})}, \\
\mathbf{c}_t &= \sum_{ij} \tilde{\omega}_{ij} \mathcal{M}_t^{ij}.
\end{aligned}$$

WRITE. The two vectors $\mathbf{r}_t$ and $\mathbf{c}_t$, along with the current state $\mathbf{z}_t$ and the map contents at the current location $\mathcal{M}[\psi(\mathbf{z}_t)]$ are passed to a neural network, producing a write-vector $\mathbf{w}_t \in \mathbb{R}^d$:

$$\mathbf{w}_t = \text{MLP}_{\text{write}}(\mathbf{z}_t, \mathbf{r}_t, \mathbf{c}_t, \mathcal{M}_{\mathbf{z}_t}).$$

UPDATE. The map cell at the current location $\mathcal{M}_{\mathbf{z}_t}$ is overwritten by the write vector:

$$\mathcal{M}_{\mathbf{z}_t} := \mathcal{M}_t[\psi(\mathbf{z}_t)] \leftarrow \mathbf{w}_t.$$

CONTROL. The three vectors $\mathbf{r}_t, \mathbf{c}_t$ and $\mathbf{w}_t$ are passed to a neural network, which outputs the next control $\mathbf{u}_t = \text{MLP}_{\text{act}}(\mathbf{r}_t, \mathbf{c}_t, \mathbf{w}_t)$.

Neural Map deviates from the previously discussed works in several ways:

- Neural Map is not a generative model, as it does not aim to model the emission or transition dynamics of the system.
- The state $\mathbf{z}$ is assumed to be measurable.
- The model is trained using reinforcement learning on a task cost.

These deviations are emblematic of the models that will be discussed in this section. We present Neural Map and its successors to show the work that has been happening to bridge the gap between the generalist approaches from the previous chapter and the special case of spatial reasoning. Looking at Neural Map from this lens, we find that its main inductive bias is the spatial organisation of the map, which is reminiscent of Murhpy's Bayesian Grid Maps and DVBF-LM's map component.

On the other hand, Neural Map heavily relies on learned components to maintain the map and to decide how the current observation should be associated with its model of the scene. Neural Map shares a similar redundancy to DVBF-LM in that the current observation only affects the map contents at the current location.

4.2.2 COGNITIVE MAPPING AND PLANNING FOR VISUAL NAVIGATION

Gupta et al. (2017) concurrently proposed a model that is similar to Neural Map, which they refer to as Cognitive Mapping and Planning (CMP). The two approaches are similar in spirit, but implemented slightly differently. Gupta et al. use convolutional neural network $\text{CNN}_{\text{map}}(\mathbf{x}_t)$ to predict an agent-frame 2D feature map $\mathbf{m}_t$. This feature map is combined with the previous one $\mathbf{m}_{t-1}$ using a weighted additive update. Feature maps from different time steps are aligned in a common reference frame using the time difference of the agent state, which is assumed to be measurable. There is no global map $\mathcal{M}$, but a series of agent-centric maps $\mathbf{m}_{1:T}$, which inform the agent's decision-making in each time step.

Perhaps the most interesting feature of this work in our narrative so far is the control algorithm. The policy is powered by a hierarchical variant of a Value Iteration Networks (VIN) (Tamar et al., 2016), which approximate value iteration using convolutional neural networks. The local maps $\mathbf{m}_t$ generated by the mapping component are passed to a VIN, which outputs the next control $\mathbf{u}_t$. The mapping component and the VIN are trained jointly using imitation learning.

4.2.3 ACTIVE NEURAL SLAM

Active Neural SLAM (ANS) by Chaplot et al. (2020) takes further steps towards specialisation by replacing the abstract feature representations that have been used in all of the previous work by a more traditional occupancy map. Similarly to the work by Gupta et al., a neural mapper outputs an agent-frame local map $\hat{\mathbf{m}}_t$ based on the current observation. Unlike before, the agent's displacement with respect to the previous step is not assumed to be known. Instead, a rough estimate $\tilde{\mathbf{z}}_t$ is available, which is refined by another neural network that has access to the current observation $\mathbf{x}_t$, preliminary state estimate $\tilde{\mathbf{z}}_t$ and the most recent two agent-frame maps $\hat{\mathbf{m}}_{t-1:t}$. After the current state estimate is refined by the neural net, it can be used to align the

current local map towards a world-frame map $\mathbf{m}_{t-1}$ from the previous time step to produce the new world-frame map $\mathbf{m}_t$. It is worth noting that all map variables are now assumed to encode the occupancy of the scene, i.e. $\hat{\mathbf{m}}_t, \mathbf{m}_t \in \mathbb{R}^{w \times l}$, as opposed to the abstract feature maps of earlier works[2].

The policy is also more structured than before. Chaplot et al. make use of a two-level controller, both implemented with neural networks. The first network takes the current map and observation and outputs a high-level landmark that the agent has to pass through to reach its target. A classical planning algorithm (Sethian, 1996) is used to compute the shortest path from the agent's current state to the landmark. Next, one point on the shortest path is selected as a local target and passed to the second network, along with the current observation, which produces a low-level control $\mathbf{u}_t$.

Both the mapping and the planning components are trained using supervised learning. The former is trained towards occupancy maps obtained using the ground-truth agent states and RGB-D observations from a set of simulated scenes. The latter makes use of optimal plans found by planning with ground-truth agent states and scene geometry from the same setup.

ANS contains perhaps the largest amount of inductive biases among all of the works reviewed in this section. Indeed, we can interpret it as an attempt to replace components of a traditional SLAM-based navigation solution with neural networks and learning these on simulated scenes.

4.3 DISCUSSION

The methods we have reviewed reveal a clear trend towards stronger specialisation in models aimed at spatial reasoning. We see neural networks being relegated to smaller tasks, and made to interface with classical algorithms. Abstract feature representations being replaced with concrete values such as occupancy. Supervised learning used to anchor learned components. It is useful to inspect ideas that show up repeatedly in this body of work.

SPATIALLY ORGANISED MEMORY. Each method has its own version of a map. GTM-SM features a non-parametric memory which is indexed using spatial information. DVBF-LM, NM, CMP and ANS rely on grid maps which correspond to a region of space. Note that these structures are unnecessary from a theoretical perspective, as the information contained in these could be included into the state space. The fact that they show up regardless is a sign that a more specialised representation is helpful.

GEOMETRIC PRIORS. Almost all of the works we have looked at contain some mechanism that explicitly relies on geometric assump-

2 The maps actually contain an additional channel which records whether the agent has visited a location. We omit this part to keep the exposition simple.

tions. DVBF-LM uses estimated agent orientations and field-of-view information to rotate and crop the features stored in its memory. CMP and ANS use agent states to align feature maps from different time steps.

CLASSICAL PLANNING. DVBF-LM proposes to use A*-search for control with its map representation. CMP relies on a neural approximation to value iteration. ANS uses a classical planning algorithm to bridge the gap between a high-level and a low-level controller.

At the same time, there is some disparity between the assumptions and ambitions behind these works. The generative models seek to explain the transition and emission dynamics of the system, while the deterministic ones are more interested in efficient methods for extracting information from observations that is relevant for decision-making. Several of the discussed works assume the agent states are measurable, while others rely on approximate dynamics models.

The role of amortisations is another critical aspect. In general, the deterministic models hinge on the presence of a set of realistic scene scans to train neural components, and seek to maximise performance on a set of held-out scenes. On the other hand DVBF-LM eschews almost all neural amortisations in favour of optimisation and classical filtering. This is in contrast to the generalist approaches of the previous chapter, which seek to solve a specific instance of a problem, and try to do so with as little training time as possible. In terms of a spatial reasoning task such as navigation, solving a specific instance is of limited interest. This would amounts to tuning model parameters for one single environment.

Is it possible to transfer a navigation policy learned in one environment to another? Currently the answer seems unclear, given the amount of variability among spatial environments, sensors and robot dynamics. One line of research (that of the deterministic agents), seeks to train on simulators with large sets of real-world 3D scans. We believe that the possibilities of inductive biases are not yet exhausted. In the following, we will look at a family of emission models which push this frontier further and then demonstrate its application to the navigation task.

5

NAVIGATION WITH SPATIAL WORLD MODELS

This chapter is partly based on the following publications:

Mirchev, Atanas, Baris Kayalibay, Patrick van der Smagt, and Justin Bayer (2021). "Variational State-Space Models for Localisation and Dense 3D Mapping in 6 DoF." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=XAS3uKeFWj.

Kayalibay, Baris, Atanas Mirchev, Patrick van der Smagt, and Justin Bayer (2022). *Tracking and Planning with Spatial World Models*. DOI: 10.48550/ARXIV.2201.10335. URL: https://arxiv.org/abs/2201.10335.

The first paper introduced the world model which is applied in the second to the task of navigation. The former is not a contribution of this thesis, but foundational to our approach. We will review the emission model and map representation of the paper, and omit details about the inference of agent states. The second paper is a joint contribution by the first the author and Atanas Mirchev, proposing a model-based control algorithm for navigation. The state estimation algorithm was conceptualised and implemented by Atanas. We will briefly review it for the sake of narrative cohesion. The author integrated the world model into a controller, adapted the state estimator to the specific case of a planar robot and designed and carried out the experimental evaluation related to navigation. The paper also contains experiments on state estimation accuracy, which were conducted by Atanas. We review the key results from these briefly and refer the interested reader to the original paper.

For the second section, we will briefly review the online SLAM method introduced in:

Mirchev, Atanas, Baris Kayalibay, Ahmed Agha, Patrick van der Smagt, Daniel Cremers, and Justin Bayer (2022). "PRISM: Probabilistic Real-Time Inference in Spatial World Models." In: *6th Annual Conference on Robot Learning*. URL: https://openreview.net/forum?id=X_qYPtJLaX8.

The remainder of the chapter will introduce a previously unpublished control algorithm, which uses this SLAM method, and demonstrate its performance in realistic simulated environments. We will close the chapter with a brief demonstration on real hardware.

5.1 SPATIAL WORLD MODELS

In the previous chapter we explored the efforts of the community in incorporating inductive biases into deep learning models. The generative models we reviewed were mostly content with letting neural networks do a large part of the heavy-lifting. DVBF-LM featured an early attempt at combining our knowledge about geometry and physics into the process We will now review an emission model and map representation proposed by Mirchev et al. (2021) where this idea is taken further.

The core of the idea is to implement ray casting, which is an elementary algorithm for rendering images, in a way that is differentiable with respect to the pose of the camera and a set of parameters which define the geometry and appearance of the environment. This idea has simultaneously appeared in a variety of communities (Bi et al., 2020; Lombardi et al., 2019; Mildenhall et al., 2020; Sitzmann et al., 2020) and the resulting research is often collected under the portmanteau term *differentiable rendering*. The exposition we provide here is based on the approach of Mirchev et al. (2021).

The environment model uses two parametric functions $f_{\mathcal{M}} : \mathbb{R}^3 \to \mathbb{R}$ and $g_{\mathcal{M}} : \mathbb{R}^3 \to \mathbb{R}^3$. These functions model the occupancy and colour over 3D space. They can be implemented using a variety of approaches from voxel-grid interpolation to neural nets to kernel methods. The main concern for the underlying model is differentiability with respect to its parameters and input. Using the functions $f_{\mathcal{M}}$ and $g_{\mathcal{M}}$, we can derive a renderer which processes images pixel by pixel.

Using the camera intrinsic matrix $\mathbf{K}$, each pixel can be mapped to a half-ray described by the function:

$$\mathbf{r}(\delta) = \mathbf{u} + \delta\mathbf{v}, \qquad \mathbf{u}, \mathbf{v} \in \mathbb{R}^3 \text{ and } \delta \in \mathbb{R}_{\geqslant 0}.$$

This ray intersects the geometry of the scene for the first time at the point:

$$\mathbf{r}^\star = \mathbf{r}(\delta^\star) = \mathbf{u} + \delta^\star\mathbf{v}, \tag{15}$$

which satisfies the condition:

$$\forall \delta \in [0, \delta^\star) : f_{\mathcal{M}}(\mathbf{r}(\delta)) \leqslant 0 \text{ and } f_{\mathcal{M}}(\mathbf{r}(\delta^\star)) > 0,$$

where $f_{\mathcal{M}}$ is the occupancy function and we define points $x \in \mathbb{R}^3$ with $f_{\mathcal{M}}(x) \leqslant 0$ as unoccupied or *free space*. The main challenge of differentiable rendering is to find the intersection point $\mathbf{r}^\star$ in a manner that is differentiable with respect to both the map parameters $\mathcal{M}$ and the ray parameters $\mathbf{u}$ and $\mathbf{v}$, which result from the camera pose.

We discretise the ray by sampling a set of equidistant points:

$$\{\mathbf{r}_k \mid \mathbf{r}_k = \mathbf{r}(\delta_k) = \mathbf{r}(k\Delta) \text{ for } k = 1, 2, \ldots, K\} \quad \text{with } \Delta > 0.$$

The parameter Δ controls the fineness of the discretisation. We find the smallest k such that $f_{\mathcal{M}}(\mathbf{r}_k) > 0$. The intersection point must lie on the line segment connecting $\mathbf{r}_{k-1}$ and $\mathbf{r}_k$. Here, we make use of

an approximation and assume that the function $f_{\mathcal{M}}$ is linear on this line segment, which leads to an analytical solution for the intersection point. This amounts to linearly interpolating between $\mathbf{r}_{k-1}$ and $\mathbf{r}_k$ using the function values $f_{\mathcal{M}}(\mathbf{r}_k)$ and $f_{\mathcal{M}}(\mathbf{r}_{k-1})$ to define interpolation weights:

$$\begin{aligned}
\mathbf{r}^\star &= \mathbf{r}(\delta^\star), \\
\delta^\star &= \alpha\delta_k + (1-\alpha)\delta_{k-1}, \\
\alpha &= \frac{f_{\mathcal{M}}(\mathbf{r}_k)}{f_{\mathcal{M}}(\mathbf{r}_k) - f_{\mathcal{M}}(\mathbf{r}_{k-1})}.
\end{aligned}$$

For rendering and RGB-D image, the depth value directly corresponds to $\delta^\star$, while the colour is found by querying the colour model at the intersection: $g_{\mathcal{M}}(\mathbf{r}^\star)$. This rendering algorithm is differentiable with respect to $\mathcal{M}$, $\mathbf{u}$ and $\mathbf{v}$ as long as the functions $f_{\mathcal{M}}$ and $g_{\mathcal{M}}$ are differentiable with respect to $\mathcal{M}$ and $\mathbf{r}$, which is satisfied by all map representations that are interesting to us. If the map $\mathcal{M}$ is a voxel grid containing occupancy and colour information, then $f_{\mathcal{M}}$ and $g_{\mathcal{M}}$ can be implemented via trilinear interpolation, which is differentiable. Likewise, if $\mathcal{M}$ is a neural network, then the output of the network is differentiable with respect to both its inputs and its weights.

Our concept of the chart $\mathbf{m}$ applies somewhat obliquely to this setup. The occupancy values along the ray $f_{\mathcal{M}}(\mathbf{r}_k)$ for each pixel and the colour value at the intersection $g_{\mathcal{M}}(\mathbf{r}^\star)$ can be seen as a local chart, as they are sufficient to explain the current camera observation. More importantly, looking at these values as a chart reveals some of the motivation behind this emission model and map representation.

A major problem with many of the models from the previous chapter was that they record information locally and redundantly. DVBF-LM uses a spatial grid of feature vectors that correspond to panoramic observations (Mirchev et al., 2019). Neural Map likewise only updates the portion of a grid map that corresponds to the agent's current location (Parisotto and Salakhutdinov, 2017). This is problematic since it prevents the agent from extrapolating to areas of the environment which have already been observed but which the agent has not physically stepped on. Of the other models, ANS trains a neural network to produce local occupancy maps via supervised learning (Chaplot et al., 2020) and CMP relies on the task cost to guide a neural network towards producing useful abstract local maps. Both of these approaches require extensive training and are arguably not able to use the observation fully due to the fixed size of the local maps. We can image a setting with a long corridor, where one observation is enough to map the whole environment with a differentiable renderer, where multiple local maps would be necessary to do the same. Finally, GTM-SM uses a non-parametric map and relies on a neural network to extrapolate to new agent states based on a lookup of the nearest neighbours. This requires the network to learn complex geometric transformations, which might be challenging. Another issue is that in an everyday environment, the nearest neighbours in terms of Euclidean distance might lie on the other side of a wall.

Differentiable rendering solves the issues of redundancy and extrapolation by minimising the work that needs to be done by learned components and relying on geometric inductive biases as much as possible. Because the chart $\mathbf{m}$ contains the occupancy values for the entire range of space that can be seen by the camera and the colour values at the surface of the scene, the agent can model an observation from any part of the environment that has been observed previously, even from a view point that has previously not been visited. The one exception to this are occlusions (i.e. we cannot model what is behind a wall until we see it from another view-point).

5.1.1 COMPARISON TO OTHER DIFFERENTIABLE RENDERERS

The rendering algorithm we just reviewed is a cave painting of the physical processes behind a camera's interaction with the world. There are other ways of writing a rendering algorithm that is differentiable. Most notably, Neural Radiance Field (NeRF) (Mildenhall et al., 2020) has become the go-to approach for differentiable rendering over recent years. It is worth noting the differences between the two approaches. First, neural radiance fields also take the viewing direction into account when modelling colour. The input to the colour model $g_{\mathcal{M}}$ contains both the location of a point and the direction it is viewed from. This allows modelling specular effects. Second, NeRFs do not explicitly define an intersection point between the camera rays and the scene. Instead, they integrate the output of the colour model along the ray using a random sample, which allows modelling transparent or semi-transparent objects.

It is clear even at first glance that NeRFs have a much more sophisticated way of rendering. Why then, deviate from this approach? The answer is in the application. While NeRFs target photo-realistic reconstruction, we are more interested in *control*, which, for the task of navigation, requires estimating the state and the scene geometry. In other words our scene representation is geared towards visual tracking and obstacle detection. The latter has no relation to visual details, while the former often suffers from it. Indeed, in many computer vision tasks it is common to smooth images to remove the exact type of high-frequency features that NeRF is trying to capture. We see similar decisions made by others who have attempted visual tracking with differentiable renderers, e.g. Sucar et al. (2021) omit the viewing direction input to a NeRF.

NeRFs also handle depth differently to how we do. They assume the observations are RGB only, without depth information. Integrating the colour over the ray might then be necessary for geometric consistency, since the model does not have access to depth observations for grounding the scene. At the same time, the depth estimates produced by a NeRF are usually not faithful to the true depth. For our purposes, having accurate depth estimates is not optional, as we need to detect the distance between the robot and obstacles in the

scene. As a result, our renderer is designed around the assumption that observations contain depth maps.

A final, subtle difference is the model used to represent $\mathcal{M}$. NeRF, as the name suggests, uses neural networks, while Mirchev et al. (2021) use a voxel grid for the occupancy and a neural network for the colour. In this work, we will use voxel grids for both occupancy and colour. The reason for this is speed, as we will experimentally verify in a later section. Our setting requires real-time state estimation, which is easier to achieve with the speed offered by voxel grids and trilinear interpolation, compared to evaluating a neural network forward pass.

5.1.2 PROBABILISTIC RENDERING

So far we have given a deterministic view of the emission model and map. In practice, we use probabilistic models that result from placing simple Gaussian or Laplace distributions on the map and the emission result. Specifically, we split the map $\mathcal{M}$ into an occupancy map and a colour map: $\mathcal{M} = (\mathcal{M}^{\text{occ}} \in \mathbb{R}^{w \times l \times d}, \mathcal{M}^{\text{col}} \in \mathbb{R}^{w \times l \times d \times 3})$. The prior over the both components is a standard Normal distribution. For approximate inference, we rely on variational distributions which factorise over the map cells and the RGB channels (for colour):

$$q(\mathcal{M}^{\text{occ}}) = \prod_{ijk} \mathcal{N}(\mathcal{M}^{\text{occ}}_{ijk} \mid \mu^{\text{occ}}_{ijk}, \sigma^{\text{occ}}_{ijk}),$$
$$q(\mathcal{M}^{\text{col}}) = \prod_{ijkl} \mathcal{N}(\mathcal{M}^{\text{col}}_{ijkl} \mid \mu^{\text{col}}_{ijkl}, \sigma^{\text{col}}_{ijkl}).$$

The rendering result becomes stochastic by adding Laplace-distributed emission noise:

$$\begin{aligned} p(\mathbf{x} \mid \mathbf{z}, \mathcal{M}) = &\prod_i \text{Laplace}(\mathbf{x}^{\text{depth}}_i \mid \delta^\star_i, \sigma^{\text{depth}}_i) \\ &\prod_i \text{Laplace}(\mathbf{x}^{\text{rgb}}_i \mid g_{\mathcal{M}}(\mathbf{r}^\star), \sigma^{\text{rgb}}_i), \end{aligned} \tag{16}$$

where i is a pixel index, $\delta^\star$ and $g_{\mathcal{M}}(\mathbf{r}^\star)$ are as defined in eq. (15) and $\sigma^{\text{depth}}_i \in \mathbb{R}_+$ and $\sigma^{\text{rgb}}_i \in \mathbb{R}^3_+$ are homoscedastic scale parameters.

If we have a dataset of images and camera poses $\mathcal{D} = \{(\mathbf{z}_i, \mathbf{x}_i)\}_{i=1}^N$ we can learn the map parameters $\mathcal{M}^{\text{occ}}$ and $\mathcal{M}^{\text{col}}$ by optimising the evidence lower-bound:

$$\arg\min_{q(\mathcal{M})} \sum_{\mathbf{z},\mathbf{x} \in \mathcal{D}} \mathbb{E}_{q(\mathcal{M})}\big[-\log p(\mathbf{x} \mid \mathbf{z}, \mathcal{M})\big] + \text{KL}[q(\mathcal{M}) \| p(\mathcal{M})].$$

We use stochastic gradient descent to that end, where stochasticity comes from the usual sources of dataset minibatching and Monte Carlo approximation of the expectation over the likelihood. An additional idiosyncratic source of stochasticity comes from random pixel subsampling. The full image likelihood from eq. (16) is typically too computationally expensive to evaluate at every update step. To speed up training, we select a random subset of pixels instead.

Assuming we have a dataset with pose-labeled RGB-D images is quite a deviation from our original goal of being able to navigate in a fully new environment. For now, we will accept this limitation and present an algorithm for tracking and planning with a map that has been learned offline. In a later chapter, we will explore going beyond this limitation and into the territory of online SLAM and control.

5.1.3 A NOTE ON TRANSITION DYNAMICS

A spatial world model is not the map representation alone. It also contains the transition dynamics, $p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t)$. In the previous chapter we saw different ways of approaching the dynamics. GTM-SM used a combination of engineered models and learned components. DVBF-LM pretrained a neural network using supervised learning. We will take an application-centric approach here. For most mobile robots, the transition dynamics can be explained using simple models such as the unicycle model or the bicycle model. What is missing is the influence of factors such as friction, the internal workings of the motors and control delay. We will therefore model the state transition using an appropriate engineered model, and rely on state estimation to resolve the unknown part.

5.2 TRACKING WITH SPATIAL WORLD MODELS

Several works have approached state estimation or visual tracking with differentiable renderers (Adamkiewicz et al., 2021; Mirchev et al., 2021; Sucar et al., 2021; Wang et al., 2021; Yen-Chen et al., 2020). The agent state $\mathbf{z}$ in this context is the 3D location and orientation of the camera. The typical method is to make use of the differentiability of the system and estimate the pose of the camera by optimising the likelihood of the corresponding image under the current map using gradient descent:

$$\arg\min_{\mathbf{z}} -\log p(\mathbf{x} \mid \mathbf{z}, \mathcal{M}), \tag{17}$$

where the likelihood term is typically approximated by subsampling the pixels. In the specific case of spatial world models, the state $\mathbf{z}$ also contains the location and angular velocity of the camera, and we have access to a transition model $p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1})$ as a prior. We are interested in approximating the posterior after observing the current image $\mathbf{x}_t$, which can be done using gradient descent on the ELBO, with a brief extension of eq. (17) (Mirchev et al., 2021):

$$\arg\min_{q(\mathbf{z}_t)} \mathbb{E}_{q(\mathbf{z}_t)}\big[\log p(\mathbf{x}_t \mid \mathbf{z}_t, \mathcal{M})\big] + \mathrm{KL}\big[q(\mathbf{z}_t) \| p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1})\big].$$

The variational distribution $q(\mathbf{z})$ is a Gaussian distribution. Adamkiewicz et al. (2021) use a similar objective, albeit for maximum a-posteriori estimation.

Whether we have access to a dynamics model or not, it is challenging to achieve real-time state estimation with this method, because differentiating through the renderer is computationally expensive (Mirchev et al., 2021; Yen-Chen et al., 2020). The only methods we are aware of, which support real-time visual tracking by differentiating through the renderer are by Sucar et al. (2021) and Adamkiewicz et al. (2021). The former relies on maintaining a set of keyframes and a hierarchical method for subsampling the image, while the latter uses ORB-features to select interesting pixels when approximating the likelihood.

Here, we will review an alternative approach which relies on approximating the likelihood with a proxy objective. The idea is that if we have a camera pose $\mathbf{z}$ and render the corresponding image $\mathbf{x}$ from our map, then we can use this image to approximate the map $\mathcal{M}$ in the local vicinity of $\mathbf{z}$. We can do this by defining a proxy emission model, which given a new pose $\mathbf{z}'$, can render the corresponding camera view by reprojecting points from $\mathbf{x}$ in to the frame of $\mathbf{z}'$. Given an observation $\mathbf{x}_1$ with an unknown pose $\mathbf{z}_1$, we first pick an anchor pose $\mathbf{z}_2$ and render its view $\mathbf{x}_2$ from the model. We then project points from $\mathbf{x}_1$ into $\mathbf{x}_2$ using the pose offset between the state estimate $\hat{\mathbf{z}}_1$ and the anchor pose $\mathbf{z}_2$. We calculate the reconstruction loss and optimise with respect to $\hat{\mathbf{z}}_1$. The benefit is that we only have to render from the model once, as opposed to optimising through the renderer, where rendering needs to take place in every gradient update. With this approach, every gradient step solely depends on simple geometric operations.

In more concrete terms, we introduce $\mathbf{T}_{\mathbf{z}_1}^{\mathbf{z}_2}(x) = \mathbf{R}_{\mathbf{z}_1}^{\mathbf{z}_2} x + \mathbf{t}_{\mathbf{z}_1}^{\mathbf{z}_2}$, which transforms a point $x \in \mathbb{R}^3$ from the frame of $\mathbf{z}_1$ into the frame of $\mathbf{z}_2$. Here, x is a 3D point obtained by using the depth map $\mathbf{x}^{\text{depth}}$ and the camera intrinsic matrix $\mathbf{K}$. We also define $\pi_{\mathbf{z}}(x)$ as the projection of the 3D point x into the image plane from the view $\mathbf{z}$. Finally, we introduce $\mathbf{n}$, which is a surface normal map that is obtained using the image gradient of $\mathbf{x}^{\text{depth}}$. Using these new constructs, the proxy likelihood term is defined as:

$$
\begin{aligned}
-\log \tilde{p}(\mathbf{x} \mid \mathbf{z}, \mathcal{M}) = \sum_i \left\| \tilde{\mathbf{x}}^{\text{rgb}} \left[\pi(\mathbf{T}_{\mathbf{z}_1}^{\mathbf{z}_2}(x_i)) \right] - \mathbf{x}^{\text{rgb}}[\pi(x_i)] \right\|_1 + \\
\sum_i \left\| \langle \tilde{x}_i - \mathbf{T}_{\mathbf{z}_1}^{\mathbf{z}_2}(x_i), \tilde{\mathbf{n}}_i \rangle \right\|
\end{aligned} \tag{18}
$$

Here, x_i is the 3D point obtained by projecting pixel i of the observation into the world frame. The point $\tilde{x}_i$ is the 3D point from the anchor image, which corresponds to x_i after projecting it into the anchor frame. Finally, $\tilde{\mathbf{n}}_i$ is the surface normal corresponding to $\tilde{x}_i$.

Another much more colloquial interpretation of this proxy objective is that we are relying on point-to-plane Iterative Closest Point (ICP) with photometric constraints (Audras et al., 2011; Chen and Medioni, 1992; Steinbrücker, Sturm, and Cremers, 2011) to align the current observation to an anchor state. Note that the presence of the map gives us a great deal of freedom in picking the anchor state. Because we have a model that can predict the RGB-D observation given

a pose, we can pick the anchor pose freely. It would even be possible to pick multiple anchors and select the one with the lowest loss after optimising eq. (18). In practice, we will simply use the previous estimate as an anchor.

We use the transition model to initialise the guess over the current state and then optimise eq. (18) and the transition prior jointly:

$$\arg\min_{\mathbf{z}_t} -\log \tilde{p}(\mathbf{x}_t \mid \mathbf{z}_t, \mathfrak{M}) - \log p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_t). \tag{19}$$

5.3 PLANNING WITH SPATIAL WORLD MODELS

In previous chapters we saw various strategies of controlling a mobile robot. DVBF-LM assumes deterministic dynamics, which allows planning with A*-search and then executing the plan exactly. CMP relied on value iteration networks as a parametric policy that includes some inductive biases. ANS used a more structured two-layer controller with learned components. The presence of parametric models in a controller requires a training phase. Because the structure or appearance of the environment is an input to the policy, the training phase also needs to include a variety of environments to be able to generalise to a new environment. We therefore omit learned components from the policy and limit ourselves to classical planning techniques.

Formally, our goal is to reach a goal coordinate $\mathbf{g} \in \mathbb{R}^2$ starting from an initial state $\mathbf{z} \in \mathbb{R}^3$, where the state includes a 2D location and a yaw angle. In other words we are working with a planar robot. For this section we will further assume that the robot can turn around freely in-space and always moves forward along its heading direction. These assumptions are fitting for a TurtleBot [1], though they exclude other systems like cars.

We will use a two-phase strategy. First, we run A*-search to plan a trajectory of landmarks that reach the goal $\mathbf{g}$. In this stage, we discretise the environment with a uniform grid and assume each cell is connected to its eight neighbours, as long as these are not occupied and maintain a minimal safety distance of Δ_{safe} to any occupied location. Occupancy and the distance to the closest occupied location are straightforward to query from our model by simulating a lidar scan around the location.

Once a list of landmarks is found, a simple low-level controller takes over, following the planned trajectory. At any point in time, the low-level controller first find the closest landmark on the plan. Any landmarks that precede the closest landmark are removed. Next, the current heading angle is corrected to have the agent facing the landmark. Since the robot has some limits in how much it can turn in one time step, this process might take several time steps. The robot does not make any forward movement until the difference between the current heading angle and the direction leading directly to the landmark is below a threshold Δ_{face}. Once the movement angle is close enough to the direct line between the agent and the landmark,

1 https://www.turtlebot.com/

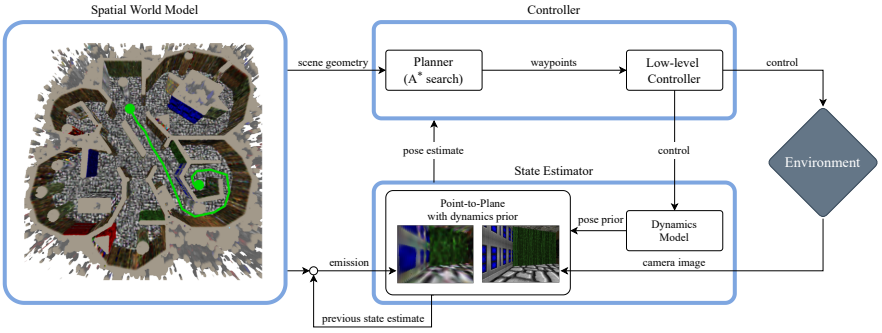

Figure 7: Overview of the method.
This figure has previously appeared in (Kayalibay et al., 2022).

the agent takes a step forward to the landmark with the largest step size available to it. As soon as the agent is closer to the landmark than a threshold Δ_{remove}, the landmark is removed and a new one is selected. The final landmark is the goal itself, where we use a more strict threshold $\Delta_{\text{goal}} < \Delta_{\text{remove}}$ to decide if the agent has reached the goal or not, at which time the navigation task is stopped.

An overview of the method is given in fig. 7. The learned model provides information about obstacles, which is used by the high-level controller. Both high- and low-level controllers rely on state estimates. These result from the dynamics and emission models and the stream of camera images.

5.4 EXPERIMENTS ON VIZDOOM

We use the ViZDoom simulator to develop a test environment (Wydmuch, Kempka, and Jaśkowski, 2018). While ViZDoom lacks visual fidelity, it allows us to easily generate levels for a set of realistic floor plans with obstacles. We convert six floor plans from the HouseExpo dataset (Li et al., 2020) into ViZDoom levels. We use a separate level for fine-tuning the algorithm. This level is not part of the HouseExpo dataset and was taken from a paper by Savinov, Dosovitskiy, and Koltun (2018). The levels used for evaluation and fine-tuning are shown in fig. 8.

To approximate a realistic scenario where the robot dynamics are known up to stochastic factors, we add noise to the motion of the robot. For a ground-based robot, simple Gaussian noise is not realistic. A control command that should move the robot forward would realistically not result in the robot moving backward. Likewise, if we ask the robot to turn left, we can expect it to turn left by some amount or stand still, but not to turn right. We therefore design a noise model which executes an action with added noise, but in a way that does not change the action's direction. Specifically, if we decompose the control into an angular velocity and a forward velocity as $\mathbf{u} = (s, v)$, the noise model behaves as:

$$\hat{s} = s + \text{clip}(\epsilon_s, -s, \Delta_s), \epsilon_s \sim \mathcal{N}(s \mid 0, \sigma_s),$$
$$\hat{v} = v + \text{clip}(\epsilon_v, -v, \Delta_v), \epsilon_v \sim \mathcal{N}(s \mid 0, \sigma_v).$$

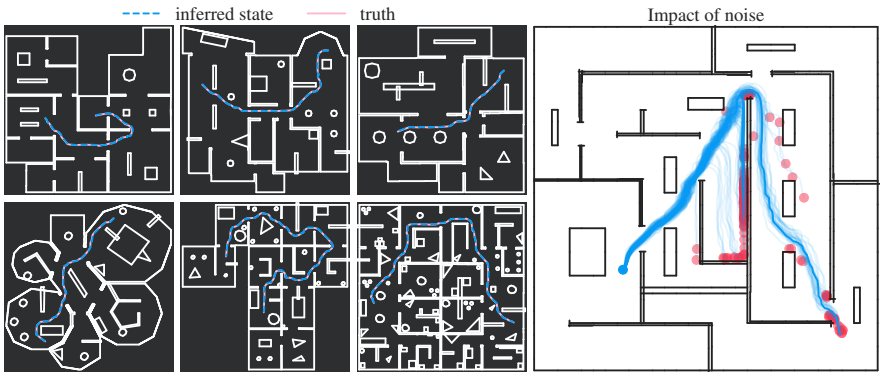

Figure 8: Evaluation (left) and fine-tuning (right) levels. The right panel demonstrates the noise model. The same plan is executed from the same starting point multiple times without re-planning. The final position of the robot is shown in red. Noisy dynamics lead to vastly different outcomes.
This figure has previously appeared in (Kayalibay et al., 2022).

Here, Δ_s and Δ_v are limits for the maximum allowed perturbation, while σ_s and σ_v control the level of noise. The noise model is demonstrated in fig. 8 (right). Here, we take an initial state and a plan which leads to the goal assuming deterministic dynamics. We then evaluate this plan in open-loop fashion multiple times under stochastic dynamics. The final location of the robot in each rollout is marked red. We find that stochastic dynamics can lead to vastly different outcomes, making state estimation a necessity.

5.4.1 TESTING STATE ESTIMATION SUCCESS

In our first experiment, we focus on the accuracy of our revised state estimation objective[2]. Figure 9 shows empirical cumulative distribution functions (CDFs) of errors made by gradient-based optimisation of the observation likelihood (referred to as *emission* in the figure) vs our method (*pred-to-obs* in the figure). We find that for both location (left) and orientation (middle) estimation the two methods are virtually identical. On the other hand, if we inspect the runtimes, as depicted in the right panel, the proposed proxy objective is over 4 times faster than tracking based on differentiation of the observation likelihood.

At the same time, we see that the choice of a voxel-based emission model is important for our method, as it allows fast rendering of a full observation. NeRF, which relies on neural networks, is prohibitively expensive here. It should be noted that there have been many recent advances in speeding up NeRFs (Cao et al., 2022; Garbin et al., 2021; Li et al., 2022; Lombardi et al., 2021; Reiser et al., 2021; Wang et al., 2022; Yu et al., 2021). It is possible for a revised version of NeRF to leverage our state estimation approach.

2 For a more complete overview of this experiment, we refer the reader to (Kayalibay et al., 2022) and would re-emphasise that both the state estimation algorithm and the experiments on state estimation accuracy are contributions of Atanas Mirchev.

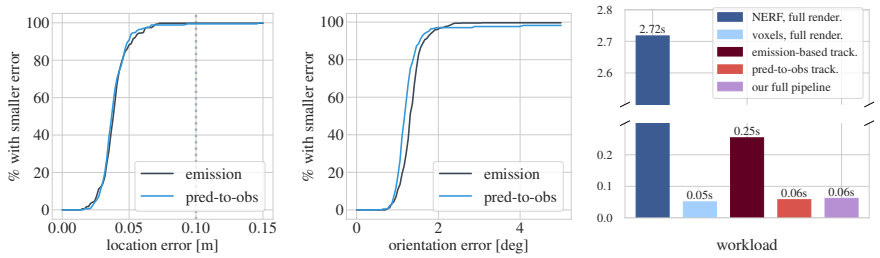

Figure 9: Cumulative distribution functions (CDFs) of location (left) and orientation (middle) errors for tracking with gradient-based optimisation of the observation likelihood (emission in the figure) vs our proxy objective (pred-to-obs in the figure). The dashed line shows half of the agent's size. The two methods have almost identical behaviour, but using our proxy objective is over 4 times faster as illustrated in the runtime breakdown (right). Using voxel-based emission models allows tractable rendering times, which is essential for our method. The rendering times of NeRF are prohibitively high.

This figure has previously appeared in (Kayalibay et al., 2022).

5.4.2 TESTING NAVIGATION SUCCESS

For checking navigation success, we follow the guidelines of Anderson et al. (2018). The agent is assumed to have a body length of 0.2m. Following that scale, the evaluation levels from fig. 8 all fit inside a $11.4 \times 11.4\text{m}^2$ area. We consider a navigation task as successful if the agent's location is within a radius of Δ_{reach} of the goal and if the agent's state estimate is within a radius of Δ_{est} of the goal. The latter is the agent's internal condition for having reached the goal, which is more stringent because we expect the state estimate to contain some error.

We sample 200 random navigation tasks. For each task, we sample a starting state and a goal. These should maintain a minimal distance of Δ_{safe} to any obstacle and they also have to be at a distance of Δ_{start} from each other to disallow trivial tasks. We use Success Weighted by Inverse Path Length (SPL) as our evaluation metric, which calculates a soft success rate, where every successful attempt is weighted by how much longer the agent's route was compared to the shortest path. In other terms, the SPL for a set of navigation tasks $\tau_{1:M}$ is:

$$\text{SPL}(\tau_{1:M}) = \sum_i \mathcal{S}(\tau_i) \frac{\text{shortest}(\tau_i)}{\text{length}(\tau_i)}.$$

Here, $\mathcal{S}(\tau_i)$ is a binary variable indicating the success of the task and $\text{shortest}(\tau_i)$ and $\text{length}(\tau_i)$ are the length of the shortest path for the task and the length of the path that the agent took.

We use three different levels of noise for the evaluation. Denoting the length of the agent's body with Δ_{body}, these are:

- **low noise.** $\sigma_s = 3°$ and $\sigma_v = 0.1\Delta_{\text{body}}$
- **medium noise.** $\sigma_s = 6°$ and $\sigma_v = 0.15\Delta_{\text{body}}$

noise	ours	no map	dynamics
high	**0.46**	0.37	0.33
medium	**0.79**	0.51	0.52
low	**0.92**	0.61	0.61

Figure 10: SPL scores obtained by our method vs tracking without a map vs tracking with the dynamics model alone.

- **high noise.** $\sigma_s = 9°$ and $\sigma_v = 0.3\Delta_{\text{body}}$

Under these conditions, we compare our method to two alternatives. The first is a variant of our method where we use the previous observation for tracking instead of rendering from the map (which we call *no map*). The second only uses the dynamics model for tracking. We call this variant *dynamics*. The results of the comparison can be found in fig. 10.

5.5 EXTENDING NAVIGATION WITH SPATIAL WORLD MODELS

The control algorithm we have proposed has two conceptual issues. The first is that the map needs to be learned offline. The second is its opaque two-stage controller.

To address the first issue, we will make use of recent work by Mirchev et al. (2022), where some ideas from earlier work (Kayalibay et al., 2022; Mirchev et al., 2021) were combined and extended to arrive at a real-time SLAM solution. The main challenge behind online map learning and tracking is that learning the map via gradient descent is time-consuming. The only method capable of SLAM with a differentiable renderer is by Sucar et al. (2021), which relies on careful sampling of pixels for both tracking and mapping, as well as a mechanism for maintaining a set of keyframes. The keyframes serve a similar purpose as experience replay in a continual learning setting (Rolnick et al., 2018).

Mirchev et al. (2022) approach the issue from a perspective of online Bayesian inference and derive closed-form updates for the map, which boil down to traditional Signed Distance Function (SDF) updates (Curless and Levoy, 1996) and borrow ideas from approximate inference on occupancy grids (Grisetti, Stachniss, and Burgard, 2005). At the same time, they extend the tracking approach from section 5.2 into a fully-probabilistic filter. We will provide a pragmatic overview of this method.

5.5.1 OVERVIEW OF THE FILTER

We use the PRISM algorithm proposed by Mirchev et al. (2022) for real-time inference of the agent state and the map. The PRISM algorithm extends the tracking algorithm from section 5.2 in two ways.

First, the posterior distribution over the agent pose is approximated using Laplace approximation of the proxy objective from eq. (18). In other words, PRISM provides a full approximate posterior distribution, while the approach from section 5.2 only provides a maximum a-posteriori estimate. PRISM also provides an approximate posterior distribution over the agent's velocity by linearising the transition function. Second, and more importantly, PRISM introduces closed-form updates for the map given a pose estimate and its respective observation. These allow for real-time inference of the map variable, which is crucial for real-time navigation in an unknown environment.

To explain how PRISM works in brief terms, let us revisit section 5.2, where we introduced eq. (19), which is an optimisation problem that provides a maximum a-posteriori estimate over the current agent location and 6-DoF orientation. From this point on PRISM, does the following steps:

- The maximum a-posterior estimate, which we denote $\mathbf{z}^\star$ is expanded into a filtering posterior $q(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{z}^\star, \Sigma_{\mathbf{z}}^\star)$ through Laplace approximation using the optimisation objective from eq. (19).
- In addition to the location and orientation, a 6-DoF agent velocity is estimated by linearising the transition distribution and solving for the mean and covariance of the velocity for the current time step using the previous pose and velocity distribution and the current pose distribution, based on a joint Gaussian assumption, which results from linearising the transition.

The current pose distribution is used to update the map. Here, instead of using gradient descent, PRISM uses closed-form updates that are similar to SDF updates (Curless and Levoy, 1996). More specifically, the map updates serve to approximate the posterior over the map given all previous observations $q_t(\mathcal{M}) \approx p(\mathcal{M} \mid \mathbf{x}_{1:t})$, which is approximated through a recursive Bayesian update:

$$q_t(\mathcal{M}) = p(\mathbf{x}_t \mid \mathbf{z}_t, \mathcal{M}) q_{t-1}(\mathcal{M}),$$

where $q_{t-1}(\mathcal{M})$ is the distribution over the map from the previous time step. The observation likelihood term is further approximated through a proxy likelihood:

$$q_t(\mathcal{M}) \approx q(\mathcal{M} \mid \mathbf{z}_t, \mathbf{x}_t) q_{t-1}(\mathcal{M}).$$

The proxy likelihood is a given by a Gaussian density over the map, such that the product with the previous map results in a new Gaussian density, whose mean and covariance can be found in closed form, which turns out to be identical to an SDF update. A more detailed discussion can be found in the original paper by Mirchev et al. (2022).

5.5.2 NAVIGATION WITH MODEL-PREDICTIVE CONTROL

The approach reviewed in section 5.3 is quite common in map-based navigation. A high-level planner produces a trajectory and a low-level

controller follows it. The two components have fundamentally different jobs and can be seen as black boxes which expect a certain input and promise a certain output. When reviewing the methods presented in this chapter, a common trend is that we seek to replace hierarchical algorithms consisting of isolated components with unified methods that optimise a common objective.

Note that the two-stage approach is difficult to modify or extend, because it is unclear at which stage we would need to intervene to change the system's behaviour. For instance, we might want to introduce constraints into the system which relate to both the agent's orientation and location. It is not possible to add these in the A*-planner since it only uses location information. On the other hand, only including the constraints inside the low-level controller might not be enough, since the planner trajectory might require violating constraints, which will require some mechanism for detecting that a plan cannot be executed without violating constraints and re-planning accordingly.

Fortunately, there is a family of controllers which rely on solving a single objective and can handle constraints. In this subsection we will derive a model-predictive control algorithm for navigation. The main idea is that we can distill the trajectory planning done by A*-search into a terminal cost function $J(\mathbf{z})$ and then rely on optimisation to traverse the resulting cost surface.

Our approach boils down to a constrained MPC objective of the form:

$$\arg\min_{\mathbf{u}_{1:T-1}} \mathbb{E}_{p(\mathbf{z}_{2:T}|\mathbf{z}_1,\mathbf{u}_{1:T-1})}\Big[\beta^{T-1}\hat{J}(\mathbf{z}_T) + \sum_{t=1}^{T-1}\beta^{t-1}\mathbf{c}_t\Big], \quad \text{s.t. } h_j(\mathbf{z}_t) \leqslant 0. \tag{20}$$

Here, $\{h_j(\mathbf{z}_t)\}_{j=1}^{M}$ are state-constraints and the expectation marginalises transition stochasticity. Note that the constraints are defined on random variables since the future state is stochastic.

5.5.2.1 Finding the Terminal Cost

The most interesting part of eq. (20) is the terminal cost $J(\mathbf{z})$. There are a variety of approaches for learning or engineering terminal cost functions. A popular method is to train a neural network, often called a critic, which approximates the cost-to-go. Since we are interested real-time control we will instead rely on dynamic programming. Finding the cost-to-go of a stochastic shortest paths problem in continuous space via dynamic programming is not possible. We will take a work-around by approximating the true problem with a proxy system, a method that is known as aggregation (Bertsekas, 2005).

Formally, we define aggregated state and control spaces $\mathcal{Z}$ and $\mathcal{U}$. The aggregated state space $\mathcal{Z}$ is related to the original state space of yaw angles and xy-coordinates $\mathbb{R}^3$ via the aggregation and disaggregation probabilities. We can take uniform discretisation as an example. Here, we take a subspace of $\mathbb{R}^3$ which correspond to our

working space and discretise it along each dimension with uniform grids. The probability of a true state $\mathbf{z}$ being aggregated into an aggregate state $\hat{\mathbf{z}}$ is then 1 if $\mathbf{z}$ falls into the bin corresponding to $\hat{\mathbf{z}}$ and zero otherwise. The probability of an aggregate state $\hat{\mathbf{z}}$ corresponding to a true state $\mathbf{z}$ is similarly 1 if $\mathbf{z}$ is the centre of the subspace or bin given by $\hat{\mathbf{z}}$ and zero otherwise. This type of aggregation is known as hard aggregation due to the binary aggregation and disaggregation probabilities. We will introduce two functions $\mathbf{A}$ and $\mathbf{D}$ to denote the process of aggregation and disaggregation. $\mathbf{A}(\mathbf{z})$ returns the aggregate state $\hat{\mathbf{z}}$ which contains $\mathbf{z}$, while $\mathbf{D}(\hat{\mathbf{z}})$ returns the centre of the bin corresponding to $\hat{\mathbf{z}}$.

In simple systems, we can get away with cruder approximations. We will consider both problems that can be explained by a unicycle model, which allows rotating in place, and those involving a bicycle model, where turning in place is not possible. For the former, we will use a simpler aggregation scheme which only discretises the location component of the state, while completely ignoring the orientation. For the latter, we will discretise all state components to capture motion constraints more accurately.

The transition and cost functions in aggregate space can be defined in different ways, depending on the type of aggregation used. In any case, the transition probabilities depend on which parts of the state space are free or occupied. We extract this information from the map $\mathcal{M}$, which is explained in the next section. The cost function of stochastic shortest path problem is a sparse indicator function, which is zero only when within a certain radius of the target and $c > 0$ elsewhere. We will generally use the same for the aggregate space, though any additional cost terms can be included. These might be motivated by favouring risk-avoidant behaviour like maximising the distance to obstacles, that goes beyond simply avoiding collisions, which is typically expressed as a constraint.

Given an aggregate problem, we find the cost-to-go over aggregate space using dynamic programming. While this procedure requires more computation than A*-search, it is highly parallelisable. In our implementation we will enable real-time control by a parallelised GPU implementation of value iteration. The cost-to-go over the aggregate problem is either a 2D or a 3D grid of scalar values. We read from this grid using bilinear or trilinear interpolation to define a continuous terminal cost $J(\mathbf{z})$.

5.5.2.2 Constraints and Obstacle Avoidance

The constraints in eq. (20) serve the purpose of obstacle avoidance. Given a state $\mathbf{z}$, we define the constraint as a function of the distance to the closest obstacle, and a safety threshold $\mathtt{tolconst}$:

$$h(\mathbf{z}) = \Delta_{\text{const}} - \delta(\mathbf{z}),$$

where $\delta(\mathbf{z})$ is the distance to the closest obstacle. We can efficiently obtain this quantity for a grid of locations using dynamic programming.

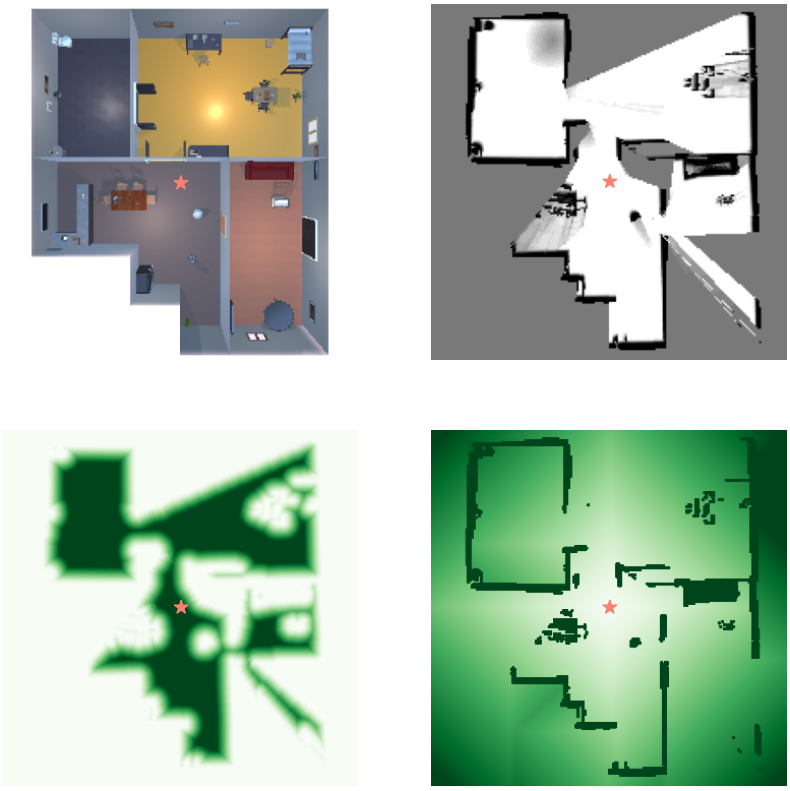

Figure 11: (Top left) Top-down view of an environment. (Top right) Estimated occupancy based on a partial exploration. Unseen locations have an occupancy probability > 0 (shown in grey). (Bottom left) Obstacle constraints. (Bottom right) Cost-to-go resulting from aggregation.

During planning, we read from the grid using bilinear interpolation, analogously to how we read from the cost-to-go grid.

Because the future state $\mathbf{Z}_t$ is a random variable due to the transition noise, the constraints need to handle stochasticity. We implement this in different ways depending on the optimisation algorithm used to solve eq. (20). We give more details on this in section 5.5.2.4.

5.5.2.3 Modelling the Transition

We model the transition dynamics using engineered models, since there are well-understood models for the systems we work on. These are the unicycle model and the kinematic bicycle model. In our simulated experiments, we use the same noise model as in section 5.4. In our experiments on real hardware, there are a series of factors that the bicycle model is agnostic to. Thus, the planning phase will contain some amount of error in either case. We rely on the iterative re-planning mechanism of MPC to account for these errors and on the state estimator to keep track of the agent state.

5.5.2.4 Practical Model-Predictive Control

Many optimisation algorithms can be used to solve eq. (20). For differentiable systems such as ours, gradient-based optimisation is par-

ticularly attractive, since it can handle constraints via the augmented Lagrangian method. At the same time, relying on gradient-based optimisation alone can be inefficient when aiming for real-time control. We take a pragmatic approach and base our planner around a constrained version of random search. The benefit of random search is that it can be parallelised, which is extremely efficient on modern GPUs.

For constrained random search, we first sample a set of candidate plans $\mathcal{C} = \{\mathbf{u}^i_{1:T-1}\}_{i=1}^{C}$. For each candidate plan, we take a Monte Carlo estimate of the expectation in eq. (20):

$$\tilde{\mathcal{L}}_{\text{mpc}}(\mathbf{u}^i_{1:T-1}, \mathbf{z}_1) = \sum_{k=1}^{K} \left[\beta^{T-1} \hat{J}(\mathbf{z}^{ki}_T) + \sum_{t=1}^{T-1} \beta^{t-1} \mathbf{c}^{ki}_t \right].$$

That is, for the same plan, we take K samples from the noisy transition dynamics and average over their total cost.

For the constraints, we take a pessimistic estimate:

$$\tilde{h}(\mathbf{z}^i_t) = \max_{k=1,\ldots,K} h(\mathbf{z}^{ik}_t).$$

Under this definition, the set of valid candidates are:

$$\mathcal{C}_{\text{valid}} = \{\mathbf{u}^i_{1:T-1} \mid \tilde{h}(\mathbf{z}^i_t) \leqslant 0 \text{ with } t = 2, \ldots, T \text{ and } i = 1, \ldots, C\}.$$

The best valid plan is then:

$$\mathbf{u}^\star_{1:T-1} = \arg \min_{\mathbf{u}_{1:T-1} \in \mathcal{C}_{\text{valid}}} \tilde{\mathcal{L}}_{\text{mpc}}(\mathbf{u}_{1:T-1}, \mathbf{z}_1).$$

Meanwhile, the safest plan is:

$$\mathbf{u}^{\text{safe}}_{1:T-1} = \arg \min_{\mathbf{u}_{1:T-1} \in \mathcal{C}} \sum_{t=2}^{T} \tilde{h}(\mathbf{z}_t).$$

Finally, the constrained random search planner returns:

$$\text{CRS}(\mathbf{u}^i_{1:T-1}, \mathbf{z}_1) = \begin{cases} \mathbf{u}^\star_{1:T-1} & \text{if } \mathcal{C}_{\text{valid}} \neq \emptyset \\ \mathbf{u}^{\text{safe}}_{1:T-1} & \text{otherwise.} \end{cases}$$

In other words, whenever a safe plan is found, we take the one with the minimum total cost. When no safe plans are available, we take the plan with the minimum constraints violation.

On robots with simple dynamics such as a unicycle model, we find random search alone to be sufficient for control. For more complex dynamics, we further refine the random search result with gradient-based optimisation, which can be done fast since the initial plan is already a good starting point.

As an illustration of constrained random search, we show results from a toy system in fig. 12. Here, the agent starts in the top left corner and has to navigate to the bottom right corner. The cost is the Euclidean distance to the goal. The agent's movement is perturbed by zero-centred Gaussian noise, and it has to avoid a three-by-three grid of circular obstacles.

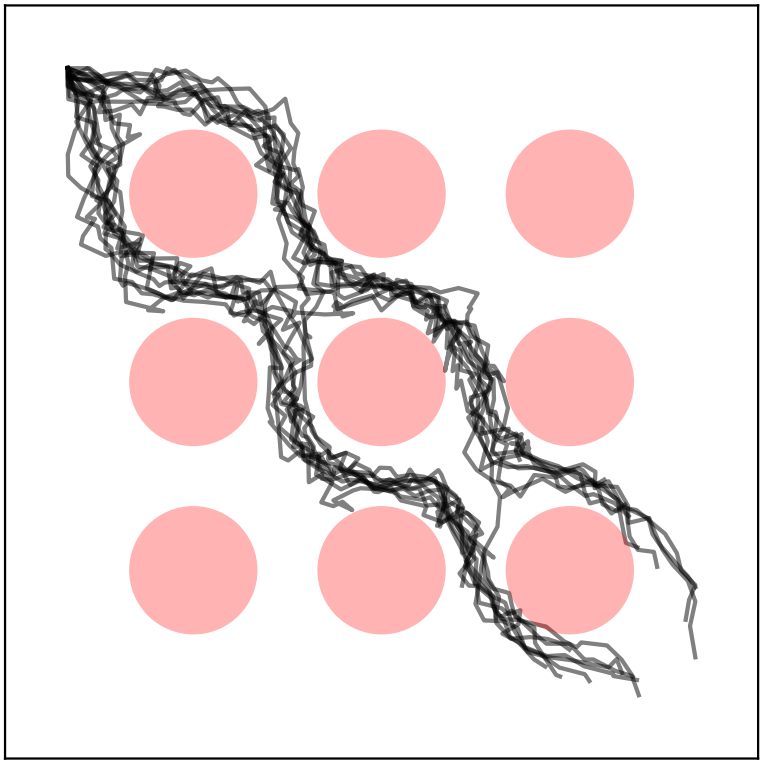

Figure 12: Multiple rollouts from a model-predictive controller using constrained random search. The agent must navigate from the top left corner to the bottom right, while avoiding a field of obstacles. Controls are 2D velocities, subject to zero-centred Gaussian noise.

Figure 13: Overview of the small evaluation environments.

5.6 EXPERIMENTS ON PROCTHOR

In our previous experiment setup, we relied on ViZDoom, because it allowed us to easily construct a set of realistically-proportioned environments based on floor plans from an open source dataset. This time, we will use the recently published ProcTHOR simulator (Deitke et al., 2022). ProcTHOR features a much higher visual fidelity than ViZDoom, while providing a set of algorithmically-generated living spaces in varying sizes.

We take 10 small-scale and 10 normal-scale environments to test our approach. These are shown in fig. 13 and fig. 14. In each environment, we sample 10 random navigation tasks. Each task consists of a start state and a goal coordinate. The two are picked to be at a Euclidean distance of at least 1m for small environments and at least 4m in normal ones, to make sure the two settings consider navigation tasks of differing complexity. As before, a task is only successful if the agent is within a radius of $\Delta_{\text{reach}} = 0.4\text{m}$ of the goal and has decided that it has reached the goal. The agent does so only if the distance between its state estimate and the goal is less than $\Delta_{\text{est}} = 0.3\text{m}$, where the stricter threshold is intended to avoid false positives.

Within each task, the agent only receives the goal coordinate and its initial state. Afterwards, it must rely on its state estimator to track itself and learns the map simultaneously. As before in the ViZDoom setup, we use a noisy version of the unicycle model inside the simulator, while the agent uses the noiseless version for state estimation

Figure 14: Overview of the normal-sized evaluation environments.

and planning. This time, we fix the noise scales at 2.86° and 0.02m. Note that the problem is much harder this time since the map is not known in advance.

5.6.1 QUALITATIVE RESULTS

We provide a bird's-eye-view of one navigation task in fig. 15. The middle panel shows the planning stage for one time step by zooming into the corresponding part of the scene. Black contours separate regions violating the safety constraint from free space. Starting from its state estimate, the agent uses its transition model to try out a number of control sequences in parallel. Plans violating constraints (shown in orange) are discarded, and the best one among the rest (shown in red) is picked. Only the first three steps of the plan are executed before reverting to planning again. The right panel shows the state estimate of the agent versus the true state.

5.6.2 QUANTITATIVE RESULTS

We now turn to a quantitative evaluation of the navigation performance. The navigation success rate of our method in each set of environments is shown in fig. 17. Our method achieves success rates of 89% and 77% in small and normal-sized environments.

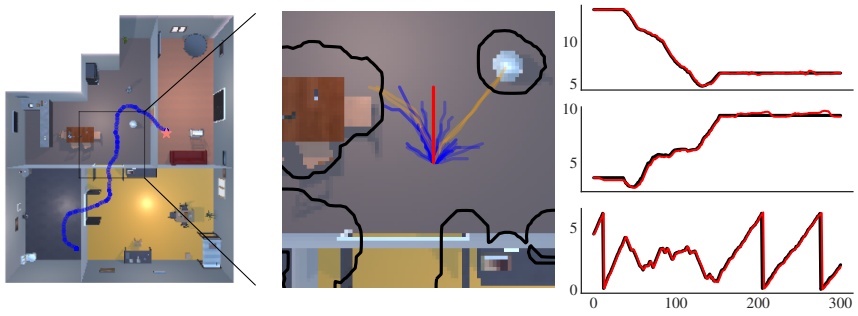

Figure 15: Bird's-eye-view of a rollout. Left, the entire trajectory with the goal shown as a pink star. Middle, one step of planning stage. Valid plans are shown in blue, invalid ones in orange. The selected plan in red. The borders between safe spaces and constraint-violating areas are shown as black contours. Right, the agent's estimates for its x- and y-coordinates and its yaw angle plotted against the true values.

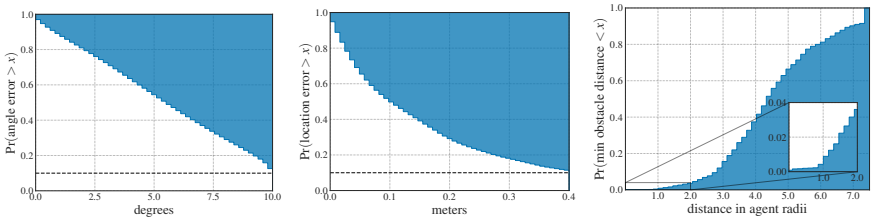

Figure 16: (Truncated) cumulative distribution functions for angle errors, location errors and the minimum distance to any obstacle. For the angle and location errors, we only show up to 90% probability to remove outliers and make the plots more legible.

scale	success rate	loc error	angle error	violation rate
small	0.8944 ± 0.0097	0.16 ± 0.01m	$12.1 \pm 0.8°$	0.003 ± 0.002
normal	0.7336 ± 0.0236	0.31 ± 0.03m	$15.8 \pm 1.5°$	0.043 ± 0.008

Figure 17: Metrics in each environment class.

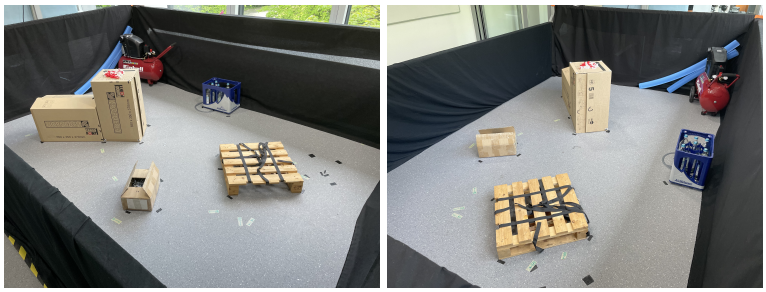

Figure 18: QCar experimental arena.

Aside from the success rate, we also examine the angle and location estimation errors, as well as the constraint violation rate. Figure 17 contains the average angle and location error and the fraction of time where the agent violated the constraints (that is, the agent's minimum distance to any obstacle was less than 1.7 times its radius). We find that in small environments, the absolute location error is less than the agent's radius (which is 0.2 m), while in normal environments it is slightly above. The average angle accuracy is similar in both classes. In small environments, the constraint violation rate is less than 1%, while in normal ones it increases to 3%.

To get a better understanding of these numbers, we report empirical cumulative distribution functions of the state estimation errors and the agent's distance to the closest obstacle in fig. 16, taken over both small and normal environments. For angle and location errors, we cut the plot at the 90% probability line, which lies at 10 degrees and 0.4 m respectively. For the distance to the closes obstacle, we find that the agent is farther away than any obstacle by two times its own body radius with more than 96% probability.

5.7 EXPERIMENTS ON REAL HARDWARE

As a final test setting, we experiment with using the method on real hardware. For that, we use a Quanser QCar [3], which is a small robot car. The QCar is equipped with an Intel D435 depth camera and an NVIDIA Jetson TX2 for computing. Though a LIDAR sensor and an IMU are also available, we do not use these in our experiments.

We build a small arena for experimentation, shown in fig. 18. The arena contains several objects which serve as obstacles. A set of OptiTrack [4] cameras are mounted on the arena. These track markers placed on the QCar, allowing us access to the location and orientation of the car. Figure 19 shows one of the OptiTrack cameras, along with the QCar.

We learn a model of the arena using the RGB-D camera and poses found by the OptiTrack. Afterwards, this model is used for planning in a model-predictive control setup, where the transition dynamics are modelled using the kinematic bicycle model. The kinematic bicycle model works with forward velocities (that is, velocity in the

3 https://www.quanser.com/products/qcar/
4 https://optitrack.com/cameras/flex-13/

Figure 19: (Left) Optitrack camera. (Right) QCar. Markers placed on the QCar allow tracking the car's location and orientation through the OptiTrack system.

Figure 20: (Left) Isometric view of the arena. (Right) Cost-to-go function overlayed on an emission.

direction of the current heading). To translate the throttle control of the car into a velocity, we manually found a throttle-to-velocity translation coefficient by looking at a data set of transitions.

Because the car's dynamics are more complex than in the ProcTHOR setup, we change the planner in two ways. First, the cost-to-go function also considers the orientation of the agent, and not just the location. Thus, the cost-to-go function is three-dimensional. Second, we refine the result of the constrained random search algorithm with gradient-based optimisation.

We examine how well the learned model lends itself to navigation based on a set of 100 randomly sampled targets. The robot navigates to each target one after the other. A human operator aborts the navigation trial if the target is not reached within 30 seconds, and a new target is sampled. In addition to navigation success, we also log each time the robot makes contact with any of the obstacles.

We find that the method is able to navigate the agent to all 100 targets, while the QCar collides against an obstacle on 6 occasions. Figure 21 shows a histogram of the speed of the QCar, which is 30 cm/s in average.

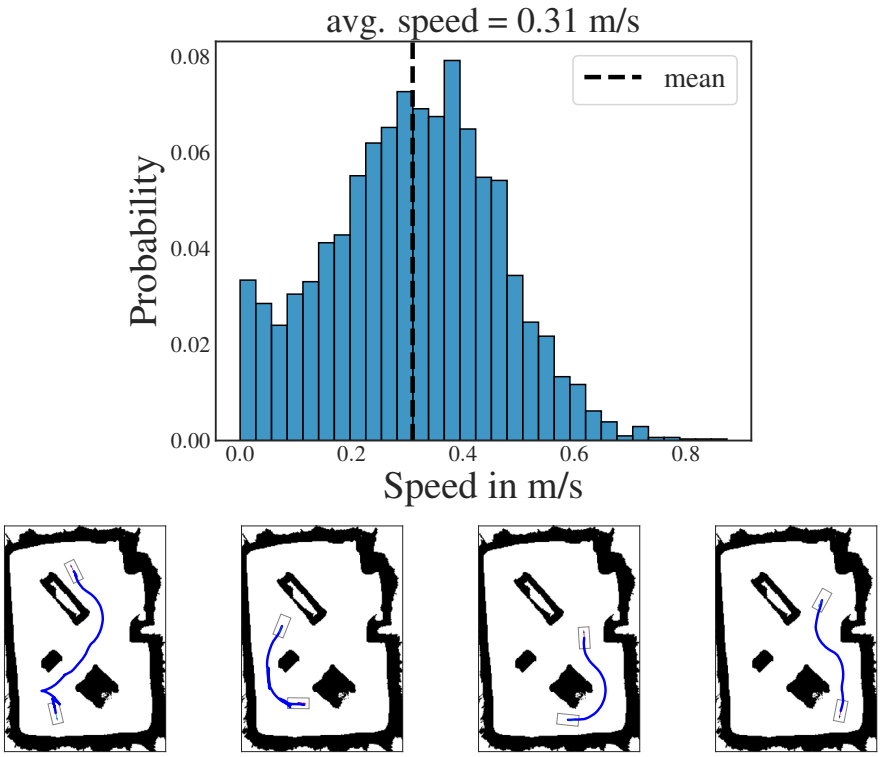

Figure 21: (Top) Histogram of the QCar's speed. (Bottom) Sample navigation tasks.

6 FILTER-AWARE MODEL-PREDICTIVE CONTROL

State estimation errors are the largest vulnerability of the controllers we saw in the previous chapter. When we rely on algorithms that contain a lot of domain knowledge, we are able to solve complex problems like mapping and motion planning with relative ease. The catch is that our solutions require accurate estimates for quantities like the agent's orientation and location, and the configuration of the obstacles in the scene. In anything but the simplest settings, it will arguably be impossible to promise perfectly accurate estimates of all system variable all of the time. In other words, when developing navigation algorithms, we have to accept a base failure rate that results from the SLAM-backend alone. The idea is that even if our planning is optimal and we are able to guarantee constraint-avoidance, all bets are off when the state estimate degenerates beyond a certain degree. Beyond some level, the only way of improving the system's performance is to build better state estimation algorithms.

On the other hand, not all state estimation errors are unavoidable. Each state estimator comes with its own set of pitfalls. Feature-based tracking or tracking with photometric errors are sensitive to what the camera sees. These approaches might fail if the agent spends too much time looking at visually uninteresting objects such as blank walls.

Usually, the controller does not reason about how its actions influence the state estimator. We see this in the model-predictive controller used in the previous chapter. From the perspective of the planning algorithm, its estimate of the system state is only a replacement for the true system state at the beginning of the planning horizon. Neither future observations nor future state estimates exist within the planning objective. This can lead to a controller check-mating itself by planning a sequence of actions that are optimal in the cost-sense, but dangerous for state estimation.

6.1 TAXONOMY OF POMDP-SOLVERS

Before we dive into the specific topic of reasoning about the relationship between a controller that operates in state-space and its state estimator, let us briefly review the myriad approaches available for solving POMDPs, which is the more general task at hand. POMDPs can be used to describe widely different settings and as such, methods developed to solve what can be described as a POMDP also differ widely in their assumptions, strengths and drawbacks. Here, we will go through the most prominent ones.

STATE-BASED CONTROL WITH A STATE ESTIMATOR By state-based control we refer to any control law that can be described as a func-

tion of the current state, $\mathbf{z}_t$. As the current state is not available in a POMDP, we often use a state estimator to infer it. The controller is then provided with either a deterministic state estimate (such as the deterministic prediction of an RNN or the output of a deterministic SLAM algorithm) or a stochastic state estimate. Where a stochastic state estimate is concerned, we might either use the mode of a distribution or take a sampling-based approach to reconcile the uncertainty. A common example for this strategy was given in section 2.5, where the limited-lookahead objective of MPC can be integrated over multiple samples from the current state estimate. It is also possible to use a parametric policy that was trained with full state observability, where at execution we provide, for instance, the mean of a Gaussian state estimate.

This family of approaches completely ignores the relationship between the state estimator and the controller. It can be well-suited to tasks where it is possible to estimate the state based on a few subsequent observations (for instance image-based Mujoco environments). It is unlikely to succeed in tasks where the controller has to actively obtain some information that it needs to solve a task. One example is a searching task where the agent is asked to find an object in a new environment.

The assumptions of this approach depend on the implementation. Where MPC is used, we assume that (a model of) the transition and cost functions are known. Specific state estimators might make additional assumptions. For instance, in particle filtering, we assume that (a model of) the emission function is known.

STATE-BASED CONTROL WITH A STATE ESTIMATOR IN AN ABSTRACT STATE SPACE By abstract state space, we refer to a state representation that is learned by a neural network. This is typically the case when a deep variational state-space model is used to learn an approximation of a POMDP based on interaction data which does not contain the true system state, as decribed in section 2.7. At first glance, this category is similar to the previous. However, as discussed in section 2.9, the fact that an abstract state space is learned makes the difference. The controller only has the limitations of state-based control *in the learned POMDP*, not necessarily in the true POMDP, which we are actually interested in controlling.

As a thought experiment, let us imagine a scenario where by the grace of stochastic gradient descent optimisation, a deep variational state space model learns a state representation which exactly corresponds to the true belief state. That is, the latent state of the learned POMDP corresponds one-to-one to the true posterior density over the state of the true POMDP. In this situation, state-based control in the learned POMDP clearly corresponds to belief-based control in the true POMDP and is therefore optimal. Unfortunately, we have no way of characterising when and why such representations could be learned, making it difficult to characterise the strengths and weaknesses of this method in the context of belief planning.

Notable methods in this category are due to Hafner et al. (2019, 2020a,b, 2023) and Becker-Ehmck et al. (2020). Other successful methods do not use a model for policy search, but do learn an abstract state space which is fed to the policy (Han, Doya, and Tani, 2020; Igl et al., 2018; Lee et al., 2019).

As for assumptions, these method assumes access to interaction data from the environment, but does not require the transition, emission and cost functions to be known, or for approximate models to be available.

BELIEF PLANNING USING THE TRUE POMDP Here, we collect approaches which assume that the POMDP that should be solved is fully-specified. That is, we have access to the true transition, cost and emission functions. Based on these, we try to solve the POMDP with belief planning. These approaches have a long history and can be subdivided into exact and approximate approaches.

Exact belief planning refers to finding the optimal policy for the POMDP. This is only possible for some POMDPs. Åström (1965) was the first to give a mathematical treatment of POMDPs. Exact solutions for POMDPs rely on the fact that the optimal cost-to-go function is piecewise-linear and convex over the belief space (Sondik, 1971). These approaches can be described by *policy trees*. A policy tree contains sequences of controls and the different observations that they might lead to.

The cost-to-go of a policy tree is often referred to as an α-vector (Kaelbling, Littman, and Cassandra, 1998). The literature on policy trees, α-vectors and efficient algorithms for exact POMDP solutions has a long history (Cassandra, Littman, and Zhang, 1998; Kaelbling, Littman, and Cassandra, 1998; Littman, 1996; Zhang and Zhang, 2001). We will not discuss this literature in detail, because it is somewhat isolated from the use-case we are interested in by how restrictive POMDPs are. As an example, note that exact POMDP solvers are typically limited problems with on the order of ten states (Roy, 2003).

Approximate optimal control (that is, sub-optimal control) is possible with different methods. These typically either restrict the number of α-vectors (Parr and Russell, 1995) or the number of beliefs that are considered (Pineau, Gordon, and Thrun, 2003). Beyond the challenge of computing a value function over beliefs, representing beliefs themselves is often challenging. Thus, one direction of research presents methods of approximate belief representations, typically through particles (Thrun, 1999). Combinations of particle filters and Monte Carlo tree search have also been investigated (Silver and Veness, 2010). This line of work has even been extended to continuous problems (Sunberg, Ho, and Kochenderfer, 2017; Sunberg and Kochenderfer, 2017).

Methods in this category generally assume access to the true transition, cost and emission functions or accurate models thereof. Many of the methods in this category are also tied to a specific type of state estimation algorithm, most notably particle filters. While recent iterations have been shown to solve continuous problems they are typically limited in the dimensionality of the observation space. As

an example a typical benchmark problem in this field might feature a robot with a LIDAR sensor (Garg, Hsu, and Lee, 2019; Wu et al., 2021), compared to a model-based RL method where most benchmarks focus on image-based observations.

LEARNING RECURRENT PARAMETRIC POLICIES By recurrent parametric policy, we mean any parametric model where the output is a function of the entire interaction history. This is typically a recurrent neural network. We focus on the type of policy used rather than the learning algorithm, as the former is more important for our purposes. This family of approaches can include an RNN policy that is trained via a policy gradient algorithm, or one that is trained on a world model on observations generated by the model. The main point is that the policy has to internalise state estimation.

Examples for works in this category are the approach of Ni, Eysenbach, and Salakhutdinov (2022), which uses model-free RL. Methods such as Stochastic Latent Actor-Critic (SLAC) (Lee et al., 2019), Deep Variational Reinforcement Learning (DVRL) (Igl et al., 2018) and Variational Recurrent Model (VRM) (Han, Doya, and Tani, 2020) are hybrid model-free model-based approaches which train recurrent networks.

This family can theoretically do optimal control, as the policy is a function of the interaction history. On the other hand, the success is highly dependent on many factors such as the sparseness of the cost, the ease at which new data can be collected or the length of memory necessary. These approaches generally assume that we can collect copious environment interaction data, similar to world model-based reinforcement learning, but potentially more intensive as the task of policy search also includes learning a state estimator.

TASK- OR STATE-ESTIMATOR-SPECIFIC SOLUTIONS In many cases a specific instance of a POMDP is interesting enough to justify a task-specific solution. Navigation is one example, where the entire field of SLAM can be seen as a task-specific POMDP solver. Many approaches try to model the relationship between a controller and its state estimator in the context of navigation.

Coastal navigation (Roy et al., 1999; Roy and Thrun, 1999) is an approach aimed at improving state estimation in map-based navigation. In the domain of Kalman filtering, *belief roadmaps* are another navigation-oriented approach, which has been deployed with an Extended Kalman Filter (EKF) (Prentice and Roy, 2009) and an Unscented Kalman Filter (UKF) (He, Prentice, and Roy, 2008) and in conjunction with constrained optimisation Zheng et al. (2021).

Aside from these, there is a great deal of research on the subject of state-estimation-aware control with Kalman filters. A number of these works use the tractable properties of Gaussian distributions to derive value learning algorithms (Berg, Patil, and Alterovitz, 2021; Platt et al., 2010; Todorov, 2005). Others use the easily quantifiable uncertainty of Gaussian beliefs to derive constraints or auxiliary objectives (Böhm,

2008; Hovd and Bitmead, 2005; Rafieisakhaei, Chakravorty, and Kumar, 2017; Rahman and Waslander, 2020).

The main drawback of these methods is their attachment to a specific state estimation algorithm (Kalman filtering) and a specific task (navigation). On the other hand, focusing on specific problems allows them to be applicable more challenging and realistic scenarios (for instance, real hardware) compared to the other approaches.

6.2 CONTROL AND STATE ESTIMATION

Among the many approaches listed in section 6.1, the first one is the most relevant to us. Specifically, we are interested in the case where MPC is combined with a state estimator. The chief drawback of this family of controllers is that the control law does not reason about its influence on the state estimator. Nonetheless, such a controller might be preferable over the other approaches depending on the task. For navigation, as an example, a model-predictive controller powered by a SLAM-backend is a general solution that can be used in any environment. Learning a parametric policy, on the other hand, would require collecting lots of data from many environments. If we could combine the generality of MPC and state estimation with a means for reasoning about beliefs, we could arrive at the best of both worlds.

Before describing our approach, we will first describe the relationship between a model-predictive controller and its state estimator mathematically. For a sound mathematical formulation of the problem, we must extend the state space of the task by any variables that factor into the state estimation algorithm. For these variables, we had introduced the concept of the *carry* in section 2.5, denoted by $\mathbf{h}$.

When we approach a partially-observable system with a specific state estimator, an optimal controller has to think about how state estimation will behave in the future depending on the controls. Formally, we can define a new POMDP where the state is expanded by the carry: $\mathbf{z}^+ = [\mathbf{h}, \mathbf{z}]$. Given a system state $\mathbf{z}_t$, a state estimator carry $\mathbf{h}_t$ and a control $\mathbf{u}_t$, the dynamics of the new POMDP combine transition, emission and state estimation:

$$\begin{aligned}
[\mathbf{h}_{t+1}, \mathbf{z}_{t+1}] &\sim p(\mathbf{z}^+_{t+1} \mid \mathbf{z}^+_t = [\mathbf{h}_t, \mathbf{z}_t], \mathbf{u}_t), \\
\mathbf{z}_{t+1} &\sim p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{u}_t), \\
\mathbf{x}_{t+1} &\sim p(\mathbf{x}_{t+1} \mid \mathbf{z}_{t+1}), \\
\mathbf{h}_{t+1} &= q(\mathbf{h}_{t+1} \mid \mathbf{x}_{t+1}, \mathbf{h}_t, \mathbf{u}_t).
\end{aligned}$$

We use the notation $h(\mathbf{h}' \mid \mathbf{x}, \mathbf{h}, \mathbf{u})$ to denote the state estimation step, where the carry is updated based on new data, analogous to section 2.5. Our exposition here is just a slightly different version of the classic Belief MDP formalism from section 2.4. The point where we diverge is that we do not assume the state estimation step is a Bayesian posterior update or that the belief is expressed as a density over the state space. This was true of the state estimator from section 5.4, and generally of any map-based navigation system which re-

lies on a SLAM-backend as a black box that provides estimates about the environment and the agent state.

More importantly, the carry $\mathbf{h}_t$ is not necessarily a sufficient statistic of the interaction history $I_t = \mathbf{x}_1, \mathbf{u}_1, \mathbf{x}_2, \mathbf{u}_2, \dots \mathbf{x}_t$. It is merely the internal state of an imperfect state estimator. Our goal, likewise, is not to solve belief planning, but to gain intuition about how a controller that is aware of its own state estimator would look. The emission function of the new POMDP, is a concatenation of the original observations and the state estimate:

$$[\mathbf{x}_t, \mathbf{h}_t] \sim p(\mathbf{x}_t, \mathbf{h}_t \mid \mathbf{z}_t^+ = [\mathbf{z}_t, \mathbf{h}_t]).$$

Naturally, an optimal controller would consider either the entire interaction history I_t or a sufficient statistic thereof. In our case, however, we are interested in policies that only work with the carry $\mathbf{h}_t$. Specifically, we are interested in the best policy that can be derived while using only the carry. We can start by writing down the expected total cost-to-go, assuming an infinite horizon:

$$\begin{aligned} J(\mathbf{h}_1, \mathbf{z}_1) = \mathbb{E}\Big[\sum_{t=1}^{\infty} \beta^{t-1} \mathbf{c}_t \Big], \\ \mathbf{z}_t &\sim p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}), \\ \mathbf{x}_t &\sim p(\mathbf{x}_t \mid \mathbf{z}_t), \\ \mathbf{h}_t &= q(\mathbf{h}_t \mid \mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{u}_{t-1}), \\ \mathbf{u}_t &= \pi(\mathbf{h}_t), \\ \mathbf{c}_t &\sim p(\mathbf{c}_t \mid \mathbf{z}_t, \mathbf{u}_t), \end{aligned}$$

for $t > 1$ and $\mathbf{u}_1 = \pi(\mathbf{h}_1)$. Note that the policy is penalised based on $\mathbf{z}_t$ but only sees $\mathbf{h}_t$. Decision-making now requires reasoning about the dynamics of our knowledge about the system as much as it does about the evolution of the system itself.

To make this more concrete, let us consider an absurd example. We have a navigation task where we estimate the agent state using an RNN which receives the agent's camera feed. On one wall in the environment there is a poster which contains an adversarial attack on the RNN, which causes it to completely misestimate the agent's location. Feeding this faulty estimate into a planner causes the agent to collide with a wall and breaks it. In this setup, the cost-to-go $J(\mathbf{h}, \mathbf{z})$ of the RNN state in front of the adversarial poster would capture this hiccup. Thus, any planner that has access to the cost-to-go could detect the problem and avoid looking at the poster.

6.3 BELIEF PLANNING AND MPC

If we implement the policy π with vanilla MPC, this is the same as optimising:

$$\mathbb{E}_{q(\mathbf{z}_1 \mid \mathbf{h}_1)} \Big[\mathbb{E}_{p(\mathbf{z}_{2:T} \mid \mathbf{z}_1, \mathbf{u}_{1:T-1})} \Big[\beta^{T-1} \hat{J}(\mathbf{z}_T) + \sum_{t=1}^{T-1} \beta^{t-1} \mathbf{c}_t \Big] \Big], \tag{21}$$

as introduced previously in section 2.5. It is easy to see that this optimisation objective does not reason about future observations—they never appear in the objective. To gain more intuition, let us consider a POMDP where using a certain control allows us to observe the true state of the system in the next time step. MPC would never select this control (except by chance under some stochastic optimisation algorithm), because it does not influence the objective.

We can extend eq. (21) using the state-carry terminal cost:

$$\mathbb{E}_{q(\mathbf{z}_1|\mathbf{h}_1)}\left[\mathbb{E}_{p(\mathbf{z}_{2:T}|\mathbf{z}_1,\mathbf{u}_{1:T-1})}\left[\beta^{T-1}\hat{J}(\mathbf{z}_T,\mathbf{h}_T)+\sum_{t=1}^{T-1}\beta^{t-1}\mathbf{c}_t\right]\right], \tag{22}$$

Note that $\mathbf{z}_T$ is still not the true state but derived from the initial state estimate using the transition model. Computing the future carry $\mathbf{h}_T$ requires computing all previous observations and carries $\mathbf{x}_{1:T-1}, \mathbf{h}_{1:T-1}$. Our goal here is to see what the platonic ideal for a state estimation-aware controller would be. In practice, eq. (22) is infeasible for multiple reasons. First, the state-carry value is not straight-forward to compute. Second, computing future observations and carries during planning inflates the cost of planning.

At this point, we return to our original goal of mitigating tracking failures. Belief planning is the gold standard in acting under partial-observability because it finds an optimal trade-off between reducing the task cost and managing information. That includes computing all the myriad ways in which information influences future costs. Solving a task might require gathering information. There might be cases in which not having a piece of information is fine, because the optimal action in any possible case is the same. In short, belief planning has to consider every possible interaction between a controller's expected future cost, and the information available to the controller.

By comparison, preventing state estimation errors is a far smaller ambition. Another way of describing our goal is information preservation. Depending on the type of state estimator, we can imagine a measure of the accuracy of state estimate and placing a threshold on it. We would like to behave in such a way that prevents the accuracy of the estimate from falling below a certain level. This is potentially much simpler, because we are not reasoning about the complete belief dynamics into the future, but only about the accuracy of the belief.

6.4 OVERVIEW OF THIS CHAPTER

In the remainder of this chapter, we will provide a mathematical treatment of the expected future accuracy of a state estimator under a controller, which we call *trackability*. We will then present filter-aware MPC, which transforms the MPC objective into a constrained optimisation problem by placing a constraint on trackability. Thus filter-aware MPC tries to minimise expected future cost, while guaranteeing a minimum level of state estimation accuracy.

We will show that trackability itself is the cost-to-go of the controller under a different POMDP, where the cost is the instantaneous

state estimation accuracy for each time step. Taking this view will allow us to learn trackability via standard function approximation techniques for value learning, given a data set of environment interactions. Finally, we will work around the issue of computing future beliefs during planning by resorting to an approximation.

We will demonstrate filter-aware MPC on a set of control tasks with different levels of difficulty. Each control task requires the controller to avoid certain conditions which lead to poor state estimation accuracy. Under these conditions, we will show that filter-aware MPC improves regular MPC, while performing either on part with or better than baselines capable of reasoning about future observations.

In addition to an experimental evaluation, we will also provide a discussion of the limitations of filter-aware MPC and draw comparisons to other belief planning approaches, as well as examine the added complexity over regular MPC.

6.5 FILTER-AWARE MPC

We aim to improve MPC by letting it distinguish between actions not just in terms of how much they reduce the cost, but also how they influence the belief. We want to find out the future accuracy of the state estimate under different plans and put a constraint on that value such that we only pick plans that guarantee a certain level of accuracy.

We have an urgent need to formalise the meaning of *state estimation accuracy*. First, we introduce $\epsilon(\mathbf{z}_t, \mathbf{h}_t)$, which is the instantaneous error of the state estimator. It is a function of the current state estimator carry $\mathbf{h}_t$ and the true system state $\mathbf{z}_t$. The implementation of $\epsilon(\mathbf{z}_t, \mathbf{h}_t)$ will depend on the specific state estimator. Typically this is a straightforward choice. If the state estimator returns a probability density function over the state space, then the error can be the negative log-likelihood $-\log q(\mathbf{z}_t \mid \mathbf{h}_t)$. A Kalman filter, for instance, falls under this category, since its state estimate is a Gaussian distribution. For state estimators that return a point-estimate, we can take the sum of squares.

We can take the expected total of this instantaneous error, starting from a specific state $\mathbf{z}$ and carry $\mathbf{h}$:

$$J^{\text{err}}(\mathbf{z}, \mathbf{h}) = \mathbb{E}\Big[\sum_{t=1}^{\infty} \beta^{t-1} \epsilon(\mathbf{z}_t, \mathbf{h}_t) \mid \mathbf{z}_1 = \mathbf{z}, \mathbf{h}_1 = \mathbf{h}\Big], \tag{23}$$

where the expectation is over both the transition and the emission noise and the controls are picked by some policy π. Future tracking errors are discounted by $\beta \in (0, 1)$.

We define $J^{\text{err}}(\mathbf{z}, \mathbf{h})$ as the trackability of $\mathbf{z}$ under the policy π and the carry $\mathbf{h}$. Using this notion of expected state estimation accuracy, we can extend eq. (27) with a constraint on the trackability:

$$\begin{aligned} \min_{\mathbf{u}_{t:t+K-1}} \quad & \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}_t|\mathbf{h}_t)}\Big[J^{\text{MPC}}(\mathbf{z})\Big], \\ \text{s.t.} \quad & \Pr\big(J^{\text{err}}(\mathbf{z}_{t+i}, \mathbf{h}_{t+i}) \leqslant \delta\big) \geqslant \Delta \quad \text{for} \quad i = 1, 2, \ldots, K. \end{aligned} \tag{24}$$

Here, δ is a threshold which defines how much tracker error we are willing to tolerate and Δ is a minimum probability of satisfying this condition that we wish to guarantee. The sudden appearance of the chance constraint (Farina, Giulioni, and Scattolini, 2016) in eq. (24) might be surprising. We formulate the constraint in terms of a probability because under stochastic dynamics, the future state $\mathbf{z}_{t+i}$ is random variable, and therefore its trackability is a random variable as well.

We refer to eq. (24) as filter-aware model-predictive control. Filter-aware MPC minimises the expected total cost while avoiding any plans that do cannot guarantee accurate state estimation with high probability. There is, however, a slight problem with eq. (24): evaluating the constraint requires computing the future state estimator carry $\mathbf{h}_{t+i}$. This places the computational cost at a similar point to regular belief planning. That is, we are spending the same amount of computation as belief planning, but are not even reaping the benefits of that effort fully.

Luckily, we can use a sensible approximation to reduce the effort drastically. In eq. (24), we can replace $J^{\text{err}}(\mathbf{z}_t, \mathbf{h}_t)$ with $J^{\text{err}}(\mathbf{z}_t, \mathbf{h}^*_{\mathbf{z}_t})$. That is, instead of checking the trackability with a sample of the future carry, we simply check how well state estimation can be done starting from $\mathbf{z}_t$ if the state estimate at that time is perfectly accurate. One way of looking at this is that we ignore state estimation errors that have happened in the past, and only reason about ones that will happen in future time steps $t' > t$. An important benefit of using $J^{\text{err}}(\mathbf{z}_t, \mathbf{h}^*_{\mathbf{z}_t})$ is that we can drop the conditioning on $\mathbf{h}^*_{\mathbf{z}_t}$ entirely, since $\mathbf{h}^*_{\mathbf{z}_t}$ is uniquely identified by $\mathbf{z}_t$. In the remainder, we drop the conditioning on $\mathbf{h}^*_{\mathbf{z}}$ and set $J^{\text{err}}(\mathbf{z}) := J^{\text{err}}(\mathbf{z}, \mathbf{h}^*_{\mathbf{z}})$ for brevity.

The benefit of formulating trackability as a function of the state alone becomes clear when we consider that we will use a multi-layer perceptron (MLP) to model $J^{\text{err}}(\mathbf{z})$. Since eq. (23) readily has the form of a cost-to-go, we can learn an MLP that approximates $J^{\text{err}}(\mathbf{z})$ using standard techniques for approximate dynamic programming. Specifically, we use TD(λ) (Sutton, 1988) to learn an MLP ϕ that approximates trackability.

6.5.1 LEARNING TRACKABILITY

Using the state estimator, we collect interaction data with a regular MPC policy. Here, we record the instantaneous system state $\mathbf{z}_t$, and state estimation error $\mathbf{e}_t = \epsilon(\mathbf{z}_t, \mathbf{h}_t)$ into a dataset of the form: $\mathcal{D} = \{\mathbf{z}^i_{1:T}, \mathbf{e}^i_{1:T-1}\}_{i=1}^N$.

Given $\mathcal{D}$, training the neural network via TD(λ) corresponds to:

$$\min_{\phi} \sum_{i}^{N} (\phi(\mathbf{z}^i_1) - J^{\text{err}}(\mathbf{z}^i_1; \lambda, \phi))^2, \tag{25}$$

where $J^{\text{err}}(\mathbf{z}_1^i; \lambda, \phi)$ is the λ-return of $\mathbf{z}_1^i$ defined as:

$$J^{\text{err}}(\mathbf{z}_1^i; \lambda, \phi) = (1-\lambda) \sum_{k=1}^{T-1} \lambda^{k-1} J_k^{\text{err}}(\mathbf{z}_1^i; \phi),$$

$$J_k^{\text{err}}(\mathbf{z}_1^i; \phi) = \sum_{t=1}^{k} \beta^{t-1} \mathbf{e}_t^i + \beta^k \phi(\mathbf{z}_{k+1}^i).$$

6.5.2 A PRACTICAL IMPLEMENTATION OF FILTER-AWARE MPC

Once the trackability net ϕ has been learned, any optimisation algorithm that can handle constraints can be used to optimise eq. (24) by replacing $J^{\text{err}}(\mathbf{z}_{t+i}, \mathbf{h}_{t+i}) \approx \phi(\mathbf{z}_{t+i})$. We use the same optimisation algorithm that was presented in section 5.5.2.4. While this does not guarantee that the trackability constraints will be satisfied all the time, we believe that being overly conservative with these constraints is unnecessary. That is because sporadic dangerous actions are unlikely to result in immediate tracking failure. At the same time, in section 5.6 we have observed the simple constraint-handling of random search to perform well enough for obstacle avoidance and therefore see it as a good fit for this task as well.

6.6 EXPERIMENTS

6.6.1 TOY SCENARIO

We start with a toy experiment that demonstrates our ideas in a simple setting.

We visualise this problem in fig. 22. We have a 2D agent that tries to reach the green box on the western side of the room. The cost is the distance to the goal zone and the agent can observe its location with some additive Gaussian noise that is significantly higher inside a circle in the center (marked by the grey circle in fig. 22 left column). The agent can control its velocity subject to additive noise.

First, we simplify the problem such that the observation noise is constant with a scale of 0.03 over the state space. The simplified problem can be solved by a model-predictive controller using a bootstrap particle filter for state estimation. The success rate comes out at 93%, shown in the bottom right panel of fig. 22. Transferring the same controller to the harder problem, where the grey circle has an observation noise of 1.0, wee see that the success rate drops to 39%. That is because vanilla MPC greedily minimises the cost by taking the shortest path, which directly leads into the grey circle, where state estimation promptly fails. At that point, the controller cannot recover, because its state estimates are not informative enough for planning.

To apply filter-aware MPC on this system, we first create a dataset of 500 rollouts with 30 time steps with the vanilla MPC policy. We then use this dataset to learn the trackability network, the output of which is shown in the top right panel of fig. 22. We can see that the learned trackability function can clearly separate safe and unsafe

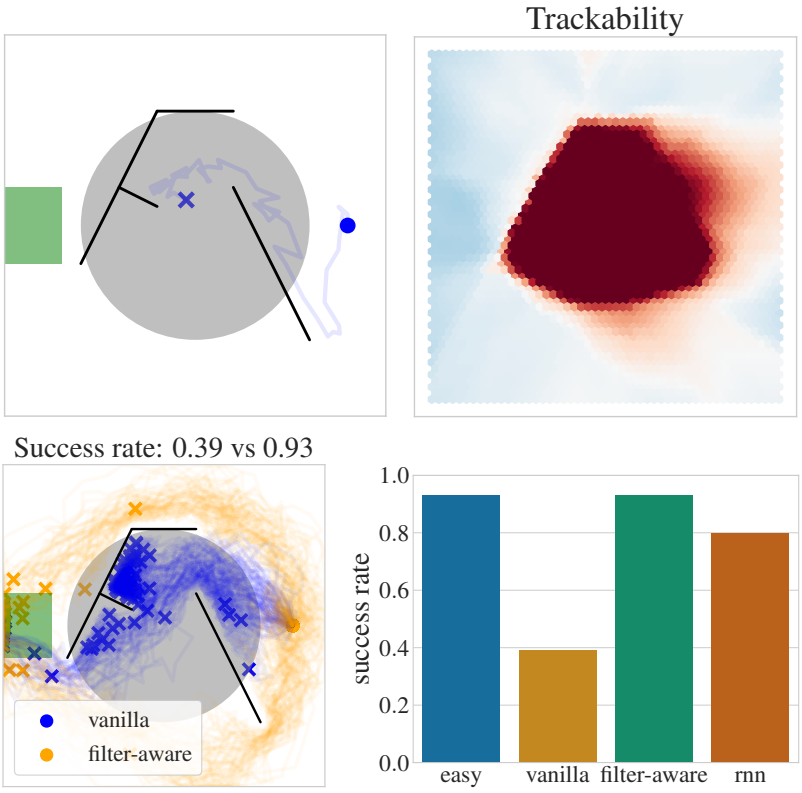

Figure 22: Toy scenario of a 2D particle aiming for the green region on the left starting from the right. The grey circle is a "dark zone" with higher observation noise. First row, left: A random walk. First row, right: Learned trackability. Second row, left: Filter-aware vs vanilla MPC. Second row, right: Success rates of vanilla MPC on an easy system with no grey zone, vanilla MPC, filter-aware MPC and an RNN policy.

This figure has previously appeared in (Kayalibay et al., 2023).

areas. Placing constraints on the network's predictions, we see that the agent can avoid the grey circle in favour of a slightly longer but safer route, shown in the bottom left panel of fig. 22.

For this experiment, we consider an RNN baseline, as shown in the bottom right panel of fig. 22. We use a differentiable implementation of the system dynamics, emission and cost to train the RNN by gradient descent to minimise the total cost over 50 time steps. Note that the RNN can implement belief planning, since it sees the entire interaction history, though this is dependent on the optimisation procedure converging to the right solution. Filter-aware MPC outperforms both regular MPC and the RNN and performs as well as regular MPC did on the easier version of the problem with no grey circle.

6.6.2 NAVIGATION IN VIZDOOM

We now turn to a visual navigation problem, which uses the ViZDoom simulator (Wydmuch, Kempka, and Jaśkowski, 2018). Here, we use the same state estimator that was used in section 5.4, with the exception that the learned emission model is replaced by the actual simulator, since our focus here is to learn a trackability critic and use it for control.

The ViZDoom environment has two rooms that are connected by two corridors. The agent is placed in a random location in one room and must go to a random location in the other room. It can see the world with an RGB-D camera and moves by picking a turning angle and a speed which is applied along its facing direction. These controls are perturbed by noise, making the dynamics stochastic. We show a sample RGB image in fig. 23 top right.

To introduce a danger against state estimation, we turn off observations in the left corridor. That is done by setting the RGB-D observation to zero. Thus, the agent can only track its state using the transition model once it enters the left corridor. The top left panel of fig. 23 shows how navigation with vanilla MPC fares under these conditions. Selecting either corridor with equal frequency, vanilla MPC ends up reaching the target only 48% of the time.

Once again, we learn the trackability function, which is shown in the bottom left panel of fig. 23. Since the trackability function clearly identifies the left corridor as a danger zone, filter-aware MPC is able to avoid that region of the state space entirely, increasing the success rate to 64% . As a theoretical upper bound of performance, we check the success rate of vanilla MPC when both corridors are safe, which lies at 69%.

6.6.3 ORBITING IN A REALISTIC ENVIRONMENT

The experiments so far take place in environments where there is a specific danger to state estimation that is somewhat artificially inserted into the setup. Our goal here was to verify that trackability can clearly delineate safe and unsafe areas. Having a visually-identifiable

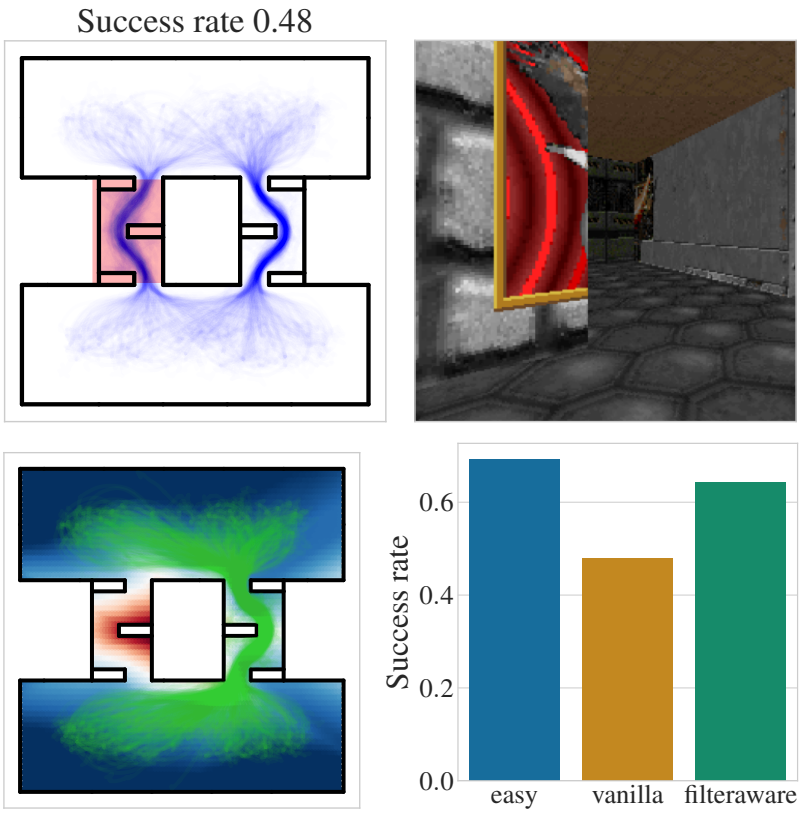

Figure 23: ViZDoom Setup. The agent starts in one room and must reach the other. First row, left: Vanilla MPC picks the left corridor where observations are corrupted. First row, right: Sample observation. Second row, left: Learned trackability function and filter-aware rollouts. Second row, right: Vanilla MPC on an easy problem where both corridors are safe vs vanilla MPC on the more difficult problem vs filter-aware MPC on the more difficult problem, where the left corridor corrupts observations.

This figure has previously appeared in (Kayalibay et al., 2023).

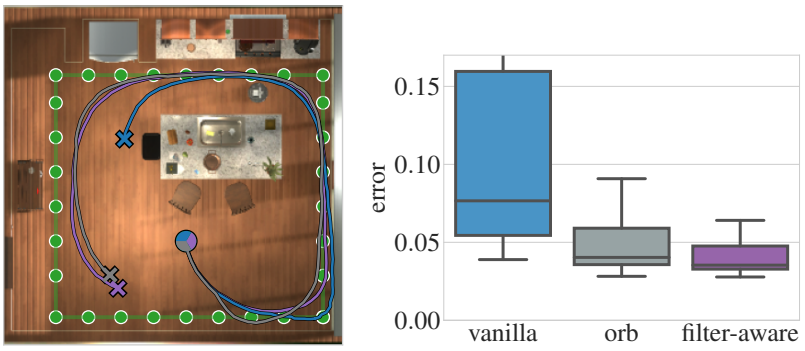

Figure 24: Orbit task. Left: Top-down view. The agent must follow the green landmarks. Blue is the average vanilla MPC trajectory, purple the filter-aware one, grey the ORB baseline. Right: Box plots for state estimation errors.
This figure has previously appeared in (Kayalibay et al., 2023).

Figure 25: Orbit task. Top: The naive strategy looks directly at the reflective surface, causing tracking failure. Bottom: Filter-aware MPC avoids looking at the reflective surface.
This figure has previously appeared in (Kayalibay et al., 2023).

region of state space that is more dangerous than the rest is helpful for that. For a more realistic setup, we turn to the AI2-THOR simulator (Kolve et al., 2017) which features models of realistic living spaces. Real-life environments naturally feature many situations that pose a difficulty for a state estimator working with images, most prominently featureless walls, glass and reflective objects. Our goal now is to check whether trackability is still helpful in a setting where the conditions that lead to poor state estimation accuracy are more subtle than a clearly defined danger zone. For that, we consider the task of following a fixed trajectory in a room.

A top-down view of the room is show in fig. 24, where the green markers show landmarks that the agent tries to follow. The agent's observations are once again RGB-D images and we use the same state estimation algorithm as in the ViZDoom experiments. The movement dynamics of the agent are similar to the ViZDoom setup as well, with one exception. The agent can now also control the facing angle of its camera, independently of its motion. The camera's viewing angle is not subject to any cost and is made to face the movement angle of the agent when we use the vanilla MPC policy. Note that vanilla MPC itself would not have any preference for moving the camera, since it does not enter into the dynamics of the body or the cost function. We face the camera in the movement direction because it is a sensible default strategy.

The eastern wall of the room features reflective surfaces, which hamper the state estimator's ability to keep track of the agent's state. Ideally, the trackability function should then be able to tell use that we should not point the camera towards these. We once again learn the trackability from rollouts collected by an MPC policy. Using the learned network in filter-aware MPC, we see that the agent is able to avoid looking at the reflective surfaces. This behaviour is compared to the default strategy in fig. 25. Here, the agent starts out looking at the table at the centre of the room. As it tries to navigate around the table, the default viewing angle is to face the reflective surface. Filter-aware MPC successfully avoids doing so, instead looking at the table which contains many interesting features that aid in state estimation.

Our main baseline for this experiment is a model-predictive controller which uses ORB features (Rublee et al., 2011) as a proxy for expected state estimation accuracy. More specifically, we train a neural network to predict the number of ORB features that in a given camera pose. This number is then used as a constraint for control, similar to how filter-aware MPC constrains trackability. This baseline was inspired by methods that are geared towards visual navigation (Falanga et al., 2018; Rahman and Waslander, 2020). Generally, the amount of ORB features correlates with how informative an image is in terms of state estimation. If the camera is pointed towards a blank wall, the number of features will be very low. If we instead point the camera towards the centre of the room, the resulting image will contain more features. We additionally compare against regular MPC as before.

In fig. 24 we draw a comparison between the three methods in terms of state estimation errors based on a set of 100 rollouts of length 200. Both the ORB baseline and filter-aware MPC improve significantly over the vanilla strategy, with the filter-aware controller performing the best in terms of state estimation. We show average rollouts for each method in fig. 24. Both the ORB baseline and filter-aware MPC complete the course, with filter-aware having a slight edge.

6.6.4 PLANAR TWO-LINK ARM WITH OCCLUDED REGIONS

Most of our experiments focus on navigation-like tasks in settings that approximate spatial environments to varying degrees. For our final experiments, we seek to illustrate the fact that filter-aware MPC is not tied to navigation problems or spatial environments. To do that, we use a two-link robot arm that tries to reach random targets, based on the Brax (Freeman et al., 2021) implementation of the classic reacher environment.

The reacher agent's observations are joint angles and velocities and the offset from the tip of its arm to the target. If any part of the arm moves into the left half of the work space, the agent's observations are set to zero. We add noise to the controls of the agent and use a particle filter for state estimation. As before, we learn a trackability

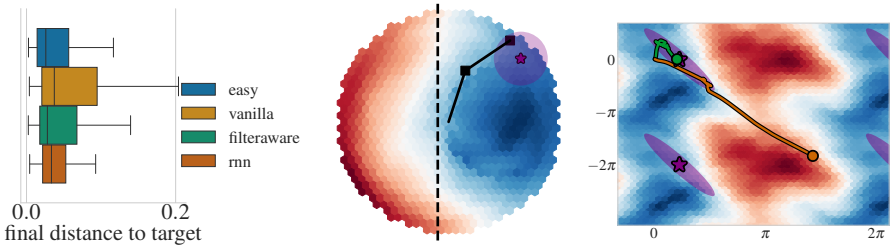

Figure 26: Reacher task. Left: Box plots. "easy" is vanilla MPC used on an easier problem. Middle: Top-down view. The circle shows a radius of 0.05 around the target. Observations are missing beyond the dashed line. The colour indicates trackability. Right: A filter-aware (green) vs a vanilla rollout (orange) in configuration space. The ellipse shows a radius of 0.05 around the target. Vanilla MPC enters the danger zone and loses track of itself.
This figure has previously appeared in (Kayalibay et al., 2023).

network, which is shown in fig. 26, both in the work space and in the space of joint angles. We find that the network is able to separate the left half of the work space. Using this information, a filter-aware controller is able to avoid the areas that lead to poor state estimation accuracy, while vanilla MPC enters into the dangerous part of the state space and thereafter loses track.

We compare filter-aware MPC, vanilla MPC and an RNN policy in fig. 26. The RNN policy follows the implementation of Ni, Eysenbach, and Salakhutdinov (2021). Both the RNN policy and filter-aware MPC improve over vanilla MPC, though the RNN has an edge here. We also present the performance of vanilla MPC in an easy setup where observations are available everywhere. Note that the RNN policy performs better than vanilla MPC even when vanilla MPC is used on the easy problem. This suggests that the performance gap between filter-aware MPC and the RNN might be due to the difference between MPC and amortised policy learning. In other words, MPC itself appears to be at a general disadvantage compared to an RNN policy in this problem.

6.7 DISCUSSION

6.7.1 COMPARISON TO OTHER POMDP METHODS

In the following, we will compare filter-aware MPC to each of the method families discussed in section 6.1.

STATE-BASED CONTROL WITH A STATE ESTIMATOR Filter-aware MPC builds on top of this family of methods by extending it with a trackability constraint, which strives to guarantee a minimum of state estimation accuracy. Unlike regular MPC and related approaches, filter-aware MPC is able to reason about the influence of the controller's influence on the state estimator, giving it an edge in applications where state estimation failures are critical and can be avoided by picking the correct controls.

As for assumptions, filter-aware MPC makes the same assumptions as regular MPC, that (a model of) the transition and cost functions are known. Additionally, filter-aware MPC requires a dataset of environment interactions to learn the trackability function from. In terms of computational complexity, filter-aware MPC is minimally more expensive than regular MPC, as it needs to evaluate the trackability function in each planning step.

STATE-BASED CONTROL WITH A STATE ESTIMATOR IN AN ABSTRACT STATE SPACE While model-based RL in abstract state-space can theoretically recover belief-planning-like behaviours, there is no guarantee that this will happen. As such, neither method performs belief planning. At the same time, model-based RL has been applied to complex problems which are out of the scope of filter-aware MPC (Hafner et al., 2023).

On the other hand, many other problems are more easily solved by model-predictive control with engineered models and state estimators. SLAM-based navigation is one example. It is unclear if model-based RL can solve such problems, due to the high complexity of real-life environments. In these cases, filter-aware MPC is still applicable.

As for assumptions, both methods rely on learning from environment interactions. One critical difference that the model-based approaches are aimed at online learning, while filter-aware MPC is more suited to learning from a static dataset.

BELIEF PLANNING USING THE TRUE POMDP Filter-aware MPC does not do belief planning. Instead, it tries to maintain the accuracy of the state estimate. The implicit assumption here is that a minimum level of accuracy is necessary to solve the task. At the same time, filter-aware MPC learns trackability as a function of the state only, and not as a function of the state and the state estimator carry. While this allows efficient planning, it further limits the applicability of the approach to problems where state estimation failures are caused by local features in a region of the state space.

As a counter example, take a simplified version of the Heaven and Hell system by Thrun (1999). In this system an agent in a 2D room must go to either the upper left or bottom right corner. At the beginning of a rollout, one of the corners is selected as the target, and the opposite becomes a danger zone with high cost. The agent can only observe which corner is heaven and which is hell when it is in the upper right corner. In all other locations, it can only observe its own location. This problem cannot be solved by filter-aware MPC, because we explicitly avoid modelling transitions in belief space and learn trackability as a function of the system state only (and not as a function of the system state *and* the carry). According to trackability, the only safe region would be the upper right corner, where the agent can observe the location of heaven and hell. That means the agent would be able to solve the first part of the problem (finding out

where the target is), but it would be unable to leave the upper right corner and actually go to the target because of the constraints.

At the same time, filter-aware MPC has some advantages over belief planning. First, belief planning approaches that are applicable to continuous domains are typically limited to particle filtering, while filter-aware MPC does not make any assumptions about which state-estimator is used. Second, filter-aware MPC does not assume the emission function is known. We only assume that a dataset of interactions is available. Finally, filter-aware MPC is applicable to much higher-dimensional observation spaces than belief planning, image-based tasks being one example.

LEARNING RECURRENT PARAMETRIC POLICIES Both approaches learn from environment interactions. Recurrent parametric policies try to solve an optimisation problem that might be harder than that of filter-aware MPC, as they need to learn both decision-making and state-estimation. Filter-aware MPC learns neither. Instead, we learn the expected future state estimation accuracy.

Control with recurrent parametric policies is theoretically more widely applicable, but similarly to model-based RL this applicability is curbed by the ease of data collection in many tasks.

TASK- OR STATE-ESTIMATOR-SPECIFIC SOLUTIONS Filter-aware MPC is closely linked to approaches like coastal navigation. The main difference is that we assume the presence of a dataset to learn the trackability function. At the same time, filter-aware MPC is slightly more general than the approaches in this category. It does not make any assumptions about which state estimator is being used and also is not limited to any particular task. However, as mentioned earlier, we are limited to problems where state estimation errors arise from features that are particular to regions of state space, such as the presence of a reflective surface at one part of the environment.

6.7.2 LIMITATIONS

In the following we will discuss several critical points and limitations of our approach.

LEARNING TRACKABILITY CONSTRAINTS. While our experiments show filter-aware MPC outperforming regular MPC in each case, in practice this requires taking care with the trackability function and the constraints. Since filter-aware MPC solves a constrained optimisation problem, we need to ensure that the constraints are not too strict. Otherwise planning will fail. In practice this means that the constraint thresholds have to be picked carefully. Our experiments feature environments where the state space is low-dimensional, such that the constraints can be easily visualised, which allows picking constraints by visual inspection. In higher-dimensional problems thresholds might

method	operations
state planning	J + h * m * (D + C)
filter-aware planning	J + h * m * (D + C + T)
belief planning	J + h * m * (D + C + E + F)

Table 1: Operations involved in planning. The letters denote evaluations of D - the dynamics model, J - the terminal cost, C - the stage cost, T - the trackability network, E - the emission model, F - the state estimator. The planning horizon is denoted by h, and the number of MC samples used to estimate expectations is denoted by m.
A similar table has previously appeared in (Kayalibay et al., 2023)

need to be picked by hyper-parameter searching, which is more cumbersome.

At the same time, learning the trackability function can be challenging. In our expreiments we relied on environment interactions. It would be much more practical to learn trackability based on model simulations, similar to how we learn policies and value functions in model-based RL. The challenge here is that this requires being able to model emissions to an accurate enough degree that the interaction between the state estimator and the observations it receives behaves the same as in the real world. If we take the example of the orbiting task, this requires modelling the reflective surface well enough that it throws off the state estimator in the same way as the real reflective surface does. While we have seen enormous leaps in the visual fidelity of differentiable renderers (Müller et al., 2022), it is unclear if it is possible to use these for learning trackability.

COMPUTATIONAL COST. We break down the steps involved in state planning (that is, regular MPC), belief planning and filter-aware planning in table 1. The cheapest of the three is state planning, where we only query the dynamics model and the cost function, in addition to the terminal cost for the last step. In belief planning, every time step also includes at least one evaluation of the emission model and at least one evaluation of the state estimate. Filter-aware MPC does not evaluate the emission and the state estimator, but instead evaluates a trackability critic.

What does this imply in practice? This depends very much on the use case. In a low-dimensional system with simple dynamics that can be summarised in a few lines of code or math, we can expect the dynamics model and the cost function to be very cheap. This is true of systems like the pendulum or our toy task. Here, filter-aware MPC might be significantly slower than vanilla MPC, as evaluating a neural network is much more costly than computing the next state of a pendulum. On the other hand, if the dynamics and cost are modelled with neural networks or when more complicated physics simulators are involved, we can expect the additional cost of evaluating a neural network to be minimal.

method	toy	ai2thor	reacher
vanilla	192 Hz	28 Hz	46 Hz
filter-aware	119 Hz	16 Hz	45 Hz

Table 2: Runtimes in each environment.
A similar table has previously appeared in (Kayalibay et al., 2023)

To provide more intuition, we report runtimes for our vanilla and filter-aware MPC in table 2.[1] We find that filter-aware MPC runs between 1.02 and 1.75 times slower than vanilla MPC. In the toy task, filter-aware MPC almost doubles the runtime because every other computation in the pipeline is quite simple. In the reacher, physics simulation takes up most of the time for planning, and neural network evaluations are less important.

6.8 OUTLOOK

We have presented filter-aware MPC, which allows a model-predictive controller to avoid certain state estimation errors. Filter-aware MPC is mostly relevant for POMDPs where the main challenge lies in preserving accurate estimates of physical quantities that are important for control. In these systems, we can work around the expense of planning in belief space while still being able to account for the actions of the agent on its ability to estimate its state. The catch is that our method is not suitable to more complex POMDPs which require seeking out new information.

The most interesting avenue of future work is to learn trackability from model simulations. This requires approximating the emission function to a high degree of fidelity. Whether this is possible in a visual navigation task remains to be seen.

1 The ViZDoom environment is excluded because both approaches run at 3 Hz in this task. That is caused by an inefficient implementation of collision-checking inside the simulator, which we had to use in lieu of the simulator's own collision handling in order to enable continuous actions.

Part III

CONCLUSION

7 CONCLUSION

This thesis has dealt with the approach of using world models to solve partially-observable Markov decision processes. We showed that learning purely neural network-based world models from interaction data is prone to a previously unexplored pitfall, the so-called conditioning gap, which prevents learning a model good enough for policy search. Though the conditioning gap is easy to fix, it is an indication that we do not yet entirely understand what makes our methods or, or what makes them fail.

Facing the intricacies of neural networks and with the goal of scaling up world models to realistic everyday spatial environments, we turned to inductive biases. We presented an evolution of models for space where neural networks appear as building blocks held together by domain knowledge about 3D geometry. This sequence of works culminated in our own work which presents a spatial world model capable of capturing real-world scenes and performing probabilistic inference about the environment and the agent state. Such spatial world models allow approaching tasks like navigation with simple model-predictive control.

At the same time, we acknowledge the fact that model-predictive control is an inadequate approach to solving POMDPs, due to its inability to reason about future observations and state estimation. To tackle this problem, we presented filter-aware MPC, which uses a constrained MPC objective to maintain a minimum degree of state estimation accuracy. Filter-aware MPC provides a balance between full belief planning, which is optimal but often infeasible, and blindly optimising the task cost given the current state estimate. Crucially, filter-aware MPC makes use of real environment interactions to learn a function of expected state estimation accuracy, which we call trackability. Can we leverage world models to learn trackability?

As world models for spatial reasoning improve, we might be able to capture the appearance of a spatial environment with enough fidelity that it becomes possible to detect at which points a state estimator will fail, when used in that environment. However as yet, this remains to be shown.

Today, model-based reinforcement learning and control is more ambitious than ever. Within the space of a couple of years, the field has gone from struggling to solve simple image-based robotics tasks to attempting complex reasoning tasks in large-scale environments which resemble the variety of the real world. In this landscape, our work focuses on two points.

The first is an emphasis on partial-observability and its implications. The textbook definition of a partially-observable system can mean many things, and unfortunately we find that the majority of established POMDP benchmarks have focused on simple systems

which do not require actively seeking out information. As we move to POMDPs with more strict partial-observability, we find that new issues emerge. Defects in our modelling which were previously permissible, the conditioning gap being one example, no longer remain viable.

The conditioning gap likely only scratches the surface of the unknowns of model-based control and policy learning in POMDPs. Are world models capable of learning state spaces which accurately capture the posterior distribution over the agent's state? As we try to scale up these methodologies to harder POMDPs, it will be necessary to tackle this question.

The second thread of our work has been inductive biases. The generalist approach of learning unstructured models from data is viable as long as we are able to pay the high price of collecting copious interaction data. As we move beyond into the realm of data scarcity, we need inductive biases to introduce valuable structure into our models. That structure allows us to apply our methods in a new environment without any environment-specific training stage. At the same time, it is that structure that allows leveraging efficient methods like dynamic programming and model-predictive control and formulating constraints to ensure safety.

While we applied this approach to the basic task of navigation, it is possible to use similar ideas for more complex tasks, where navigation emerges as a sub-task. Similar to how we model properties like occupancy and colour over space, we can also model other quantities, such as semantics, affordances (Qi et al., 2020) or the density of interesting visual features. Modelling the environment as a random variable also enables us to reason about the posterior distribution over unknown parts of the environment given the already explored part, which allows a Bayesian approach to searching in novel environments.

Part IV

APPENDIX

Part V

APPENDIX

A

LESS SUBOPTIMAL LEARNING AND CONTROL IN VARIATIONAL POMDPS

A.1 HYPER-PARAMETER SEARCH SPACE

hyper-parameter	value
model	
hidden units	$\{32, 64, 96\}$
number of layers	$\{1, 2\}$
activation	{softsign, relu, elu}
emission type	{learned, extracting}
emission activation	{softsign, relu, elu}
RNN units	$\{8, 16, 32, 64\}$
observation scale	$\{0.01, 0.02\}$
cost scale	$\{0.5, 0.75, 1.0\}$
learning rate	$\{0.0003, 0.001, 0.003\}$
initial flows	$\{3, 5, 8\}$
uses initial KL	{True, False}
batch size	128
time steps	16
policy	
RNN units	$\{8, 16, 32, 64\}$
batch size	$\{32, 64, 128\}$
training rollout length	$\{15, 30, 40\}$
learning rate	$\{0.0001, 0.0003\}$
hidden units	$\{8, 16, 32, 64\}$
number of layers	$\{1, 2\}$
number of configurations	
Dark Room	2400
Mountain Hike	1200
Meta Pendulum	300

Figure 27: Hyper-parameters search spaces used in our experiments.

Figure 27 lists the hyper-parameters and search spaces used in our experiments. We will briefly explain what each of these mean in the context of the learning algorithm explained in section 2.7.

- **hidden units.** Refers to the number of hidden units used in all MLPs in the model.
- **number of layers.** Refers to the number of layers used in all MLPs in the model.
- **activation.** Refers to the activation function used in all MLPs in the model, with the exception of the emission.
- **emission type.** We use two types of emission functions. In both, the standard deviation of the Gaussian likelihood is a hyper-

parameter, and the emission function produces the mean of the Gaussian. *learned* refers to a simple MLP mapping from states to observations. $\widehat{\text{extracting}}$ refers to a model which copies a number of state units into the observation. In this model part of the state learns to subsume the observation. In mathematical terms, if d_x and d_a are the size of an observation and a number of additional state dimensions, then the state space has $d_x + d_a$ many dimensions: $\mathbf{z} \in \mathbb{R}^{d_x + d_a}$, and the extracting emission function selects the first d_x dimensions of the state as the mean of the Gaussian likelihood over the observation.

- **emission activation.** Refers to the activation function used in emission MLP, if present.
- **RNN units.** The number of recurrent units used in the feature extracting RNN.
- **observation scale.** The standard deviation of the Gaussian likelihood over the observation.
- **cost scale.** The standard deviation of the Gaussian likelihood over the cost.
- **learning rate.** Step size of the Adam optimiser.
- **initial flows.** The number of RealNVP transformations used for the initial state distribution.
- **uses initial KL.** Whether we optimise the KL divergence between the prior over the initial state and its approximate posterior or not. This is the first KL term in eq. (10). We have found this term to be unstable during optimisation and thus explore the option of ignoring it.
- **batch size.** Batch size during model learning.
- **time steps.** The sequence length of traning data.

Similarly, for the policy:

- **RNN units.** Refers to the number of recurrent units used in policy.
- **batch size.** Number of rollouts used to approximate the expectation in eq. (11).
- **training rollout length.** The length of a training rollout.
- **learning rate.** Step size of the Adam optimiser.
- **hidden units.** Refers to the number of hidden units used in all MLPs in the policy. We use one MLP to embed the observation and another to map from the policy's recurrent state to the space of controls.
- **number of layers.** Refers to the number of layers used in all MLPs in the policy.

A.2 HYPER-PARAMETERS OF SIMPLE

We use a procedure similar to the Simple algorithm (Kaiser et al., 2020). For each environment we use rollouts of length 50 for collecting data, with the exception of Mountain Hike, where we use 75 steps. We perturb controls selected by the policy with zero-centred Gaussian noise with a standard deviation of 0.025. We kick-start the learning process with 25 rollouts collected through a random policy and afterwards add 5 new rollouts in each iteration, up to a total of 100 iterations. Each iteration contains a model-learning phase and a policy learning phase. In the model-learning phase the model parameters are updated for 2500 iterations and in the policy learning phase the policy parameters are updated for 50000.

B

NAVIGATION WITH SPATIAL WORLD MODELS

B.1 EXPERIMENTS IN VIZDOOM

B.1.1 MODEL DETAILS

We rely on voxel grids to model occupancy and color. In the ViZDoom experiments, each grid has spatial dimensions of $200 \times 200 \times 20$. The color grid has three feature dimensions that correspond to RGB values.

For ViZDoom we found approximate values for the camera parameters by projecting depth observations and aligning the resulting point cloud with the known level geometry. Thus, the focal length of the camera is 150, and the principal offsets in x and y are 160 and 120 respectively. The height and width of a camera image are 320 and 240. The camera is mounted on the body of the player at a quaternion orientation of $[0.5, -0.5, 0.5, -0.5]^T$. Finally, When casting rays for each pixel, we consider a maximum depth of 20 and discretise the interval $[0, 20]$ with 200 points.

In ViZDoom we map each environment using a set of 5000 pose-labeled RGB-D images. The locations and orientations of these are sampled uniformly from within the free-space of the scene. For ViZDoom, we optimise the occupancy and color models for 10000 steps using the Adam optimiser with a learning rate of 0.05.

In either setup, we use a batch size of 25 and subsample images by picking 200 pixels. The scale parameters of the Laplace distribution over color and occupancy σ_1 and σ_2 are modeled by setting $\sigma_1 = \sigma_2/5$. With σ_2 initialised to 2.4 and thereafter learned using the Adam optimiser with a learning rate of 0.01.

B.1.2 MOTION PLANNING AND LOW-LEVEL CONTROL DETAILS

We start A*-search from the starting location sampled for the respective task and explore the free space of the world model by stepping along the eight cardinal directions. The step size used here is 80% of the agent's own body length. We disallow stepping into any location that is closer to any obstacle than a safety distance of 1.2 times the agent's body length. The search is stopped as soon as we reach a position that is closer to the goal location than 80% of the agent's body. Both the occupancy grid we use for checking if a location is navigable and the set of obstacles we use to maintain the safety distance are based on a horizontal slice of the occupancy voxel grid.

The low-level controller first selects the closest waypoint from the trajectory planned by A* as a local target. Given the agent's inferred 2D location l, we first define the desired movement vector as $v =$

$l - t$, where t is the local target. We then check the difference between the angle of this vector and the agent's current heading o. If that difference is below a threshold of 5°, we allow the agent to take a step forward, respecting a maximum velocity constraint of 80% of the agent's body length. If the difference is higher than said threshold, the agent stays in-place and rotates to match o with the angle ν, while obeying a maximum angular velocity constraint. As soon as the agent is closer than a certain distance to the local target, we remove it from the set of waypoints and pick a new local target. Finally, if the agent can reach the target by moving in a straight line according to the occupancy model, we define the target as our only landmark, and do no further A*-based planning.

B.1.3 STATE ESTIMATION DETAILS

The agent's starting position is known. Thereafter, for each control executed in the simulator, we use the imperfect dynamics model to predict the next state of the agent. Then, we optimise eq. (19) using this predicted next state, the new camera image received from the simulator and the RGB-D image emitted from the model using the last inferred state. We use the Adam optimiser for 100 steps with a learning rate of 0.02. At every optimization step, we compute the loss for 1000 randomly selected pixels from the new image observation $\mathbf{x}_t$. We sample a different set of pixels in each of the 100 optimisation steps, but keep these 100×1000 pixel indices fixed for all invocations of the state estimator. Note that the subsampling happens only in the observed image, the predicted previous-step image $\tilde{\mathbf{x}}_{t-1}$ needs to remain whole, as we need its gradient information. We use bilinear interpolation of $\tilde{\mathbf{x}}_{t-1}$ to make the photometric term in eq. (18) differentiable w.r.t. the camera pose. We use the numeric gradients of $\tilde{\mathbf{x}}_{t-1}$ to compute the normals necessary for the second point-to-plane term in eq. (18). For the sake of robustness, we threshold the absolute values of the geometric and photometric errors in eq. (18). For a pixel with index i, the photometric error term was defined in eq. (18) as:

$$\left\| \tilde{\mathbf{x}}^{\text{rgb}} \left[\pi(\mathbf{T}_{z_1}^{z_2}(x_i)) \right] - \mathbf{x}^{\text{rgb}} [\pi(x_i)] \right\|_1 .$$

We ignore this term for any pixel where it exceeds a value of 1.0. Likewise, the geometric term is:

$$\left| \langle \tilde{x}_i - \mathbf{T}_{z_1}^{z_2}(x_i), \tilde{\mathbf{n}}_i \rangle \right|,$$

and it too is ignored for any pixel where it is higher than 5.0.

B.1.4 NAVIGATION

We sample 200 navigation tasks from each environment. Here, we only allow tasks where the start and target are farther apart than three times the agent's body length. Likewise, we do not allow start and target locations that are closer to any obstacle than 1.2 times the

agent's body. For this part, we access the actual scene geometry of the simulator in ViZDoom to define obstacles.

In ViZDoom, we implement continuous control via the teleportation and rotation commands. These do not check for collisions, so we check these manually by calculating distances to the obstacles, which we extract from the simulator in the form of a set of line segments. As soon as a collision occurs, we prohibit further movement.

B.2 EXPERIMENTS IN PROCTHOR

B.2.1 PRISM HYPER-PARAMETERS

hyper-parameter	value
map inference	
grid size	$200 \times 200 \times 200$
color priority	5.0
initial emission scale	2.4
occupancy update scale	0.1
color update scale	0.5
map update period	5
anchor update period	2
color map init. mean	0.5
color map init. stddev	50.0
occupancy map init. mean	-0.001
occupancy map init. stddev	50.0
state inference	
optimisation steps	1000
subsample size	200
location learning rate	0.001
orientation learning rate	0.00036
photometric residual threshold	0.18
geometric residual threshold	0.45
color scale	0.1
geometric scale	0.02
location initial stddev	0.33
orientation initial stddev	0.16
velocity initial stddev	0.01
orientation velocity init.$_{\mathrm{stddev}}$	0.0066
velocity stddev	0.01
orientation velocity stddev	0.01
location integration stddev	0.016
orientation integration stddev	0.0066

Figure 28: Hyper-parameters of the PRISM algorithm.

B.2.2 CONTROL HYPER-PARAMETERS AND SYSTEM CONFIGURATION

hyper-parameter	value
MPC	
replan interval	3
planning horizon	10
candidate plans	100
parallal rollouts	1
discount	1.0
TD(λ) - λ	1.0
system parameters	
max turn	10°
max speed	20cm
agent radius	20cm
steering noise scale	2.86°
throttle noise scale	2mm

Figure 29: Control hyper-parameters and system configuration.

B.2.3 RANDOM SAMPLING OF PLANS

We define an efficient sampling distribution for picking random plans. We sample a random turning command and repeat it for a random number of time steps within the planning horizon. At the same time, we sample a random time index where the agent starts moving forward and then move forward by a random amount. Since the turning and forward motion command can overlap, this results in movement that can start with an arc, afterwards transitioning into straight lines. In precise terms, our proposal distribution is defined as:

$$
\begin{aligned}
s &\sim \mathcal{U}(s \mid -\tau_{\text{turn}}, \tau_{\text{turn}}) \\
T_{\text{turn}} &\sim \mathcal{U}(\{1, \dots, T\}) \\
v &\sim \mathcal{U}(v \mid 0, \tau_{\text{throttle}}) \\
i_{\text{throttle}} &\sim \mathcal{U}(\{1, \dots, T\}) \\
s_t &= \begin{cases} s & t < T_{\text{turn}} \\ 0 & \text{otherwise} \end{cases} \\
v_t &= \begin{cases} v & t > i_{\text{throttle}} \\ 0 & \text{otherwise} \end{cases} \\
\mathbf{u}_t &= \begin{bmatrix} s_t & v_t \end{bmatrix}.
\end{aligned}
$$

Note that we do not allow moving backwards, as the agent could then bump into obstacles it has not seen yet. The maximum steering angle and maximum speed τ_{turn} and τ_{throttle} are specified in fig. 30 (as "max turn" and "max speed").

B.2.4 MODIFICATIONS TO THE PRISM ALGORITHM

A slight modification to the PRISM algorithm was necessary to enable online map and state estimation during control. PRISM applies map

updates at fixed intervals. Similarly, the anchor image used to estimate the agent state is updated at fixed intervals. We also use fixed intervals but additionally trigger an immediate map and anchor image update whenever the agent's model does not contain enough information about the current camera pose. The definition of "enough information" is tied to the fraction of pixels in the current anchor image that have non-zero colour. Because the colour map is initialised at zero, a zero-colour prediction implies that the ray for the respective pixel has landed in an area of the map that has not been learned yet. As soon as more than 10% of the image has zero colour, we trigger an immediate update of both the map and the anchor image.

B.3 EXPERIMENTS ON REAL HARDWARE

B.3.1 CONTROL HYPER-PARAMETERS

hyper-parameter	value
MPC	
replan interval	1
planning horizon	25
candidate plans	500
parallal rollouts	1
discount	1.0
gradient updates	500
control learning rate	0.1
Lagrange multiplied learning rate	0.001
dynamic programming	
location grid size	100
angle grid size	18

Figure 30: Control hyper-parameters .

B.3.2 RANDOM SAMPLING OF PLANS

We generate random plans for the bicycle model using a stochastic algorithm, which composes a plan out of left-turning arcs, right-turning arcs and straight lines. Every plan produced contains three chunks: one left-arc, one right-arc and one straight line, in a random order. The duration of each of these three chunks is random, with the condition that the total length of the plan is fixed to the length of the planning horizon used in MPC. The magnitude of the steering and throttle controls are sampled randomly for each plan and kept fix for the duration of the plan.

Only the sign of the steer and throttle varies from chunk to chunk, where a left-arc has negative steering, a right arc positive and a straight line has a sign of zero. The sign of the throttle control for each chunk is sampled randomly per chunk. That means the car can start moving in reverse gear at the beginning of a plan, then switch to forward gear and then return to reverse gear and so on.

```python
import random

def proposal(horizon, max_steer, max_throttle):
    steer_signs = [-1, 1, 0]  # left, right, straight
    steer = random.uniform(-max_steer, max_steer)
    throttle = random.uniform(-max_throttle, max_throttle)
    steers = []
    throttles = []
    time_left = horizon
    for chunk in range(3):
        steer_sign = random.choice(steer_signs)
        steer_signs.remove(steer_sign)
        throttle_sign = random.choice([-1, 1])
        duration = random.randint(1, time_left - (2 - chunk))
        time_left -= duration
        steers += [steer * steer_sign] * duration
        throttles += [throttle * throttle_sign] * duration

    effective_horizon = random.randint(1, horizon)
    steers = [
        s if i < effective_horizon else 0.0
        for i, s in enumerate(steers)
    ]
    throttles = [
        t if i < effective_horizon else 0.0
        for i, t in enumerate(throttles)
    ]
    return steers, throttles
```

Figure 31: Simplified python pseudo-code for the random plan sampler used in the QCar experiments. The actual implementation in our code base uses vectorised code and samples multiple plans in parallel.

Finally, we sample an *effective chunk length*, which allows sampling plans that are shorter than the planning horizon by setting all throttle controls to zero after a random time index, which is the effective length of the plan. The reason we set controls to zero instead of producing a shorter plan is to enable plans to be evaluated in parallel, which requires fixed-size memory chunks. Having a variable plan length is useful because some situations require a much shorter plan length than others. For instance, when the car is surrounded by obstacles, random plans beyond a certain length are more likely to collide with an obstacle. On the other hand, when the car is in a large free area, longer plans explore the state space better. Varying the plan length offers the best of both worlds.

We provide a Python-based pseudo-code implementation of the random plan sampler in fig. 31.

C

FILTER-AWARE MODEL-PREDICTIVE CONTROL

C.1 TRACKABILITY LEARNING

We use TD(λ) to learn a trackability network. Different papers have translated the TD(λ)-update rule into different neural network update schemes. The one we used in this paper combines gradient descent with the parameter-averaging method used by Fujimoto, Hoof, and Meger (2018).

The objective that we minimise is:

$$\mathcal{L}(\phi) = \sum_{i}^{N} (\phi(\mathbf{z}_1^i) - J^{\text{err}}(\mathbf{z}_1^i; \lambda, \phi'))^2,$$

with:

$$J^{\text{err}}(\mathbf{z}_1^i; \lambda, \phi') = (1-\lambda) \sum_{k=1}^{T-1} \lambda^{k-1} J_k^{\text{err}}(\mathbf{z}_1^i; \phi'),$$

$$J_k^{\text{err}}(\mathbf{z}_1^i; \phi') = \sum_{t=1}^{k} \beta^{t-1} \mathbf{e}_t^i + \beta^k \phi'(\mathbf{z}_{k+1}^i).$$

Here, we introduced ϕ', which is a different copy of ϕ that is only used to create targets for $\mathcal{L}(\phi)$. The learning rule mixes gradient updates on $\mathcal{L}(\phi)$ with parameter-averaging updates for ϕ':

$$\begin{aligned} \phi &\leftarrow \phi + \alpha \nabla \mathcal{L}(\phi) \\ \phi' &\leftarrow \eta * \phi' + (1-\eta) * \phi. \end{aligned} \tag{26}$$

The parameter-averaging factor $\eta \in (0, 1)$ controls the speed at which the targets of ϕ will change. We wrote a standard gradient descent update in eq. (26) for simplicity. In practice we use the Adam optimiser (Kingma and Ba, 2017).

C.2 MPC AND TERMINAL COSTS

We use terminal costs for two of our experiments: the toy scenario and ViZDoom. These problems have sparse costs, which makes it necessary to use a terminal cost, which is then also the driving factor when planning. This is because the cost signal has the same value for every plan that is not close to the target. The standard MPC objective with a terminal cost is:

$$\mathbb{E}_{\mathbf{z}_t | \mathbf{x}_{1:t}, \mathbf{u}_{1:t-1}} \left[\sum_{k=t}^{t+K} \beta^{k-t} c(\mathbf{z}_t, \mathbf{u}_t) + \beta^{K+t+1} \hat{J}(\mathbf{z}_K) \right]. \tag{27}$$

In practice we have found that we get better performance by using the λ-return as our objective. This is given by the following equations:

$$J^{\text{MPC}}(\mathbf{z}_1^i; \lambda) = (1-\lambda) \sum_{k=1}^{T-1} \lambda^{k-1} \hat{J}_k^{\text{MPC}}(\mathbf{z}_1^i),$$

$$\hat{J}_k^{\text{MPC}}(\mathbf{z}_1^i) = \sum_{t=1}^{k} \beta^{t-1} \mathbf{c}_t^i + \beta^k \hat{J}(\mathbf{z}_{k+1}^i).$$

This is a weighted average over MPC objectives with different planning horizons. We believe it worked better than the regular MPC objective due to the sparseness of the cost and the stochastic dynamics. When the cost is sparse, the terminal cost is the only factor that guides planning, until the agent is close enough to the target to reach it within the planning horizon. This is an issue under stochastic dynamics because the transition error will build up towards the end of the planning horizon. The end of the planning horizon will have the highest prediction error, though it will also be the only place where we check the terminal cost.

We can reduce the noise by shortening the planning horizon, but that in turn can hurt the performance if the terminal cost is not a perfect approximation of the true future total cost. This is the case in the ViZDoom environment, where the terminal cost is just the geodesic distance from a location to the target, assuming the agent can move in any direction that is not blocked by a wall. The geodesic distance doesn't take into account that the agent can only move along its facing direction and can only change that direction by a limited amount in each time step. In some places the agent needs time to turn and face a better direction. A longer planning horizon is necessary to make sure the agent has enough time to reorient itself and then also move for long enough to reach a lower terminal cost.

The λ-return is a middle-ground between a long planning horizon that has noisier planning and a low horizon that allows less complex maneuvers. This only applies to the two environments with sparse costs however, and in the rest of the environments we used the regular MPC objective without terminal costs.

C.3 EXPERIMENT DETAILS

C.3.1 TOY SETUP

PROBLEM SETTING. The system state is the 2D location of the point. The observations are the state with additive Gaussian noise. The controls are 2D velocities, with a speed limit of 0.05. The cost is 0 when the point is inside the goal area, 1 otherwise. Walls block the agent's movement. Each control is perturbed by additive Gaussian noise with a scale of 0.03 before being used. The observation noise is 0.03 outside of the dark zone and 1.0 inside. The dark zone is centered at $(0.5, 0.5)$ and has a radius of 0.3. The environment is contained in the square area $[0, 1]^2$.

DYNAMICS EQUATIONS. Let $\mathbf{z} \in \mathbb{R}^2$ be a state, $\mathbf{x} \in \mathbb{R}^2$ and observation and $\mathbf{u} \in \mathbb{R}^2$ a control with $\|\mathbf{u}\|_2 \leqslant 0.05$. The system dynamics are:

$$\begin{aligned}
\tilde{\mathbf{u}}_t &= \mathbf{u}_t + \mathbf{w}_t \qquad \text{with } \mathbf{w}_t \sim \mathcal{N}(0, 0.03), \\
\mathbf{z}_{t+1} &= \begin{cases} \mathbf{z}_t + \tilde{\mathbf{u}}_t & \text{no collision} \\ \mathbf{z}_t - 0.01\tilde{\mathbf{u}}_t / (\|\tilde{\mathbf{u}}_t\|_2 + 0.0001) & \text{collision} \end{cases}, \\
\mathbf{x}_t &= \mathbf{z}_t + \mathbf{v}_t \qquad \text{with } \mathbf{v}_t \sim \mathcal{N}(0, \sigma_{\mathbf{w}}), \\
\sigma_{\mathbf{w}} &= \begin{cases} 0.03 & \|\mathbf{z}_t - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}\|_2 \geqslant 0.3 \\ 1.0 & \text{otherwise} \end{cases}.
\end{aligned}$$

Here, "collision" means the line segment that starts at $\mathbf{z}_t$ and ends at $\mathbf{z}_t + \tilde{\mathbf{u}}_t$ intersects one of the obstacles. In this case we take a small step in the opposite direction by normalising $\tilde{\mathbf{u}}_t$ to have unit length and scaling it up to 0.01. The additive constant 0.0001 is to avoid a division by zero.

STATE ESTIMATOR. We use a bootstrap particle filter. The proposal distribution is a Gaussian. Its mean is given by the transition function's prediction and its scale is the same as the transition noise: 0.03. We use 512 particles for estimating the initial state and 128 particles after that.

TERMINAL COST. The sparse cost requires using a terminal cost. We use the geodesic distance to the target area as a terminal cost. For filter-aware MPC, we re-calculate geodesic distances by respecting the constraint. If a location violates the constraints, we don't allow the shortest path to lead through that location. This is the same as treating the constrained areas as obstacles while calculating shortest distances.

RANDOM SEARCH HYPER-PARAMETERS. The proposal distribution samples a random control from a uniform distribution for the first time step and repeats it for the rest of the horizon. We take 100 candidates with a horizon of 10. For each plan we estimate the expected future total cost with 50 Monte Carlo samples. We execute the first control only, before re-estimating the state and re-planning. The MPC objective is the λ-return with $\lambda = 1.0$ and $\beta = 1.0$.

TRACKABILITY DATA. We collect rollout for trackability learning by using MPC on the true system. Here, the planning horizon is set to 5. We take 500 rollouts with 30 steps. The initial location is sampled uniformly over the space.

TRACKABILITY LEARNING. We estimate trackability with TD(λ), setting $\lambda = 0.95$ and $\beta = 0.8$. We divide the training rollouts into chunks of length 5. The tracking error is defined as the weighted sum

of the squared error of each particle, weighted by its particle weight. The parameter-averaging factor (see appendix C.1) is set to 0.995 and we use 5000 gradient updates. The learning rate is set to 0.001. We use use 128 hidden units and two hidden layers with relu activations. The batch size is 512.

COMPARATIVE STUDY. We compare filter-aware MPC and regular MPC using 100 rollout of length 50. The *easy* setting uses an emission noise of 0.03 everywhere, omitting the dark zone. For filter-aware MPC, the constraint threshold δ is set to 0.6.

RNN BASELINE. The RNN baseline is trained by gradient descent to minimise the total cost in a horizon of length 50. The initial state is sampled uniformly to give the agent access to experience from all over the environment. The cost is modified to be the length of the shortest path to the goal, i.e. the geodesic distance to the goal. We found this to be crucial for solving the task with a parametric policy. We precompute geodesic distances to the goal using a uniform grid and index the grid by discretising the current location of the agent. We define a custom gradient for this operation using finite differencing.

C.3.2 VIZDOOM

PROBLEM SETTING. The system state is the 2D location and 1D yaw of the agent. The observations are RGB-D images. The controls are the turning angle and forward velocity, with a maximum angle of 6° and a maximum speed of 0.015. The cost is 0 when the agent is within a radius of 0.1 of the goal, 1 otherwise. The controls are perturbed by clipped Gaussian noise. The clipping ensures that noise does not flip the direction of the control. If the agent tries to turn left by 2°, the noise is clipped to be in $[-2, \infty)$, such that noise cannot cause it to turn right instead of left. Likewise, if the agent tries to move forward with a speed of 0.01, the speed noise is clipped to be in $[-0.01, \infty]$ such that it cannot move backward instead of forward. When the agent enters the left corridor, every pixel in the RGB-D observation is set to zero. The environment is contained in the square area $[0, 1]^2$.

DYNAMICS EQUATIONS. Let $\alpha \in [0, 360°]$ be an orientation and $l \in \mathbb{R}^2$ a location. The system state is a concatenation of these: $\mathbf{z} = [\alpha, l]^T$. Let $\dot{\alpha} \in [-6°, 6°]$ be a turning angle and $s \in [-0.015, 0.015]$ a forward speed. The control is a concatenation of these: $\mathbf{u} = [\dot{\alpha}, s]^T$. The dynamics are then:

$$\alpha_{t+1} = \alpha_t + \text{sign}(\dot{\alpha}_t) \max(0, |\dot{\alpha}_t| + \epsilon^\alpha_t) \text{ with } \epsilon^\alpha_t \sim \mathcal{N}(0, 2°),$$

$$l_{t+1} = l_t + \begin{bmatrix} \cos(\alpha_{t+1}) \\ \sin(\alpha_{t+1}) \end{bmatrix} \max(0, s_t + \epsilon^s_t) \text{ with } \epsilon^s_t \sim \mathcal{N}(0, 0.0035).$$

The way in which noise is injected into the dynamics was previously used by Kayalibay et al. (2022). Note that these dynamics are further subject to collision handling with the environment. Here, we only allow moving from l_t to l_{t+1} if the line segment connecting these points does not intersect any wall. If it does intersect a wall, we only allow moving up to the intersection point. The observations $\mathbf{x}$ are RGB-D images created by the ViZDoom simulator.

STATE ESTIMATOR. We use colored point-to-plane ICP (Audras et al., 2011; Chen and Medioni, 1992; Steinbrücker, Sturm, and Cremers, 2011) for state estimation.

TERMINAL COST. The sparse cost requires using a terminal cost. We use the geodesic distance to the goal as a terminal cost. For filter-aware MPC, we re-calculate geodesic distances by respecting the constraint. If a location violates the constraints, we don't allow the shortest path to lead through that location. This is the same as treating the constrained areas as obstacles while calculating shortest distances.

RANDOM SEARCH HYPER-PARAMETERS. The proposal distribution samples a random direction between left and right and turns in that direction for a random number of time steps using the maximum turning angle. We also sample a random time index to start moving forward in and then move forward with the maximum speed for a random number of time steps. We take 200 candidates with a horizon of 20. For each plan we estimate the expected future total cost with a single Monte Carlo sample. We execute the three controls of each plan, before re-planning. The MPC objective is the λ-return with $\lambda = 1.0$ and $\beta = 1.0$.

TRACKABILITY DATA. We collect rollout for trackability learning by using MPC on the true system. We take 900 rollouts with 200 steps.

TRACKABILITY LEARNING. We estimate trackability with TD(λ), setting $\lambda = 0.95$ and $\beta = 0.95$. We divide the training rollouts into chunks of length 5. We only accept a chunk into the training set if the tracking error at the second time step (i.e. after using the first control) is less than 0.99. The tracking error is defined as the squared distance between the agent's true and inferred locations. The network's input is the agent's location only (i.e. we disregard the orientation component). The parameter-averaging factor (see appendix C.1) is set to 0.995 and we use 10000 gradient updates. The learning rate is set to 0.0001. We use use 128 hidden units and two hidden layers with relu activations. The batch size is 512.

COMPARATIVE STUDY. We compare filter-aware MPC and regular MPC using 90 rollout of length 200. The *easy* setting uses the normal, uncorrupted RGB-D image in both corridors. For filter-aware MPC, the constraint threshold δ is set to 3.5.

C.3.3 AI2-THOR

PROBLEM SETTING. The state is the 2D location, 1D robot facing angle (yaw), and 1D camera angle relative to that. The observations are RGB-D images. Controls are 2D velocity, robot turning angle, camera turning angle. The maximum speed is 0.04 and the maximum turning angles are 10°. The cost is designed to make the agent follow a loop, which is specified by a list of landmarks. Given the agent's location l_t at time t, we find the closest landmark p_t^1 and the next landmark from the list p_t^2. We define a local target $p_t^* = p_t^1 + 0.55 * (p_t^2 - p_t^1)$. The final cost is then:

$$\begin{aligned} c(\mathbf{z}_t, \mathbf{u}_t, \mathbf{z}_{t+1}) = &\|p_{t+1}^* - l_{t+1}\| \\ &+ 10 * \langle v_{t+1}, p_{t+1}^2 - p_{t+1}^1 \rangle \\ &+ 10 * \langle p_{t+1}^2 - p_{t+1}^1, p_{t+1}^* - p_t^* \rangle, \end{aligned}$$

where v_{t+1} is the velocity component of the control $\mathbf{u}_{t+1}$ and $\langle \cdot, \cdot \rangle$ is the scalar product. The robot is initialised at (0.0, -2.5) facing the southern wall. We use the scene "FloorPlan10" from the set of AI2-THOR environments. We add zero-centered Gaussian noise with a scale of 0.02 to the velocity controls.

DYNAMICS EQUATIONS. Let $\alpha, \psi \in [0, 360°]$ be orientations and $l \in \mathbb{R}^2$ be a location. The state concatenates these: $\mathbf{z} = [\alpha, \psi, l]^T$. Let $\dot{\alpha}, \dot{\psi} \in [-10°, 10°]$ be turning angles and $v \in \mathbb{R}^2$ a velocity vector with $\|v\|_2 \leqslant 0.04$. The control concatenates these: $\mathbf{u} = [\dot{\alpha}, \dot{\psi}, v]^T$. Then, the dynamics are:

$$\begin{aligned} \alpha_{t+1} &= \alpha_t + \dot{\alpha}_t, \\ \psi_{t+1} &= \psi_t + \dot{\psi}_t, \\ l_{t+1} &= l_t + v_t + \mathbf{w}_t \qquad \text{with } \mathbf{w}_t \sim \mathcal{N}(0, 0.02). \end{aligned}$$

The observations $\mathbf{x}$ are RGB-D images produced by the AI2-THOR simulator. For that we teleport the camera to the 3D location $[l_t, 0.9]^T$ and yaw angle $\alpha_t + \psi_t$, where 0.9 is the constant elevation of the camera. Note that these dynamics are subject to collision handling done by AI2-THOR.

STATE ESTIMATOR. We use the same state estimator as in the ViZ-Doom experiments. We assume that the initial state of the system is known.

TERMINAL COST. Since the cost is not sparse, no terminal cost was necessary in this environment.

RANDOM SEARCH HYPER-PARAMETERS. We use a combination of two optimisers for filter-aware MPC. The first plans the controls for moving the robot: the velocity and turning angle. The second takes

the movement controls as given and only plans the camera angle. Only the second controller looks at the constraints. The first planner is implemented the same way as in the ViZDoom experiments. The second planner's proposal distribution first picks a random camera angle within the set of angles that can be reached in the planning horizon. Then it finds the individual turning angles at every time step that are required to reach that camera angle and maintain it. Both planners use the same hyper-parameters: 5000 candidate plans, a horizon of 5, 1 Monte Carlo sample per plan and a replanning interval of 3 time steps. For regular MPC we only use the first planner and set the relative camera angle to zero, so that the camera is always facing the direction of movement.

TRACKABILITY DATA. We collect training data by using regular MPC on the true system, taking 4500 rollouts of length 30. For this procedure we place the agent at a random location and facing angle.

TRACKABILITY LEARNING. We estimate trackability with TD(λ), setting $\lambda = 0.95$ and $\beta = 0.95$. We do not further divide the training rollouts into subsequences, and use the original rollouts themselves. We only accept a rollout into the training set if the tracking error at the second time step (i.e. after using the first control) is less than 0.99. The tracking error is defined as the squared distance between the agent's true and inferred locations. The network's input is the agent's true location and absolute camera angle (i.e. robot facing angle plus the relative camera angle) converted to cosine-sine representation. The parameter-averaging factor (see appendix C.1) is set to 0.995 and we use 10000 gradient updates. The learning rate is set to 0.001. We use use 128 hidden units and two hidden layers with relu activations. The batch size is 512.

COMPARATIVE STUDY. We compare filter-aware and regular MPC with 200 rollouts of length 100. The constraint threshold δ is set to -10.0, making sure the constraint is always active. This is an appropriate choice here because we can optimise for safe trackability by turning the camera alone which can be done without impeding the movement of the robot.

ORB BASELINE. The ORB baseline trains a neural network to predict the number of ORB features that will be visible under a given camera pose. Here, we use the same rollouts that were used to train the trackability critic. For each image, we detect ORB features with OpenCV (Bradski, 2000) and compute their number. We record the camera's yaw and 2D location in a dataset, along with the number of features that were detected. We train a neural network to map the camera pose to the feature count. The feature count is normalised by a division by 500, the maximum number of features that is allowed in the feature detection phase. The rest of the algorithm is identical to filter-aware MPC, with the trackability critic replaced by the ORB network.

C.3.4 REACHER

PROBLEM SETTING. The state is the 52-dimensional physics state used by Brax (referred to as *QP* in the Brax documentation). The observations are joint angles and the vector from the tip of the arm to the target. The controls are joint actuation signals for each joint. The cost is the absolute distance between the tip of the arm and the target. Whenever this distance drops below 0.05, the agent gets a bonus cost of -100 for one time step. If the distance rises above 0.05 again, this bonus is canceled out by a one-time penalty of 100. The initial state of the arm is as defined by the Brax implementation of the reacher. The target sampling is modified to make sure that targets are never in the dark zone, where observations are corrupted. We inject noise into the dynamics by perturbing the controls with zero-centered Gaussian noise with a scale of 0.1. Whenever the x-coordinate of any point on the arm is less than -0.02 (detected by checking 20 equally-spaced points on the arm), the observations are replaced by zero.

STATE ESTIMATOR. We use a bootstrap particle filter with Gaussian distributions placed on the transition and emission functions. The transition scale is 0.005 and the emission scale is 0.001. We assume the initial state is known, and use 100 particles for all future time steps. Though the state-space if 52-dimensional, many of the components are static and need to have specific values for accurate physics simulation. We set these static components to their ground-truth values and keep them fixed.

TERMINAL COST. Since the cost is not sparse, no terminal cost was necessary.

RANDOM SEARCH HYPER-PARAMETERS. The proposal distribution samples a random control for each joint and then applies that control for a random duration and from a random start. Some movements require both joints to be actuated at the same time, either with the same signal or with different signals (e.g. one joint turns left while the other turns right). To make sure we get enough samples where this is the case, we use same random time interval for both links in one third of the candidates and use both the same random interval and the same random control signal in another third. The remaining third samples different random intervals and controls for each joint. We use 300 candidates and a planning horizon of 22. We take 5 MC-samples per plan and use the first 3 controls of every plan before replanning.

TRACKABILITY DATA. The training data is generated by using regular MPC on the true system. We sample 5000 rollouts of length 50. The initial arm position is random and uniform across the state space (including the dark zone) to ensure a good coverage.

TRACKABILITY LEARNING. We estimate trackability with TD(λ), setting $\lambda = 0.95$ and $\beta = 0.95$. We do not further divide the training rollouts into subsequences, and use the original rollouts themselves. The tracking error is defined as the weighted sum of the squared error of each particle, weighted by its particle weight, as in the toy experiment. The network's input is the true joint angles converted to cosine-sine representation. The parameter-averaging factor (see appendix C.1) is set to 0.995 and we use 20000 gradient updates. The learning rate is set to 0.00001. We use use 128 hidden units and two hidden layers with relu activations. The batch size is 512.

COMPARATIVE STUDY. We compare filter-aware MPC and regular MPC with 100 rollouts of length 200. The constraint threshold δ is set to 9.0 for filter-aware MPC.

RNN BASELINE. The RNN baseline follows the design of Ni, Eysenbach, and Salakhutdinov (2021). We use the same model that was used in their "standard POMDP" experiments.

BIBLIOGRAPHY

Adamkiewicz, Michal, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager (2021). *Vision-Only Robot Navigation in a Neural Radiance World*. DOI: 10.48550/ARXIV.2110.00168. URL: https://arxiv.org/abs/2110.00168.

Anderson, Peter et al. (2018). *On Evaluation of Embodied Navigation Agents*. arXiv: 1807.06757 [cs.AI].

Åström, Karl Johan (1965). "Optimal control of Markov processes with incomplete state information." In: *Journal of Mathematical Analysis and Applications* 10.1, pp. 174–205.

Audras, Cedric, A Comport, Maxime Meilland, and Patrick Rives (2011). "Real-time dense appearance-based SLAM for RGB-D sensors." In: *Australasian Conf. on Robotics and Automation*. Vol. 2, pp. 2–2.

Bayer, Justin, Maximilian Soelch, Atanas Mirchev, Baris Kayalibay, and Patrick van der Smagt (2021). *Mind the Gap when Conditioning Amortised Inference in Sequential Latent-Variable Models*. arXiv: 2101.07046 [cs.LG].

Becker-Ehmck, Philip, Maximilian Karl, Jan Peters, and Patrick van der Smagt (2020). "Learning to Fly via Deep Model-Based Reinforcement Learning." In: *CoRR* abs/2003.08876. arXiv: 2003.08876. URL: https://arxiv.org/abs/2003.08876.

Bellman, Richard (1957). "A Markovian Decision Process." In: *Indiana Univ. Math. J.* 6 (4), pp. 679–684.

Bengio, Yoshua, Nicholas Léonard, and Aaron Courville (2013). *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation*. arXiv: 1308.3432 [cs.LG].

Berg, Jur van den, Sachin Patil, and Ron Alterovitz (2021). "Efficient Approximate Value Iteration for Continuous Gaussian POMDPs." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 26.1, pp. 1832–1838. DOI: 10.1609/aaai.v26i1.8371. URL: https://ojs.aaai.org/index.php/AAAI/article/view/8371.

Bertsekas, Dimitri P. (2005). *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific. ISBN: 1886529264. URL: http://www.worldcat.org/oclc/314894080.

Bi, Sai, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi (2020). *Neural Reflectance Fields for Appearance Acquisition*. DOI: 10.48550/ARXIV.2008.03824. URL: https://arxiv.org/abs/2008.03824.

Bradski, G. (2000). "The OpenCV Library." In: *Dr. Dobb's Journal of Software Tools*.

Böhm, Christoph (July 2008). *Avoidance of Poorly Observable Trajectories: A Predictive Control Perspective*. DOI: 10.3182/20080706-5-KR-1001.00332.

Cadena, Cesar, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard (2016). "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age." In: *IEEE Transactions on Robotics* 32.6, pp. 1309–1332. DOI: 10.1109/tro.2016.2624754. URL: https://doi.org/10.1109%2Ftro.2016.2624754.

Cao, Junli, Huan Wang, Pavlo Chemerys, Vladislav Shakhrai, Ju Hu, Yun Fu, Denys Makoviichuk, Sergey Tulyakov, and Jian Ren (2022). "Real-Time Neural Light Field on Mobile Devices." In: *arXiv preprint arXiv:2212.08057*.

Cassandra, A., M. L. Littman, and N. L. Zhang (1998). "Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes." In: *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 54–61.

Chaplot, Devendra Singh, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov (2020). *Learning to Explore using Active Neural SLAM*. DOI: 10.48550/ARXIV.2004.05155. URL: https://arxiv.org/abs/2004.05155.

Chen, Yang and Gérard Medioni (1992). "Object modelling by registration of multiple range images." In: *Image and vision computing* 10.3, pp. 145–155.

Cho, Kyunghyun, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio (2014). *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. arXiv: 1409.1259 [cs.CL].

Curless, Brian and Marc Levoy (1996). "A Volumetric Method for Building Complex Models from Range Images." In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 303–312. ISBN: 0897917464. DOI: 10.1145/237170.237269. URL: https://doi.org/10.1145/237170.237269.

Deitke, Matt et al. (2022). *ProcTHOR: Large-Scale Embodied AI Using Procedural Generation*. DOI: 10.48550/ARXIV.2206.06994. URL: https://arxiv.org/abs/2206.06994.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2017). *Density estimation using Real NVP*. arXiv: 1605.08803 [cs.LG].

Engel, J., V. Koltun, and D. Cremers (2016). "Direct Sparse Odometry." In: *arXiv:1607.02565*.

Falanga, Davide, Philipp Foehn, Peng Lu, and Davide Scaramuzza (2018). "PAMPC: Perception-Aware Model Predictive Control for Quadrotors." In: *CoRR* abs/1804.04811. arXiv: 1804.04811. URL: http://arxiv.org/abs/1804.04811.

Farina, Marcello, Luca Giulioni, and Riccardo Scattolini (2016). "Stochastic linear model predictive control with chance constraints–a review." In: *Journal of Process Control* 44, pp. 53–67.

Fraccaro, Marco, Danilo Jimenez Rezende, Yori Zwols, Alexander Pritzel, S. M. Ali Eslami, and Fabio Viola (2018). *Generative Temporal Models with Spatial Memory for Partially Observed Environments*. DOI: 10.48550/ARXIV.1804.09401. URL: https://arxiv.org/abs/1804.09401.

Freeman, C. Daniel, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem (2021). *Brax – A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. Version 0.0.12. URL: http://github.com/google/brax.

Fujimoto, Scott, Herke van Hoof, and David Meger (2018). "Addressing Function Approximation Error in Actor-Critic Methods." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 1587–1596. URL: https://proceedings.mlr.press/v80/fujimoto18a.html.

Garbin, Stephan J., Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin (2021). "FastNeRF: High-Fidelity Neural Rendering at 200FPS." In: *https://arxiv.org/abs/2103.10380*.

Garg, Neha Priyadarshini, David Hsu, and Wee Sun Lee (2019). "DESPOT-Alpha: Online POMDP Planning with Large State and Observation Spaces." In: *Proceedings of Robotics: Science and Systems*. Freiburg im Breisgau, Germany. DOI: 10.15607/RSS.2019.XV.006.

Grisetti, G., C. Stachniss, and W. Burgard (2005). "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling." In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2432–2437. DOI: 10.1109/ROBOT.2005.1570477.

Gupta, Saurabh, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik (2017). *Cognitive Mapping and Planning for Visual Navigation*. DOI: 10.48550/ARXIV.1702.03920. URL: https://arxiv.org/abs/1702.03920.

Ha, David and Jürgen Schmidhuber (2018). "World Models." In: DOI: 10.5281/ZENODO.1207631. URL: https://zenodo.org/record/1207631.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. arXiv: 1801.01290 [cs.LG].

Hafner, Danijar, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson (2019). "Learning Latent Dynamics for Planning from Pixels." In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 2555–2565. URL: http://proceedings.mlr.press/v97/hafner19a.html.

Hafner, Danijar, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba (2020a). "Mastering Atari with Discrete World Models." In: *CoRR* abs/2010.02193. arXiv: 2010.02193. URL: https://arxiv.org/abs/2010.02193.

Hafner, Danijar, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi (2020b). *Dream to Control: Learning Behaviors by Latent Imagination*. arXiv: 1912.01603 [cs.LG].

Hafner, Danijar, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap (2023). *Mastering Diverse Domains through World Models*. arXiv: 2301.04104 [cs.AI].

Han, Dongqi, Kenji Doya, and Jun Tani (2020). "Variational Recurrent Models for Solving Partially Observable Control Tasks." In: *8th*

International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net. URL: https://openreview.net/forum?id=r1lL4a4tDB.

Hansen, N. and A. Ostermeier (1996). "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation." In: *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 312–317. DOI: 10.1109/ICEC.1996.542381.

He, Ruijie, Sam Prentice, and Nicholas Roy (2008). "Planning in information space for a quadrotor helicopter in a GPS-denied environment." In: *2008 IEEE International Conference on Robotics and Automation*, pp. 1814–1820. DOI: 10.1109/ROBOT.2008.4543471.

Hovd, Morten and Robert Bitmead (Nov. 2005). *Interaction between control and state estimation in nonlinear MPC*.

Igl, Maximilian, Luisa M. Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson (2018). "Deep Variational Reinforcement Learning for POMDPs." In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 2122–2131. URL: http://proceedings.mlr.press/v80/igl18a.html.

Jordan, Michael I., Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul (1999). "An Introduction to Variational Methods for Graphical Models." In: *Learning in Graphical Models*. Cambridge, MA, USA: MIT Press, 105–161. ISBN: 0262600323.

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). "Planning and Acting in Partially Observable Stochastic Domains." In: *Artif. Intell.* 101.1-2, pp. 99–134. DOI: 10.1016/S0004-3702(98)00023-X. URL: https://doi.org/10.1016/S0004-3702(98)00023-X.

Kaiser, Lukasz et al. (2020). *Model-Based Reinforcement Learning for Atari*. arXiv: 1903.00374 [cs.LG].

Karl, Maximilian, Maximilian Soelch, Philip Becker-Ehmck, Djalel Benbouzid, Patrick van der Smagt, and Justin Bayer (2017). *Unsupervised Real-Time Control through Variational Empowerment*. arXiv: 1710.05101 [stat.ML].

Kayalibay, Baris, Atanas Mirchev, Ahmed Agha, Patrick van der Smagt, and Justin Bayer (2023). *Filter-Aware Model-Predictive Control*. DOI: 10.48550/ARXIV.2201.10335. URL: https://arxiv.org/abs/2201.10335.

Kayalibay, Baris, Atanas Mirchev, Patrick van der Smagt, and Justin Bayer (2021). "Less Suboptimal Learning and Control in Variational POMDPs." In: *Self-Supervision for Reinforcement Learning Workshop - ICLR 2021*. URL: https://openreview.net/forum?id=oe4q7ZiXwkL.

Kayalibay, Baris, Atanas Mirchev, Patrick van der Smagt, and Justin Bayer (2022). *Tracking and Planning with Spatial World Models*. DOI: 10.48550/ARXIV.2201.10335. URL: https://arxiv.org/abs/2201.10335.

Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Kingma, Diederik P and Max Welling (2022). *Auto-Encoding Variational Bayes*. arXiv: 1312.6114 [stat.ML].

Kolve, Eric, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi (2017). "AI2-THOR: An Interactive 3D Environment for Visual AI." In: *arXiv*.

Lee, Alex X, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine (2019). "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model." In: *arXiv preprint arXiv:1907.00953*.

Li, Chaojian, Sixu Li, Yang Zhao, Wenbo Zhu, and Yingyan Lin (2022). "RT-NeRF: Real-Time On-Device Neural Radiance Fields Towards Immersive AR/VR Rendering." In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2022)*.

Li, Tingguang, Danny Ho, Chenming Li, Delong Zhu, Chaoqun Wang, and Max Q. H. Meng (2020). *HouseExpo: A Large-scale 2D Indoor Layout Dataset for Learning-based Algorithms on Mobile Robots*. arXiv: 1903.09845 [cs.RO].

Littman, Michael Lederman (1996). *Algorithms for Sequential Decision Making*.

Lombardi, Stephen, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh (2019). "Neural volumes." In: *ACM Transactions on Graphics* 38.4, pp. 1–14. DOI: 10.1145/3306346.3323020. URL: https://doi.org/10.1145%2F3306346.3323020.

Lombardi, Stephen, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih (2021). *Mixture of Volumetric Primitives for Efficient Neural Rendering*. arXiv: 2103.01954 [cs.GR].

Mania, Horia, Aurelia Guy, and Benjamin Recht (2018). *Simple random search provides a competitive approach to reinforcement learning*. arXiv: 1803.07055 [cs.LG].

Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng (2020). *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. DOI: 10.48550/ARXIV.2003.08934. URL: https://arxiv.org/abs/2003.08934.

Mirchev, Atanas, Baris Kayalibay, Ahmed Agha, Patrick van der Smagt, Daniel Cremers, and Justin Bayer (2022). "PRISM: Probabilistic Real-Time Inference in Spatial World Models." In: *6th Annual Conference on Robot Learning*. URL: https://openreview.net/forum?id=X_qYPtJLaX8.

Mirchev, Atanas, Baris Kayalibay, Patrick van der Smagt, and Justin Bayer (2021). "Variational State-Space Models for Localisation and Dense 3D Mapping in 6 DoF." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=XAS3uKeFWj.

Mirchev, Atanas, Baris Kayalibay, Maximilian Soelch, Patrick van der Smagt, and Justin Bayer (2018). *Approximate Bayesian inference in*

spatial environments. DOI: 10.48550/ARXIV.1805.07206. URL: https://arxiv.org/abs/1805.07206.

Mirchev, Atanas, Baris Kayalibay, Maximilian Soelch, Patrick van der Smagt, and Justin Bayer (2019). *Approximate Bayesian inference in spatial environments*. arXiv: 1805.07206 [stat.ML].

Müller, Thomas, Alex Evans, Christoph Schied, and Alexander Keller (July 2022). "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding." In: *ACM Trans. Graph.* 41.4, 102:1–102:15. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127.

Mur-Artal, Raúl, J. M. M. Montiel, and Juan D. Tardós (2015). "ORB-SLAM: A Versatile and Accurate Monocular SLAM System." In: *IEEE Transactions on Robotics* 31.5, pp. 1147–1163. DOI: 10.1109/TRO.2015.2463671.

Murphy, Kevin P (1999). "Bayesian Map Learning in Dynamic Environments." In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press. URL: https://proceedings.neurips.cc/paper/1999/file/66be31e4c40d676991f2405aaecPaper.pdf.

Ni, Tianwei, Benjamin Eysenbach, and Ruslan Salakhutdinov (2021). *Recurrent Model-Free RL Can Be a Strong Baseline for Many POMDPs*. DOI: 10.48550/ARXIV.2110.05038. URL: https://arxiv.org/abs/2110.05038.

Ni, Tianwei, Benjamin Eysenbach, and Ruslan Salakhutdinov (2022). *Recurrent Model-Free RL Can Be a Strong Baseline for Many POMDPs*. arXiv: 2110.05038 [cs.LG].

Parisotto, Emilio and Ruslan Salakhutdinov (2017). *Neural Map: Structured Memory for Deep Reinforcement Learning*. DOI: 10.48550/ARXIV.1702.08360. URL: https://arxiv.org/abs/1702.08360.

Parr, R. and S. Russell (1995). "Approximating optimal policies for partially observable stochastic domains." In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.

Pineau, Joelle, Geoff Gordon, and Sebastian Thrun (2003). "Point-Based Value Iteration: An Anytime Algorithm for POMDPs." In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI'03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 1025–1030.

Platt, Robert, Russell Louis Tedrake, Leslie P. Kaelbling, and Tomas Lozano-Perez (2010). "Belief Space Planning Assuming Maximum Likelihood Observations." In: *Proceedings of the Robotics: Science and Systems Conference (RSS)*.

Prentice, Samuel and Nicholas Roy (2009). "The Belief Roadmap: Efficient Planning in Linear POMDPs by Factoring the Covariance." In: *Proceedings of the 12th International Symposium of Robotics Research (ISRR)*.

Pritzel, Alexander, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell (2017). *Neural Episodic Control*. DOI: 10.48550/ARXIV.1703.01988. URL: https://arxiv.org/abs/1703.01988.

Qi, William, Ravi Teja Mullapudi, Saurabh Gupta, and Deva Ramanan (2020). *Learning to Move with Affordance Maps*. arXiv: 2001.02364 [cs.RO].

Rafieisakhaei, Mohammadhussein, Suman Chakravorty, and P. R. Kumar (2017). "T-LQG: Closed-loop belief space planning via trajectory-optimized LQG." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 649–656. DOI: 10.1109/ICRA.2017.7989080.

Rahman, Shatil and Steven L. Waslander (2020). "Uncertainty-Constrained Differential Dynamic Programming in Belief Space for Vision Based Robots." In: *CoRR* abs/2012.00218. arXiv: 2012.00218. URL: https://arxiv.org/abs/2012.00218.

Reiser, Christian, Songyou Peng, Yiyi Liao, and Andreas Geiger (2021). *KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs*. arXiv: 2103.13744 [cs.CV].

Rolnick, David, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne (2018). *Experience Replay for Continual Learning*. DOI: 10.48550/ARXIV.1811.11682. URL: https://arxiv.org/abs/1811.11682.

Roy, Nicholas (2003). "Finding Approximate POMDP solutions Through Belief Compression." PhD thesis. Pittsburgh, PA: Carnegie Mellon University.

Roy, Nicholas, Wolfram Burgard, Dieter Fox, and Sebastian Thrun (1999). "Coastal navigation-mobile robot navigation with uncertainty in dynamic environments." In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)* 1, 35–40 vol.1.

Roy, Nicholas and Sebastian Thrun (1999). "Coastal Navigation with Mobile Robots." In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press. URL: https://proceedings.neurips.cc/paper/1999/file/df9028fcb6b065e000ffe8a4f03eeb38-Paper.pdf.

Rubinstein, Reuven Y. (1997). "Optimization of computer simulation models with rare events." In: *European Journal of Operational Research* 99.1, pp. 89–112. ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(96)00385-2. URL: https://www.sciencedirect.com/science/article/pii/S0377221796003852.

Rublee, Ethan, Vincent Rabaud, Kurt Konolige, and Gary Bradski (2011). "ORB: An efficient alternative to SIFT or SURF." In: *2011 International Conference on Computer Vision*, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

Savinov, Nikolay, Alexey Dosovitskiy, and Vladlen Koltun (2018). *Semi-parametric Topological Memory for Navigation*. DOI: 10.48550/ARXIV.1803.00653. URL: https://arxiv.org/abs/1803.00653.

Sethian, J A (1996). "A fast marching level set method for monotonically advancing fronts." In: *Proceedings of the National Academy of Sciences* 93.4, pp. 1591–1595. DOI: 10.1073/pnas.93.4.1591. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.93.4.1591. URL: https://www.pnas.org/doi/abs/10.1073/pnas.93.4.1591.

Silver, David and Joel Veness (2010). "Monte-Carlo Planning in Large POMDPs." In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2010/file/edfbe1afcf9246bb0d40eb4d8027d90f-Paper.pdf.

Sitzmann, Vincent, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein (2020). *Implicit Neural Representations with Periodic Activation Functions*. DOI: 10.48550/ARXIV.2006.09661. URL: https://arxiv.org/abs/2006.09661.

Sondik, Edward (Jan. 1971). *The optimal control of partially observable decision processes*.

Srinivas, Aravind, Michael Laskin, and Pieter Abbeel (2020). *CURL: Contrastive Unsupervised Representations for Reinforcement Learning*. arXiv: 2004.04136 [cs.LG].

Steinbrücker, Frank, Jürgen Sturm, and Daniel Cremers (2011). "Real-time visual odometry from dense RGB-D images." In: *2011 IEEE international conference on computer vision workshops (ICCV Workshops)*. IEEE, pp. 719–722.

Sucar, Edgar, Shikun Liu, Joseph Ortiz, and Andrew J. Davison (2021). *iMAP: Implicit Mapping and Positioning in Real-Time*. DOI: 10.48550/ARXIV.2103.12352. URL: https://arxiv.org/abs/2103.12352.

Sunberg, Zachary N., Christopher J. Ho, and Mykel J. Kochenderfer (2017). "The value of inferring the internal state of traffic participants for autonomous freeway driving." In: *2017 American Control Conference (ACC)*, pp. 3004–3010. DOI: 10.23919/ACC.2017.7963408.

Sunberg, Zachary and Mykel J. Kochenderfer (2017). "POMCPOW: An online algorithm for POMDPs with continuous state, action, and observation spaces." In: *CoRR* abs/1709.06196. arXiv: 1709.06196. URL: http://arxiv.org/abs/1709.06196.

Sutton, R. S. (1988). "Learning to predict by the methods of temporal differences." In: *Machine Learning* 3, pp. 9–44.

Tamar, Aviv, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel (2016). "Value Iteration Networks." In: DOI: 10.48550/ARXIV.1602.02867. URL: https://arxiv.org/abs/1602.02867.

Thrun, Sebastian (1999). "Monte Carlo POMDPs." In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS'99. Denver, CO: MIT Press, 1064–1070.

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press. ISBN: 0262201623.

Todorov, Emanuel (2005). "Stochastic Optimal Control and Estimation Methods Adapted to the Noise Characteristics of the Sensorimotor System." In: *Neural Comput.* 17.5, 1084–1108. ISSN: 0899-7667. DOI: 10.1162/0899766053491887. URL: https://doi.org/10.1162/0899766053491887.

Wang, Huan, Jian Ren, Zeng Huang, Kyle Olszewski, Menglei Chai, Yun Fu, and Sergey Tulyakov (2022). "R2L: Distilling Neural Ra-

diance Field to Neural Light Field for Efficient Novel View Synthesis." In: *ECCV*.

Wang, Zirui, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu (2021). *NeRF–: Neural Radiance Fields Without Known Camera Parameters*. DOI: 10.48550/ARXIV.2102.07064. URL: https://arxiv.org/abs/2102.07064.

Williams, Grady, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou (2016). "Aggressive driving with model predictive path integral control." In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277.

Wu, Chenyang, Guoyu Yang, Zongzhang Zhang, Yang Yu, Dong Li, Wulong Liu, and Jianye Hao (2021). "Adaptive Online Packing-guided Search for POMDPs." In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., pp. 28419–28430. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/ef41d488755367316f04fc0e0e9dc9fc-Paper.pdf.

Wu, Philipp, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel (2022). *DayDreamer: World Models for Physical Robot Learning*. arXiv: 2206.14176 [cs.RO].

Wydmuch, Marek, Michał Kempka, and Wojciech Jaśkowski (2018). "ViZDoom Competitions: Playing Doom from Pixels." In: *IEEE Transactions on Games*.

Yen-Chen, Lin, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin (2020). *INeRF: Inverting Neural Radiance Fields for Pose Estimation*. DOI: 10.48550/ARXIV.2012.05877. URL: https://arxiv.org/abs/2012.05877.

Yu, Alex, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa (2021). "PlenOctrees for Real-time Rendering of Neural Radiance Fields." In: *arXiv*.

Zhang, N. L. and W. Zhang (2001). "Speeding Up the Convergence of Value Iteration in Partially Observable Markov Decision Processes." In: *Journal of Artificial Intelligence Research* 14, pp. 29–51. DOI: 10.1613/jair.761. URL: https://doi.org/10.1613%2Fjair.761.

Zhao, Tony Z., Anusha Nagabandi, Kate Rakelly, Chelsea Finn, and Sergey Levine (2021). *MELD: Meta-Reinforcement Learning from Images via Latent State Models*. arXiv: 2010.13957 [cs.LG].

Zheng, Dongliang, Jack Ridderhof, Panagiotis Tsiotras, and Ali-akbar Agha-mohammadi (2021). "Belief Space Planning: A Covariance Steering Approach." In: *CoRR* abs/2105.11092. arXiv: 2105.11092. URL: https://arxiv.org/abs/2105.11092.