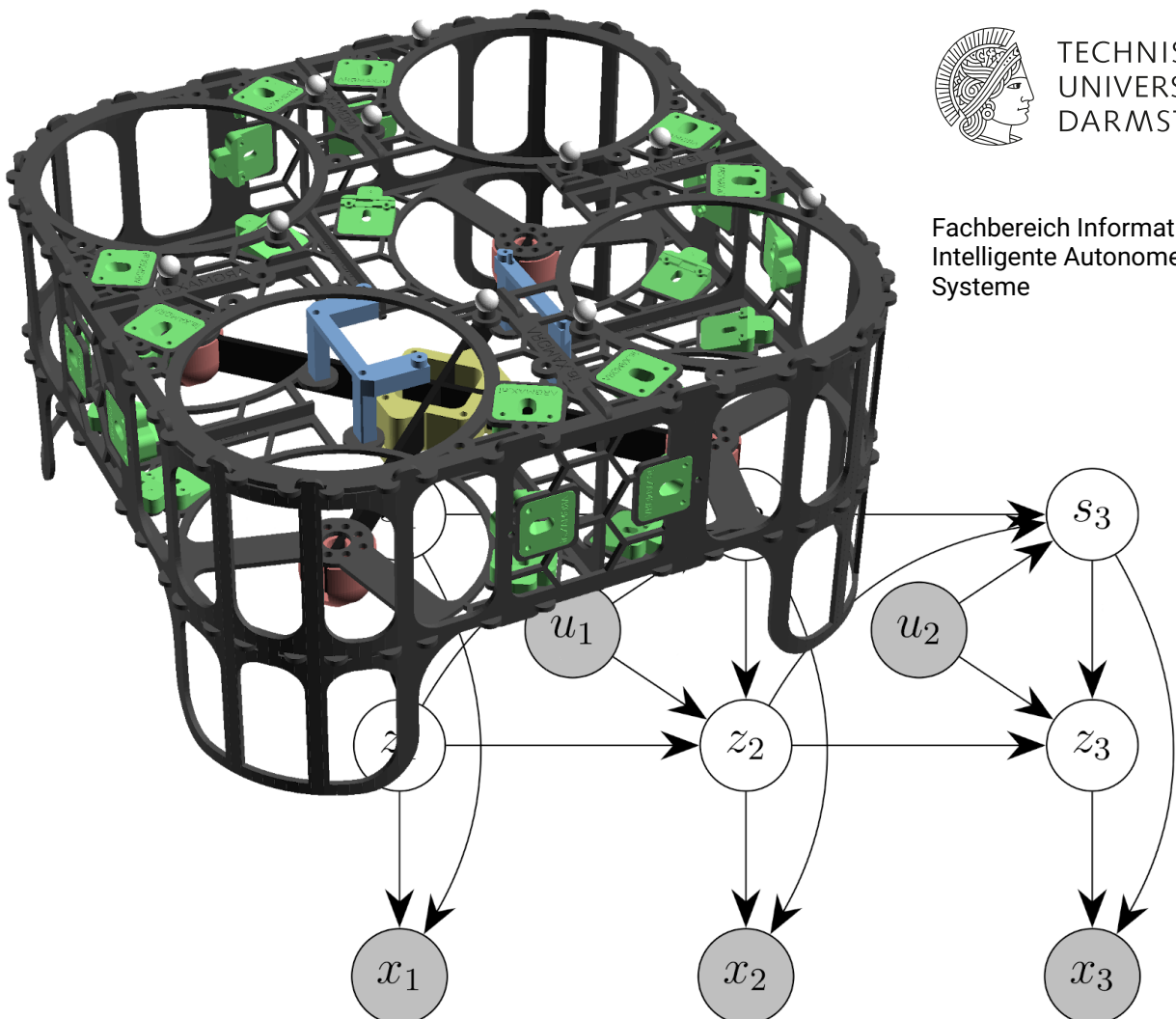


# Latent State-Space Models for Control

## Regelung in latenten Zustandsräumen

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)  
genehmigte Dissertation von Philip Becker-Ehmck aus München  
Tag der Einreichung: 28.07.2022, Tag der Prüfung: 26.09.2022

1. Gutachten: Prof. Jan Peters, Ph.D.
2. Gutachten: Prof. Dr. Marco Hutter  
Darmstadt – D 17



Latent State-Space Models for Control  
Regelung in latenten Zustandsräumen

genehmigte Dissertation von Philip Becker-Ehmck

1. Gutachten: Prof. Jan Peters, Ph.D.
2. Gutachten: Prof. Dr. Marco Hutter

Tag der Einreichung: 28.07.2022

Tag der Prüfung: 26.09.2022

Darmstadt – D 17

Bitte zitieren Sie dieses Dokument als:

URN: urn:nbn:de:tuda-tuprints-urn:nbn:de:tuda-tuprints-224895

URL: <http://tuprints.ulb.tu-darmstadt.de/22489>

Dieses Dokument wird bereitgestellt von tuprints,  
E-Publishing-Service der TU Darmstadt  
<http://tuprints.ulb.tu-darmstadt.de>  
[tuprints@ulb.tu-darmstadt.de](mailto:tuprints@ulb.tu-darmstadt.de)

Die Veröffentlichung steht unter folgender Creative Commons Lizenz:  
Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International  
<https://creativecommons.org/licenses/by-sa/4.0/>



---

## Erklärungen laut Promotionsordnung

### §8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

### §8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

### §9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

### §9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

München, 28.07.2022

---

P. Becker-Ehmck

---



---

# Abstract

---

Learning to control robots without human supervision and prolonged engineering effort has been a long-term dream in the intersection of machine learning and robotics. If successful, it would enable many novel applications from soft robotics over human-robot interaction to quick adaptation to unseen tasks or robotic setups. A key driving force behind this dream are inherent limitations of classical control algorithms that restrict applicability to low-dimensional and engineered state-spaces, prohibiting the use of high-dimensional sensors such as cameras or touchpads. As an alternative to classical control methods, Reinforcement Learning (RL) presumes no prior knowledge of a robot's dynamics and paired with deep learning opens the door to use high-dimensional sensory information of any kind. Yet, reinforcement learning has only achieved limited impact on real-time robot control due to its high demand for real-world interactions (among other reasons). Model-based approaches promise to be much more data efficient, but present the challenge of engineering accurate simulators. As building a simulator comes with many of the same challenges as designing a controller, using engineered simulators is not a satisfactory solution for the generic goal of "*learning to control*"; most of the engineering work would still have to be done to build the simulator. Instead, learning such a model, in particular a Latent State-Space Model (LSSM), promises to resolve us from engineering a simulator while still reaping the benefits of having one. A learned latent space can compactly represent high-dimensional sensor information and store all relevant information for prediction and control.

In this thesis, we show how to perform system identification of complex and nonlinear systems based on high-dimensional observations purely from raw sensory data. Despite their complexity, such systems can often be approximated well by a set of linear dynamical systems if broken into appropriate subsequences. This mechanism not only helps us find good approximations of dynamics, but also gives us deeper insight into the underlying system. Combining Bayesian inference, Variational Autoencoders and Concrete relaxations, we show how to learn a richer and more meaningful state-space, for example by encoding joint constraints or collisions with walls in a maze, from partial and high-dimensional observations. In a setting with time-varying dynamics, we show how our inference method for continuous switching variables can infer changing but unobserved physical properties

---

---

that govern the dynamics of a system, such as masses or link lengths in robotic simulations. This inference happens online in our learned filter without retraining or fine-tuning of model parameters. Quantitatively, we find that such representations translate into a gain of accuracy of learned dynamics showcased on various simulated tasks and that they promise to be helpful for policy optimization.

Building on this work, we show how this LSSM can be used to learn a probabilistic model of real-world robot dynamics, such as from a self-built drone and a 7 degree of freedom (DoF) robot arm. No prior knowledge of the flight dynamics or kinematics is assumed. On top, we propose a novel Model-Based Reinforcement Learning (MBRL) method where both a parameterized policy and value function are optimized entirely by propagating stochastic analytic gradients through generated latent trajectories. Our learned thrust-attitude controller can fly a drone to a randomly placed marker in an enclosed environment, and steer a joint velocity controlled robot arm to random end effector positions in Cartesian space. This can be achieved with less than an hour of interactions on the real system. The control policy is learned entirely in the learned simulator and can be applied without modification or fine-tuning to the real system.

Last, we propose a novel exploration criterion for the development of autonomous agents: Empowerment Gain. Different to other exploration criteria, this approach ties together an agent's entire perception-control loop and its current capabilities to act. Perspectively, this method will help us learn models of the world that are actually relevant to realizing an agent's influence in the world. As a key insight, our learned models do not actually have to be perfect simulators of the entire world and all of its processes, rather they need to convey the information necessary to enable an agent to interact with the world around him. We show how this criterion compares to, and in some ways incorporates, other intrinsic motivations such as novelty seeking, surprise minimization and learning progress. While our method still ensures exploration of the entire space, it prefers regions with greater potential for realizing an agent's influence in the world.

In conclusion, we give answers to three major questions: (1) how do we learn a LSSM from raw sensory data, (2) how do we use it for control and (3) what parts of the world do we need to explore and model in the first place. While the last part remains in a theoretical and conceptual stage, we demonstrate the first two on two different real-world robotic platforms. We focused on proposing general purpose methods that are as broadly applicable as they can be, but are still successful in a real-world setting.

---

# Zusammenfassung

---

Das Lernen der Regelung von Robotern ohne menschliche Betreuung und zeitraubende spezifische Entwicklungsarbeit ist ein langwährender Traum in der Schnittstelle von Machine Learning und Robotik. Falls von Erfolg gekrönt, würde dies viele neue Anwendungsfälle eröffnen, wie zum Beispiel Soft Robotics, Mensch-Maschine-Interaktionen oder die schnelle Anpassung an neuartige Aufgaben oder neue robotische Systeme. Getrieben ist dieser Traum von unüberwindbaren Limitierungen klassischer Regelungsverfahren, wie der Beschränkung auf niedrigdimensionale Zustandsräume. Als Alternative zu klassischen Regelungsverfahren benötigt Reinforcement Learning kein Vorwissen über etwaige Roboterdynamiken und zusammen mit Deep Learning Methoden öffnet es die Tür zur Verwendung hochdimensionaler Sensorinformation wie Kameras oder Behrührungssensoren. Trotz dieser Versprechungen hat Reinforcement Learning (RL) bisher nur begrenzten Erfolg in der Regelung von echten Robotern aufgrund seiner Dateneffizienz (unter anderem). Modellbasierte Methoden versprechen hier Abhilfe zu schaffen, setzen aber die Erstellungen eines präzisen Simulators voraus. Eben diese kommt allerdings bereits mit vielen der Herausforderungen, die auch klassische Regelungsverfahren unterlegen sind, und daher sind sie keine ideale Option, wenn man tatsächlich universelle Methoden zur Regelung entwickeln möchte. Stattdessen erscheint das Lernen eines Solchen die eigentliche Lösung. Ein gelernter Zustandsraum kann hochdimensionale Sensoren kompakt darstellen, während er immernoch alle Informationen beinhaltet, die für die Vorhersage und Regelung von Bedeutung sind.

In dieser Arbeit zeigen wir, wie man Systemidentifikation von komplexen und nicht-linearen Systemen basierend auf hochdimensionalen Sensordaten durch allgemeine Lernverfahren bewerkstelligen kann. Trotz ihrer Komplexität sind solche Systeme oft durch eine Anzahl an linearen Systemen gut zu approximieren – falls man den Zustandsraum entsprechend aufteilen kann. Dieser Mechanismus erlaubt uns nicht nur eine gute Approximation der Dynamik zu finden, er gibt uns auch Einsicht in das zugrundeliegende System. Mittels Variational Inference, Deep Learning und einer kontinuierlichen Approximation diskreter Zufallsvariablen, zeigen wir, wie man eine interpretierbare Zustandsraumdarstellung lernt, die Konzepte wie Geschwindigkeit und Beschränkungen durch Wände aus hochdimensionalen und unvollständigen Beobachtungen explizit extrahiert. In Szenarien

---

---

mit zeitvarianten Systemdynamiken zeigen wir, wie unsere Methode die der Veränderung zugrundeliegenden, aber unbeobachteten, Systemvariablen, wie zum Beispiel Massen oder Längen in Robotiksimulationen, automatisch inferieren kann. Diese Inferenz geschieht dabei als Teil unseres gelernten Filters und ohne erneutes Anpassen der Modellparameter. Diese gelernte Zustandsraumdarstellung führt nicht nur zu einer verbesserten Vorhersagefähigkeit, sondern stellt sich auch als gute Informationsquelle für eine gelernte Policy heraus.

Aufbauend auf dieser Arbeit zeigen wir, wie man diese Methode verwenden kann, um echte Roboterdynamiken zu approximieren – wie die einer selbst gebauten Drohne und die eines Roboterarms. Kein Vorwissen über die Flugdynamik oder die Kinematik wird vorausgesetzt. Darauf aufgesetzt zeigen wir, wie man eine parametrisierte Policy und Value Function damit optimieren kann; ganz allein basierend auf simulierten Erfahrungen und deren Gradienten. Solch eine gelernte Policy kann eine Drohne zu einem bestimmten Ort fliegen oder den Endeffektor eines Roboterarms an einen zufälligen kartesischen Ort durch direkte Kontrolle der Gelenke fahren. Das Ganze wird erreicht mit unter einer Stunde an Erfahrungen in der echten Welt. Die gelernte Policy ist ausschließlich im gelernten Simulator optimiert und kann auf die echte Welt ohne weitere Veränderungen erfolgreich übertragen werden.

Anschließend schlagen wir ein neuartiges Kriterium zur Exploration für autonome Agenten vor: Empowerment Gain. Anders als andere Explorationskriterien umfasst diese Methode die komplette Wahrnehmungs-Handlungs-Schleife, sowie die aktuellen Handlungsfähigkeiten eines Agenten. Perspektivisch wird diese Methode uns helfen Zustandsraumdarstellungen zu lernen, die tatsächlich relevant für die Regelung von Systemen sind. Eine Schlüsselbeobachtung dieser Arbeit ist, dass wir kein Modell der Welt in ihrer Gesamtheit lernen müssen, wir benötigen lediglich ein Modell, dass uns erfolgreiche Interaktion mit der Umwelt erlaubt. Wir vergleichen dieses Kriterium zu bekannten Kriterien wie Novelty Seeking, Surprise Maximization oder Learning Progress und zeigen auf, wie es jene, in einer gewissen Weise, beinhaltet. Während unsere Methode weiterhin garantiert den gesamten Beobachtungsraum zu erforschen, bevorzugt sie jene Bereiche, in der ein Agent mehr potenziellen Einfluss auf die Welt entwickeln kann.

Schlussendlich geben wir in dieser Arbeit eine Antwort zu drei großen Fragen: (1) wie lerne ich Zustandsraumdarstellungen basierend auf beliebigen Sensordaten, (2) wie verwende ich diese Darstellung zur Regelung und (3) welche Teile der Welt muss ich eigentlich explorieren und modellieren. Während der letzte Teil in der konzeptuellen und theoretischen Phase verweilt, demonstrieren wir die ersten beiden Antworten in der echten Welt auf zwei ganz unterschiedlichen Systemen. Obwohl wir eine erfolgreiche Anwendung auf zwei konkreten Systemen beleuchten, sind die entwickelten Methoden sehr allgemein gehalten und deutlich genereller anwendbar.



---

## Acknowledgments

---

I would like to thank my advisor Jan Peters for his openness in letting me pursue my own research directions and his valuable general guidance. Similarly, I would like to thank my supervisor Patrick van der Smagt for the opportunity to conduct this research at the Volkswagen Machine Learning Research Lab and his ongoing support and trust. Special gratitude is reserved for my friend and colleague Maximilian Karl with whom I have collaborated closely on many occasions during this thesis. I also want to thank all my other colleagues at Volkswagen, in particular Maximilian Soelch, Djalel Benbouzid, Alexandros Paraschos, Botond Cseke, Richard Kurle, Felix Frank, Alexej Klushyn, Nutan Chen and Justin Bayer, for being great colleagues and for many thought-provoking discussions. Lastly, special thanks goes to my family who supported and endured me during this exciting and challenging time.



---

# Contents

---

<b>Abstract</b>	<b>i</b>
<b>Zusammenfassung</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>v</b>
<b>Abbreviations and Symbols</b>	<b>xi</b>
<b>1. Introduction</b>	<b>1</b>
1.1. Objective and Approach . . . . .	3
1.2. Contributions . . . . .	4
1.2.1. Switching Deep Variational Bayes Filter . . . . .	4
1.2.2. Learning to Control Real Robots via Deep Model-Based Reinforcement Learning . . . . .	4
1.2.3. Exploration via Empowerment Gain: Combining Novelty, Surprise and Learning Progress . . . . .	5
1.3. Thesis Outline . . . . .	6
<b>2. Switching Deep Variational Bayes Filter</b>	<b>9</b>
2.1. Background . . . . .	10
2.1.1. Switching Linear Dynamical Systems . . . . .	10
2.1.2. Stochastic Gradient Variational Bayes . . . . .	11
2.1.3. The Concrete Distribution . . . . .	12
2.2. Related Work . . . . .	12
2.3. Proposed Approach . . . . .	14
2.3.1. Generative Model . . . . .	14
2.3.2. Inference . . . . .	15
2.3.3. Training . . . . .	19
2.4. Experiments . . . . .	20
2.4.1. Multiple Bouncing Balls in a Maze . . . . .	21
2.4.2. Reacher . . . . .	23

---

2.4.3. Ball in a Box on Image Data . . . . .	23
2.4.4. FitzHugh-Nagumo . . . . .	24
2.4.5. Susceptibility to the Scale of Temporal Discretization . . . . .	25
2.4.6. Time-Varying Dynamics . . . . .	27
2.5. Discussion . . . . .	32
2.6. Conclusion . . . . .	32
<b>3. Learning to Control Real Robots via Deep Model-Based Reinforcement Learning</b>	<b>35</b>
3.1. Background . . . . .	37
3.1.1. Backpropagation through Stochastic Variables . . . . .	37
3.1.2. Reinforcement Learning . . . . .	38
3.2. Variational Latent Dynamics . . . . .	39
3.3. Learning a Controller . . . . .	40
3.3.1. Value Estimation . . . . .	41
3.3.2. Policy . . . . .	42
3.4. Experimental Platform: Our Drone . . . . .	43
3.4.1. High and Low-Level Controller . . . . .	44
3.4.2. Environment . . . . .	45
3.5. Drone Experiments . . . . .	47
3.5.1. Setup . . . . .	47
3.5.2. Fly to Marker . . . . .	50
3.6. Robot Arm Experiments . . . . .	53
3.6.1. Environment . . . . .	55
3.6.2. Setup . . . . .	56
3.6.3. Results . . . . .	57
3.7. Related Work . . . . .	58
3.8. Conclusion . . . . .	60
<b>4. Exploration via Empowerment Gain: Combining Novelty, Surprise and Learning Progress</b>	<b>63</b>
4.1. Background . . . . .	65
4.1.1. Empowerment . . . . .	65
4.1.2. Information Gain . . . . .	66
4.2. Empowerment Gain (EG) . . . . .	67
4.2.1. Model-Free Empowerment Gain . . . . .	68
4.2.2. Model-Based Empowerment Gain . . . . .	69
4.3. Experiments . . . . .	70
4.3.1. Deterministic Environments . . . . .	72

---

4.3.2. Prison States . . . . .	72
4.3.3. Actions of Varying Reliability . . . . .	75
4.3.4. State-Dependent Action Noise . . . . .	76
4.3.5. Redundancies in the Action Space . . . . .	76
4.4. Related Work . . . . .	78
4.5. Discussion & Limitations . . . . .	80
4.6. Conclusion . . . . .	81
<b>5. Conclusion</b>	<b>83</b>
5.1. Outlook . . . . .	83
5.1.1. Robustness to Sensor Noise and Partial Observability . . . . .	83
5.1.2. Universally Applicable Reward Functions . . . . .	84
5.1.3. Learned Low-Level Control . . . . .	84
5.1.4. What Does Really Matter in (Model-Based) Reinforcement Learning? . . . . .	85
5.1.5. Empowered Exploration . . . . .	85
5.1.6. A Universal Agent . . . . .	86
<b>A. Appendix</b>	<b>87</b>
A.1. Appendix for Chapter 2 . . . . .	87
A.1.1. Lower Bound Derivation . . . . .	87
A.1.2. Experimental Setup . . . . .	89
A.1.3. On Scaling Issues of Switching Linear Dynamical Systems . . . . .	92
A.2. Appendix for Chapter 3 . . . . .	93
A.2.1. Implementation & Training . . . . .	93
A.2.2. Onboard Computational Cost . . . . .	93
A.3. Appendix for Chapter 4 . . . . .	95
A.3.1. Derivations . . . . .	95
A.3.2. Model-Based Empowerment Gain . . . . .	97
A.3.3. Experiments . . . . .	100
A.3.4. An Alternative Empowerment Gain Formulation . . . . .	100
A.3.5. Empowerment in Partially Observable Environments . . . . .	101
<b>Bibliography</b>	<b>101</b>
<b>List of Algorithms</b>	<b>119</b>
<b>List of Figures</b>	<b>121</b>
<b>List of Tables</b>	<b>123</b>

---

**Publications**

**125**

**Curriculum Vitae**

**127**

---

# Abbreviations and Symbols

---

## Abbreviations

<b>Notation</b>	<b>Description</b>
DoF	degree of freedom
DVBF	Deep Variational Bayes Filter
EG	Empowerment Gain
ELBO	Evidence Lower Bound
IG	Information Gain
IMU	Inertial Measurement Unit
KL	Kullback-Leibler
KVAE	Kalman Variational Autoencoder
LSSM	Latent State-Space Model
MBAC	Model-Based Actor-Critic
MBRL	Model-Based Reinforcement Learning
MDP	Markov Decision Process
MPC	Model Predictive Control
MSE	Mean Squared Error
ORC	Open Robot Communication
PID	Proportional–Integral–Derivative

---

RL	Reinforcement Learning
RNN	Recurrent Neural Network
RSSM	Recurrent State-Space Model
SAC	Soft Actor Critic
SLAC	Stochastic Latent Actor Critic
SLDS	Switching Linear Dynamical System
VAE	Variational Autoencoder
VI	Variational Inference

## Symbols

This table includes the basic notation and important symbols used throughout this thesis. If a symbol is overloaded across chapters, the correct meaning should be apparent from the context.

Notation	Description
$u_t$	A control input $u$ to a system at time $t$ .
$\mathcal{E}(s)$	Empowerment of state $s$ .
$\mathbb{H}(X)$	Shannon entropy of random variable $X$ .
$\mathbb{E}_{x \sim p}[f(x)]$	Expectation of $f$ over probability density $p$ .
$\mathcal{I}(X, Y)$	Mutual information between to random variables $X$ and $Y$ .
$\mathcal{IG}(\theta, d)$	Information gain about parameters $\theta$ after observing data $d$ .
$\mathcal{N}(\mu, \sigma)$	Normal or Gaussian distribution with mean $\mu$ and variance $\sigma$ .
$x_t$	An observation $x$ of the environment at time $t$ .
$\theta$	Vector of parameters from a probability distribution.
$\pi_\theta(a_t   s_t)$	Policy $\pi$ parameterized by parameters $\theta$ computing an action $a_t$ based on state $s_t$ .
$r_t$	A scalar reward signal $r$ at time $t$ .



---

$x_{1:T}$	A sequence of a variable (in this case observations $x$ ), $x_{1:T} = (x_1, x_2, \dots, x_{T-1}, x_T)$
$\bar{x}$	The overline is an alternative way we denote a sequence of a variable (in this case observations $x$ ) if $x_{1:T}$ would be too verbose.
$z_t$	An internal or latent state $z$ .
$Q(s, a)$	State-action value function $Q$ of state $s$ and action $a$ .
$V(s)$	State value function $V$ of state $s$ .
$\tau$	A trajectory consisting of observations $x$ and control inputs $u$ , $\tau = (x_1, u_1, x_2, u_2, \dots, u_{T-1}, x_T)$ .
$q(x)$	Probability distributions denoted by $q(x)$ are variational approximations of the true distribution denoted by $p(x)$ .



---

# 1. Introduction

---

Classical control methods rely on carefully designed state-spaces to enable precise and robust control policies. They shine in setups where robots are stiff, interactions are limited and observations of the state are low-dimensional and easily transformed into the underlying state-space. As an alternative to classical control methods, deep RL makes far fewer presumptions about what has to be known a priori and has been shown, when paired with powerful function approximators such as neural networks, to work directly on high-dimensional observations. It has enjoyed remarkable success in various applications such as in Atari (Mnih et al., 2015), Go (Silver et al., 2017) or StarCraft II (Vinyals et al., 2019), however deep RL has only achieved limited impact on real-time robot control. There are various reasons for this (Dulac-Arnold et al., 2020), most often mentioned is the high demand for real-world interactions, but other reasons such as delays, noise, partial observability and continuous and high-dimensional state and action spaces are other important factors. Poor sample efficiency is seemingly the price to pay for a general purpose learning method. For robotics, this cost becomes prohibitive as interactions are not easily scaled up like in simulations or games. Fortunately and possibly due to some of these reasons, model-based reinforcement learning approaches have become more and more prominent (Schrittwieser et al., 2020; Hafner et al., 2020; Lee et al., 2020a), matching their model-free counterparts in absolute performance while being more sample efficient. Having an explicit world model comes with numerous potential advantages. It allows for generation of synthetic data, enables planning of trajectories, reasoning about outcomes and facilitates encoding of high-dimensional observations into a low-dimensional state-space. Given these potential advantages, we hypothesize that compressing high-dimensional and often highly redundant information into a small *latent* representation capturing relevant information for control is a key ingredient for scaling these approaches to real-world applications.

Learning representations can be approached in numerous ways, in this thesis we particularly focus on variational LSSMs (Krishnan et al., 2015). These are probabilistic models that can learn both a latent representation and latent dynamics model from raw and high-dimensional data such as video streams or tactile data. Thus, they not only give us a low-dimensional representation to work with, but they represent a fully fledged simulator

---

---

that can even be computationally cheaper to execute than their engineered counterparts. They can deal with aleatoric and epistemic uncertainty in a principled manner, although transferring this theory to practice remains an ongoing challenge. Importantly, these learned simulators allow us to generate more data without actually operating a robot, potentially cutting down the need for real-world interactions. While these learned simulators may be black boxes, they can be designed to be differentiable, enabling us to use first (or possibly higher) order information of the dynamics for optimizing a controller. Furthermore, they can also be used as a filter for online state estimation, providing a belief over the current state which in turn may be used to condition a policy.

However, while theoretically very promising, there are huge practical problems that prohibited this approach from being widely successful so far. Often named first is the questionable accuracy of learned simulators paired with the inherit problem of accumulating prediction errors over time. Much more popular have thus been model-free RL methods that are learned in an engineered simulation first and then later on transferred to the real robot. Here, huge effort is spent in ensuring the simulator adequately corresponds to the real dynamics of the one specific robot one would like to operate. Still, even then the transfer step from simulation to the real world may still be accompanied by quite a few adjustments to account for simulation inaccuracies. Thus, for robotic simulations to be useful for learning a controller for the real world, a technique called domain randomization is often employed. This technique varies various parameters of the simulation to cover possible inaccuracies, in the end producing a robust but potentially suboptimal policy that hopefully works in all of these varied settings – including the real world. Despite these drawbacks, there have been quite a few successes using this approach, e. g. Peng et al. (2018), Andrychowicz et al. (2020), Tan et al. (2018), Chebotar et al. (2019), Lin et al. (2019), Hwangbo et al. (2019) and Lee et al. (2020b) representing the current state of the art.

However, there are further issues with this approach that limits its general applicability and its potential when combined with general learning methods even if the manpower and expertise to build an accurate simulator is available. First, they limit us in the robots that we use. Dynamics of stiff and rigid robots are relatively simple to write down, while soft robotics are for this reason typically excluded. The simulation of interactions is intrinsically difficult and computationally expensive. Simulating high-dimensional sensors such as video streams – while possible – typically is far away from being realistic and the transfer from simulation to the real world remains an open challenge. So, while there have been great successes, these are some essential arguments to look for an alternative approach that is possibly more in line with the vision of generic methods for learning to control. With the rise of deep learning methods over the last decade in various domains, one wonders why one can not also learn world models directly from raw sensory data.

---

---

While learned models may also be inaccurate, they are at least fitted based on real-world data and the methodologies are theoretically applicable to systems where there exists no analytical derivation of the system dynamics. Their inaccuracies and uncertainties can principally be estimated and in a Bayesian setting a posterior over possible dynamics could even replace often crude ways of domain randomization.

In this thesis, we will try to convince the reader that learned simulators, in particular LSSMs, are a promising option that can be used to learn controllers in real-world scenarios. While huge obstacles have to be addressed, their promise of broad applicability and the freedom to incorporate any sensory information are just too big to ignore.

## 1.1. Objective and Approach

The principal objective of this thesis is to move closer towards learned and universally applicable control methods using latent world models. We address three major questions stemming from this objective:

1. How should such a world model look like and how can it be learned?
2. How can we learn a controller with it?
3. What parts of the world do we need to explore and model?

Each of these three questions is addressed in one of the main chapters of this thesis.

Resulting from this objective, some guiding principles can be derived and they were followed throughout this thesis. First of all, all developed methods strive to be as universally applicable as they can be, not only applicable beyond a specific setting, task or robot, but to any agent in any setting. While specific scenarios and robots were necessarily chosen to demonstrate a method's capabilities, no effort was spared to minimize the modifications and engineering required to make it work. The focus was on developing principled frameworks and algorithms implemented by deep learning methods that enable broad applicability. Hence, we always directly learn on minimally preprocessed sensor data. The latent representation itself, beyond being a proper state-space with its Markov property, is never provided with any prior knowledge of the system. That is not to say that such hybrid approaches are not useful or valid – as they clearly are – just that for our objective and this thesis, it was not the correct thing to do. Despite this generality, it was important to us to scale and demonstrate the methods on real robotic systems. While they may not yet be able to match state of the art of engineered or hybrid approaches in

---

---

absolute performance, the goal was to clearly demonstrate the feasibility and promise of learned latent state-space models for control.

## **1.2. Contributions**

The contributions of this thesis are separated into three main chapters. Chapter 2 introduces a novel method for learning LSSMs with Switching Linear Dynamical Systems (SLDS) by auto encoding high-dimensional data. Chapter 3 applies this method to real-world robotic settings and develops a method for learning a controller. Chapter 4 asks and attempts to answer the fundamental question of what parts of the world should be explored and modeled in a latent space by taking into account an agent’s entire perception-control loop.

### **1.2.1. Switching Deep Variational Bayes Filter**

In this work, we show how to learn LSSMs with SLDS from raw observations in an end-to-end manner using stochastic analytic gradients. Combining Bayesian inference, Variational Autoencoders and Concrete relaxations, we show how to learn a richer and more meaningful state-space, for example by encoding joint constraints or collisions with walls in a maze, from partial and high-dimensional observations. In a setting with time-varying dynamics, we show how our inference method for continuous switching variables can infer changing but unobserved physical properties that govern the dynamics of a system, such as masses or link lengths in robotic simulations. This inference happens online in our learned filter without retraining or fine-tuning of model parameters. Quantitatively, we find that such representations translate into a gain of accuracy of learned dynamics showcased on various simulated tasks and that they promise to be helpful for policy optimization. The final model may act as a filter for online state estimation which can be used for control as we demonstrate later on.

### **1.2.2. Learning to Control Real Robots via Deep Model-Based Reinforcement Learning**

In this work, by leveraging the previously introduced framework to learn a probabilistic model of drone dynamics, we achieve thrust-attitude quadrotor control via model-based reinforcement learning. Without encoding any physics knowledge into the latent state-space, we show how to learn a drone dynamics model from raw sensory data. This model is good enough to learn a controller entirely in simulation that can be executed

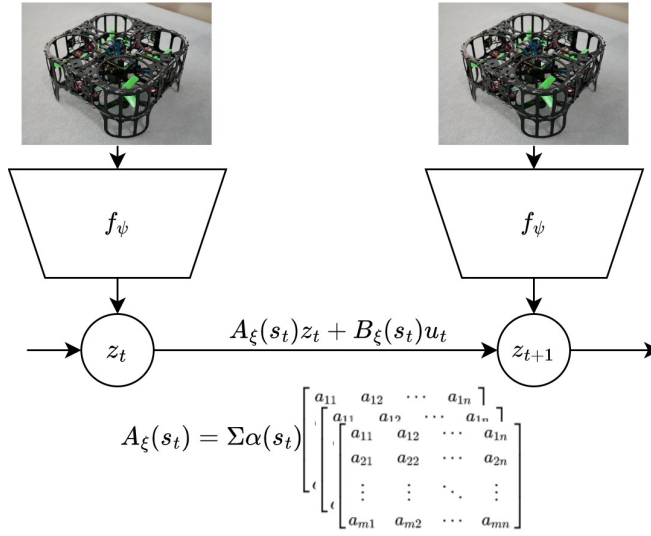


Figure 1.1.: Conceptual overview of Switching DVBF (Chapter 2) with latent state  $z_t$  and control inputs  $u_t$ . The state matrix  $A_\theta$  and control matrix  $B_\theta$  are chosen as a linear combination of a set of base matrices via a stochastic switching variable  $s_t$ .

on a real drone. The controller is capable of flying the drone to observed goal positions using only onboard sensors and computational resources. This can be achieved with less than 30 minutes of experience on a single, self-built drone. The optimization scheme is based on stochastic analytic gradients enabled by implementing model, actor and critic as differentiable function approximators. The methodology is showcased on various drone configurations, but is applicable more broadly to other robotic setups without the need for algorithmic changes. We demonstrate this by applying the same methodology to a 7 DoFs robotic arm where we use joint velocity control to reach random Cartesian end effector positions and orientations. Again, this can be achieved with limited real-world experience, requiring only up to an hour of real-world interaction data.

### 1.2.3. Exploration via Empowerment Gain: Combining Novelty, Surprise and Learning Progress

Here, we present a novel exploration criterion called *Empowerment Gain* for exploration and develop its connection to well known intrinsic motivations such as novelty seeking, surprise maximization and information gain. We show how it incorporates these methods

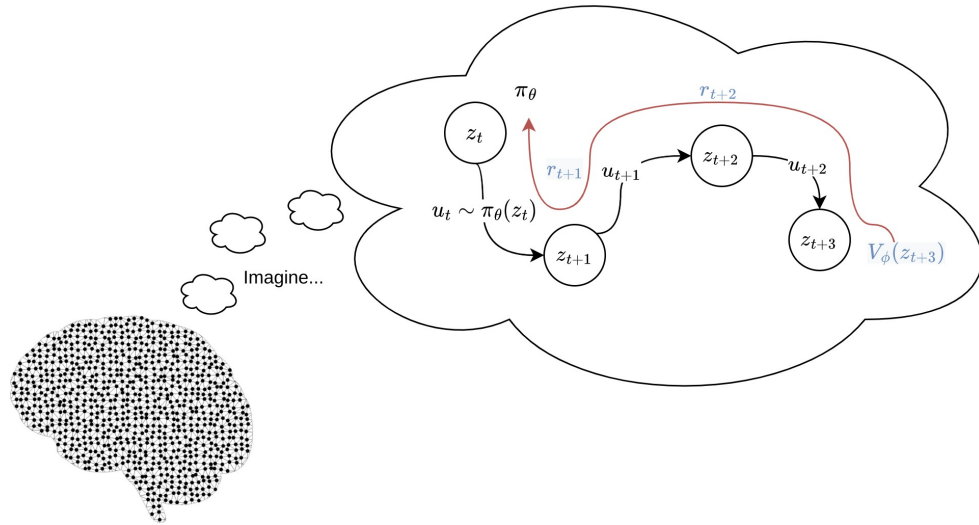


Figure 1.2.: Model-Based Actor-Critic (Chapter 3): optimization of a policy by backpropagating (gradient path indicated in red) through short imagined latent rollouts.

in a specific manner, but also demonstrate its key differences and advantages. In particular, our method steers the exploration process based on agent’s entire perception-control loop and directly encourages an agent to perform actions to recognize its capability to interact with the world. This results in both novelty seeking and surprise maximization in a particular trade-off. We demonstrate how such an exploration scheme avoids prison cells, prefers exploring actions and regions of the state-action space of lower stochasticity and prefers areas with a greater number of available options to the agent. Using both the theoretical underpinning and the experimental results, we try to convince the reader that increasing an agent’s capability to interact with the environment is the ultimate goal of task-agnostic exploration. We believe it to be vital to the development of autonomous agents as it allows us to learn world models that are useful for control.

### 1.3. Thesis Outline

Although the chapters are mostly self-contained, in particular chapters 2 and 3 heavily build on each other, we recommend reading in the presented order. However, each chapter is accompanied by its own appendix whose reading may be deferred to a later point.



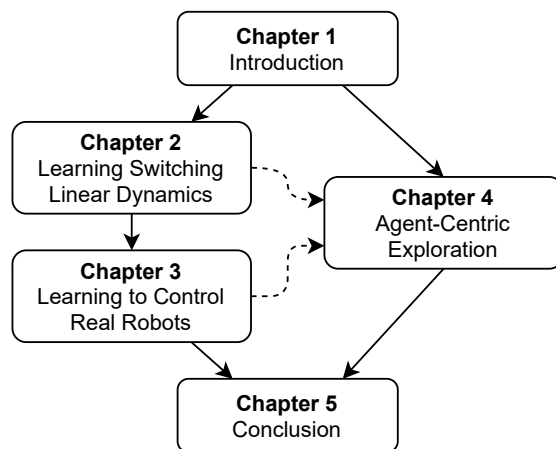


Figure 1.3.: The structure of this thesis. Chapters 2 and 3 should be read in order. Chapter 4 is self-contained, but relates to the previous chapters in various ways.


**Chapter 1** outlines this thesis. We introduce the general motivation, objective and state the contributions of this thesis.

In **Chapter 2**, we show how to learn a LSSM with SLDS from raw data. We use neural Variational Inference (VI) methods to learn this model end-to-end and showcase its performance on various simulated tasks. The resulting model is not only good for prediction, but may also act as an online filter for state estimation.

In **Chapter 3**, we show how to use the previous chapter’s methodology for control of two different robotic systems. We show how to learn a neural controller using a novel MBRL variant that backpropagates stochastic analytic gradients through learned dynamics. The entire learned system can be deployed on a self-built quadrotor, flying it to an observed goal marker, and on a 7 DoF robot arm for joint velocity control. Either system can be learned without encoding prior knowledge into the latent space and by only collect up to an hour of real-world interactions.

In **Chapter 4**, we propose a novel criterion for autonomous exploration in the absence of a concrete task. Our criterion, Empowerment Gain (EG), directly tries to improve an agent’s capability to interact with the world and incorporates well known intrinsic motivations such as novelty seeking, surprise maximization and information gain in a specific manner. We argue that this exploration mechanism is beneficial for learning latent variable models of the world that are useful for control.

Finally, we conclude in **Chapter 5** by summarizing the main contributions of our thesis. We give an outlook on promising directions for future work that either directly extend



---

this work or take it to new areas of research.

---

## 2. Switching Deep Variational Bayes Filter

---

Learning dynamics from raw data (also known as system identification) is a key component of model predictive control and model-based reinforcement learning. Problematically, environments of interest often give rise to very complex and highly nonlinear dynamics which are seemingly difficult to approximate. However, switching linear dynamical systems (SLDS) approaches claim that those environments can often be broken down into simpler units made up of areas of equal and linear dynamics (Ackerson and Fu, 1970; Chang and Athans, 1978).

Not only have those approaches demonstrated good predictive performance in various settings, which often is the sole goal of learning a system’s dynamics, they also encode useful information into so called *switching variables* which determine the dynamics of the next transition. For example, when looking at the movement of an arm, one is intuitively aware of certain restrictions of possible movements, e.g., constraints to the movement due to joint constraints or obstacles. The knowledge is present without the need to simulate; it is explicit. Exactly this kind of information will be encoded when successfully inferring switching variables. Our goal in this work will therefore entail the search for richer representations in the form of latent state space models which encode knowledge about the underlying system dynamics. In turn, we expect this to improve the accuracy of our simulation as well. Such a representation alone could then be used in a reinforcement learning approach that possibly only takes advantage of the learned latent features but not necessarily its learned dynamics.

To learn richer representations, we identify one common problem with prevalent recurrent Variational Autoencoder models (Chung et al., 2015; Fraccaro et al., 2016; Karl et al., 2017; Krishnan et al., 2015): the non-probabilistic treatment of the transition dynamics often modeled by a powerful nonlinear function approximator. From the history of the Autoencoder to the Variational Autoencoder, we know that in order to detect robust features in an unsupervised manner, probabilistic treatment of the latent space is paramount (Dai et al., 2017; Kingma et al., 2014). As our starting point, we will build on previously proposed approaches by Krishnan et al. (2017) and Karl et al. (2017). The latter already made use of locally linear dynamics, but only in a deterministic fashion. We extend their approaches by a stochastic SLDS model with structured inference and show that such

treatment is vital for learning richer representations and simulation accuracy.

Expanding on this, we show how our model and in particular the inference of these separate latent switching variables that determine the latent dynamics can be used to infer unobserved time-varying parameters governing system dynamics. In particular, we show how our switching variables can detect hidden physical properties such as masses, link lengths or friction coefficients from partial observations of the system dynamics alone. Representing multiple dynamical systems by a single model can be viewed as a kind of meta-learning where each dynamical system represents a different task. Different to popular meta-learning approaches such as MAML (Finn et al., 2017) our approach does not require a couple of gradient steps to adapt to a specific task. When using the trained filter in the end, the detection of these quantities is amortized into the inference procedure and happens online without any retraining of model parameters. This makes our filter directly applicable to adaptive control scenarios. We demonstrate this by learning a controller with SLAC (Lee et al., 2020a), a state of the art model-based reinforcement learning method, on a pendulum environment with time-varying dynamics. This gives support to our claim that our learned latent space representation is also superior for policy optimization when compared to other approaches.

## 2.1. Background

We consider discretized time-series data consisting of continuous observations  $x_t \in \mathcal{X} \subset \mathbb{R}^{n_x}$  and control inputs  $u_t \in \mathcal{U} \subset \mathbb{R}^{n_u}$  that we would like to model by corresponding latent states  $z_t \in \mathcal{Z} \subset \mathbb{R}^{n_z}$ . We will denote sequences of variables by  $x_{1:T} = (x_1, x_2, \dots, x_T)$ .

### 2.1.1. Switching Linear Dynamical Systems

A Switching Linear Dynamical Systems (SLDS) model enables us to model nonlinear time series data by splitting it into subsequences governed by linear dynamics. At each time  $t = 1, 2, \dots, T$ , a discrete switch variable  $s_t \in 1, \dots, M$  chooses of a set of  $M$  LDSs a system which is to be used to transform the continuous latent state  $z_{t-1}$  to the next time step (Barber, 2012). Formally, we define

$$\begin{aligned} z_t &= A(s_t)z_{t-1} + B(s_t)u_{t-1} + \epsilon(s_t), \\ x_t &= H(s_t)z_t + \eta(s_t) \end{aligned} \tag{2.1}$$

with  $\eta(s_t) \sim \mathcal{N}(0, R(s_t)), \quad \epsilon(s_t) \sim \mathcal{N}(0, Q(s_t)).$

Here  $A \in \mathbb{R}^{n_z \times n_z}$  is the state matrix,  $B \in \mathbb{R}^{n_z \times n_u}$  is the control matrix,  $\epsilon$  is the transition noise with covariance matrix  $Q$ ,  $\eta$  is the emission/sensor noise with covariance matrix

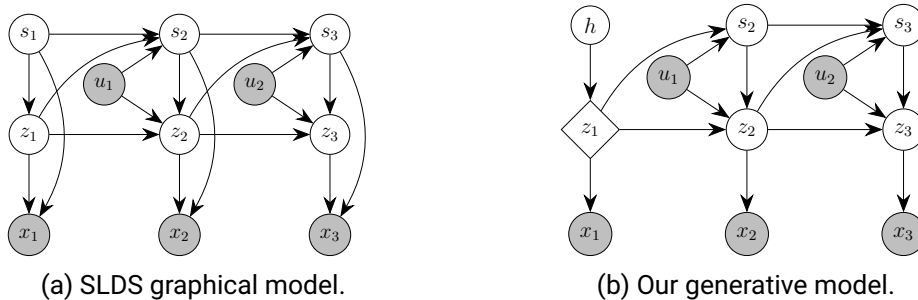


Figure 2.1.: (a)  $s_t$  denote discrete switch variables,  $z_t$  are continuous latent variables,  $x_t$  are continuous observed variables,  $u_t$  are (optional) continuous control inputs. (b) By introducing a special latent variable  $h$  used for initial state inference, we want to make explicit that the first step is treated differently from the rest of the sequence.

$R$  and the observation matrix  $H \in \mathbb{R}^{n_x \times n_z}$  defines a linear mapping from latent to observation space. These equations imply the joint distribution

$$p(x_{1:T}, z_{1:T}, s_{1:T} \mid u_{1:T}) = \prod_{t=1}^T p(x_t \mid z_t, s_t) p(z_t \mid z_{t-1}, u_{t-1}, s_t) p(s_t \mid z_{t-1}, u_{t-1}, s_{t-1})$$

with  $p(s_1 \mid z_0, u_0, s_0) = p(s_1)$  and  $p(z_1 \mid z_0, u_0, s_1) = p(z_1 \mid s_1)$  being initial state distributions. The discrete switching variables are usually assumed to evolve according to Markovian dynamics, i.e.  $\Pr(s_t \mid s_{t-1} = k) = \pi_k$ , which optionally may be conditioned on the continuous state  $z_{t-1}$ . The corresponding graphical model is shown in Figure 2.1a.

### 2.1.2. Stochastic Gradient Variational Bayes

Given the simple graphical model

$$p(x) = \int p(x, z) dz = \int p(x \mid z) p(z) dz, \quad (2.2)$$

Kingma and Welling (2014) and Rezende et al. (2014) introduced the *Variational Autoencoder* (VAE) which overcomes the intractability of posterior inference of  $p(z \mid x)$  by maximizing the evidence lower bound (ELBO) of the model log-likelihood

$$\log p(x) \geq \mathcal{L}_{\text{ELBO}}(x; \theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] - \text{KL}(q_\phi(z \mid x) \parallel p(z)).$$

---

Their main innovation was to approximate the intractable posterior  $p(z | x)$  by a *recognition network*  $q_\phi(z|x)$  from which they can sample via the *reparameterization trick* to allow for stochastic backpropagation through both the recognition and generative model at once. Assuming that the latent state is normally distributed, a simple transformation allows us to obtain a Monte Carlo gradient estimate of  $\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)]$  w.r.t.  $\phi$ . Given that  $z \sim \mathcal{N}(\mu, \sigma^2)$ , we can generate samples by drawing from an auxiliary variable  $\epsilon \sim \mathcal{N}(0, 1)$  and applying the deterministic and differentiable transformation  $z = \mu + \sigma\epsilon$ .

### 2.1.3. The Concrete Distribution

One simple and efficient way to obtain samples  $d$  from a  $k$ -dimensional categorical distribution with class probabilities  $\alpha$  is the Gumbel-Max trick

$$d = \text{one\_hot}(\text{argmax}[g_i + \log \alpha_i]) \quad \text{with } g_1, \dots, g_k \sim \text{Gumbel}(0, 1).$$

However, since the derivative of the argmax is 0 everywhere except at the boundary of state changes, where it is undefined, we cannot learn a parameterization by backpropagation. The Gumbel-Softmax trick approximates the argmax by a softmax which gives us a probability vector leading to what is since known as the Concrete distribution (Jang et al., 2017; Maddison et al., 2017). We can then draw samples via

$$d_k = \frac{\exp((\log \alpha_k + g_k)/\lambda)}{\sum_{i=1}^n \exp((\log \alpha_i + g_i)/\lambda)}.$$

This softmax computation approaches the discrete argmax as temperature  $\lambda \rightarrow 0$  while for  $\lambda \rightarrow \infty$  it approaches a uniform distribution. In the same vein, the bias of the estimator decreases with decreasing  $\lambda$  while its variance increases.

## 2.2. Related Work

Our model can be viewed as a Deep Kalman Filter (Krishnan et al., 2015) with structured inference (Krishnan et al., 2017). In our case, structured inference entails another stochastic variable model with parameter sharing inspired by Karl et al. (2019) and Karl et al. (2017) which pointed out the importance of backpropagating the reconstruction error through the generative transition. Marino et al. (2018) adopts this idea by starting an iterative amortized inference scheme with the prior prediction. Iterative updates are then only conditioned on the gradient of the loss, allowing the observation only to adjust, but not completely overrule, the prior prediction. We are different to a number of stochastic

---

---

sequential models like Bayer and Osendorfer (2014), Chung et al. (2015), Goyal et al. (2017) and Shabanian et al. (2017) by directly transitioning the stochastic latent variable over time instead of having an RNN augmented by stochastic inputs. Fraccaro et al. (2016) proposes a transition over both a deterministic and a stochastic latent state sequence, wanting to combine the best of both worlds.

Previous models (Fraccaro et al., 2017; Karl et al., 2017; Watter et al., 2015) have already combined locally linear models with recurrent Variational Autoencoders, however they provide a weaker structural incentive for learning latent variables determining the transition function. Van Steenkiste et al. (2018) approach a similar multi bouncing ball problem (see Section 2.4.1) by first distributing the representation of different balls into their own entities without supervision and then structurally hardwiring a transition function with interactions based on an attention mechanism. Kurle et al. (2020) learned a switching Gaussian linear systems with a Rao-Blackwellised particle filter and also explored Gaussian instead of discrete switch variables.

Recurrent switching linear dynamical systems (Linderman et al., 2017) uses message passing for approximate inference, but has restricted itself to low-dimensional observations and a multi-stage training process. Nassar et al. (2019) builds on this work and suggests a tree structure for enforcing a locality prior on switching variables where subtrees share similar dynamics. Johnson et al. (2016) propose a similar model to ours but combine message passing for discrete switching variables with a neural network encoder for observations learned by stochastic backpropagation. Dong et al. (2020) proposed a method for switching nonlinear dynamical systems by learning an inference network for the continuous latent variables, but performing exact marginalization over the discrete latent variables.

One feature an SLDS model may learn are interactions which have recently been approached by employing Graph Neural Networks (Battaglia et al., 2016; Kipf et al., 2018). These methods are similar in that they predict edges which encode interactions between components of the state space (nodes). Tackling the problem of propagating state uncertainty over time, various combinations of neural networks for inference and Gaussian processes for transition dynamics have been proposed (Doerr et al., 2018; Eleftheriadis et al., 2017). However, these models have not been demonstrated to work with high-dimensional observation spaces like images.

Our goal is not only to learn a single dynamical system, but be able to represent a whole class of systems and adapt to a change of dynamics without retraining or fine-tuning of parameters. This can be interpreted as a meta-learning problem where modeling each instance of the environment is seen as a different task. One popular approach for meta-learning is based on Model-Agnostic Meta-Learning (MAML) and its various improvements (Finn et al., 2017; Liu et al., 2019; Rakelly et al., 2019; Song et al., 2020),

---

a method for finding a set of parameters that are easily finetuned by a couple of gradient steps to a specific instance of a task or environment. Different and more along the line of our approach, Zhao et al. (2020) argues that one may use the training procedure to directly amortize this adaptation process into the inference network. Kumar et al. (2021) similarly learn an amortized adaptation module to cope with time-varying dynamics but it assumes knowledge of the ground truth parameters governing the dynamics during training time. Lee et al. (2020c) suggest learning an explicit context vector conditioning a learned dynamics model that is learned via a forward and backward prediction loss, but structurally they do not enforce the role of the context vector in shaping the transition.

## 2.3. Proposed Approach

We propose learning an SLDS model through a recurrent Variational Autoencoder framework which approximates switching variables by a Concrete distribution (Jang et al., 2017; Maddison et al., 2017). This leads to a model that can be optimized entirely by stochastic backpropagation through time. For inference, we propose a time-factorized approach with a specific computational structure, reusing the generative model. This allows us to learn good transition dynamics. Our generative model is shown in Figure 2.1b and our inference model in Figure 2.2a.

### 2.3.1. Generative Model

Our generative model for a single  $x_t$  is described by

$$p(x_t) = \int_{s_{\leq t}} \int_{z_{\leq t}} p(x_t | z_t) p(z_t | z_{t-1}, s_t, u_{t-1}) p(s_t | s_{t-1}, z_{t-1}, u_{t-1}) p(z_{t-1}, s_{t-1}) \quad (2.3)$$

which is close to the original SLDS model (see Figure 2.1a). Latent states  $z_t$  are continuous and represent the state of the system while states  $s_t$  are switching variables determining the transition. In order to use end-to-end backpropagation, we approximate the discrete switching variables by a continuous relaxation, namely the Concrete distribution. Different from the original model, we do not condition the likelihood of the current observation  $p_\xi(x_t | z_t)$  directly on the switching variables. This limits the influence of the switching variables to choosing a proper transition dynamic for the continuous latent space. The likelihood model is parameterized by a neural network with, depending on the data, either a Gaussian or a Bernoulli distribution as output.



---

As outlined in (2.3), we need to learn separate transition functions for the continuous states  $z_t$  and for the discrete states  $s_t$ . For the continuous state transition  $p(z_t | z_{t-1}, s_t, u_{t-1})$  we follow (2.1) and maintain a set of  $M$  base matrices  $\{(A^{(i)}, B^{(i)}, Q^{(i)}) | \forall i, 0 < i < M\}$  as our linear dynamical systems to choose from. Our transition is then defined as

$$p_\xi(z_t | z_{t-1}, s_t, u_{t-1}) = \mathcal{N}(\mu, \sigma^2)$$

where  $\mu = A_\xi(s_t)z_{t-1} + B_\xi(s_t)u_t, \quad \sigma^2 = Q_\xi(s_t)$ .

For the transition on discrete latent states  $p(s_t | s_{t-1}, z_{t-1}, u_{t-1})$ , we conventionally require a Markov transition matrix. However, since we approximate our discrete switching variables by a continuous relaxation, we can parameterize this transition by a neural network. We define

$$p_\xi(s_t | s_{t-1}, z_{t-1}, u_{t-1}) = \text{Concrete}(\alpha, \lambda_{\text{prior}}) \quad \text{where } \alpha = g_\xi(z_{t-1}, s_{t-1}, u_{t-1}).$$

Finally, the question arises how we determine our transition matrices  $A, B$  and  $Q$  since our Concrete samples  $s_t$  are now probability vectors and not one-hot vectors anymore. We could execute the forward pass by choosing the linear system corresponding to the highest value in the sample (hard forward pass) and only use the relaxation for our backward pass. This, however, means that we would follow a biased gradient. Alternatively, we can use the relaxed version for our forward pass and aggregate the linear systems based on their corresponding weighting

$$A_t(s_t) = \sum_{i=1}^M s_t^{(i)} A^{(i)}, \quad B_t(s_t) = \sum_{i=1}^M s_t^{(i)} B^{(i)} \quad \text{and} \quad Q_t(s_t) = \sum_{i=1}^M s_t^{(i)} Q^{(i)}. \quad (2.4)$$

Here, we lose the discrete switching of linear systems, but maintain a valid lower bound. We note that the hard forward pass has led to worse results and focus on the soft forward pass for this thesis.

### 2.3.2. Inference

As previously stated, the inference structure is critical for performance. In particular, we require the reconstruction loss gradient to flow through the generative transition model which is not naturally the case for these types of models. Without a properly structured inference scheme, only the KL divergence would guide the generative transition model. To achieve this, we formulate our inference scheme as a local optimization around the prior prediction where information from the observation only adjusts our prior prediction.

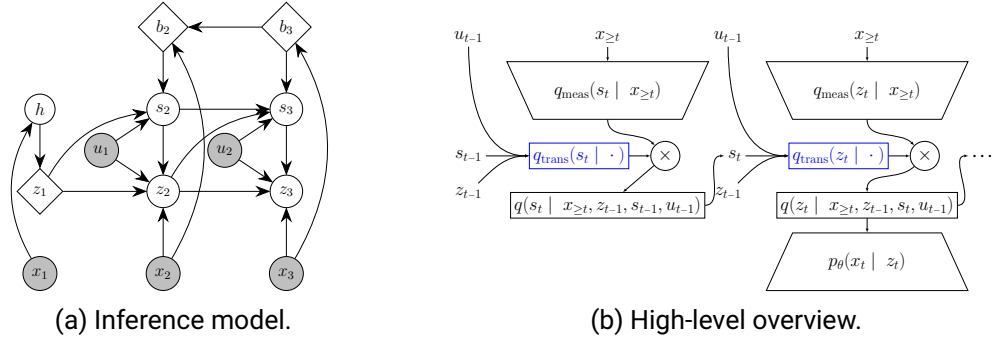


Figure 2.2.: (a) Depicts the inference model.  $b_t$  is the hidden state of the backward RNN of  $q_{\text{meas}}(s_t | x_{\geq t}, u_{\geq t})$ . Initial inference of  $h$  may be conditioned on the entire sequence of observations, or just a subsequence. We omitted the arrows for sake of clarity for the rest of the graph. (b) Shows schematically how we combine the transition with the inverse measurement model in the inference network. Transitions (in blue) are shared with the generative model.

Generally, we split the inference model into two parts: 1) the generative transition model and 2) encoding the current (and optionally future) observations. Both parts will independently predict a distribution over the next latent state which are then combined in a manner inspired by a Bayesian update. In the following, we will go through the specific construction for both normal and Concrete distributions. The overall inference structure is depicted in Figure 2.2b.

### Structured Inference of Gaussian Latent State

Starting from the factorization of our true posterior, our approximate posterior takes the form  $q_{\psi}(z_t | z_{t-1}, s_t, x_{\geq t}, u_{\geq t-1})$  where we notice that observations up to the last time step can be omitted as they are summarized by the last Markovian latent state  $z_{t-1}$ . As mentioned, the inference model splits into two parts: 1) transition model  $q_{\text{trans}}(z_t | z_{t-1}, s_t, u_{t-1})$  and 2) inverse measurement model  $q_{\text{meas}}(z_t | x_{\geq t}, u_{\geq t})$  as previously proposed in Karl et al. (2019). This split allows us to reuse our generative transition model in place of  $q_{\text{trans}}(z_t | z_{t-1}, s_t, u_{t-1})$ . For practical reasons, we only share the computation of the transition mean  $\mu_{\text{trans}}$  but not the variance  $\sigma_{\text{trans}}^2$  between inference and generative model. Both parts,  $q_{\text{meas}}$  and  $q_{\text{trans}}$ , will give us independent predictions about the new state  $z_t$  which will be combined in a manner akin to a Bayesian update in a

---

Kalman Filter. Formally, our filter is defined as

$$\begin{aligned}
q_\psi(z_t \mid z_{t-1}, s_t, x_{\geq t}, u_{\geq t-1}) &= \mathcal{N}(\mu_q, \sigma_q^2) \propto q_{\text{meas}}(z_t \mid x_{\geq t}, u_{\geq t}) \times q_{\text{trans}}(z_t \mid z_{t-1}, s_t, u_{t-1}) \\
q_{\text{meas}}(z_t \mid x_{\geq t}, u_{\geq t}) &= \mathcal{N}(\mu_{\text{meas}}, \sigma_{\text{meas}}^2) \\
\text{where } [\mu_{\text{meas}}, \sigma_{\text{meas}}^2] &= h_\psi(x_{\geq t}, u_{\geq t}) \\
q_{\text{trans}}(z_t \mid z_{t-1}, s_t, u_{t-1}) &= \mathcal{N}(\mu_{\text{trans}}, \sigma_{\text{trans}}^2) \\
\text{where } \mu_{\text{trans}} &= A_\theta(s_t)z_{t-1} + B_\xi(s_t)u_t, \quad \sigma_{\text{trans}}^2 = Q_\psi(s_t).
\end{aligned}$$

The densities of  $q_{\text{meas}}$  and  $q_{\text{trans}}$  are multiplied resulting in another Gaussian density defined by

$$\mu_q = \frac{\mu_{\text{trans}}\sigma_{\text{meas}}^2 + \mu_{\text{meas}}\sigma_{\text{trans}}^2}{\sigma_{\text{meas}}^2 + \sigma_{\text{trans}}^2} \quad \text{and} \quad \sigma_q^2 = \frac{\sigma_{\text{meas}}^2\sigma_{\text{trans}}^2}{\sigma_{\text{meas}}^2 + \sigma_{\text{trans}}^2}.$$

To enable online filtering, we chose to condition the inverse measurement model  $q_{\text{meas}}(z_t \mid x_{\geq t}, u_{\geq t})$  solely on the current observation  $x_t$  instead of the entire remaining trajectory. We found empirically that despite being theoretically suboptimal, this can yield good results in many cases, in particular in case of actually Markovian settings which are plentiful in the physical world. In this case, this methodology can be used for real-time state filtering in online feedback control.

For the initial state  $z_1$  we do not have a conditional prior from the transition model as in the rest of the sequence. Other methods (Krishnan et al., 2015; Fraccaro et al., 2016) have used a standard normal prior, however this led to a large divergence for us between approximate posterior and its prior in the first time step. We therefore decided that instead of predicting  $z_1$  directly, we predict an auxiliary variable  $h$  that is then mapped deterministically to a starting state  $z_1$ . The initial inference step is defined by

$$\begin{aligned}
q_\psi(h \mid x_{1:T}, u_{1:T}) &= \mathcal{N}(h; \mu_w, \sigma_w^2) \\
\text{where } [\mu_h, \sigma_h^2] &= i_\psi(x_{1:T}, u_{1:T}) \\
\text{and } z_1 &= t_\psi(h).
\end{aligned}$$

A standard Gaussian prior is then applied to  $h$ . Alternatively, we could have specified a more complex or learned prior for the initial state like the VampPrior (Tomczak and Welling, 2018) which we defer to future work.

### Inference of Switching Variables

Following Maddison et al. (2017) and Jang et al. (2017), we can reparameterize a discrete latent variable with the Gumbel-softmax trick. Again, we split our inference network

$q_\psi(s_t \mid s_{t-1}, z_{t-1}, x_{\geq t}, u_{\geq t-1})$  in an identical fashion into two components: 1) transition model  $q_{\text{trans}}(s_t \mid s_{t-1}, z_{t-1}, u_{t-1})$  and 2) inverse measurement model  $q_{\text{meas}}(s_t \mid x_{\geq t}, u_{\geq t})$ . The transition model is again shared with the generative model and is implemented via a neural network as we potentially require quick changes to chosen dynamics. The inverse measurement model is parametrized by a backward LSTM (or an MLP in the online filtering setting). For Concrete variables, we let each network predict the logits of a Concrete distribution and our inverse measurement model  $q_\psi(s_t \mid x_{\geq t}, u_{\geq t})$  produces an additional vector  $\gamma$ , which determines the value of a gate deciding how the two predictions are to be weighted. Formally, our approximate posterior is defined by

$$\begin{aligned}
q_\psi(s_t \mid s_{t-1}, z_{t-1}, x_{\geq t}, u_{\geq t-1}) &= \text{Concrete}(\alpha, \lambda_{\text{posterior}}) \\
&\text{with } \alpha = \gamma\alpha_{\text{trans}} + (1 - \gamma)\alpha_{\text{meas}}, \\
q_{\text{meas}}(s_t \mid x_{\geq t}, u_{\geq t}) &= \text{Concrete}(\alpha_{\text{meas}}, \lambda_{\text{posterior}}) \\
&\text{where } [\alpha_{\text{meas}}, \gamma] = k_\psi(x_{\geq t}, u_{\geq t}) \\
q_{\text{trans}}(s_t \mid s_{t-1}, z_{t-1}, u_{t-1}) &= \text{Concrete}(\alpha_{\text{trans}}, \lambda_{\text{prior}}), \\
&\text{where } \alpha_{\text{trans}} = g_\xi(z_{t-1}, s_{t-1}, u_{t-1}).
\end{aligned} \tag{2.5}$$

The temperatures  $\lambda_{\text{posterior}}$  and  $\lambda_{\text{prior}}$  are set as a hyperparameter and can be set differently for the prior and approximate posterior. The gating mechanism gives the model the option to balance between prior and approximate posterior. If the prior is good enough to explain the next observation,  $\gamma$  will be pushed to 1 which will ignore the measurement and will minimize the KL between prior and posterior by only propagating the prior. If the prior is not sufficient, information from the inverse measurement model can flow by decreasing  $\gamma$  and incurring a KL penalty.

### Reinterpretation as a Hierarchical Model

Lastly, we could step away from the theory of SLDS and instead view our model simply as an hierarchical graphical model where some variables should explain the current observation while others should determine the locally linear transition over said variables. When viewed in this manner it gives more freedom to model the switching variables in various ways, e.g. also as normally distributed with a specific decoder structure predicting some kind of transition parameters. Gaussian switching variables have been found to work well also by Kurle et al. (2020). If this worked better, it would either highlight still existing optimization problems of concrete relaxations of discrete random variables or be indicative that the hard switching behavior is too restrictive or not as easily learned. Additionally, it would support our initial claim that stochastic treatment of the transition

dynamics in general is important, irrespective of the specific implementation. Although any number of parameterizations are now viable, the one that we explore here is to use the Gaussian switching variables in the following way. Using just a single linear layer, we predict mixing coefficients

$$\alpha = \text{softmax}(W s_t + b) \in \mathbb{R}^M \quad (2.6)$$

for our transition matrices. Alternatively, one may also use the *sigmoid* instead of the *softmax* activation function for even more flexibility and expressiveness, allowing us to superimpose any number of linear dynamical systems (for a more thorough discussion see Appendix A.1.3). And then our transition matrices are defined by

$$A_t(s_t) = \sum_{i=1}^M \alpha_i(s_t) A^{(i)}, \quad B_t(s_t) = \sum_{i=1}^M \alpha_i(s_t) B^{(i)} \quad \text{and} \quad Q_t(s_t) = \sum_{i=1}^M \alpha_i(s_t) Q^{(i)}.$$

In this scenario, our inference scheme for normally distributed switching variables is then analogous to the one described in Section 2.3.2. Another advantage of this approach is that it allows explicit encoding of continuous hidden variables that govern the dynamics of a system. We will explore this in detail in Section 2.4.6 where we show how these latent variables can be used to encode properties governing the dynamics of robotic simulations.

### 2.3.3. Training

Our objective function is the commonly used evidence lower bound

$$\begin{aligned} \mathcal{L}_{\theta, \psi}(x_{1:T} \mid u_{1:T}) &\geq \mathbb{E}_{q_{\psi}(z_{1:T}, s_{2:T} \mid x_{1:T})} [\log p_{\theta}(x_{1:T} \mid z_{1:T}, s_{2:T}, u_{1:T})] \\ &\quad - \text{KL}(q_{\psi}(z_{1:T}, s_{2:T} \mid x_{1:T}, u_{1:T}) \parallel p(z_{1:T}, s_{2:T} \mid u_{1:T})). \end{aligned}$$

for our hierarchical model. As we chose to factorize over time, the loss for a single observation  $x_t$  becomes

$$\begin{aligned} \mathcal{L}_{\xi, \psi}(x_t \mid u_{1:T}) &= \mathbb{E}_{s_t} [\mathbb{E}_{z_t} [\log p_{\xi}(x_t \mid z_t)]] \\ &\quad - \mathbb{E}_{s_{t-1}, z_{t-1}} [\text{KL}(q_{\psi}(s_t \mid \cdot) \parallel p_{\xi}(s_t \mid s_{t-1}, z_{t-1}, u_{t-1}))] \\ &\quad - \mathbb{E}_{z_{t-1}} [\mathbb{E}_{s_t} [\text{KL}(q_{\psi}(z_t \mid \cdot) \parallel p_{\xi}(z_t \mid z_{t-1}, s_t, u_{t-1}))]]. \end{aligned} \quad (2.7)$$

The full derivation can be found in Appendix A.1.1. We generally approximate the expectations with one sample by using the reparametrization trick, the exception being the KL between two Concrete random variables in which case we take 10 samples for the approximation. For the KL on the switching variables, we further introduce a scaling factor

Table 2.1.: Mean squared error (MSE) on predicting future observations. Static refers to constantly predicting the first observation of the sequence.

Prediction steps	Reacher			3-Ball Maze		
	1	5	10	1	5	10
Static	5.80E-02	5.36E-01	1.25E+00	1.40E-02	5.74E-01	2.65E+00
LSTM	3.07E-02	3.67E-01	1.02E+00	7.20E-03	1.58E-01	2.60E-01
DVBF	1.10E-02	3.06E-01	6.05E-01	6.20E-03	1.36E-01	1.82E-01
DVBF Fusion	4.90E-03	2.97E-02	8.25E-02	4.33E-03	2.03E-02	4.88E-02
Ours (Concrete)	1.06E-02	5.73E-02	1.56E-01	2.28E-03	1.22E-02	3.40E-02
Ours (Normal)	<b>3.39E-03</b>	<b>1.85E-02</b>	<b>4.97E-02</b>	<b>1.30E-03</b>	<b>5.52E-03</b>	<b>1.38E-02</b>

$\beta < 1$  (as first suggested in Higgins et al. (2016), although they suggested increasing the weighting of the KL term) to scale down its importance. This decision can be justified by other work which has demonstrated theoretically and empirically the inadequacy of maximum likelihood training on the ELBO for learning latent representations (Alemi et al., 2018).

## 2.4. Experiments

In this section, we evaluate our approach on a diverse set of physics simulations based on partially observable system states or high-dimensional images as observations. We show that our model outperforms previous models and that our switching variables learn meaningful representations.

Models we compare to are Deep Variational Bayes Filter (DVBF) (Karl et al., 2017), DVBF Fusion (Karl et al., 2019) (called fusion as they do the same Gaussian multiplication in the inference network) which is closest to our model but does not have a stochastic treatment of the transition, the Kalman VAE (KVAE) (Fraccaro et al., 2017) and a vanilla LSTM (Hochreiter and Schmidhuber, 1997).

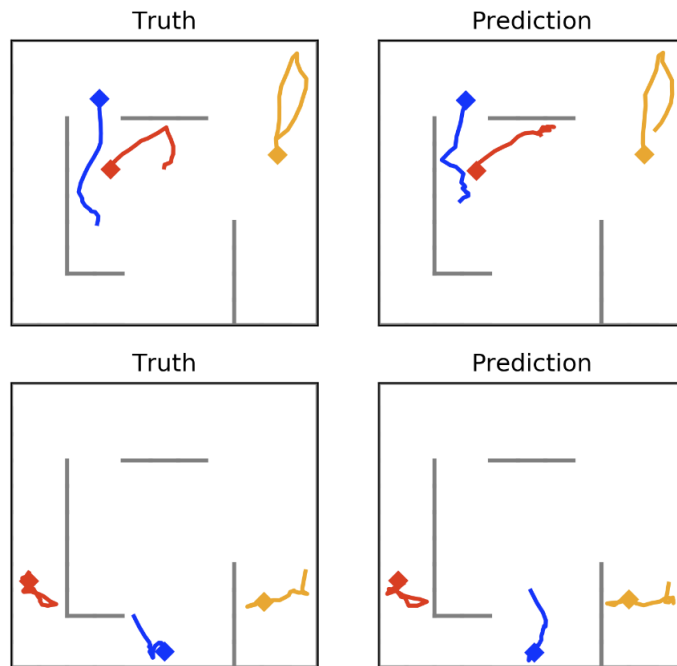


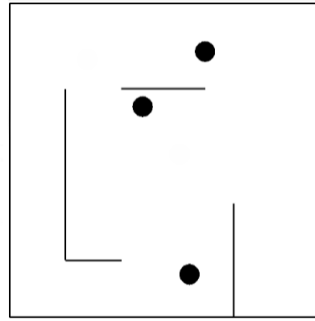
Figure 2.3.: Comparison of actual and predicted 20 step trajectories. The diamond marker denotes the starting position of a trajectory. These results have been produced with Concrete switching variables.

### 2.4.1. Multiple Bouncing Balls in a Maze

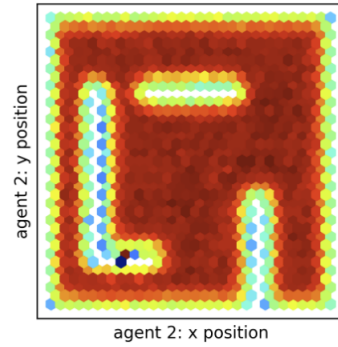
Our first experiment is a custom 3-agent maze environment simulated with Box2D. Each agent is fully described by its  $(x, y)$ -coordinates and its current velocity and may accelerate in either direction. We learn in a partially observable setting and limit the observations to the agents' positions, therefore  $x \in \mathbb{R}^6$  while the true state space is in  $\mathbb{R}^{12}$  and  $u \in \mathbb{R}^6$ .

Our first objective is to evaluate the learned latent space. We start by training a linear regression model on the latent space  $z$  to see if we have recovered a linear encoding of the unobserved velocities. Here, we achieve an R2 score of 0.92 averaged over all agents and velocity directions.

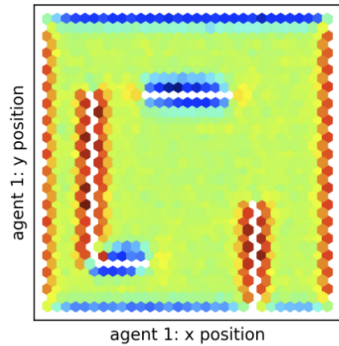
We now shift our focus to the switching variables that we anticipated to encode interactions with walls. We provide a qualitative confirmation of that in Figure 2.4 where we see switching variables encoding space where there is no interaction in the next time step and variables which encode walls, distinguishing between vertical and horizontal ones. In



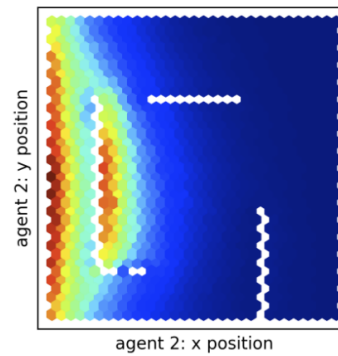
(a) Multi agent maze environment.



(b) Variable encoding free space for agent 2.



(c) Variable encoding walls for agent 1.



(d) System activation for deterministic transition.

Figure 2.4.: Figures (b) and (c) depict an agent's position colored by the average value of a single latent variable  $s$  marginalized over all controls and velocities. Figure (d) shows a typical activation map for a single transition system for deterministic treatment of transition dynamics. It does not generalize to the entire maze and stays fairly active near the wall.

Figure 2.4d one can see that if the choice of locally linear transition is treated deterministically, we do not learn global features of the same kind. To confirm our visual inspection, we train a simple decision tree based on latent space  $s$  in order to predict an interaction with a wall. Here, we achieve an F1 score of 0.46. It is difficult to say what a good value should look like as collisions with low velocity are virtually indistinguishable from no collision, but certainly a significant portion of collisions has been successfully captured. We were unable to capture collisions between two agents which can be explained by their



---

---

rare occurrence and much more complicated ensuing dynamics.

Going over to quantitative evaluation, we compare our prediction quality to several other methods in Table 2.1 where we outperform all of our chosen baselines. Also, curiously, modeling switching variables by a Gaussian distribution outperforms the Concrete distribution in all of our experiments. Aside from known practical issues with training a discrete variable via backpropagation, we explore one reason why that may be in Section 2.4.5, which is the greater susceptibility to the chosen scale of temporal discretization.

### 2.4.2. Reacher

We then evaluate our model on the RoboschoolReacher (OpenAI, 2017) environment. Again, we learn only on partial observations, removing velocities and leaving us with just the positions or angles of the joints as observations. The reacher’s dynamics are globally the same unless the upper and lower joints collide which is again a feature we expect our switching variables to detect. Similar to before, we inspect the learned latent space in Figure 2.5 visually where show linear encoding of shoulder joint’s velocity and encoding of collision dynamics. The chosen dynamics are unaffected by the shoulder angle, but are sensitive to the relative angle of the elbow to the upper link. To confirm our qualitative analysis, we again learn a linear classifier based on latent space  $s$  and reach an F1 score of 0.53. The predictive quality of our model is compared to other methods in Table 2.1.

### 2.4.3. Ball in a Box on Image Data

Here, we evaluate our method on high-dimensional image observations using the single bouncing ball environment used by Fraccaro et al. (2017). They simulated 5000 sequences of 20 time steps each of a ball moving in a two-dimensional box, where each video frame is a  $32 \times 32$  binary image. There are no forces applied to the ball, except for the fully elastic collisions with the walls. Initial position and velocity are randomly sampled.

In Figure 2.7a we compare our model to both the smoothed and generative version of the KVAE. The smoothed version receives the final state of the trajectory after the  $n$  predicted steps which is fed into the smoothing capability of the KVAE. One can see that our model learns a better transition model, even outperforming the smoothed KVAE for longer sequences. For short sequences, KVAE performs better which highlights the value of it disentangling the latent space into separate object and dynamics representation. A sample trajectory is plotted in Figure 2.6.

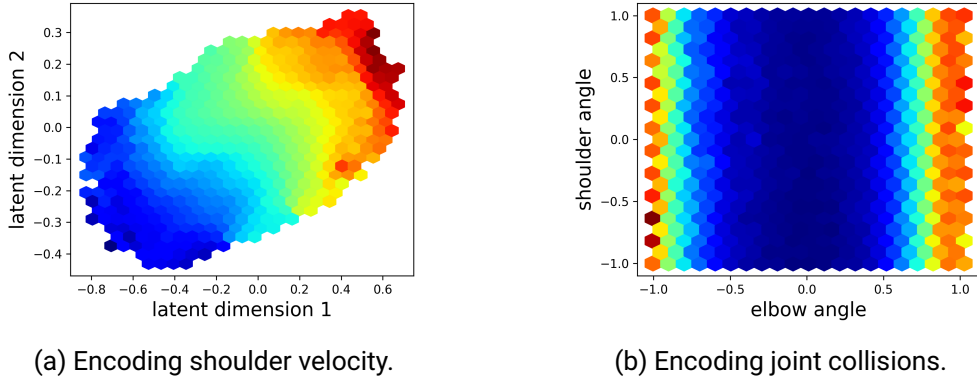


Figure 2.5.: Encoding features of the reacher environment. Figure (a) shows two latent dimensions colored by the ground truth shoulder velocity. The model captures the shoulder’s velocity purely out of provided joint angle data. Figure (b) highlights the activation of a Concrete switching variable. Note that the elbow’s angle is provided relative to the upper link, meaning that a (normalized) value of -1 or 1 leads to a collision with the upper link.

#### 2.4.4. FitzHugh-Nagumo

To compare to recurrent SLDS (rSLDS, Linderman et al. (2017)) and tree-structured SLDS (TrSLDS, Nassar et al. (2019)), we adopt their FitzHugh-Nagumo (FHN, FitzHugh (1961)) experimental setup. FHN is a 2-dimensional relaxation oscillator commonly used throughout neuroscience to describes a prototype of an excitable system (e.g., a neuron). It is fully described by the following system of differential equations:

$$\dot{v} = v - \frac{v^3}{3} - w + I_{\text{ext}}, \quad \tau \dot{w} = v + a - bw \quad (2.8)$$

Following their setup, we set the parameters to  $a = 0.7$ ,  $b = 0.8$ ,  $\tau = 12.5$  and external stimulus  $I_{\text{ext}} \sim \mathcal{N}(0.7, 0.04)$ . We create 100 trajectories of length 430 where the last 30 time steps are withheld during training and used for evaluation. Starting states are drawn uniformly from  $[-3; 3]^2$ . Since they restrict their observation model to be linear, we do the same for this experiment. In Figure 2.7b, we compare the models based on normalized multi-step predictive performance where our model matches the performance of TrSLDS.

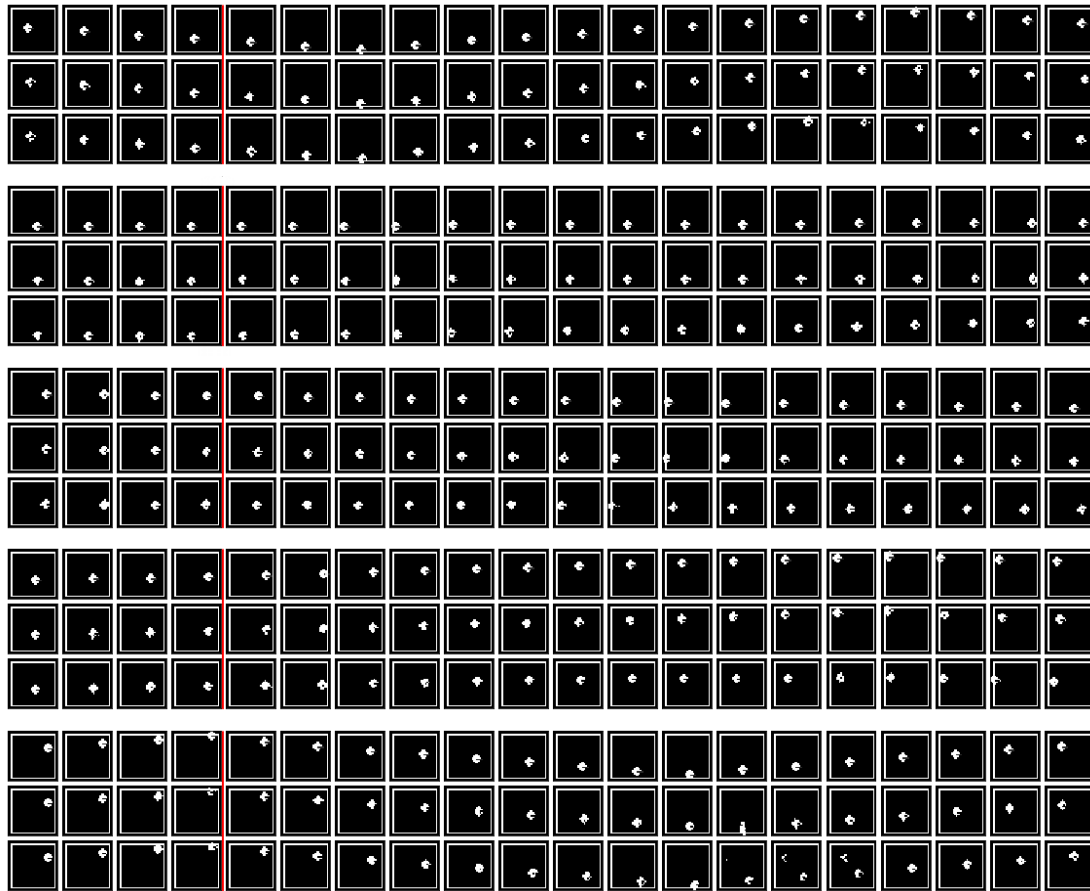
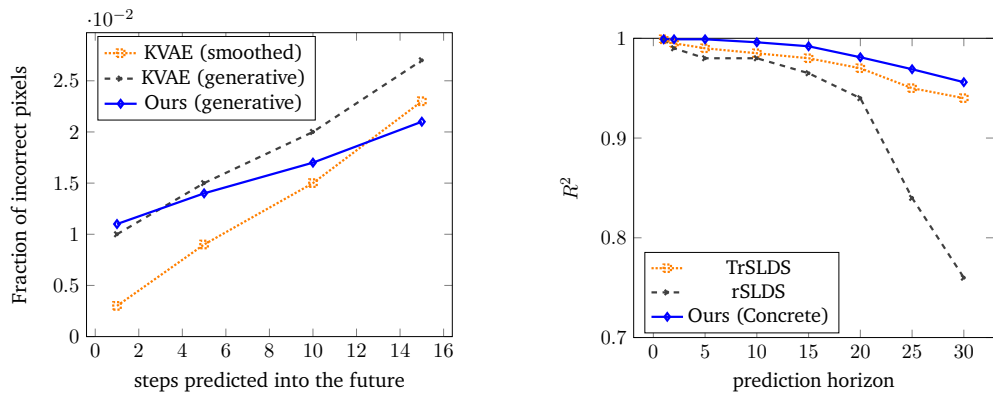


Figure 2.6.: First row: data, second row: reconstructions, third row: predictions. The first 4 steps are used to find a stable starting state, predictions start with step 5 (after the red line). These results have been produced with Gaussian distributed switching variables.

#### 2.4.5. Susceptibility to the Scale of Temporal Discretization

In this section, we would like to explore how the choice of  $\Delta t$  when discretizing a system influences our results. This crucial factor is often ignored or presumed to be chosen appropriately, although there has been some recent work addressing this issue specifically (Jayaraman et al., 2019; Neitz et al., 2018).

In particular, we suspect our model with discrete (Concrete) switching latent variables



(a) Fraction of incorrectly predicted pixels. (b) FitzHugh-Nagumo multi-step prediction.

Figure 2.7.: (a) Our dynamics model is outperforming even the smoothed KVAE for longer trajectories. (b) Our models performs as well as TrSLDS. (c) Modeling switching variables as Concrete random variables scales less favorably with increasing time discretization intervals.

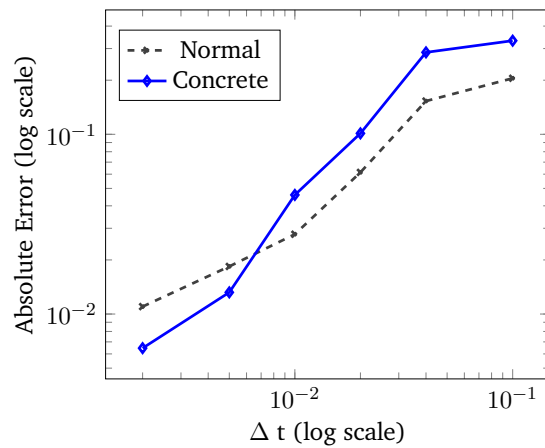


Figure 2.8.: Modeling switching variables as Concrete random variables scales less favorably with increasing time discretization intervals.

to be more susceptible to scaling than when modeled by a normal distribution. A possible explanation is that in the latter case the switching variables can scale the various matrices more freely, while in the former scaling up one system necessitates scaling down another.

Table 2.2.: Root mean squared error (RMSE) on predicting 10 time steps into the future. Results are averaged over 10 training runs,  $\pm 1$  standard deviation. To fairly evaluate, the filter is run for 80 steps before starting the prediction. The dynamics are changed once after 50 time steps.

	Pendulum	Reacher	Cheetah
RNN	$0.058 \pm 0.020$	$0.0508 \pm 0.0084$	$0.168 \pm 0.043$
LSTM	$0.063 \pm 0.032$	$0.0319 \pm 0.0052$	$0.111 \pm 0.041$
RSSM (Dreamer)	$0.053 \pm 0.019$	$0.0217 \pm 0.0080$	<b><math>0.076 \pm 0.004</math></b>
HVAE (SLAC)	$0.051 \pm 0.012$	$0.0300 \pm 0.0074$	$0.102 \pm 0.006$
DVBF Fusion	$0.044 \pm 0.006$	$0.0157 \pm 0.0028$	$0.130 \pm 0.008$
Ours	<b><math>0.037 \pm 0.005</math></b>	<b><math>0.0150 \pm 0.0019</math></b>	$0.098 \pm 0.005$

For empirical comparison, we go back to our custom maze environment (this time with only one agent as it is not pertinent to our question at hand) and learn the dynamics on various discretization scales. Then we compare the absolute error’s growth for both approaches in Figure 2.8 which supports our hypothesis. While the discrete approximation even outperforms for small  $\Delta t$ , there is a point where it rapidly becomes worse and gets overtaken by the normally distributed approximation. This suggests that  $\Delta t$  was simply chosen to be too large in both the reacher and the ball in a box with image observations experiment.

#### 2.4.6. Time-Varying Dynamics

Now, we shift the focus to time-varying dynamics and to the question whether our model can capture hidden variables governing the dynamics in its switching variables. Here, we compare to models that have been used in popular model-based reinforcement learning approaches, in particular, we compare to RSSM (Hafner et al., 2020) and SLAC (Lee et al., 2020a). The latter has also been used in Zhao et al. (2020) for meta-learning. Different to us, they use it to infer an unobserved task variable instead of changing dynamical properties, however they argue similar to us for an approach to amortize this process in the inference model. For these experiments, we adapted the DeepMind Control Suite

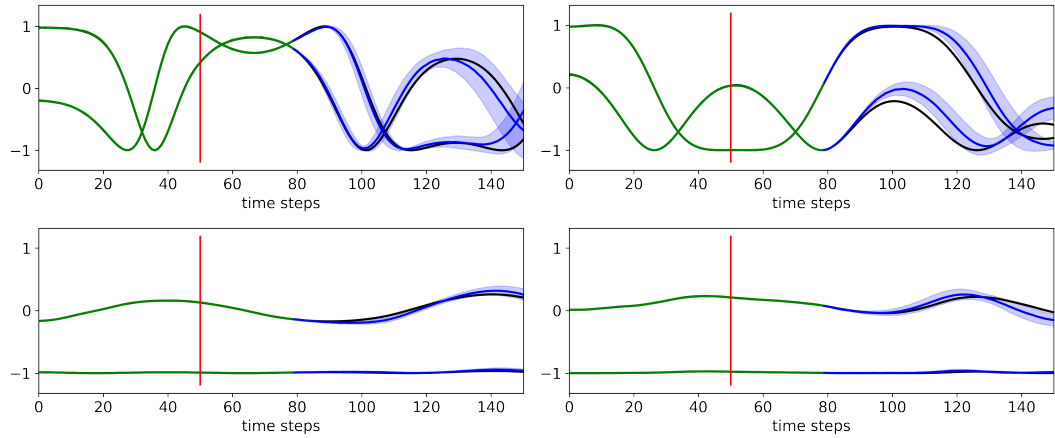


Figure 2.9.: Model predictions of the pendulum. The two lines represent  $\cos(\theta)$  and  $\sin(\theta)$  of the pendulum’s angle  $\theta$ . Black lines are the ground truth data, green lines are filtered reconstructions, blue lines are open-loop predictions from the filtered part onward. The shaded blue area is the estimated uncertainty over multiple rollouts. The vertical red line indicates a change of system dynamics.

(Tassa et al., 2020) and use modified versions of the pendulum, reacher and cheetah environment. Depending on the system, we change the systems mass, damping coefficients, stiffness or link lengths (see Appendix A.1.2 for specifics). At every time step, there is a probability  $p = 0.005$  that these underlying parameters are resampled, abruptly changing the dynamics of the system. As observations, we use (joint) positions, but remove time derivative data (velocities). The time-varying physics parameters also remain unobserved.

### Model Learning

For model training, we collect a data set consisting of 1000 episodes of 500 steps by sampling actions from an Ornstein-Uhlenbeck process to get a diverse data set of the environment. For evaluation, we first run our filter for 80 steps and predict the system from then on forwards. After the first 50 time steps, we change the dynamics once and keep them fixed afterwards as we want to fairly evaluate open-loop prediction quality. This evaluation then showcases not only that the model can learn various dynamics but also that it can adapt online to changes to the dynamical system. This evaluation scheme is graphically presented in Figure 2.9 on some exemplary trajectories of the pendulum. The

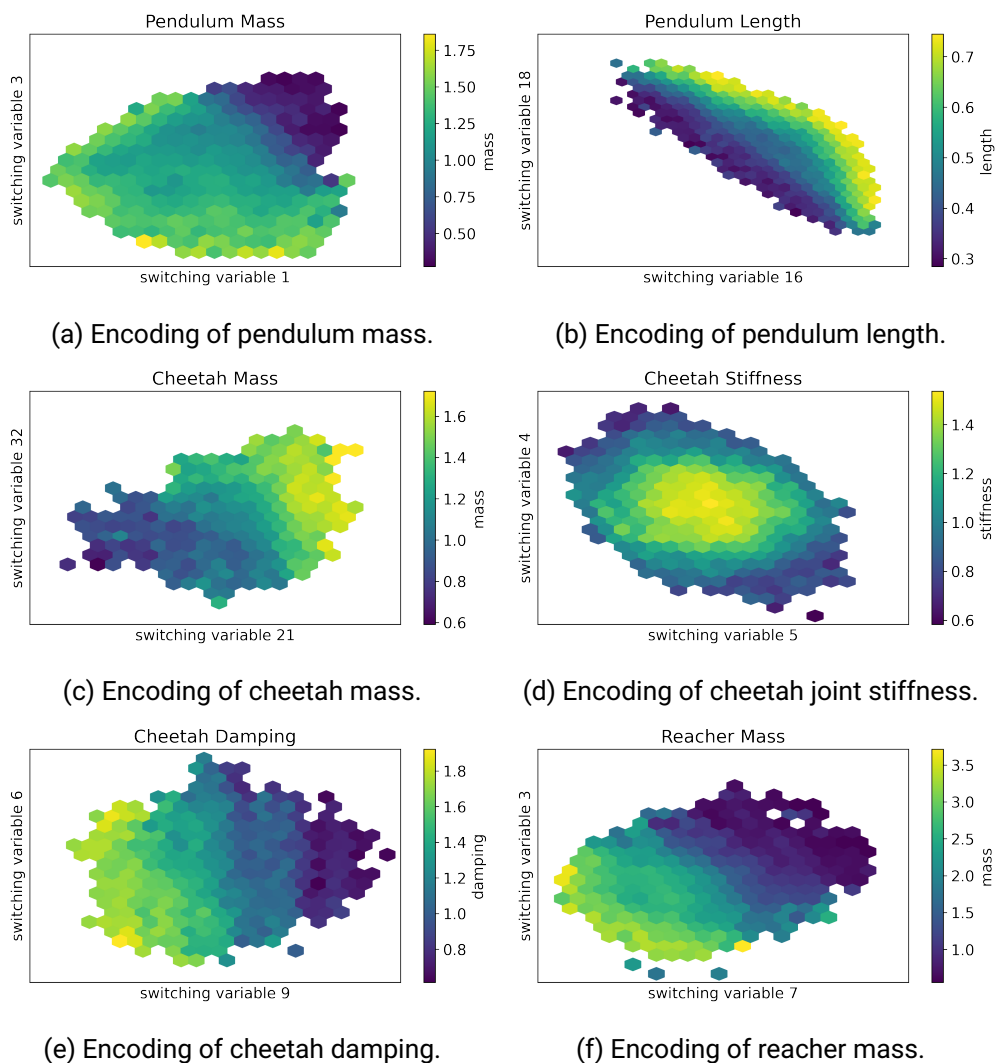


Figure 2.10.: Encoding of unobserved variables governing the physics of a system into our latent switching variables.

numerical results are presented in Table 2.2, where we can see that our method performs best in 2 out of the 3 environments. We average the evaluation over 10 separate training runs and compare the RMSE on predicting 10 time steps into the future. For the Cheetah, the RSSM performed best, this may indicate possible scaling issues of globally switching

Table 2.3.: The best performing agent of each of the 10 runs evaluated over 100 episodes. The initial state and random seed for changing the physics is identical for the evaluation of each algorithm.

	<b>Avg. Reward</b>
SAC	$-1136 \pm 69$
SLAC LSTM	$-1371 \pm 382$
SLAC DVBF Fusion	$-1088 \pm 108$
SLAC	$-1027 \pm 97$
SLAC SLDS-DVBF (Ours)	<b><math>-950 \pm 61</math></b>

linear dynamics to more complicated systems, however our method still comes in second.

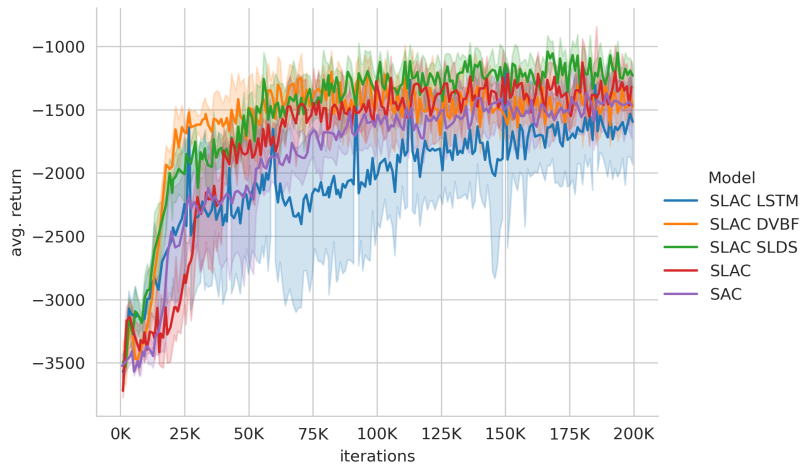
Qualitatively, we inspected the learned latent spaces and highlighted some of them in Figure 2.10. Here, we plot some switching variables with the highest correlation to some unobserved quantities governing the system dynamics, clearly showing successful encoding of mass, link length, joint stiffness and damping coefficients. Despite the high dimensionality of the switching latent space (between 32 and 64), we found these individual correlations without further post-processing or dimensionality reduction of the latent space. As we suspected based on our structural prior, all of these encodings were found in switching latent variables  $s$  and not in latent variables  $z$ .

### Latent Representation for Control

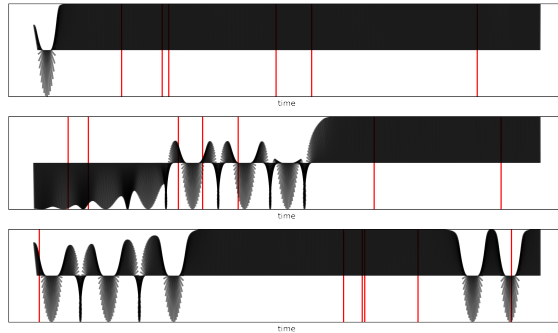
Lastly, we want to give some support to the claim that our filter and learned latent representation can be useful for control. For that, we try to learn an agent in the time-varying pendulum environment using SLAC, a model-based reinforcement learning method. SLAC can be described as a model-based Soft Actor-Critic (SAC) where the latent state representation of the model is used to inform the critic, but other than that the model is not used further for either backpropagation or generating synthetic data. Different to the original paper, for our case we found it necessary to also condition the policy on the latent representation. Hence, this algorithm and setting allows an isolated comparison of the effect of the learned latent state representations on learning of an agent.

Our results are presented in Figure 2.11 showing performance over training and high-





(a) SLAC performance on the time-varying pendulum environment.



(b) Exemplary pendulum rollouts, red lines indicate a change of physics.

Figure 2.11.: Reinforcement learning results in the time-varying pendulum environment.

lighting a few actual rollouts of the final policy. For each algorithm, we performed 10 training runs. In Table 2.3, we compare best performing agent of each training run on 100 episodes. For a fair comparison, both the initial state and the change of physics were changed identically for each agent in the 100 episodes.

It should be noted that, given the control constraints, some configurations of the pendulum are actually impossible to swing-up while for others the pendulum is not underactuated anymore. Thus, only relative performance and not necessarily the absolute performance should be compared. We found that not only did our latent representation

---

---

lead to the best performance, but also the performance across runs were more stable than with most other models. Although the final difference in performance is not huge, we found it to be quite consistent over the training runs. For a model-free baseline, we also included SAC where we stacked 4 observations to address partial observability. Overall, this study gives some first indication that such an explicit encoding as enforced through our model’s structural prior can be helpful for model-based reinforcement learning.

## 2.5. Discussion

We want to emphasize some subtle differences to previously proposed architectures that make an empirical difference, in particular for the case when  $s_t$  is chosen to be continuous. In Watter et al. (2015) and Karl et al. (2017), the latent space is already used to draw transition matrices, however they do not extract features such as walls or joint constraints. There are a few key differences to our approach. First, our latent switching variables  $s_t$  are only involved in predicting the current observation  $x_t$  through the transition selection process. The likelihood model therefore does not need to learn to ignore some input dimensions which are only helpful for reconstructing future observations but not the current one. There is also a clearer restriction on how  $s_t$  and  $z_t$  may interact:  $s_t$  may now only influence  $z_t$  by determining the dynamics, while previously  $z_t$  influenced both the choice of transition function as well as acted inside the transition itself. These two opposing roles lead to conflicting gradients as to what should be improved. Furthermore, the learning signal for  $s_t$  is rather weak so that scaling down the KL-regularization was necessary to detect good features. Lastly, a locally linear transition may not be a good fit for variables determining dynamics as such variables may change very abruptly. Therefore, it might be beneficial to have part of the latent space evolve according to locally linear dynamics and other parts according to a general purpose neural network transition. Overall, our structure of choosing a transition gives a stronger inductive bias towards learning such features when compared to other methods.

## 2.6. Conclusion

In this chapter we show how to learn a latent state-space model with switching linear dynamical systems via neural variational inference and end-to-end backpropagation. We showed that latent switching variables are able to extract and represent joint constraints or walls in a maze from raw data streams of an agent’s position alone. In the case of time-varying dynamics, we showed that continuous switching variables can not only infer

---

---

changing hidden variables governing the dynamics such as masses or link lengths, but also infer those variables via online filtering; no retraining or fine-tuning of model parameters is required. When comparing switching variables modeled by either a concrete relaxation or by a Gaussian distribution, we showed that the latter is seemingly more robust in practice while still retaining a lot of the interpretability of a strict switching behavior. The inferred state-space representations are not only readily interpretable but also improved simulation accuracy in various tasks when compared to other approaches. This makes this model a promising candidate for model-based reinforcement learning methods also when tackling time-varying dynamics or for adaptive control problems in general – which we demonstrated in a first small case study. Further developing and showcasing this potential application of this model for control is the focus of our future work.



---

### 3. Learning to Control Real Robots via Deep Model-Based Reinforcement Learning

---

Designing controllers for robots requires years of expertise and effort, fine tuning modules for state estimation, model-predictive control and contact scheduling. As a final product, one has designed a controller that works only on that one particular robot. In contrast, deep RL presumes no prior knowledge of a robots dynamics and promises broad applicability. However, applying deep RL to robotic systems faces many challenges, among them poor sample efficiency. This shortcoming is often circumvented by finding ways to parallelise the learning over many agents (i. e. many identical robotic setups) and by automatic resetting of the environment (Gu et al., 2017; Kalashnikov et al., 2018). As an alternative to expensive real-world rollouts, simulators have commonly been employed, allowing us to generate more data without actually operating the robot. This approach has been the most successful way to deploy learned controllers on real hardware (Peng et al., 2018; Andrychowicz et al., 2020; Tan et al., 2018; Chebotar et al., 2019; Lin et al., 2019). However, this just avoids the underlying issue that we wanted to address: while controller learning is now generic, most of the expert knowledge is still required to build the simulator. Thus, a truly generic solution using little to no prior knowledge requires us to also learn the simulator.

All of these restrictions become obvious when working on drones. Simulating the dynamics of a drone, esp. a self-built one, is complex. Recording many flights with a drone is cumbersome because of frequent battery changes; flying many drones in parallel is difficult because of space constraints, or just costly. We therefore focus on an approach that is more data-efficient as we have to reduce data collection needs to a minimum. We learn a full dynamics simulation of the drone, employing neural variational inference methods to train a latent state-space model from observational data (Krishnan et al., 2017; Karl et al., 2017; Fraccaro et al., 2017). This generic framework is applicable to all kinds of dynamical systems as it is agnostic to sensorimotor configuration, and this we demonstrate by applying the method to a 7-DoFs robot arm.

As model bias represents a huge challenge of this approach, most prior work on learning controllers for quadrotors has been conducted using a predefined drone dynamics model,

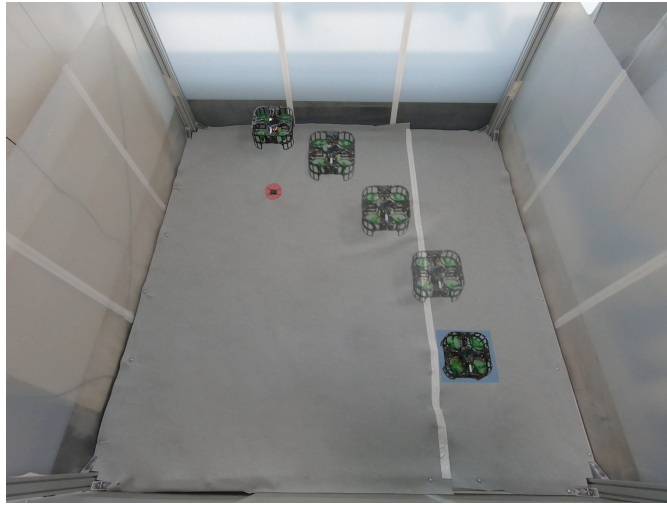


Figure 3.1.: A learned controller flying a quadrotor to a goal marker. The controller was optimized using rollouts from a learned dynamics model and is executed onboard.

e. g. Hwangbo et al. (2017), Lin et al. (2019) and Koch et al. (2019). Typically, they use either imitation learning or model-free RL algorithms where the predefined simulator is used as a data generator. As a drawback, the simulator’s internal structures can usually not be exploited for optimization and its gap to reality may be high. In contrast, our learned simulator is fully differentiable and, having been trained on data of the real system, may also help reduce the simulation-to-reality gap. Further, truly model-free methods have no possibility of building an internal model of the world, unless equipped with a hidden state which allows them to build a world model implicitly. They are thus harder to apply in partially observed settings, however this drawback can be mitigated in simple scenarios by providing the policy with the last few observations instead of just the most recent one.

Our principal contribution is a methodology for learning a controller using a learned LSSM without the necessity to provide prior information in form of a simulator or predefined differential equations. The robot-agnostic dynamics model is learned from real-world data and can deal with noisy and partial observations. We evaluate this methodology on a self-built drone that has to fly to marked positions (see Figure 3.1) using various sensor types and combinations. Using a novel model-based Actor-Critic approach, the controller and value function are optimized entirely in the latent rollouts of the learned dynamics model. The controller acts on the same level a human would operate a quadrotor, provid-

---

ing roll, pitch, yaw and throttle control inputs. It is supported by a low-level controller which keeps the drone hovering in absence of control inputs—a feature which mainly simplifies the initial exploration policy. When deployed, both the learned model—used for online state estimation—and controller are executed in real-time on the drone itself using only embedded computational resources (cq. a Raspberry Pi 4).

To show that the algorithm is not limited to quadrotors, we perform some ablation studies using a Franka Emika Panda robot arm. Being a stationary robot without the need for changing batteries or any other human interaction, we can conduct a comparison to other methods. The task is kept similar to the drone case, meaning the agent has to move the arm’s end effector to an observed and randomly sampled Cartesian position and orientation. Control is performed by commanding joint velocities in a gravity- and Coriolis-compensated robot. The main challenge compared to the drone lies in the higher number of DoFs and their complicated interaction.

Even though we have to learn a (global) dynamics model, by leveraging it for policy optimization we can be sufficiently data efficient for a real-world application. For the drone, we require only 30 minutes of real-world flight (equivalent to approximately 25.000 model steps at  $\sim 14$  Hz) to find our final policy. In the robot arm scenario, we require 60.000 steps at 10 Hz in the environment which corresponds to 100 minutes of interaction on a real robot.

## 3.1. Background

Throughout this chapter, we consider observations  $x_t$  and control inputs  $u_t$  that form a trajectory or episode  $\tau = (x_1, u_1, x_2, u_2, \dots, u_{T-1}, x_T)$ . We denote a sequence of variables as  $\bar{x} = x_{1:T} = (x_1, x_2, \dots, x_T)$ .

### 3.1.1. Backpropagation through Stochastic Variables

Computing  $\nabla_{\theta} \mathbb{E}_{q_{\theta}(z|\cdot)}[f(z)]$ , i.e. a gradient w. r. t. to some parameters  $\theta$  where the expectation is taken with respect to a distribution parameterized by  $\theta$  is key for learning stochastic representations by gradient descent. Function  $f$  is assumed to be integrable and smooth. One way to approximate this gradient is by employing the score function estimator (also called REINFORCE (Williams, 1992) or likelihood ratio method (Glynn, 1990))

$$\nabla_{\theta} \mathbb{E}_{q_{\theta}(z|\cdot)}[f(z)] = \mathbb{E}_{q_{\theta}(z|\cdot)}[f(z) \nabla_{\theta} \log q_{\theta}(z | \cdot)]. \quad (3.1)$$

In practice, this estimator often suffers from high variance. In this work, we instead make use of a pathwise Monte Carlo gradient estimator, a technique repopularized by Kingma

and Welling (2014) and Rezende et al. (2014). Instead of sampling from  $z$  directly, one can sample from an auxiliary noise variable  $\epsilon$  that is independent from the parameters  $\theta$  and transform the sample into a sample of the original distribution

$$z = g_\theta(\epsilon) \quad \text{with} \quad \epsilon \sim p(\epsilon). \quad (3.2)$$

Such a transformation  $g$  exists for many continuous distributions and is in general required to be smooth and invertible. This allows us to compute a Monte Carlo estimates of an expectation

$$\mathbb{E}_{q_\theta(z|x)}[f(z)] = \mathbb{E}_{p(\epsilon)}[f(g_\theta(\epsilon))] \quad (3.3)$$

by sampling from  $p(\epsilon)$  instead of  $q_\theta(z|x)$ . In this form, the gradient operator  $\nabla_\theta$  can trivially be moved inside the expectation. We made use of this technique, often called *reparameterization trick*, both for learning a latent variable model and for optimizing a stochastic policy.

### 3.1.2. Reinforcement Learning

We consider the regular Markov Decision Process (MDP) (Bellman, 1957) where both state space  $\mathcal{Z}$  and action space  $\mathcal{U}^1$  are continuous. An agent starts in an initial state  $z_0 \sim p(z_0)$ . At every time  $t$ , the agent samples a control  $u_t$  from its policy  $\pi(u_t | z_t)$  and transitions to a new state  $z_{t+1}$  according to the dynamics  $p(z_{t+1} | z_t, u_t)$ . The agent receives a bounded reward  $r_t \sim r(z_t, u_t)$  reinforcing or punishing the behavior. The reward-to-go  $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$  is the accumulated discounted reward from now on, with a discount factor  $\gamma \in [0; 1)$ . Reinforcement Learning seeks to learn a policy that maximizes the expected reward-to-go

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)}[R_t], \quad (3.4)$$

where  $p_\pi(\tau)$  represents the distribution over trajectories induced by following policy  $\pi$ . Optimizing directly w. r. t. eq. (3.4), as is often done in policy search methods (Deisenroth et al., 2013), can suffer from high variance since we are using a single sampled trajectory to evaluate the performance of our policy. Value-based methods introduce state or state-action value estimators which can reduce variance while introducing a bias. The value of a state  $z$  under policy  $\pi$  is defined as the expected reward for following the policy from state  $z$ , i.e.  $V^\pi(z) = \mathbb{E}_{\tau \sim p_\pi(\tau)}[R_t | z_t = z]$ . The state-action value is defined as  $Q^\pi(z_t, u_t) = r(z_t, u_t) + \gamma \mathbb{E}_{z_{t+1} \sim p(z_{t+1}|z_t, u_t)}[V^\pi(z_{t+1})]$ . To scale to real-world problems, state and state-action value functions are typically represented by function approximators

<sup>1</sup>More common is the notation with state space  $\mathcal{S}$  and action space  $\mathcal{A}$ , but we remain with  $\mathcal{Z}$  and  $\mathcal{U}$  for sake of consistency.



such as neural networks. Actor-Critic methods are a hybrid approach that learn both a policy (called *actor*) and value function (called *critic*) that bootstrap each other.

When working in continuous action spaces, optimization is most commonly done using policy gradients where the policy parameters  $\theta$  are updated in direction of the performance gradient (Sutton et al., 2000)

$$\mathbb{E}_{\tau \sim p_{\pi}(\tau)}[\nabla_{\theta} \log \pi(u_t \mid z_t) Q^{\pi}(z_t, u_t)]. \quad (3.5)$$

Notably, the policy gradient does not depend on the gradient of the state distribution.

MBRL distinguishes itself from model-free methods by exploiting a (possibly) learned dynamics model  $p(z_{t+1} \mid z_t, u_t)$  to optimize a policy. There are numerous ways of doing so. In Dyna (Sutton, 1991) and derivative work, the learned dynamics model is used exclusively as a data generator for optimizing the policy using standard model-free algorithms. Real-world data is here only used to fit the dynamics model. Another category of algorithms uses model derivatives either for policy search or improved value estimation (Heess et al., 2015; Feinberg et al., 2018; Byravan et al., 2019).

## 3.2. Variational Latent Dynamics

As a model for the environment, we re-use the method introduced in Chapter 2. However, to be useful in our reinforcement learning setting, on top of learning a model of the system dynamics, we also require a (differentiable) reward function. In practice, the reward function is often based on the entire system state or even some additional quantities like contact points (Brockman et al., 2016) to provide an informative learning signal. However, we would also like to learn in settings with only partial observations where not all of these quantities are explicitly available to the agent. Thus, we can not directly compute the rewards based on predicted observations of an imagined trajectory. Instead, we must build a model of the reward function which is based on the learned latent state. Here, even in the partially observed cases, we expect the latent space to contain all necessary information for prediction and thus also for modeling a reward function. We introduce a separate neural network for reward prediction  $p_{\xi}(r_t \mid z_t, u_t)$  and extend the general lower bound in eq. (2.7) by a term for the reward function

$$\mathcal{L}_{\text{Reward}}(r_{1:T} \mid u_{1:T}) = \sum_{t=1}^T \mathbb{E}_{z_t \sim q_{\psi}}[\log p_{\xi}(r_t \mid z_t, u_t)]. \quad (3.6)$$

Note that we stop the gradient such that this term does not shape the latent space as this changes and possibly simplifies the underlying problem significantly. Additionally, this would render the model non-transferable to different tasks.

---

### 3.3. Learning a Controller

Generally, our algorithm falls into the category of Actor-Critic methods as introduced in Section 3.1.2, meaning we learn both a parameterized policy (actor) and value function (critic) that evaluates the actor. In terms of design choices, having a generative model of the environment gives us every freedom in shaping our policy optimization scheme. However, for policy and value function optimization, we limit ourselves to data generated by the learned simulator for optimization. The observed data is thus only used for fitting the generative model, but not directly for learning the controller. This demonstrates what is possible with a purely model-based approach which is clearer to evaluate if we do not make use of real-world rollouts for policy optimization. A hybrid approach, where real-world data is used in an off-policy fashion, is also possible and the extension for that is quite straightforward using importance sampling (see e. g. Heess et al. (2015)).

Going more into detail, Algorithm 1 sketches a high-level overview of the algorithm. After collecting an initial data set and optimizing a preliminary dynamics model, we start with the main optimization loop. Note that one could also perform only a single collection step at the start and then learn in an offline setting. For optimizing the controller we root every dreamed rollout, i. e. a rollout within the latent dynamics model, in a real-world experience by randomly choosing points from the data set and computing the corresponding latent state by filtering up to that point. From then on we follow the policy for a fixed and small number of steps in our latent dynamics model. This rollout is used to optimize both actor and critic via stochastic analytic gradients.

The length of the rollout is a trade-off between exploiting the model’s internal structure for gradient computation and limiting the impact of the model’s accumulating prediction error. Long imagined trajectories are generally not necessary for two reasons. First, we use a learned value function as a critic to evaluate the policy’s performance instead of relying entirely on a Monte Carlo estimate of the reward. Second, we are free to start a rollout at any observed state that we have ever seen before, which is in contrast to e. g. a real-world setup where the robot always resets to the same starting position. On the other hand, imagining just a single step limits our use of the model severely. Generating data this way has numerous advantages: we are always on-policy, we can create arbitrarily many experiences and have no need for a replay buffer.

Note that the various updates of model, policy and value function are not tied to each other and may be performed at independent intervals. Fujimoto et al. (2018) have noted that updating the actor less frequently than the critic may be beneficial.

```

Initialize model parameters  $\theta, \phi, \xi, \psi$  randomly;
for each data collection do
  \ \ Collect data in real environment;
  for each episode do
    for  $t=1..T$  do
      Get state estimate  $z_t \sim q_\psi(z_t \mid x_t, z_{t-1}, u_{t-1})$ ;
      Evaluate policy  $u_t \sim \pi(u_t \mid z_t)$ ;
      \ \ Execute control in env.;
       $x_{t+1}, r_t \leftarrow \text{execute}(u_t)$ ;
    Add episode  $(\bar{x}, \bar{u}, \bar{r})$  to data set  $\mathcal{D}$ .;
  \ \ Optimize model and controller;
  for each training iteration do
    \ \ Fit dynamics model;
    Sample  $b$  episodes  $(\bar{x}, \bar{u}, \bar{r}) \sim \mathcal{D}$ ;
    Update model parameters  $\xi, \psi$  based on eq. (2.7);
    \ \ Fit actor and critic;
    Imagine a batch of trajectories starting from a random subset of filtered
      states;
    Update policy  $\theta$  with  $\nabla_\theta$  of eq. (3.13);
    Update value function  $\phi$  with  $\nabla_\phi$  of eq. (3.9);
    Update target value func.  $\phi' \leftarrow \alpha_{\phi'} \phi + (1 - \alpha_{\phi'}) \phi'$ ;

```

**Algorithm 1:** Model-Based Actor-Critic

### 3.3.1. Value Estimation

As mentioned, we optimize the value function based on on-policy rollouts in our latent dynamics model, where a rollout is formally defined as

$$\tau_{\theta, \xi} \sim p(z_1) \prod_{t=1}^{H-1} \pi(u_t \mid z_t) p_\xi(z_{t+1} \mid z_t, u_t). \quad (3.7)$$

Note that for notational convenience we subsume our hierarchical latent space consisting of  $z$  and  $s$  into a single latent state  $z$  for the remainder of this chapter. Given these rollouts, state values can be estimated in numerous ways, most commonly by minimizing the

temporal difference error (TD-error) (Sutton et al., 1998)

$$\mathbb{E}_{\tau_{\theta,\xi}} \left[ \left( V_{\phi}^{\pi}(z_t) - (r_{\xi}(z_t, u_t) + \gamma V_{\phi}^{\pi}(z_{t+1})) \right)^2 \right]. \quad (3.8)$$

For faster convergence, we make use of an n-step variant (Watkins, 1989)

$$\mathbb{E}_{\tau_{\theta,\xi}} \left[ \left( V_{\phi}^{\pi}(z_t) - y_t \right)^2 \right] \\ \text{with } y_t = \sum_{i=0}^{H-1} \gamma^i r_{\xi}(z_{t+i}, u_{t+i}) + \gamma^H V_{\phi'}^{\pi}(z_{t+H}), \quad (3.9)$$

which reduces the bias at the cost of increased variance. Since we sample from both policy and latent dynamics model, we can minimize this objective w. r. t. parameters  $\phi$  by computing analytic stochastic gradients using the reparameterization trick. To stabilize this regression, we introduce a *target* value network  $V_{\phi'}^{\pi}(z_{t+H})$  parameterized by  $\phi'$  (Mnih et al., 2015). The target network’s parameters  $\phi'$  are slowly updated towards  $\phi$  using a small learning rate  $\alpha_{\phi'} \ll 1$  and thus the target values of the regression are changing more slowly. The update is defined by an exponential decay

$$\phi' \leftarrow \alpha_{\phi'} \phi + (1 - \alpha_{\phi'}) \phi'. \quad (3.10)$$

We parameterize the value function by a neural network.

### 3.3.2. Policy

Using the critic, we can optimize our policy, which we choose to be represented as a diagonal multivariate Gaussian distribution, i. e.

$$\pi(u_t \mid z_t) \sim \mathcal{N}(\mu_{\theta}(z_t), \sigma_{\theta}(z_t)). \quad (3.11)$$

The mean  $\mu_{\theta}(z_t)$  and standard deviation  $\sigma_{\theta}(z_t)$  are again parameterized by dense neural networks. In general, any distribution where the reparameterization trick is applicable may be used instead. Similar to optimizing the value function, we make use of short policy rollouts within the learned dynamics model. These rollouts do not need to be the same or be of the same length as used for learning the critic. As the general objective for policy optimization in Actor-Critic methods, one chooses the gradient of the critic, i. e.

$$\nabla_{\theta} \mathbb{E}_{\tau_{\theta,\xi}} \left[ r_{\xi}(z_t, u_t) + \gamma V_{\phi}^{\pi}(z_{t+1}) \right]. \quad (3.12)$$

Here again, we make a slightly different compromise between Monte Carlo estimation and relying on the critic by choosing to sum up predicted rewards before truncating the

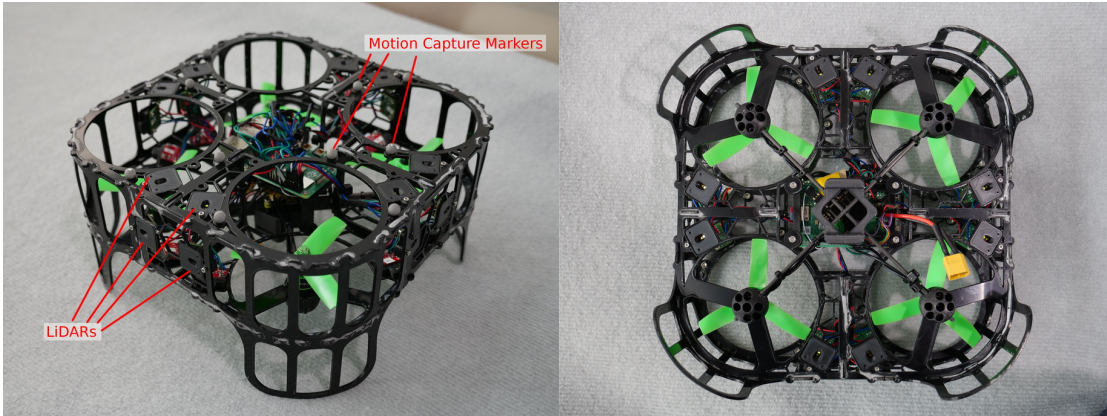


Figure 3.2.: Our quadrotor with 24 LiDARs (8 of them facing to the top, side and bottom), motion capture markers and a Raspberry Pi 4. The flight controller including the IMU is placed directly beneath the Raspberry Pi. The frame makes the drone withstand collisions with the walls or floor.

series with the (discounted) critic of the trajectory’s terminal state. Formally, this n-step term expressed by

$$\mathbb{E}_{\tau_{\theta, \xi}} \left[ \sum_{i=0}^{H-1} \gamma^i r_{\xi}(u_{t+i}, z_{t+i}) + \gamma^H V_{\phi}^{\pi}(z_{t+H}) \right] \quad (3.13)$$

where a typical rollout length  $H$  used in our experiments is between 3 and 10. Analogously to the optimization of the critic, this objective can be maximized w. r. t. policy parameters  $\theta$  by backpropagation using the reparameterization trick.

### 3.4. Experimental Platform: Our Drone

We built our own quadrotor (see Figure 3.2) from scratch as learning autonomous flight starting from random exploration places unique demands on the hardware. In particular, we fitted our quadrotor with a robust polyamide frame, allowing it to bump into walls or the ground while staying fully operational. This simplifies initial exploration and deployment of (preliminary) policies. Moreover, this frame also allows for mounting various sensors such as cameras or LiDARs. For this work, the drone is equipped with 24 VL53L1X time-of-flight sensors which are connected via I2C directly to a Raspberry Pi 4 which performs all computing tasks required for our entire model. In practice, the LiDARs

---

---

Table 3.1.: Drone hardware components

Component	Description
<b>Flight Electronics</b>	
Flight Controller	CLRACING F7
Motors	4×1808-2600 kv
ESC	4-In-1 30A with BLHeli-S
Propellers	4 × 4" Bull Nose, 3 Blades
Battery	4S LiPo, 1400 mAh, 65C
<b>Perception &amp; Computation</b>	
Onboard	Raspberry Pi 4
LiDAR	24×VL53L1X
IMU	ICM-20602

give us readings between 0.05–3.50 m and take up to 50 ms to measure. The drone is further equipped with a CLRACING F7 flight controller which comes with an ICM-20602 inertial measurement unit (IMU) that combines a three-axis gyroscope and a three-axis accelerometer. Similar to the LiDARs, the flight controller is connected to the Raspberry Pi. Lastly, the drone frame is fitted with motion capture markers for an external tracking system. Overall the drone weighs 640 g without the battery, which weighs another 181 g. An overview of all key hardware components is given in Table 3.1.

The drone was placed in a 3.0 m × 3.0 m × 2.4 m cage with acrylic glass walls (see Figure 3.5). We artificially limited the drone’s throttle such that it cannot fly higher than the visible foil at the walls which end at 1.2 m. The foil’s main purpose is to improve the LiDAR readings. The cage is fitted with a motion capture system (OptiTrack) which operates at up to 120 Hz. Its data is relayed to the drone over WiFi using Open Robot Communication (ORC) (Frank et al., 2019). An overview of the various components and their communication is visualised in Figure 3.3.

### 3.4.1. High and Low-Level Controller

Our control scheme consists of a high and low-level controller, where only the high-level controller is learned (as described in Section 3.3). The high-level controller is a

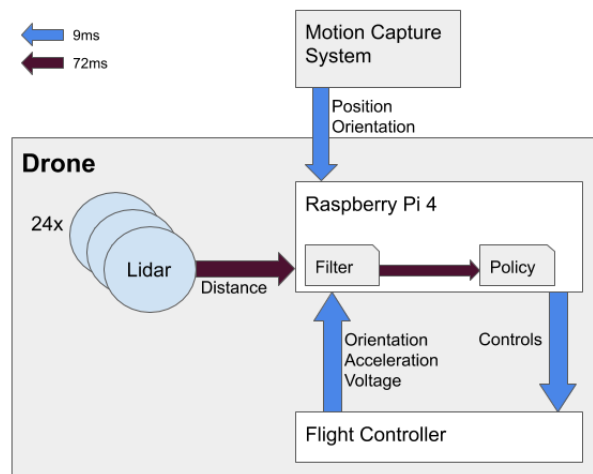


Figure 3.3.: Visualization of communication between sensors, onboard computational hardware and external motion capture system. Motion capture data is sent over WiFi using Open Robot Communication (Frank et al., 2019).

thrust-attitude controller, i. e. it defines throttle, roll, yaw and pitch commands which are then translated to motor torques. For low-level control we use angle mode of Betaflight 3.5.6 (Betaflight, 2020) on our flight controller which keeps the drone hovering when no control inputs are sent. In this mode, roll and pitch inputs are translated into fixed desired angle of the drone and a throttle application of 0 approximately keeps the drone hovering. In practice, variations due to individual battery performance and battery charge require some adjustments from a controller to achieve a stable hovering position. When no (i. e. 0) attitude commands are sent to the flight controller, it will level itself. This behavior is mainly useful for initial exploration as it simplifies the policy design greatly, but is admittedly also helpful for hovering at a goal position after it has been reached. This general setting is identical to how humans typically learn to operate a drone at the beginning.

### 3.4.2. Environment

Since our learned high-level controller sends thrust-attitude commands to the flight controller, there exists an underlying PID controller that translates them into motor currents. This PID controller being limited in its operational speed, we would like to

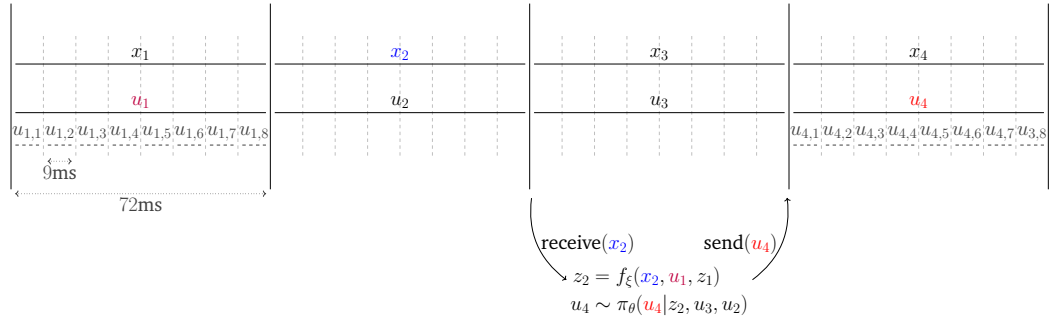


Figure 3.4.: Timing of the control loop. The big intervals denote high-level control where we update our state estimation and compute a new control input based on our learned policy. The smaller 9 ms intervals represent the communication interval with the flight controller. Here, we send a control signal and receive filtered IMU data.

avoid a jittery control signal—something that is not enforced by our Gaussian policy. We therefore introduce a small moving average over the last 6 values (or 54 ms) to smoothen out the control signal. This also ensures that our policy stays within the data distribution of our initial exploration policy which is important for valid model predictions. Further, the control signal has to obey certain control limits of the real system. We enforced this by transforming the control input by a tanh function and then scaling it to the required interval.

The low-level controller runs at 2 kHz while the high-level controller sends control commands every 9 ms. Since computation of our neural state estimation and policy takes longer than 9 ms, we run the high-level controller at 72 ms intervals which predicts the next 8 actions in an open-loop fashion. This control loop is shown in Figure 3.4.

Finally, when applying RL methods, it is often wrongfully assumed that the agent’s state does not change during action selection (Ramstedt and Pal, 2019; Travník et al., 2018; Walsh et al., 2009). In practice, this oversight is often not disastrous as the time required for action selection is negligible. We follow Ramstedt and Pal (2019) which allows an agent exactly one time-step to select an action and where the original MDP is augmented by the previous action(s). Thus, the policy actually takes the form

$$\pi_{\theta}(u_{t+1} \mid z_t, u_t, u_{t-1}). \quad (3.14)$$

This recovers the theoretical applicability of the underlying mathematical framework.



Table 3.2.: Drone sensor streams with  $n_d$  as the number of dimensions of the sensor,  $n_d^{\text{eff}}$  as the effective dimension and  $\Delta t$  as the measurement period in milliseconds.

Sensor Stream	$n_d$	$\Delta t$	$n_d^{\text{eff}}$
LiDAR	24	72	24
IMU orientation	3	9	24
IMU acceleration	3	9	24
Battery Voltage	1	72	1
(Simulated) Compass	1	9	8
Mocap Drone Position	7	9	56
Mocap Drone Velocity	3	9	24
Mocap Goal Marker	7	72	7

### 3.5. Drone Experiments

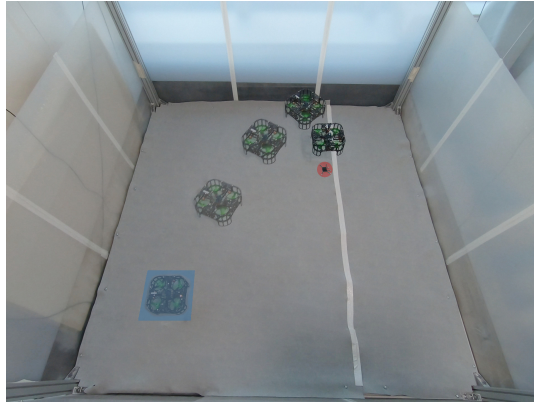
We showcase our methodology in various variations of the same scenario: our quadrotor flying from any position to a randomly placed goal marker in an enclosed environment. This scenario was completed in two settings with either disabled or enabled yaw actuation. In both settings, drones using different subsets of the available sensory information were compared to each other.

#### 3.5.1. Setup

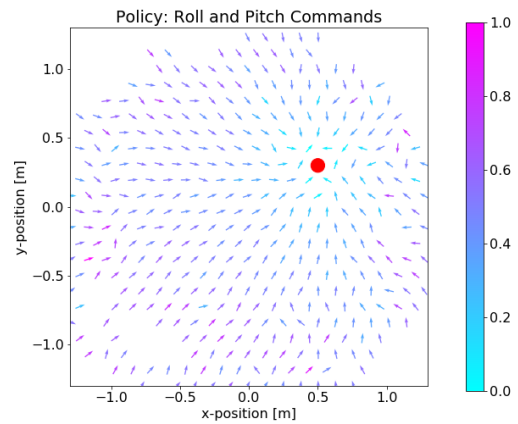
In this section, we introduce various drone configurations, how we collect data and how we preprocess the data for learning.

#### Drone Configurations

We compared three different drone configurations, using only part of all sensory information as listed in Table 3.2. Note that some sensors give measurements more frequently than others (see also Figure 3.3), but our model works at a fixed interval length of 72 ms. If multiple measurements are available, we simply provide our model with all of them. Our three configurations are defined as follows:



(a)



(b)

Figure 3.5.: (a) Shows the drone lifting off and flying to the marked goal in the upper right section of the cage. (b) Shows a corresponding quiver plot looking at the cage from a birds perspective and indicates the policy's roll and pitch commands averaged over a validation data set given the goal position of (a). The arrow's color denotes the relative magnitude of the control input.

- *MocapVel* observes motion capture estimates for position and Cartesian velocity. Velocity is computed using backward differences on filtered motion capture positions.
- *Mocap* observes only motion capture positions, but no velocity. Thus, it only observes a partial state and has to infer its velocity indirectly.
- *LiDARVel* uses only onboard sensors, including all 24 LiDARs and the IMU, but no motion capture position or velocity. However, to make unique and global position identification at all possible, we simulate a compass using our motion capture system, i. e. we provide the drone's z-orientation in the global motion capture frame. Lastly, we supply the model with a LiDAR-based Cartesian velocity estimate based on averaging estimates of LiDARs pointing in the same or opposite direction.

Further, all drones receive a current battery voltage measurement which is updated approximately every 400 ms and the motion capture position of the goal marker as an observation.

---

## Data Collection

Exploration in this setting is a difficult endeavor as we require prolonged actions in one direction for the drone to do something meaningful. Random control commands, as is often done in RL research, would thus lead to the drone mostly sitting on the floor or being stuck at one of the walls. This natural behavior of a random agent in a bounded environment is aggravated by the suction effect near the walls. On the other hand, it is very easy to accelerate the drone to high speeds by excessively tilting in one direction which leads to uncontrollable flight. As both scenarios are undesirable, we start out by collecting an initial data set with a PID controller and continue with the learned policy in subsequent recording steps.

For the initial data set, a high-level PID controller based on the motion capture system flies to randomly chosen targets. To facilitate learning, we update each component of the goal, which consists of  $(x, y, z)$ -position and  $z$ -orientation, independently after a randomly drawn interval length (on average around 1 s). This leads to more decorrelated actuation of the control dimensions when compared to changing the entire goal at once. Note that it is not important for the PID controller to actually reach a target or do a good job at controlling the drone at all. We merely require varied and stable drone flight for our initial data set to learn a first dynamics model. After recording an initial data set, we can record subsequent data using the learned policy. We collected about 10 of overall 30 minutes of flight data using this initial exploration scheme. Maximum velocity reached in any direction during exploration was around  $1.5 \frac{m}{s}$ .

Further, our algorithm allows us to reuse data between drone configurations. In particular, we perform the described data collection loop only using the *MocapVel* configuration and learn the other configurations entirely offline based on the already collected data.

## Data Processing

Minimising engineering effort being a primary motivation of our work, we wanted to limit data processing requirements as much as possible; nevertheless a few processing steps are necessary. As is common when working with neural networks, we normalized all input data to the interval  $[-1; 1]$  based on known sensor limits.

Both the motion capture data and IMU signal are already filtered by their respective sources and we applied no further filtering. The tracking system gives us the orientation of the drone and goal marker in quaternions. Quaternions are difficult for neural networks to learn as there are discontinuities and ambiguity in representation. Instead of learning on them directly, we followed Zhou et al. (2019) and learned on a unique and smooth 6D representation which is based on 2 columns of the corresponding rotation matrix. For

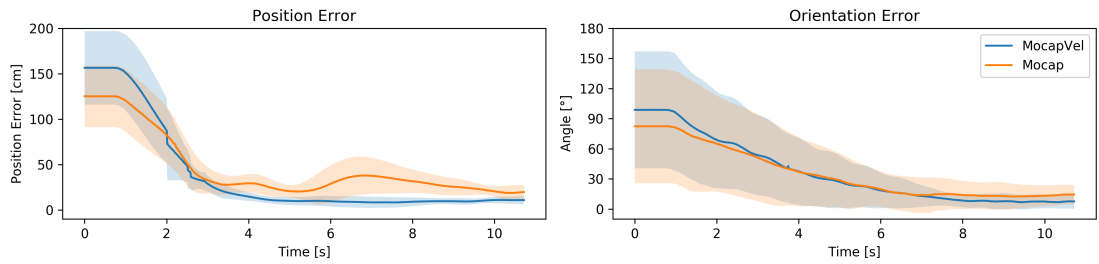


Figure 3.6.: Comparison of the position and z-orientation error over time averaged across all successful test flights. Shaded area marks one standard deviation. One can see how *Mocap* requires a bit longer to settle, oscillating a bit before converging to a slightly larger error than the fully observable *MocapVel* configuration.

the motion capture data, we mapped the drone’s position from the tracking frame to the drone’s frame. This helps the model in so far as the actions and motion capture data are represented in the same frame of reference. The goal marker position was not translated or otherwise processed.

LiDAR data is filtered using a weighted moving average over the last 5 measurements, both for computational simplicity and minimal added delay. Based on this filtered signal, we estimate their rate of change using backward differences over the same window. Then, LiDARs pointing in the same or opposite direction are averaged to compute a Cartesian velocity estimate as we found the use of singular velocity estimates to be too noisy for the model.

For training of the sequential latent variable model, we applied a sliding window on the recorded trajectories, each of which is approximately a minute long, that yields subsequences of ca. 3–6 s duration (i. e. 40–80 discrete time steps). This increases the diversity of observed initial states and accelerates training. For each subsequence, we augmented the data set by overwriting the recorded goal marker position by a randomly drawn one, increasing the diversity of observed goals. Accordingly, we recomputed the observed reward to match the new goal.

### 3.5.2. Fly to Marker

Our goal scenario is for a drone to fly to a marker, i. e. from a standstill, it is supposed to lift off, fly and hover at a desired height over a goal marker that may be put anywhere in the cage. We present results on two variations of this scenario, one simplified scenario where

Table 3.3.: Fly to marker: quantitative results (yaw disabled)

Config.	Success		Avg. Final Error	
	Rate	Time [s]	Pos. [cm]	Orien. [°]
MocapVel	100%	2.5 ± 0.3	6.2 ± 2.4	–
Mocap	100%	7.6 ± 3.2	6.5 ± 3.3	–
LiDARVel	44%	17.2 ± 6.6	7.3 ± 1.5	–

the drone’s yaw actuation was disabled and one where it was not. We do this because some drone configurations were unable to complete the harder scenario successfully.

The global position of the goal marker is observed by the external motion capture system and relayed to the drone. That goal may be treated just as any other observation and be concatenated to the model input. Alternatively, as we do in our experiments, it may be used directly as a condition for the policy and during training, the reward and value function. A scenario is completed successfully if the drone hovers within a certain distance and orientation (in the scenario where yaw actuation is enabled) for a duration of 3 s. Specifically, its distance may not exceed 30 cm and its orientation may not exceed 15 degrees within the time frame. If an episode is successful, we compute the average final error over the following second. These metrics are then used to showcase both the absolute achieved performance and to compare between the different drone configurations.

As a reward function, we define a common one across all scenarios as

$$-(p - p_{\text{goal}})^2 - 0.1d(\theta_z, \theta_{\text{goal}})^2 - 0.1v^2 - 0.001u^2, \quad (3.15)$$

where  $p[\text{m}]$  is the drone’s Cartesian position in the global tracking frame,  $\theta_z[\text{rad}]$  its z-orientation,  $v[\text{m/s}]$  its Cartesian velocity and  $u$  the normalized control input. The quantities  $p_{\text{goal}}$  and  $\theta_{\text{goal}}$  refer to the position and orientation of the goal marker. Function  $d(\theta_z, \theta_{\text{goal}})$  computes the difference between the two angles. This last term is omitted for the scenario with disabled yaw actuation. In the other scenario, the goal orientation is fixed and not dependent on the goal marker’s orientation. If successful, the white stripe of the drone should always face upwards in the end (see Figure 3.5).

### Disabled Yaw

In this simplified scenario, the drone’s yaw actuation was disabled and it was placed axis-aligned in the environment, i. e. facing directly in one of the four cardinal directions.

Table 3.4.: Fly to marker: quantitative results (yaw enabled)

Config.	Success		Avg. Final Error	
	Rate	Time [s]	Pos. [cm]	Orien. [°]
MocapVel	100%	6.1 ± 2.8	4.4 ± 1.7	6.9 ± 2.8
Mocap	67%	10.5 ± 4.5	5.2 ± 2.1	7.8 ± 3.7

An example of this scenario is shown in Figure 3.1. We report our main metrics in Table 3.3 which were averaged over 9 different start and goal positions. The same start and goal positions were used for each drone configuration.

We see that *MocapVel* is successful in all cases, achieving success much faster and with lower error than the other two configurations. Without an observed velocity (*Mocap*), the learned controller is still good enough to complete the task, albeit it requires significantly more time and concludes with a slightly higher final error. Having no velocity observation, it has a much harder time to find a perfect stand still above the goal and oscillates around the goal. This behavior can readily be seen in Figure 3.7, displaying the remaining error over time of successful trajectories.

Lastly, while *LiDARVel* possesses a velocity estimate, both its position and velocity measurements are much more noisy than the motion capture data. Thus, it represents the hardest setting as can be inferred by its lower rate of success. Here, we found that our learned controller in all our attempts only flew well while oriented in some of the four possible fixed orientations. The underlying issue we suspect to be related to the translation from its local position measurement to the global goal frame, but we were unable to resolve it. Other failures take the form of oscillating around the goal, leaving the desired goal region periodically within 3s. In general, it took much longer until the drone stayed within the defined goal region for a sufficient amount of time and even drifted away from it after achieving a momentary standstill. We also tried a fourth configuration *LiDAR* where we removed the velocity estimate from the observation and tried to control based on LiDAR-sensors alone, but were unable to complete the scenario.

### Activated Yaw

When activating yaw actuation, we were only able to get results in the *MocapVel* and *Mocap* configuration. An example of a successful completion of this scenario is shown in Figure 3.5, where the drone now also has to rotate to specific z-orientation. Analogously

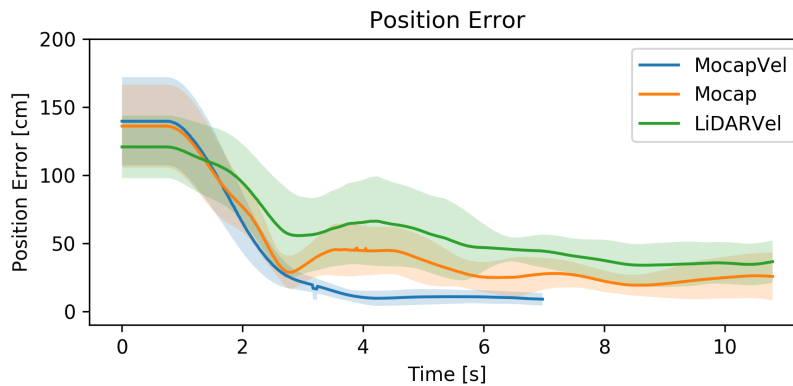


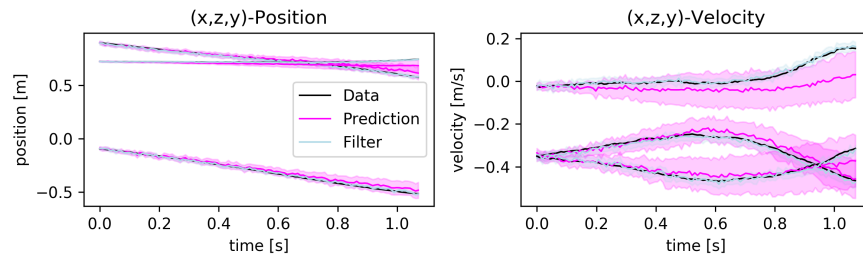
Figure 3.7.: Comparison of position error over time in the scenario where yaw actuation is disabled. One can see the greater oscillation of *Mocap* and *LiDARVel* compared to the perfect and complete information provided in *MocapVel*. Nevertheless, they do eventually converge to the desired position within a reasonable tolerance level.

to before, we report final errors in Table 3.4 and show the remaining error of position and orientation over time in Figure 3.6. Here, we can see that *MocapVel* is still successful in all cases, while *Mocap* is only successful in two thirds of the scenarios. The failures were mainly due to the drone not rotating all the way to the required goal orientation and rarely due to excessive oscillation around the goal position. Again, we can see that *Mocap* takes a bit longer to settle at the goal, with slightly higher final errors in both position and orientation. To get a better impression of the actual performance, we recommend taking a look at the video ([https://youtu.be/e5buJL\\_DYgA](https://youtu.be/e5buJL_DYgA)) where we show both successful and failed episodes of these various sensors configurations.

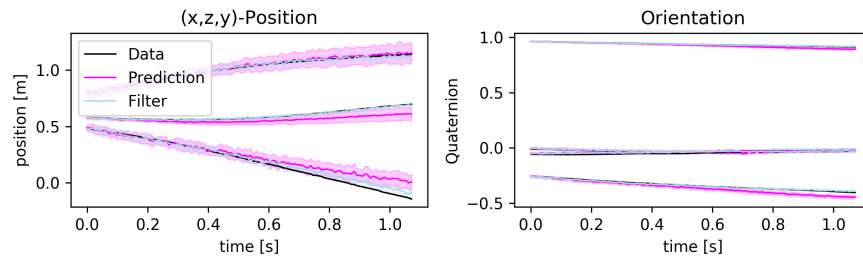
Lastly, in Figure 3.8 we highlight our model’s predictive and filtering performance in various drone configurations. Hyperparameters for all drone configurations are shared and can be found in Table A.4 in the Appendix.

### 3.6. Robot Arm Experiments

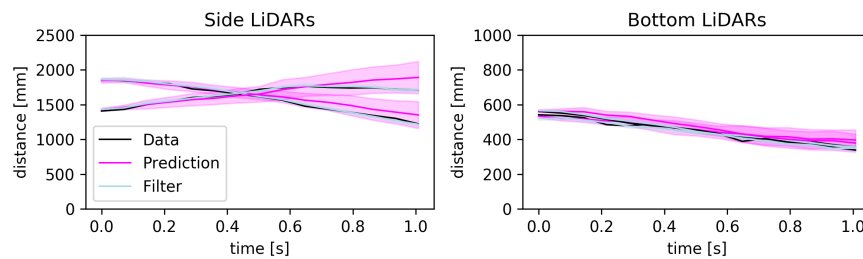
To showcase more general applicability and to compare to a model-free method, we perform some more experiments on an entirely different platform. The Franka Emika Panda robot is a 7 DoFs robot arm intended to be used in a human robot collaboration



(a) *MocapVel*



(b) *Mocap*



(c) *LiDARVel*

Figure 3.8.: These plots showcase the filtering and predictive performance of our probabilistic model. The black line denotes the data, the magenta line denotes the prediction average and its shaded area marks one standard deviation based on 20 sampled trajectories. For predictions, we condition on a filtered starting state and future control inputs and then let it simulate without further feedback for 1s. The cyan line denotes the state estimation of our filter.

environment. For our purposes, the scenario was to move the robot's end effector to a randomly sampled position and orientation within a certain range using joint velocity control



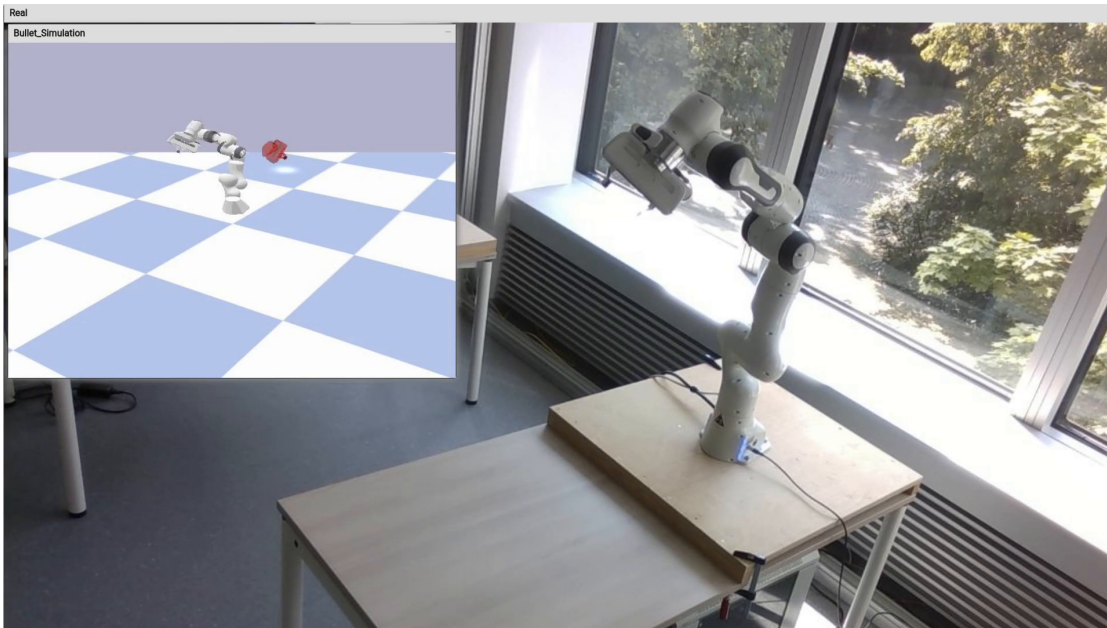


Figure 3.9.: Our Panda task environment. The arm is in a random starting position and the desired end effector position and orientation is visualised in translucent red in the small rendered frame. The target is also randomly sampled within some constraints.

(see Figure 3.9). Again, this should be achieved without providing prior information about the kinematic or dynamic behavior and by directly learning on the real robot instead of a simulation. We compare our method to SAC (Haarnoja et al., 2018), a state of the art model-free RL algorithm, and also perform two ablation studies: replacing our variational LSSM (which we refer to as SLDS-DVBF) with a recurrent neural network (RNN), and replacing our parameterized critic by a simple Monte Carlo estimation of the remaining rewards. Note that we also tried to perform these model-based ablation studies on the drone, but were unsuccessful in getting any quantifiable results.

### 3.6.1. Environment

Both the goal and initial robot configuration are sampled from two different ranges. The goal end effector position is sampled from a semi circle doughnut around the robot's base. It is at least 30 cm and at most 80 cm away, and between 20 cm and 80 cm high.

Table 3.5.: Ranges used for normalization of robot arm observations.

Observation	Range
<b>Joint Angle</b>	
$j_1$	[ -2.8973; 2.8973 ]
$j_2$	[ -1.7628; 1.7628 ]
$j_3$	[ -2.8973; 2.8973 ]
$j_4$	[ -3.0718; -0.0698 ]
$j_5$	[ -2.8973; 2.8973 ]
$j_6$	[ -0.0175; 3.7525 ]
$j_7$	[ 2.8973; 2.8973 ]
<b>Endeffector position</b>	
x, y	[ -1; 1 ]
z	[ 0; 2 ]

The orientation is rotated from its neutral position (pointing down) by up to 45 degrees independently around all 3 axes in either direction. All values are sampled uniformly from these ranges for each rollout. Lastly, we ensure that the sampled goal is actually reachable and re-sample if it is not. For starting position, the bottom joint is sampled from  $[-1.5; 1, 5]$  rad, approximately covering the semi circle, all other joints are sampled within 0.2 rad of their neutral position. This mainly simplifies initial exploration, but also requires a more versatile controller that can handle random starting positions as compared to a single neutral position. As observations, we measure the integrated joint angles (but no velocities) and the end effector position and orientation.

### 3.6.2. Setup

Just as in the drone experiments, we transform the end effector orientation given in quaternions into a smooth 6D representation. Angles, end effector position and orientation are clipped at their minimum and maximum ratings and are normalized to be in the interval  $[-1; 1]$  based on Table 3.5. Importantly, different to our drone experiment, the initial dataset here is recorded by simply executing actions from a randomly initialized policy instead of an engineered exploration policy.

Table 3.6.: Comparison of results in the Panda arm environment after 100 minutes of interaction with the environment. Displayed is the achieved average performance over 100 rollouts of 5s length.

Algorithm	Avg. Reward
MBAC+SLDS-DVBF (Ours)	-166.5
MBAC+SLDS-DVBF, Monte Carlo Critic	-168.6
MBAC+RNN	-179.5
SAC	-192.6

Our reward function takes the form

$$-(p - p_{\text{goal}})^2 - d(\theta_z, \theta_{\text{goal}})^2 - 0.25u^2 - (u_t - u_{t-1})^2. \quad (3.16)$$

$p$  [m] is the Cartesian position of the end effector of the robot arm,  $\theta$  [rad] its orientation and  $u$  is the control input (velocity of the joints in [m/s]). All values are normalized for reward computation. To compute a scalar for the orientation error  $d(\theta_z, \theta_{\text{goal}})^2$ , the distance is computed in the smooth 6D space simply using the mean squared error. The last term rewards smoothness of the trajectory which we found necessary to run the otherwise jerky policy on a real robot arm without damaging it.

Our learned controller operates at 10 Hz where it sends joint velocity commands and receives observations. The environment is automatically terminated and reset after 5 s (50 steps) or just before it would collide with anything in the workspace to keep the robot safe. We limited the overall interaction time to 100 minutes (60,000 steps) and spread our rollouts across 3 identical robot arms. To account for partial observability (since velocities are unobserved), we stack the last 3 observations for the model-free SAC method.

### 3.6.3. Results

Our main results are displayed in Table 3.6. There, we compare the achieved average reward over 100 rollouts of 5 s length for all 4 methods. We find that our approach works the best, but is virtually matched by the Monte Carlo critic – very different to our drone experiments where we could not make this approach work at all. We believe this is explained by a comparatively very accurate predictive model that we can learn in this setting as the robot arm’s dynamics and its sensors are less chaotic and noisy. Also, the action space is simpler, allowing direct control of the velocity instead of just the attitude

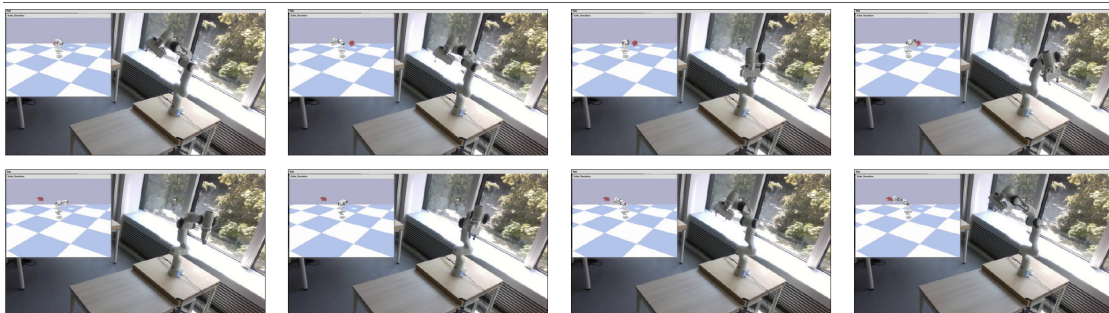


Figure 3.10.: Snapshots of two exemplary rollouts of the final policy.

of the drone. When we replace our variational LSSM with an RNN, we observed a steeper drop off in performance and SAC performs worse than all the model-based approaches. Note that this level of performance is not the best that can be reached with these methods, but it represents the performance after 100 minutes of interaction (we found in simulation that all methods would continue to improve with more data). SAC performing worse than model-based approaches goes hand in hand with other work (e. g. (Hafner et al., 2020)) finding model-based approaches to be more data efficient in general.

Figure 3.10 shows two rollouts using our approach, for a visual comparison of all methods we recommend to take a look at our video (<https://youtu.be/huW0LzU5D0k>).

### 3.7. Related Work

The potential of classical reinforcement learning methods has already been demonstrated by learning autonomous helicopter flight (Bagnell and Schneider, 2001; Ng et al., 2006; Abbeel et al., 2007). Later on, deep learning methods enabled further applications. In particular, neural networks helped enhance and support classical PID controllers. The Neural Lander (Shi et al., 2019) achieves stable drone landing by approximating higher level dynamics, such as air disturbances close to the ground, with a neural network. Kaufmann et al. (2018) uses imitation learning to train an onboard CNN to predict waypoints on a drone racing track. Loquercio et al. (2019) extends this work, dealing with a changing track (e. g. moving gates the drone has to pass through). Li et al. (2018) has a quadrotor following a person based on a camera input. They use model-free reinforcement learning to give goals to a low-level PID controller which translates these into motor commands.

Instead of using a PID controller to steer towards the goal, some approaches use neural

---

---

networks to predict either thrust-attitude controls or motor commands directly. Imitation learning was successfully employed in this fashion to train a thrust-attitude controller that is capable of flying through a narrow gap (Lin et al., 2019). Lambert et al. (2019) replaced the PID controller with MPC in a learned model using onboard sensors only and hence achieved hovering capability for up to 6 seconds. Neuroflight (Koch et al., 2019) suggested to replace the engineered PID controller of Betaflight (Betaflight, 2020) with a learned one that can adapt to the current state of the aircraft. Probably closest to us is pioneering work by Hwangbo et al. (2017) where they also learn a neural network controller that directly computes motor torques based on state inputs. They manage to stabilize a quadrotor from random starting conditions even without using an underlying PID controller (for the final policy, they do use one during training/exploration) as in our case. However, they achieve this by using a predefined instead of learned model for simulation.

Model-free deep RL has received a tremendous amount of attention ever since  $Q$ -learning was successfully applied to playing Atari games directly from raw input images with the use of deep convolutional neural networks (Mnih et al., 2015). One key ingredient to its success was its stabilization of its neural network based  $Q$ -values using a slowly updating target network. Work on further stabilizing deep  $Q$ -learning and coping with its other shortcomings like its overestimation bias has since been central to further inquiry (Wang et al., 2016; Van Hasselt et al., 2016; Lillicrap et al., 2016; Fujimoto et al., 2018). Much of the early deep RL work being limited to discrete action spaces, DDPG (Lillicrap et al., 2016) revives the idea of deterministic policy gradient and hence, extends  $Q$ -learning to continuous action spaces. A3C (Mnih et al., 2016) takes an Actor-Critic approach and promotes parallelised asynchronous updates. Noticeably, the paper also explores  $n$ -step value updates, similarly to our value-update schema in Section 3.3.1.

Changing focus to MBRL, learned models have been used for forward planning (Chua et al., 2018; Haarnoja et al., 2019) which have recently closed the gap to model-free methods in simulated physics environments while being more sample efficient. Looking at gradient-based approaches, Stochastic Value Gradients (Heess et al., 2015) shows how to use a model just for gradient computation of real-world trajectories by backsolving for noise variables. Model-based value estimation (Feinberg et al., 2018) uses short imagined rollouts within the dynamics model to improve the value estimation. Imagined value gradients (Byravan et al., 2019) use the dynamics model’s gradient to optimize the value function. Concurrent to our work, Hafner et al. (2020) demonstrates an optimization scheme similar to ours with great success directly on pixel inputs of simulated environments. Morgan et al. (2021) proposes a hybrid approach, leveraging model learning and optimized model predictive rollouts as the data source for policy optimization in a Soft Actor-Critic algorithm.

---

---

In robotics applications, end-to-end training of deep visuomotor policies was first shown to be feasible in Levine et al. (2016). Zhang et al. (2019) demonstrated MBRL directly on images of a robotic arm for Cartesian velocity control on various manipulation tasks. Haarnoja et al. (2019) taught a real-world Minitaur robot to walk in 2 hours using a slightly modified Stochastic Actor Critic method (Haarnoja et al., 2018). Andrychowicz et al. (2020) and Akkaya et al. (2019) learn dexterous in-hand manipulation in a simulator and deploy on a Shadow Dexterous Hand that is robust to various (previously unseen) disturbances. Closely related to us, Piergiovanni et al. (2019) optimized a real-world policy by dreaming in a learned, but deterministic dynamics model.

Watter et al. (2015) started by learning a locally linear LSSM using neural variational inference, but they learn on pairs instead of sequences and do not optimize a lower bound of  $p(x)$ . Krishnan et al. (2017) and Karl et al. (2017) showed how to learn LSSMs using neural variational inference directly on data sequences using a time-factorized Evidence Lower Bound (ELBO), the latter pointing out the importance of propagating the reconstruction error through the transition model for improved learning of dynamics. Fraccaro et al. (2017) showed how to combine neural variational inference with an underlying Kalman smoother in latent space. Before LSSMs, some methods used sequential VAEs for augmenting recurrent neural networks with a stochastic component allowing for multi-modality (Bayer and Osendorfer, 2014; Chung et al., 2015). A multi-stage training process was used in world models Ha and Schmidhuber (2018) to learn a controller for game scenarios based on image observations. Aside from latent variable models, there have also been efforts to inform or restrict deep learning methods using our physical understanding of the world (Raissi, 2018; de Avila Belbute-Peres et al., 2018; Lutter et al., 2019). Robot arm dynamics have been successfully learned with Graph Neural Networks by Sanchez-Gonzalez et al. (2018). Using this model they perform model predictive control on desired velocities to solve a similar task as ours both in simulation and on a real robot arm.

### 3.8. Conclusion

Even without encoding any physics knowledge into the latent-state space, we have shown how to learn a drone dynamics model from raw sensor observations. This model is good enough to learn a controller entirely in simulation that can be executed on a real drone. The controller is capable of flying the drone to observed goal positions; in the case without yaw actuation even using only onboard sensors while the goal is specified in a global coordinate frame. The optimization scheme is based on stochastic analytic gradients enabled by implementing model, actor and critic as differentiable function approximators.

---

---

To our knowledge, our work is the first MBRL approach deployed on a real drone where both the model and controller are learned by deep learning methods. The methodology was showcased on various drone configurations, but is applicable more broadly to other robotic setups without the need for algorithmic changes. We demonstrated this in a robotic arm scenario where we highlighted its data efficiency when compared to a model-free method. Thus, we believe that this represents an important step towards learning robots from the ground up using minimal engineering.

However, there still remain some small engineered parts which prevent our method from being truly robot-agnostic such as the reliance on a stabilizing PID controller for exploration or performing thrust-attitude control instead of operating directly on motor currents. The gap in performance to more engineered methods is still readily apparent, but we hope that this contribution can inspire more work in the direction of combining machine learning and control.





---

## 4. Exploration via Empowerment Gain: Combining Novelty, Surprise and Learning Progress

---

In RL we tend to give an agent a specific task to solve, and use exploration heuristics to speed up training (Thrun, 1992; Arulkumaran et al., 2017). While these heuristics may be useful, biological systems have a more efficient approach. Even when faced with no concrete task, natural autonomous agents (like children) explore the world playfully to acquire new skills that may be used later (Chu and Schulz, 2020). This exploratory behavior is an autonomous and active endeavor guided by intrinsic motivation which forms the core of a system for task-independent learning (Oudeyer et al., 2007; Oudeyer and Kaplan, 2009).

Intrinsic motivations have been formalized in RL literature in various ways (Aubret et al., 2019). In particular, the concepts of *novelty* and *surprise* are believed to be vital to exploration (Berlyne, 1960; Barto et al., 2013). We follow a common distinction (Berlyne, 1960; Barto et al., 2013; Xu et al., 2021) in defining the key difference of the terms, although the formalization of novelty is particularly difficult and controversial. Following common intuition, novelty may be defined as being a statistical outlier; something is completely novel if we have not seen it before. One way this has typically been formalized in machine learning (Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017) is

$$\text{Novelty}(s) \propto -\log p(s), \quad (4.1)$$

where  $p(s)$  models the visitation frequency of state  $s$  based on previous experience.

However, not everything that is novel is necessarily surprising. And complex states may reveal something surprising even after having encountered them often before. Surprise (also called *contextual novelty* or *curiosity*) requires an internal world model that formulates an expectation about the future. The deviation between these predictions and the observed reality quantifies the amount of surprise. One way to formalize this is to use a forward model  $p_{\xi}(s_{t+1} | s_t, a_t)$  (parameterized by parameters  $\xi$ ) for computing the inverse likelihood of the observation  $s_{t+1}$  (Schmidhuber, 1991a; Lopes et al., 2012; Stadie

et al., 2015; Achiam and Sastry, 2017; Pathak et al., 2017)

$$\text{Surprise}(s_{t+1} | s_t, a_t) \propto p_{\xi}(s_{t+1} | s_t, a_t)^{-1}. \quad (4.2)$$

But just seeking novelty or surprise faces a problem. Consider static TV noise: it is highly entropic and unpredictable, hence remains highly surprising and novel. To address this, it has been suggested to consider an agent’s *learning progress* (Schmidhuber, 1991a; Storck et al., 1995; Lopes et al., 2012; Achiam and Sastry, 2017). Since collecting more data in this environment will not lead to better prediction of the noisy observation, an explorer should avoid these areas. We should rather track and optimize the learning progress of the agent, which has been defined as

$$\log p_{\xi'}(s_{t+1} | s_t, a_t) - \log p_{\xi}(s_{t+1} | s_t, a_t) \quad (4.3)$$

where  $\xi'$  and  $\xi$  are the updated and old parameters, after and before observing some new data. A related formulation of learning progress can be derived from a Bayesian definition of surprise (information gain, see Section 4.1.2) that similarly has been used for exploration (Houthoofd et al., 2016).

While learning progress is an important insight, we would like to offer an alternative reason as to why the noisy TV screen is uninteresting for an exploring agent: it is really the failure of increasing its capability to interact with the world. While improving one’s world model is an absolutely vital part of that process, it alone is not sufficient – the goal is not just to become a perfect simulator. Even a predictable pattern (e. g. a movie) remains uninteresting without meaningful interaction (e. g. TV remote) or without informing the agent about how to act in the world. As another example, consider an agent exploring a chair. It may not be necessary to predict every nuance of its physical appearance, but it is crucial to find out about its important practical uses such as exploring which surface is suitable for sitting.

One formalism to measure an agent’s capability to interact with the world is called *empowerment* (Klyubin et al., 2005a,b). It is an information-theoretic concept that measures the maximal flow of information over an agent’s perception–action loop. It is formally defined as the channel capacity (a tight upper bound on the rate of information that can be transmitted over a channel) between reachable terminal states  $S'$  and the possible action sequences  $A_{1:T}$  over a horizon  $T$ , that is

$$\mathcal{E}^T(s) = \max_{A_{1:T}} \mathcal{I}(S', A_{1:T} | s) = \max_{A_{1:T}} [\mathbb{H}(S' | s) - \mathbb{H}(S' | s, A_{1:T})] \quad (4.4)$$

where  $\mathcal{I}$  denotes the mutual information and  $\mathbb{H}$  denotes entropy. Intuitively, an agent is empowered if it can predictably reach many states (high  $\mathbb{H}(S' | s)$ , low  $\mathbb{H}(S' | s, A_{1:T})$ ),

---

---

and it has low empowerment if its actions have little to no perceived influence on the world (high  $\mathbb{H}(S' | s, A_{1:T})$ ).

In this work, we propose a novel criterion for exploratory behavior in autonomous agents which we call *Empowerment Gain (EG)*. Its goal can be stated as follows: in the absence of a concrete task, an agent should take those actions in the environment which provide the most information for improving its empowerment estimator, i. e. it should take those actions that maximize the increase in perceived influence over its environment after observing new data. *New experiences should help me recognize my capability to interact with the world.* We show how EG combines and puts into relation common notions of novelty seeking, surprise maximization and learning progress in a single formulation. This formulation takes into account an agent’s limitations in actuation and sensation as well as inherent stochasticity of the environment. We provide a number of illustrative experiments in simple and discrete environments which support our theoretical findings and provide some more intuition about the EG criterion. In particular, we show how EG guides an agent towards those areas of the state and action space where it has more potential for improving its influence in the world. Since EG considers an extended time horizon  $T$  instead of a single time step, we show that it tends to avoid inescapable traps in the environment that limit its future control. Both of these behaviors cannot be achieved by novelty seeking, surprise maximization or learning progress alone.

## 4.1. Background

As a necessary background, we introduce the basic formulation and concepts of empowerment. Then, we discuss ways to quantify information gain. A combination of these two concepts leads to our proposed method later on.

### 4.1.1. Empowerment

*Empowerment* (Klyubin et al., 2005a,b; Salge et al., 2014) is a measurement of an agent’s control over its perceived environment. It is defined in terms of an agent’s embodiment; the coupling of sensors and actuators via the environment (perception–action loop).

**Definition 1** *Empowerment for state  $s$  is defined as the channel capacity between terminal*

state distribution  $S'$  and the possible action sequences  $A_{1:T}$  over a time horizon  $T$

$$\mathcal{E}^T(s) = \max_{A_{1:T}} \mathcal{I}(S', A_{1:T} \mid s) \quad (4.5)$$

$$= \max_{A_{1:T}} [\mathbb{H}(S' \mid s) - \mathbb{H}(S' \mid s, A_{1:T})] \quad (4.6)$$

$$= \max_{A_{1:T}} [\mathbb{H}(A_{1:T} \mid s) - \mathbb{H}(A_{1:T} \mid s, S')] \quad (4.7)$$

where  $\mathcal{I}$  denotes mutual information and  $\mathbb{H}$  denotes entropy.

Interpreting the environment as a communication channel, the agent finds a *source distribution* over action sequences  $A_{1:T}$  such that its sensors at time  $T + 1$  can recover the most information about them. Writing the mutual information as differences of entropy terms gives an intuitive understanding. Interpreting eq. (4.6), empowerment finds balance between diversity and predictability. It wants to maximize the diversity of reachable states (entropy of final states  $S'$ ) while still being able to predict the outcome when conditioned on the taken action sequence. In highly stochastic environments, our empowerment will always be relatively small, because even though we (accidentally) reach a lot of states, we can not predict the outcome. By symmetry of mutual information, in eq. (4.7) we want to maximize the diversity of action sequences, but every action sequence should ideally lead to a unique final state  $s'$  such that the action sequence can be recovered from first and final state. For discrete and deterministic environments, empowerment simplifies significantly and becomes proportional to the number of reachable states within a horizon  $T$  and the source distribution learns how to reach those states uniformly. In this sense, we can view an action sequence  $a_{1:T}$  as a skill that transforms the world state from  $s$  to  $s'$  and our source distribution as a distribution over those skills.

#### 4.1.2. Information Gain

Given a model parameterized by  $\theta \in \Theta$  of random variable  $\Theta$ , we are interested in selecting data  $d$  that is maximally informative. To quantify this, one can define various measure of *information gain (IG)*, two of which we introduce here. First, one can look at the difference in entropies of parameters before and after observing new data  $d$ ,

$$\mathcal{IG}(\Theta, d) = \mathbb{H}(\Theta) - \mathbb{H}(\Theta \mid d). \quad (4.8)$$

The greater the reduction in uncertainty, the more information we have gained about our parameters  $\theta$ . Alternatively, in a Bayesian inference setting one typically defines

$$\text{KL}(p(\theta \mid d) \parallel p(\theta)). \quad (4.9)$$

as a measure of the information gained by revising one's beliefs from the prior distribution  $p(\theta)$  to the posterior distribution  $p(\theta | d)$ . These two measures are equal in expectation when considering new data  $d$  (MacKay, 1992). Information gain is another common way to formalize and quantify surprise (e. g. when  $\theta$  are the parameters of a world model) and is aptly named Bayesian surprise (Itti and Baldi, 2005, 2006, 2009). Different to the definition of surprise in eq. (4.2), this definition is directly related to an agent's learning progress.

## 4.2. Empowerment Gain (EG)

In this section, we develop an exploration criterion based on expected improvement of an agent's empowerment estimator. We show how it encapsulates and relates to other intrinsic motivations such as novelty seeking, surprise maximization and learning progress. We start out by looking at the components that make up empowerment estimation starting from the well-known equations for the *model-free* setting

$$\hat{\mathcal{E}}_{\theta}^{\text{MF},T}(s) = \max_{\omega_{\phi}(a_{1:T}|s)} \mathbb{E}_{\omega_{\phi}(a_{1:T}|s)p(s_{T+1}|s,a_{1:T})} [ \log p_{\psi}(a_{1:T} | s, s_{T+1}) - \log \omega_{\phi}(a_{1:T} | s) ], \quad (4.10)$$

using real world transition  $p(s_{T+1} | s, a_{1:T})$  or *model-based* setting

$$\hat{\mathcal{E}}_{\theta}^T(s) = \max_{\omega_{\phi}(a_{1:T}|s)} \mathbb{E}_{\omega_{\phi}(a_{1:T}|s)p_{\xi}(s_{T+1}|s,a_{1:T})} [ \log p_{\xi}(s_{T+1} | s, a_{1:T}) - \log p_{\phi,\xi}(s_{T+1} | s) ]. \quad (4.11)$$

using an approximated transition model  $p_{\xi}(s_{T+1} | s, a_{1:T})$ . These estimations consists of various familiar distributions, the *source distribution*  $\omega_{\phi}(a_{1:T} | s)$ , the transition or *forward model*  $p_{\xi}(s_{T+1} | s, a_{1:T})$ , the *inverse model* or *planning distribution*  $p_{\psi}(a_{1:T} | s, s_{T+1})$  and the *final state marginal* distribution  $p_{\phi,\xi}(s_{T+1} | s)$  where we have split our parameters  $\theta = \{\phi, \xi, \psi\}$ .

The idea of empowerment gain (EG) is that an agent should act such that it maximizes its expected improvement of its empowerment estimator. Concretely, in any given state  $s$ , it should perform action  $a^*$  such that it maximizes its expected improvement of its empowerment estimator  $\hat{\mathcal{E}}_{\theta}^T(s)$  after updating the parameters  $\theta$  using the newly collected data  $d$ . Formally, the objective is defined as

$$a^* = \arg \max_a \mathbb{E}_{d=(s,a,s' \sim p(s'|s,a))} [ \hat{\mathcal{E}}_{\theta'}^T(s) - \hat{\mathcal{E}}_{\theta}^T(s) ] \quad (4.12)$$

where  $\theta'$  are the updated parameters after observing new data  $d$ . Next, by rewriting this objective in various ways, we develop a deeper understanding of the EG objective and discover that other intrinsic motivations such as novelty seeking and learning progress of a forward or inverse model are contained as part of the EG objective in a specific way.

### 4.2.1. Model-Free Empowerment Gain

Let us first rewrite EG in the model-free setting that is broadly applicable to any parameterized estimate of one's empowerment. We can define EG as

$$\begin{aligned} \hat{\mathcal{E}}_{\theta'}^{\text{MF},T}(s) - \hat{\mathcal{E}}_{\theta}^{\text{MF},T}(s) &\approx \mathbb{H}(A_{1:T}^{\phi'} \mid s) - \mathbb{H}(A_{1:T}^{\phi} \mid s) \\ &+ \mathbb{E}_{\omega_{\phi'}^*(a_{1:T}|s)p(s_{T+1}|s,a_{1:T})} [ \\ &\quad \log p_{\psi'}(a_{1:T} \mid s, s_{T+1}) - \log p_{\psi}(a_{1:T} \mid s, s_{T+1})] \\ &\text{where } A_{1:T}^{\phi'} \text{ is defined by pdf } \omega_{\phi'}(a_{1:T} \mid s) \end{aligned} \quad (4.13)$$

where the approximation is relatively tight for small updates of the source parameters  $\phi$  (for derivation see the supplementary material). Note that we introduce this approximation mainly for didactical reasons to better grasp the components that make up EG. While this approximation may also be useful to derive an efficient exploration algorithm later on, this is not the focus of this work and we compute the exact EG value for our experiments in Section 4.3. We see that the empowerment estimate can be improved in two distinct ways.

First, by increasing the entropy of the empowerment-realizing source distribution  $\omega_{\phi'}(a_{1:T} \mid s)$ , which should be a highly entropic action distribution of skills (action sequences). The second part of the equation describes the learning progress of the inverse model. This ensures that skills are distinguishable by their outcome/final state  $s'$ . Finding novel ways to do something that we can already do is not enough. Importantly, the learning progress of the inverse model is taken in expectation over our learned skills, meaning the model quality outside of the space induced by our learned skills is of no relevance. This is desirable because we are not interested in learning a perfect (inverse) model of everything, but learn how to act in the environment. These two terms form a trade-off where more diverse actions are only helpful if they lead to distinguishable outcomes and a perfect inverse model alone is not worth much if we are limited to a small number of skills.

## 4.2.2. Model-Based Empowerment Gain

While the previous model-free formulation is theoretically interesting, practically it is not very useful as one needs to sample from the real world for its estimation. Hence, let us now consider a different formulation where we formulate EG estimated in a parameterized and learned model  $p_\xi(s_{T+1} \mid s, a_{1:T})$ . We define model-based EG as

$$\begin{aligned} \hat{\mathcal{E}}_{\theta'}^T(s) - \hat{\mathcal{E}}_\theta^T(s) &\approx \mathbb{H}(S_{T+1}^{\phi', \xi'} \mid s) - \mathbb{H}(S_{T+1}^{\phi, \xi} \mid s) \\ &+ \mathbb{E}_{\omega_{\phi'}^*(a_{1:T} \mid s)}[\text{KL}(p_{\xi'}(s_{T+1} \mid s, a_{1:T}) \parallel p_\xi(s_{T+1} \mid s, a_{1:T}))] \end{aligned} \quad (4.14)$$

where  $S_{T+1}^{\phi', \xi'}$  is defined by pdf

$$\int \omega_{\phi'}^*(a_{1:T} \mid s) p_{\xi'}(s_{T+1} \mid s, a_{1:T}) da_{1:T}$$

where the approximation is relatively tight for small updates of  $\phi$  and  $\xi$  (for derivation see Appendix A.3.1). Again, we see that the empowerment estimate can be improved in two distinct ways.

First by (potential) novelty seeking, i. e. by increasing reachable state entropy, which can be achieved by either detecting new states or by realizing how to reach them more uniformly. Note that we are not strictly novelty seeking as we do not directly act according to  $\omega_\phi(a_{1:T} \mid s)$ , rather we want to realize that we could maximize novelty, if we wanted to – hence the term potential novelty seeking. Whether actually visiting all reachable states uniformly or not is the best way to achieve this goal is then up to the agent and the learning procedure.

Second, by improving the forward model (learning progress). Different to other methods focusing on learning progress on a forward model, the divergence between current and past forward model is put into context by the empowerment-realizing source distribution  $\omega_{\phi'}(a_{1:T} \mid s)$ . That is, the improvement of the forward model is weighted in so far as it helps the agent realize its influence in the world. So even if learning progress of a forward model is large, if it does not help the agent in attaining states more reliably, EG may still be small. Inversely, a small update in the forward model can still lead to a large EG if the improved transition is central to many skills (samples of  $\omega_{\phi'}(a_{1:T} \mid s)$ ). Jointly optimizing these two terms results in a trade-off between novelty seeking and learning progress.

A drawback of optimizing empowerment within a model is that the empowerment estimate is no longer bounded by the empowerment value of the correct dynamics. The model may become overly optimistic, leading to high empowerment estimates and avoiding counterfactuals that disprove its perceived empowerment. If one assumes an uninformed Bayesian model, this problem vanishes in deterministic, but not necessarily in stochastic,

---

---

settings. To address this, an alternative heuristic would be to use the absolute value of the difference in eq. (4.12) as the objective instead, encouraging just a big change in one’s empowerment estimate. Having stated this potential shortcoming, we did not find this to be problematic in our experiments. Note that this problem is also faced in a way by the non-Bayesian formulation of learning progress (eq. (4.3)).

Also, while our discussion has been entirely in terms of states and actions, EG is readily applicable to partially observable environments as discussed in Appendix A.3.5.

### 4.3. Experiments

In our experiments, we investigated the exploratory behavior of an EG-maximizing agent (as discussed in Section 4.2.2, but we compute the exact value of EG rather than the introduced approximation) in various simple but illustrative grid world scenarios. We looked at the effect of various types of action noise, the shape of the state space, traps (sinks in the state space the agent cannot get out of) and empowerment’s computation horizon. We compared this to agents maximizing other types of intrinsic motivation as introduced in the introduction. Namely, 1) *IG explorer*, an agent maximizing its information gain on a forward model, 2) *Novelty explorer*, an agent that chooses the so far least visited reachable state and 3) *Surprise explorer*, an agent maximizing the prediction error of a forward model. Our main goal was to illustrate and compare what these exploration criteria would do in a perfect information setting, but it should be noted that efficient and scalable algorithms may be developed on top of existing work on empowerment approximation (Mohamed and Rezende, 2015; Karl et al., 2019). This, however, is not the focus of our work and while its development is discussed a bit in Section 4.5, it is mainly deferred to future work.

Thus, for our purposes we remain in a simple grid world setting that allows for exact computation of empowerment gain, information gain and closed-form solutions for model updates. This setting also leads to the exploration algorithm (see Algorithm 2) where we made some simplifying assumptions. In particular, when we explore, we first try out each action an agent could take in the real world, update our parameterized model(s), and compute either expected empowerment gain or information gain. Then we only execute and count the best action according to the respective criterion. While this is impractical, we think this study is very illustrative of the behavior induced by the various exploration criteria and gives an intuitive understanding of the theoretical results of the previous section.

Going through Algorithm 2 in detail, `blahut_arimoto(m)` refers to an implementation of the Blahut-Arimoto algorithm (Arimoto, 1972; Blahut, 1972), an iterative procedure



---

```

Initialize forward model  $m = p_{\xi}(s_{T+1} \mid s, a_{1:T})$  with Dirichlet( $\alpha$ ) prior,  $\alpha = 0.01$ .;
for each episode do
   $s = \text{env.reset}()$ ;
  for  $t=1..T$  do
    empowerment = blahut_arimoto( $m$ );
    for  $a$  in Actions do
      # Try out all actions and compute their empowerment gain;
       $s' = \text{env.step}(a)$ ;
       $m' = m.\text{update}(s, a, s')$ ;
      empowerment' = blahut_arimoto( $m'$ );
      gain[ $a$ ] = empowerment' - empowerment;
      env.set_state( $s$ );
    # Perform the best action for exploration;
     $a = \text{argmax}_a \text{gain}$ ;
     $s' = \text{env.step}(\text{best\_action})$ ;
     $m = m.\text{update}(s, a, s')$ ;
     $s = s'$ ;

```

**Algorithm 2:** Exploration Algorithm

to compute the channel capacity (and thus empowerment) of a discrete channel. The channel in our case is the currently approximated world model  $m$ . We perform just one long rollout without resetting the environment where overall the agent takes 50,000 steps in the environment (10,000 steps in the chain environment) in a whole experiment. For IG exploration the algorithm works analogously, but naturally we compute information gain instead of empowerment gain. To limit the effect of randomness in stochastic environments, we collect 10 instead of just 1 sample to collect data for the model update. For all experiments, we executed 10 separate runs to ensure reproducible results. As the results are quite stable across runs and our results are mostly qualitative in nature, we picked representative plots from individual runs at random. Where quantitative measures are mentioned, they are averaged across runs.

Our grid worlds, built on top of MiniGrid (Chevalier-Boisvert et al., 2018), have a discrete state space  $\mathcal{S}$  and action space  $\mathcal{A}$ . The agent may perform five different actions that move to a neighboring cell (left, right, up, down) or keep the agent where it is (stay). An action that leads into a wall has no effect on the agent's position. As observations we use the global  $x$  and  $y$  coordinates.

---

---

For estimating our criteria, we maintain a Bayesian forward model (aside from Novelty exploration where we just maintain a visitation frequency map). For each cell, the forward model is realized by a Categorical distribution with a Dirichlet prior. The parameterization of the forward model is, of course, essential to the resulting behavior of these exploration criteria and should be taken into account when interpreting the results. With our choice, the agent cannot generalize information among states as would be the case with e. g. a neural network. We use the iterative Blahut-Arimoto algorithm (Arimoto, 1972; Blahut, 1972) to directly compute an empowerment estimate, including the distribution parameters of the source  $\omega_\phi(a_{1:T} | s)$  and inverse model  $p_\psi(a_{1:T} | s, s_{T+1})$ . Information gain can be computed in closed form since the Dirichlet distribution is a conjugate prior to the Categorical distribution. For more details we refer to Appendix A.3.3.

Note that we do not compare to an Empowerment maximizing agent as this does not result in exploratory behavior, but just leads the agent to the most empowered state in the environment. In our setting, where initially empowerment is estimated to be 0 everywhere, it results in the agent hopping back and forth between the first two states it encounters.

#### 4.3.1. Deterministic Environments

We start out by discussing the fully observable and deterministic setting. For the IG and Surprise explorer, in every state the agent performs each action uniformly which also leads to a uniform visitation frequency of every state (see Figure 4.1a) in this particular setting. This matches the uniform state visitation of novelty seeking. In contrast, rather than performing each action uniformly, the EG explorer, from any state, visits each reachable state uniformly. This leads to a visitation frequency that qualitatively resembles the empowerment landscape for the environment in that more empowered states (center of the room) are visited more frequently during exploration than less empowered states (bordering walls). Importantly, the EG explorer still makes sure to visit every state and does not remain strictly on high empowered states, it just shifts the exploration focus towards them. This remains true for more complicated wall structures and is also affected by the empowerment horizon as shown in Figures 4.1c and 4.1d. Since a state being more empowerment means having more ways to interact with the world, exploring these states more thoroughly should in general be more helpful for downstream tasks.

#### 4.3.2. Prison States

What happens if we have detrimental states in the environment that severely limit an agent’s control? Avoiding such states should generally be helpful for exploration. We investigated the extreme scenario of perfect sinks where an agent, once entered, cannot

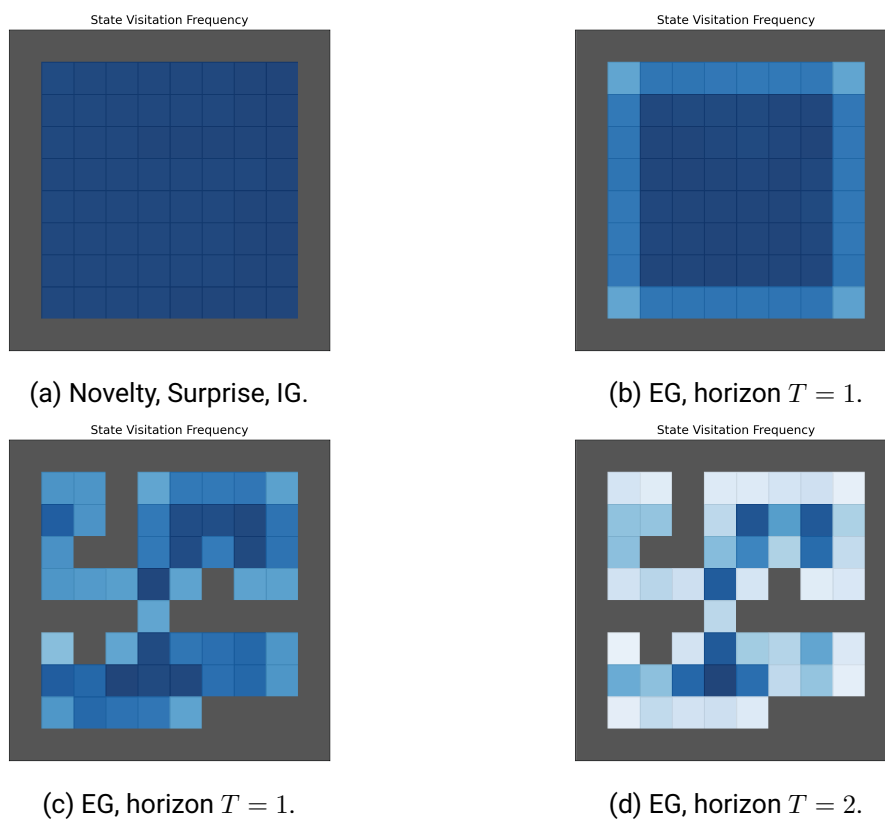


Figure 4.1.: Comparison of state visitation frequencies (dark blue visited most often, white least often) of the EG and the three other explorers (Novelty, Surprise, IG) in deterministic and discrete environments.

get out again (prison states). In our grid world, we model these states as lava cells (Figure 4.2a). Different to the other scenarios, we randomly reset the agent once it is stuck in such a cell (the agent can not observe this reset).

We can see in Figure 4.2c that, while the IG and Surprise explorer skew away from the lava cells to some degree, this is just due to the random walk being reset randomly whenever an agent enters the lava. Pure novelty seeking (Figure 4.2b) is an especially poor choice in this environment as it actively ensures that all lava cells are visited just as often as any other state. For the EG explorer, avoidance of lava is actively pursued for an empowerment horizon greater than 1. One may ask: why is EG not fully avoiding the lava since the agent's empowerment in this state is 0 as its actions have no influence at all?

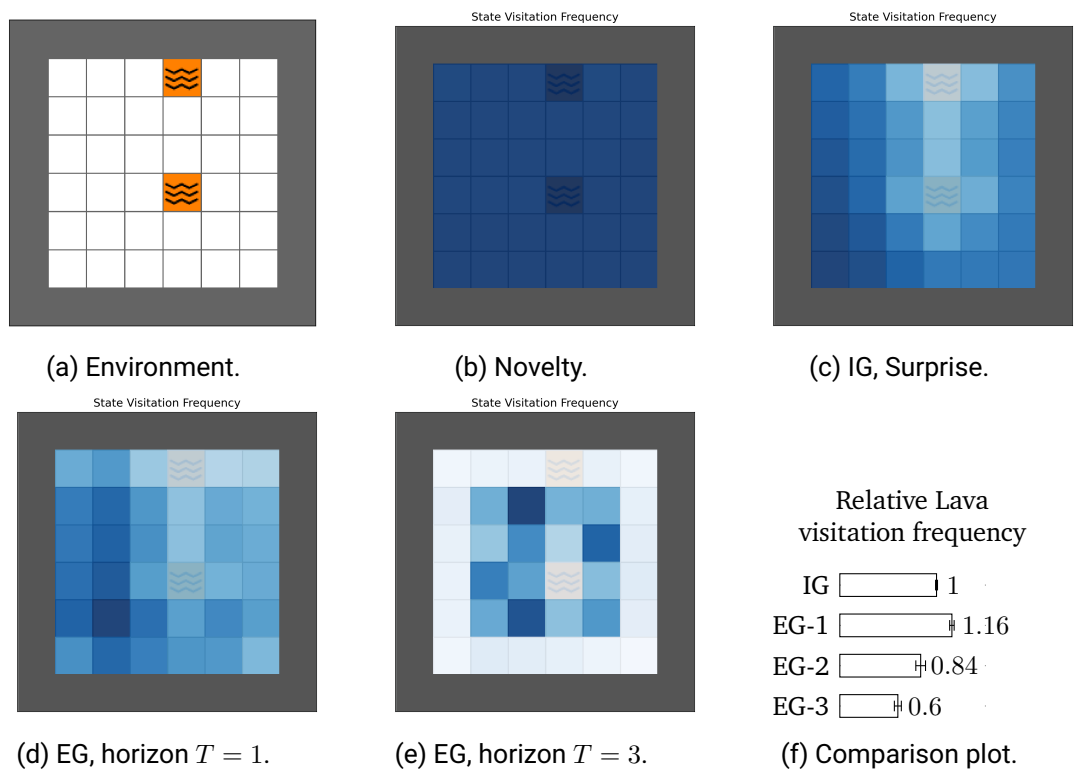


Figure 4.2.: (a) Rendering of the empty environment, orange cells are lava cells (prisons states). (b) Novelty seeking ensures uniform visitation frequency. (c) IG and Surprise do not avoid lava, its visitation frequency skews away from the lava in so far as the random walk resets randomly whenever the agent enters the lava. (d,e) EG over horizon 3, however, actively avoids the lava trap to some extent, while horizon of 1 is too myopic. (f) shows how often lava states are visited compared to all other states (normalized to the ratio of the IG explorer).

This is because the lava cell is still a state in and of itself, and it is still more empowering to know that you can reach that state more reliably. However, improving empowerment after having entered a lava cell is impossible and hence the increase of the estimated empowerment for the rest of the trajectory is 0. That is why a horizon of 1 is not enough to realize the detrimental effect and longer horizon skew more and more away from the lava as shown in Figure 4.2f.

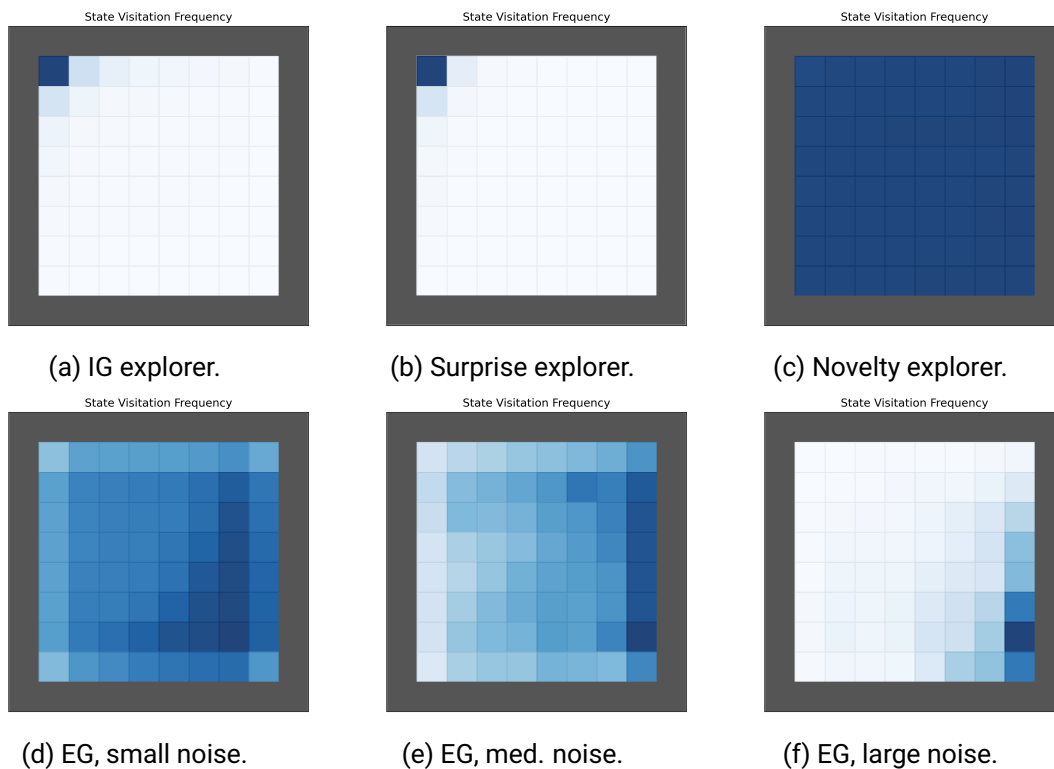


Figure 4.3.: Comparison of different action noises, there is no noise for actions going down and right, but there is noise going up (with probability  $p_a$ ) and left (with probability  $2p_a$ ). (a,b) IG and Surprise explorer are much more attracted in performing the more noisy actions. (c) Novelty explorer still maintains a uniform visitation frequency. (d) When introducing a bit of noise, EG prefers going to the bottom right, but still stays away from the walls as in the deterministic setting. (e, f) When noise on the actions is increased even further, at some point this effect dominates.

### 4.3.3. Actions of Varying Reliability

Next, we investigated the influence of varying action noises, i. e. in any state different actions are augmented with different level of noise. Concretely, when taking action  $a$ , there is a probability  $p_a$  that a random action  $a' \in \mathcal{A}$  is performed instead. We share the same action noise model for every state in the environment.

---

---

Novelty seeking is ignorant of the action space and still (tries to) ensure a uniform visitation frequency. For IG exploration, we observe that the more noisy action is taken more frequently. This is because occasionally we observe an unexpected event which leads to a bigger change to the model and thus to high information gain. Similarly, the Surprise explorer prefers the noisy action as well because the prediction error for noisy actions remains larger than for less noisy ones. Interestingly, expected EG skews in the opposite direction, generally preferring the more reliable actions as they have greater potential of increasing one’s empowerment estimate. This effect augments the results we found in the deterministic setting and also depends on the empowerment’s horizon, which push the agent away from the walls (see Figures 4.3d to 4.3f). Again, we argue that this exploration behavior is more desirable as exploring the actions that have a more reliable effect should in general be more helpful for interacting with the world.

#### **4.3.4. State-Dependent Action Noise**

Let us now look at state-dependent action noise, in which all actions in one state share the same noise model, but differ among different states. For the Surprise and IG explorer we observe the same behavior as in the deterministic setting, as these criteria cannot see ahead enough; after all, all actions in any state have an identical effect (except next to a wall). Conversely, we found that an EG explorer skews towards the part of the state space with less noise (Figure 4.4). While it still explores the whole state space, it focuses more on areas with greater potential for influencing the world. However, as with the previously discussed sinks (lava), EG needs to be computed over at least a horizon of 2 in order to realize that going towards the less noisy region helps us with increasing our empowerment estimate to a greater degree. For this experiment, we used a slightly different objective that considered the impact of action sequences on the empowerment estimate instead of just an individual actions as we detail in Appendix A.3.4.

#### **4.3.5. Redundancies in the Action Space**

Last, we wanted to take a look at the effect of an enlarged action space containing multiple actions with the same effect on the state space. As far as improving the transition model is concerned, these actions are all equally important, but for an agent to act in the environment, it is really sufficient to learn any one of the identical actions. To illustrate this, we explore a simple chain environment as depicted in Figure 4.5 where only a single action leads either left or remains in the current state, while 8 actions lead to its right neighbor. The agent starts out in the rightmost state and is never reset.

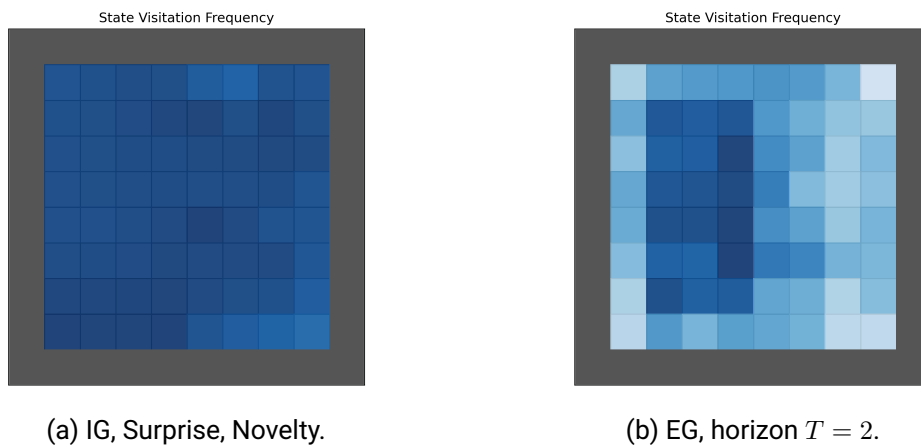


Figure 4.4.: (a) - (b) The left half of the room is without noise, while the right half is augmented by linearly increasing state-dependent action noise, i. e. the column right of the center line has noise probability  $p_a = 0.1$  and the rightmost column of the room  $p_a = 0.4$ . State visitation frequency for EG is skewed towards the less noisy part of the state space while the other 3 intrinsic motivations cannot account for this effect.

Here, we found the result as shown in Table 4.1. The Novelty explorer is naturally unaffected by the change in action space and still ensures uniform visitation frequency of all states. It chooses one of the 8 options leading right at random. However, the Surprise and IG explorer are heavily affected and their uniform action selection preference leads to a strongly skewed state visitation frequency to the rightmost state, barely visiting the leftmost state in comparison. The EG explorer, even of horizon 1, acts similarly to the Novelty explorer. The resulting visitation frequency is analogous to the one we found in Section 4.3.1, meaning the border states at both ends of the chain are visited a bit less frequently than the inner states as they have fewer neighbors. Different to the Novelty explorer, the EG explorer focuses on one of the actions leading right, ignoring the others, maximally improving the agent’s capability to interact. Note, that this complete focus will only occur in this complete hindsight information setting in deterministic environments. If the actions leading right were augmented by varying levels of noise, EG would focus on the least noisy one (in contrast to e. g. the Surprise explorer as seen in Section 4.3.3).

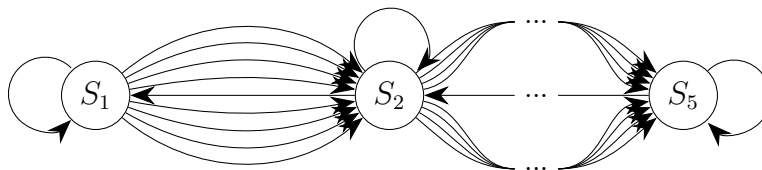


Figure 4.5.: Our chain environment where many options lead right, but only a single one in each state to the left.

Method	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$
Novelty	2000	2000	2000	2000	2000
Surprise	26	95	409	1771	7669
IG	4	18	138	1095	8745
EG, $T = 1$	1538	2307	2308	2308	1539

Table 4.1.: State visitation frequency in the chain environment.

#### 4.4. Related Work

Numerous ways of motivating exploration have been suggested over the years. Among the first was Schmidhuber (1991b) which based reward signals on prediction errors of predictive models. This prediction error has been used in various ways using function representation learning (Stadie et al., 2015; Pathak et al., 2017). Pathak et al. (2017) formulates surprise as the error in an agent’s ability to predict the consequence of its own actions in a visual feature space learned by a self-supervised inverse dynamics model. Focusing purely on prediction error has the downside of being attracted unpredictable traps (static TV noise problem), where the error remains large even if it has been observed countless times. Hence, learning progress of a predictive model has been suggested and formalized in various works (Schmidhuber, 1991a; Lopes et al., 2012; Stadie et al., 2015; Achiam and Sastry, 2017). Houthoof et al. (2016) formulates learning progress in a Bayesian setting as information gain.

More recently, ensembles of predictors have been used to form some kind of internal disagreement metric as opposed to directly comparing to the real world difference. Pathak et al. (2019) and Shyam et al. (2019) train ensembles of dynamics models and incen-



---

---

tivize the agent to explore such that the disagreement of those ensembles is maximized. Plan2Explore (Sekar et al., 2020) performs latent rollouts using an ensemble of learned world models, then rewarding action plans by their latent disagreements. Ratzlaff et al. (2020) is modeling uncertainty about dynamics using a generative model and incentivizes exploration where uncertainty is large. Blaes et al. (2019) combines surprise with task selection based on learning progress.

Different from these surprise based approaches, various ways of novelty seeking have similarly been formalized over the years. Bellemare et al. (2016) (extended by Ostrovski et al. (2017)) proposed using a pseudo-count using density estimation for novelty estimation in high-dimensional state spaces. Tang et al. (2017) suggested a computationally simple but effective generalization of count-based approaches using hashing. Lee et al. (2019) recast exploration as a problem of state marginal matching, where they aim to learn a policy for which the state marginal distribution matches a given target state distribution (often a uniform distribution). Similarly, Hazan et al. (2019) defines the maximum-entropy exploration policy as the policy whose induced state distribution has the maximum possible entropy. Savinov et al. (2018) defines novelty through reachability within a certain time horizon from observations in a memory where reachability is approximated using a siamese network architecture.

However, there are also other exploration approaches based on skill acquisition that are related to our method. Sharma et al. (2019) encourages novel skill acquisition via mutual information maximization between skills and states. DIAYN (Eysenbach et al., 2019) encourages exploration by incentivizing different skills that lead to distinguishable outcomes. Choi et al. (2021) generalizes various methods for goal-conditioned exploration (such as DIAYN) in a common framework and derives novel objectives based on variational mutual information maximization.

Empowerment (Klyubin et al., 2005a,b) as an intrinsic motivation has been gaining more and more attention over the recent years. Initially mainly formulated in the discrete domain, empowerment has since been applied in the continuous domain by Jung et al. (2011) where empowerment is approximated using a learned model and Monte Carlo sampling. Salge et al. (2013a) introduced a more efficient approximation based on linear Gaussian channels. Salge et al. (2013b) extends this method by accounting also for state-dependent noise where they find that empowerment maximizing agents will avoid uncertain states or unpredictable interactions, similar to what we have found in our exploration with EG. Later, variational methods for approximation have been suggested using the Barber-Agakov bound on mutual information (Agakov, 2004) in model-free (Mohamed and Rezende, 2015; Gregor et al., 2016) and model-based settings (Karl et al., 2019). Zhao et al. (2021) approximate empowerment via a learned Gaussian channel by convex optimization.

---

## 4.5. Discussion & Limitations

There are various shortcomings of our proposed framework. Estimating empowerment (and hence, EG) is very complex and computationally prohibitive. Even if one found an efficient way of approximating empowerment, directly trying to estimate improvement in empowerment can be problematic. While it is true that increasing one’s options will eventually improve one’s empowerment, this becomes true only after some initial learning phase, e. g. when interacting with novel objects this can at first be detrimental to our understanding of the world and thus decrease our empowerment at first. This is particularly true for our current machine learning models where outlier observations can radically change a model’s behavior, even on the previously seen data distribution. As empowerment estimation consists of multiple components, it may take some time to converge and realize its greater influence in the world.

Even though EG is computationally expensive, it is still a heuristic. Optimizing it may not be the best way to recognize one’s empowerment. In particular, the maximal local improvement of Empowerment does necessarily not guarantee fast global convergence. Alternatively, one could imagine computing EG not on a singular state but over a representative distribution of states and maximizing the average change, but that would increase the computational cost significantly. The impact of the horizon of empowerment is critical but often neglected since the computational complexity effectively limits the horizon. In particular, the formulation is indifferent to when an agent can reach a certain state, as long as it can be attained at time  $T$ . That means, it does not care about reaching a certain state within a shorter time horizon – something we can imagine to be of great importance to an agent in general. How to choose the best horizon or maybe interleave multiple time horizons remains an open question.

In our experiments, we parameterize the forward model of each cell independently. This is for both simplicity’s sake to allow for exact computation and to get a clear and isolated understanding of the exploration criterion. However, a forward model realized by a (Bayesian) Neural Network or a Gaussian Process that allows to generalize experience in the neighborhood or even to unseen areas would drastically change the exploratory behavior. Given that these types of model are ubiquitous, a study thereof when combined with EG is of great importance. Moreover, only limited conclusions can be drawn from these simple and discrete settings when transferring the agent to more complicated scenarios with continuous state and action spaces.

As far as scaling this approach to high-dimensional or continuous domains, various ways of efficiently approximating have been proposed (Mohamed and Rezende, 2015; Karl et al., 2019; Zhao et al., 2021). Using these approaches, a more practical algorithm can be formulated by computing EG in hindsight after performing a (batch of) actions in the real

---

---

world. EG can then be approximated by doing a few gradient steps with the newly acquired data and then computing the EG as an (additional) reward term to the agent. However, while feasible, this approach is still computationally expensive compared to many other exploration heuristics. Potentially more promising as a direction, we suggest to use the exploration characteristics derived from this framework as a foundation for future research, e. g. the consideration of relevancy for improving the world model based on an agent's current capabilities to interact with the world is crucial. More crude approximations of these characteristics may turn out to be more fruitful in practice.

## 4.6. Conclusion

In this work, we propose a novel framework for embodiment driven exploration. Building on the universally applicable and information-theoretic measure of an agent's perceived influence in the world – empowerment – we suggest an exploration criterion based on the expected improvement of one's estimation thereof. Theoretically, we show that it captures and puts into relation various previously suggested exploration criteria such as novelty seeking, surprise maximization and learning progress. Different to the use of these individual criteria, EG accounts for the agent's current capabilities and boundedness, and focuses directly on what we are interested in – recognizing one's capability to interact with the world. In various discrete grid world environments featuring different noise models, we showcase our theoretical findings where an agent's exploration is focused on areas with greater potential for increasing its influence. However, this work is just the very start of this investigation. For future work, the focus will be on smart approximations that can capture the spirit of EG while being scalable to scenarios of interest for the current state of the art research.



---

## 5. Conclusion

---

In this thesis, we show the full picture of how to “*learn to control*” using a learned LSSM on two very different robotic platforms: a self-built drone and a robotic arm. While minimizing engineering effort and maximizing use of machine learning methods, we demonstrate that some relatively simple tasks can be learned with less than an hour of real-world interaction data and without expert domain knowledge. Starting from raw sensory data, we learn a LSSM that acts as both a simulator of dynamics and a filter for online state estimation. Using this model, we show how to learn a controller relying entirely on the simulation accuracy of the learned model. Without modification, this controller can be deployed and successfully perform relatively simple tasks in the real world. Last, in a more theoretical contribution, we address the question of what should be explored and modeled in the first place. In particular, we hinge the answer to this question on an agent’s potential to increase its empowerment estimator, i. e. to increase its recognized potential to influence the world around him.

### 5.1. Outlook

Naturally, in a single thesis there is only so much that can be explored. In the following, we give an outlook on promising directions of research. These are either rather direct follow-ups of the presented work or present related issues that were untouched in this work, but are required to fulfill the dream of autonomous robots that can learn on their own.

#### 5.1.1. Robustness to Sensor Noise and Partial Observability

We (and others) have shown how to extract various ground truth state information using a LSSM without encoding prior knowledge into the latent space. For example, we showed that we can linearly encode simple time derivative information, such as velocities from positional data. However, the performance gap to an agent that is provided with observed velocities can be quite significant, in particular as the observations become more and

---

---

more noisy. Most of the work in the field is done on either simulated data or very well engineered robotic systems with sophisticated methods for preprocessing raw sensor data. As we experienced with the LiDARs on the drone, which required very low-weight and thus comparatively low-quality sensors, a low-pass preprocessing step was helpful to attain a viable controller. It is somewhat surprising that such a simple operation was not learned to an equal level of quality by our proposed modeling approach. It should be investigated how learning methods can be made more robust such that we can avoid engineering such simple preprocessing steps in the future.

### **5.1.2. Universally Applicable Reward Functions**

Success of RL depends heavily on the given reward function. For state of the art results or just to produce a workable outcome on more complicated system, algorithms usually require highly-tuned reward functions. We tried to keep the reward engineering to a minimum, but on top of rewarding the desired position, punishing high velocities and accelerations was almost always helpful or even crucial in our experiments. Other RL methods, in particular for quadruped walking, often come with up to a dozen reward terms encouraging and punishing all kinds of different behaviors (Lee et al., 2020b; Kumar et al., 2021). Naturally, this level of reward engineering is not a satisfactory state of affairs in RL, especially as tuning the reward function incurs the cost of training the agent repeatedly. In effect, this mitigates some of the potential benefit from not engineering other parts of the system. What we really hope to find is a more general purpose reward function. Candidates for such universal reward terms may be energy minimization (Fu et al., 2021), pain avoidance (or rather its translation to robotic systems) and other intrinsic motivations such as novelty seeking, surprise maximization and empowerment (gain). Formulating and combining these objectives with the actual task objective has been proposed numerous times, but the best results on a given task are usually still achieved with hand-tailored reward functions.

### **5.1.3. Learned Low-Level Control**

In this thesis, we showcased a learned controller on two real robotic platforms where most components were learned without encoding prior knowledge or strong inductive biases into the model. Nevertheless, a few parts still relied on engineered methods, one of particular note were the action spaces. Both the learned thrust-attitude controller and joint velocity controller require underlying PID controllers to transform these signals into torques and then pulse width modulation signals. For the chosen systems, these components are well understood and can be engineered precisely. However, for a method

---

---

to be truly generic and universally applicable, we cannot rely on such components. They may not exist for the next system we want to solve, and maybe the necessity for their existence limits us unnecessarily in our design of robotic systems. Unfortunately, learning low-level control end-to-end has been shown to be challenging, not only because the action to state space interaction becomes more complicated, but also because lower level control typically requires (much) higher frequency of control. Instead of learning end-to-end, this transformation of the control signal as previously been framed and learned as a separate supervised learning problem (Hwangbo et al., 2019). While this approach has had some successes, addressing the general problem of action spaces has seen very little attention in the RL community and remains an open challenge to fulfill its promises of universal applicability.

#### **5.1.4. What Does Really Matter in (Model-Based) Reinforcement Learning?**

Having so many different parts in (MB)RL makes it really difficult to attribute any method's success or failure to individual design choices. In our work, we found our choice of LSSM and backpropagating through this dynamics model for optimization of the policy to be of vital importance. Others have reported good results (in different settings) with a variety of approaches, not only limited to model-based methods. Be it Model Predictive Control (MPC) or model-free RL methods where the policy may be conditioned on otherwise compressed representations of high-dimensional observations. It will probably be a long accompanying investigation on what truly makes these approaches work in order to routinely strip the methods down to their core essentials.

#### **5.1.5. Empowered Exploration**

The key observation of our EG contribution was that the exploration should be based on an agent's capability to interact with the world and how this understanding can be improved. It takes into account an agent's perception, actuation and current limitations of its modeling and control. It shows what parts of the world are fundamentally interesting and on the other side irrelevant to encode into a world model. Increasing an agent's capability to act is a fundamental exploration principle for which we suggested one formalism based on empowerment. However, we believe that we just scratched on the surface of this principle. We believe finding a scalable and efficient method realizing this key principle is central to the task of developing autonomous agents.

---

---

### 5.1.6. A Universal Agent

Training a completely new agent from a blank slate for a specific task on a specific platform is a time-consuming task. While this mode of operation is still the default in machine learning, it is in stark contrast to how humans learn to act in the world. Intuitively, one would imagine learning tasks in a curriculum from easy to hard should lead to natural synergies and speed up training. Despite this intuition and this obviously conflicting approach in a lot of machine learning research, there remains a good reason for this mode of operation: learning to solve multiple tasks with a single agent is a very complicated endeavor often leading to complete failure to solve any task, or unlearning previously mastered tasks when acquiring new tasks. MT-Opt (Kalashnikov et al., 2021) carefully detailed a successful process of a robotic arm learning a handful of tasks consisting of lifting various objects, rearranging or covering them in various ways. While ultimately successful, its learning procedure turned out to require a lot of care and attention to detail. More recently, GATO (Reed et al., 2022), a neural network architecture based on transformers (Vaswani et al., 2017), showed how a single agent can play Atari, caption images, chat and stack blocks with a real robot arm based on expert demonstrations. Undoubtedly, this demonstrated an unprecedented level of multi-task competency and encouragingly the model consisting of  $79M - 1.1B$  parameters ended up relatively small for modern deep learning standards. However, the evaluation of adaptation on unseen tasks still raises a lot of questions. For example, GATO requiring 1000 episodes of expert demonstration to match the expert's performance on the cart-pole swing up task suggests that while the authors may have found an architecture that can successfully encode policies for a lot of different tasks, multi-task learning may still lead to negative rather than positive synergies. Thus, a long road remains ahead.



---

# A. Appendix

---

## A.1. Appendix for Chapter 2

### A.1.1. Lower Bound Derivation

We derive the sequential ELBO

$$\begin{aligned} & \log p(x_{1:T}) \\ &= \log \int_{z_{1:T}} \int_{s_{2:T}} q_\phi(z_{1:T}, s_{2:T} \mid x_{1:T}) \frac{p_\theta(x_{1:T} \mid z_{1:T}) p_\theta(z_{1:T}, s_{2:T})}{q_\phi(z_{1:T}, s_{2:T} \mid x_{1:T})} \\ &\geq \int_{z_{1:T}} \int_{s_{2:T}} q_\phi(z_{1:T}, s_{2:T} \mid x_{1:T}) \log \frac{p_\theta(x_{1:T} \mid z_{1:T}) p_\theta(z_{1:T}, s_{2:T})}{q_\phi(z_{1:T}, s_{2:T} \mid x_{1:T})} \\ &= \int_{z_{1:T}} \int_{s_{2:T}} q_\phi(z_{1:T}, s_{2:T} \mid x_{1:T}) \log p_\theta(x_{1:T} \mid z_{1:T}) \\ &\quad + \int_{z_{1:T}} \int_{s_{2:T}} q_\phi(z_{1:T}, s_{2:T} \mid x_{1:T}) \log \frac{p_\theta(z_{1:T}, s_{2:T})}{q_\phi(z_{1:T}, s_{2:T} \mid x_{1:T})} \\ &= \int_{z_1} \int_{s_2} \cdots \int_{s_T} \int_{z_T} q(z_1 \mid x_{1:T}) q(s_2 \mid z_1, x_{1:T}) \cdots q(s_T \mid z_{T-1}, s_{T-1}, x_{1:T}) \\ &\quad \quad \quad q(z_T \mid z_{T-1}, s_T, x_{1:T}) \log p_\theta(x_1 \mid z_1) \cdots p_\theta(x_T \mid z_T) \\ &\quad - \text{KL}(q(z_{1:T}, s_{2:T} \mid x_{1:T}) \parallel p_\theta(z_{1:T}, s_{2:T})) \\ &= \mathbb{E}_{z_1 \sim q(z_1 \mid \cdot)} [\log p(x_1 \mid z_1)] \\ &\quad + \sum_{t=2}^T \mathbb{E}_{s_t \sim q(s_t \mid s_{t-1}, z_{t-1}, x_t)} [\mathbb{E}_{z_t \sim q(z_t \mid z_{t-1}, s_t, x_t)} [\log p(x_t \mid z_t)]] \\ &\quad - \text{KL}(q(z_{1:T}, s_{2:T} \mid x_{1:T}) \parallel p(z_{1:T}, s_{2:T})) \end{aligned}$$

for our model where we omit we omit conditioning on control inputs  $u_{1:T}$  for brevity.

## Factorization of the KL Divergence

The dependencies on data  $x_{1:T}$  and  $u_{1:T}$  as well as parameters  $\phi$  and  $\theta$  are omitted in the following for convenience.

$$\begin{aligned}
& \text{KL}(q(z_1, s_2, \dots, s_T, z_T) \parallel p(z_1, s_2, \dots, s_T, z_T)) \\
& \text{(Factorization of the variational approximation)} \\
&= \int_{z_1} \int_{s_2} \cdots \int_{s_T} \int_{z_T} q(z_1)q(s_2 \mid z_1) \dots q(s_T \mid z_{T-1}, s_{T-1})q(z_T \mid z_{T-1}, s_T) \\
& \quad \log \frac{q(z_1)q(s_2 \mid z_1) \dots q(z_T \mid z_{T-1}, s_T)}{p(z_1, s_2, \dots, s_T, z_T)} \\
& \text{(Factorization of the prior)} \\
&= \int_{z_1} \int_{s_2} \cdots \int_{s_T} \int_{z_T} q(z_1)q(s_2 \mid z_1) \dots q(s_T \mid z_{T-1}, s_{T-1})q(z_T \mid z_{T-1}, s_T) \\
& \quad \log \frac{q(z_1)q(s_2 \mid z_1) \dots q(z_T \mid z_{T-1}, s_T)}{p(z_1)p(s_2 \mid z_1) \dots p(z_T \mid z_{T-1}, s_T)} \\
& \text{(Expanding the logarithm by the product rule)} \\
&= \int_{z_1} q(z_1) \log \frac{q(z_1)}{p(z_1)} + \int_{z_1} \int_{s_2} q(z_1)q(s_2 \mid z_1) \log \frac{q(s_2 \mid z_1)}{p(s_2 \mid z_1)} \\
& \quad + \sum_{t=2}^T \int_{z_1} \int_{s_2} \cdots \int_{s_T} \int_{z_T} q(z_1)q(s_2 \mid z_1) \dots q(z_T \mid z_{T-1}, s_T) \log \frac{q(z_t \mid z_{t-1}, s_t)}{p(z_t \mid z_{t-1}, s_t)} \\
& \quad + \sum_{t=3}^T \int_{z_1} \int_{s_2} \cdots \int_{s_T} \int_{z_T} q(z_1)q(s_2 \mid z_1) \dots q(z_T \mid z_{T-1}, s_T) \log \frac{q(s_t \mid z_{t-1}, s_{t-1})}{p(s_t \mid z_{t-1}, s_{t-1})} \\
& \text{(Ignoring constants)} \\
&= \text{KL}(q(z_1) \parallel p(z_1)) \\
& \quad + \mathbb{E}_{z_1 \sim q(z_1)} [\text{KL}(q(s_2 \mid z_1) \parallel p(s_2 \mid z_1))] \\
& \quad + \sum_{t=2}^{T-1} \mathbb{E}_{z_{t-1} \sim q(z_{t-1} \mid \cdot)} \left[ \mathbb{E}_{s_t \sim q(s_t \mid \cdot)} [\text{KL}(q(z_t \mid z_{t-1}, s_t) \parallel p(z_t \mid z_{t-1}, s_t))] \right] \\
& \quad + \sum_{t=3}^{T-1} \mathbb{E}_{s_{t-1} \sim q(s_{t-1} \mid \cdot)} \left[ \mathbb{E}_{z_{t-1} \sim q(z_{t-1} \mid \cdot)} [\text{KL}(q(s_t \mid z_{t-1}, s_{t-1}) \parallel p(s_t \mid z_{t-1}, s_{t-1}))] \right]
\end{aligned}$$

Table A.1.: Dimensionality of environments.

Dimensionality of	Observation Space	Control Input Space	State Space
(Bullet) Reacher	7	2	9
Hopper	8	3	15
Multi Agent Maze	6	6	12
Image Ball in Box	$32 \times 32$	0	4
FitzHugh-Nagumo	2	1	2
Time-Varying Pendulum	2	1	2
Time-Varying Reacher	4	2	6
Time-Varying Cheetah	8	6	17

### A.1.2. Experimental Setup

Overall, training the Concrete distribution has given us the biggest challenge as it was very susceptible to various hyperparameters. We made use of the fact that we can use a different temperature for the prior and approximate posterior (Maddison et al., 2017) and we do independent hyperparameter search over both. For us, the best values were 0.75 for the posterior and 2 for the prior. Additionally, we employ an exponential annealing scheme for the temperature hyperparameter of the Concrete distribution. This leads to a more uniform combination of base matrices early in training which has two desirable effects. First, all matrices are scaled to a similar magnitude, making initialization less critical. Second, the model initially tries to fit a globally linear model, leading to a good starting state for optimization.

With regards to optimizing the KL-divergence, there is no closed-form analytical solution for two Concrete distributions. We therefore had to resort to a Monte Carlo estimation with  $n$  samples where we tried  $n$  between 1 and 1000. While using a single samples was (numerically) unstable, using a large number of samples also didn't result in observable performance improvements. We therefore settled on using 10 samples for all experiments.

In all experiments, we train everything end-to-end with the ADAM optimizer (Kingma and Ba, 2015). We start with learning rate of  $5e-4$  and use an exponential decay schedule with rate  $\lambda \in \{0.95, 0.97, 0.98\}$  every 2000 iterations.

Table A.2.: Overview of hyperparameters.

Parameter	Multi Agent Maze	Reacher	Image Ball in Box
# episodes	50000	20000	5000
episode length	20	30	20
batch size	256	128	256
dimension of $z$	32	16	8
dimension of $s$	16	8	8
posterior temp.	0.75	0.75	0.67
prior temp.	2	2	2
temp. annealing steps	100	100	100
temp. annealing rate	0.97	0.97	0.98
$\beta$ (KL-scaling)	0.1	0.1	0.1

### Roboschool Reacher

To generate data, we follow a Uniform distribution  $\mathcal{U} \sim [-1, 1]$  as the exploration policy. Before we record data, we take 20 warm-up steps in the environment to randomize our starting state. We take the data as is without any other preprocessing.

### Multi Agent Maze

Observations are normalized to be in  $[-1, 1]$ . Both position and velocity is randomized for the starting state. We again follow a Uniform distribution  $\mathcal{U} \sim [-1, 1]$  as the exploration policy.

### Time-Varying Dynamics: Pendulum, Reacher, Cheetah

As exploration policy we use and Ornstein-Uhlenbeck process with damping coefficient 0.05 and standard deviation 0.2. We generate 1000 episodes of 500 steps. For the pendulum

Table A.3.: Overview of hyperparameters for experiments with time-varying dynamics.

Parameter	Pendulum	Reacher	Cheetah
# episodes	1000	1000	1000
episode length	500	500	500
batch size	256	256	256
dimension of $z$	16	16	32
dimension of $s$	32	32	64
$\beta$ (KL-scaling)	0.1	0.1	0.1

we vary the mass, length and damping coefficient from 50% to 200% of the default values. For the cheetah, we sample the mass, damping and joint stiffness from 50% to 200% of the default values. For the reacher, we vary the mass between 25% and 400% of the default value.

### Network Architecture

For most networks, we use MLPs implemented as residual nets (He et al., 2016) with ReLU activations. The following specifications were used for networks in the reacher and maze experiments:

- $q_{\text{meas}}(z_t \mid x_{\geq t}, u_{\geq t})$ : MLP consisting of two residual blocks with 256 neurons each. We only condition on the current observation  $x_t$  although we could condition on the entire sequence. This decision was taken based on empirical results.
- $q_{\text{trans}}(z_t \mid z_{t-1}, u_{t-1}, s_t)$ : In the case of Concrete random variables, we just combine the base matrices and apply the transition dynamics to  $z_{t-1}$ . For the Normal case, the combination of matrices is preceded by a linear combination with softmax activation. (see equation 2.6)
- $q_{\text{meas}}(s_t \mid x_{\geq t}, u_{\geq t})$ : is implemented by a backward LSTM with 256 hidden units. We reuse the preprocessing of  $q_{\text{meas}}(z_t \mid x_t)$  and take the last hidden layer of that

---

network as the input to the LSTM. For the work on time-varying dynamics, we chose a forward LSTM instead to get an actual filter for control.

- $q_{\text{trans}}(s_t \mid s_{t-1}, z_{t-1}, u_{t-1})$ : MLP consisting of one residual block with 256 neurons.
- $q_{\text{initial}}(w \mid x_{1:T}, u_{1:T})$ : MLP consisting of two residual block with 256 neurons optionally followed by a backward LSTM. We only condition on the first 3 or 4 observations for our experiments.
- $q_{\text{initial}}(s_2 \mid x_{1:T}, u_{1:T})$ : The first switching variable in the sequence has no predecessor. We therefore require a replacement for  $q_{\text{trans}}(s_t \mid s_{t-1}, z_{t-1}, u_{t-1})$  in the first time step, which we achieve by independently parameterizing another MLP.
- $p(x_t \mid z_t)$ : MLP consisting of two residual block with 256 neurons.
- $p(z_t \mid z_{t-1}, u_{t-1}, s_t)$ : Shared parameters with  $q_{\text{trans}}(z_t \mid z_{t-1}, u_{t-1}, s_t)$ .
- $p(s_t \mid s_{t-1}, z_{t-1}, u_{t-1})$ : Shared parameters with  $q_{\text{trans}}(s_t \mid s_{t-1}, z_{t-1}, u_{t-1})$ .

We use the same architecture for the image ball in a box experiment, however we increase number of neurons of  $q_{\text{meas}}(z_t \mid x_{>t}, u_{>t})$  to 1024. For the FitzHugh-Nagumo model we downsize our model and restrict all networks to a single hidden layer with 128 neurons.

For the reinforcement learning experiments, we use MLPs for the policy and critic network consisting of 2 layers with 256 neurons and with tanh activation functions.

### A.1.3. On Scaling Issues of Switching Linear Dynamical Systems

Let's consider a simple representation of a ball in a rectangular box where its state is represented by its position and velocity. Given a small enough  $\Delta t$ , we can approximate the dynamics decently by just 3 systems: no interaction with the wall, interaction with a vertical or horizontal wall (ignoring the corner case of interacting with two walls at the same time). Now consider the growth of required base systems if we increase the number of balls in the box (even if these balls cannot interact with each other). We would require a system for all combinations of a single ball's possible states:  $3^2$ . This will grow exponentially with the number of balls in the environment.

One way to alleviate this problem that requires only a linear growth in base systems is to independently turn individual systems on and off and let the resulting system be the sum of all activated systems. A base system may then represent solely the transition for a single ball being in specific state, while the complete system is then a combination of  $N$

---

---

such systems where  $N$  is the number of balls. Practically, this can be achieved by replacing the *softmax* by a *sigmoid* activation function or by replacing the categorical variable  $s$  of dimension  $M$  by  $M$  Bernoulli variables indicating whether a single system is active or not. We make use of this parameterization in the multiple bouncing balls in a maze environment and also for the section about time-varying dynamics.

Theoretically, a preferred approach could be to disentangle multiple systems (like balls, joints) and apply transitions only to their respective states. This, however, would require a proper and unsupervised separation of (mostly) independent components. We defer this to future work.

## A.2. Appendix for Chapter 3

### A.2.1. Implementation & Training

For the drone experiments, the model was implemented using TensorFlow (Abadi et al., 2016) and TensorFlow Probability (Dillon et al., 2017). The ablation study on the Panda robot arm was done in PyTorch (Paszke et al., 2019). Mostly the same parameters were used across all experiments and drone configurations. All individual neural networks were parameterized by dense neural networks with either 1 or 2 hidden layers with ReLU activations and 256 neurons. For the policy and value function we used a tanh activation function instead. The encoder network was chosen to be an RNN in cases where the drone’s velocity was unobserved. We used ADAM (Kingma and Ba, 2015) for optimization of all our model components.

In case of the drone, the training can be done on a single GPU and takes up to a few days. The best hyperparameters (see Table A.4) were chosen using the real data in an offline reinforcement learning setting.

For the Panda Arm, training is distributed across three arms and can be completed within a few hours. Its joint velocities are limited to the range  $[-0.6; 0.6] \frac{rad}{s}$ . All observations are normalized based on Table 3.5. The best hyperparameters (see Table A.4 and Table A.5) were chosen based on a search in a simulated replication of the environment using the Bullet physics engine (Coumans and Bai, 2016).

### A.2.2. Onboard Computational Cost

For online control of the robot, we do not require execution of all parts of the model. In particular, we only need the policy  $\pi_{\theta}(a_t | s_t)$  and the filter consisting of inverse measurement model  $q_{\text{meas}}(z_t | x_t)$  and transition models  $p_{\xi}(z_t | z_{t-1}, s_t, u_{t-1})$  and  $p_{\xi}(s_t | s_{t-1}, z_{t-1}, u_{t-1})$ . Not needed are the likelihood model for observations  $p_{\xi}(x_t | z_t)$

Table A.4.: Hyperparameters for drone and arm experiments.

Parameter	Drone	Panda
<b>Dynamics Model</b>		
batch size	64	256
dimensions of latents $z_t$	32	64
dimensions of latents $s_t$	32	64
learning rate	$3 \times 10^{-4}$	$3 \times 10^{-4}$
# of base matrices	32	64
<b>Policy</b>		
batch size	128	256
learning rate	$1 \times 10^{-4}$	$1 \times 10^{-4}$
rollout horizon	5	10
discount factor	0.95	0.99
<b>Value Function</b>		
batch size	128	256
learning rate	$3 \times 10^{-4}$	$3 \times 10^{-4}$
rollout horizon	5	10
target network learning rate $\alpha_{\phi'}$	$1 \times 10^{-3}$	$5 \times 10^{-3}$
<b>Data Collection</b>		
collect data every n iterations	-	100
env. steps per collection	-	200
replay buffer initial size	-	1,000
replay buffer capacity	-	60,000

and rewards  $r_{\xi}(z_t, u_t)$  and the value function as they are only required during training. We are therefore free to choose the parameterization of the latter while the former need to be of limited complexity so that they can be executed in real-time on a Raspberry Pi 4.

For the Panda experiments, we do not deploy the model to an embedded device, rather



Table A.5.: SAC hyperparameters for experiments on the Panda arm.

Parameter	Value
batch size	256
policy/critic layers	2
policy/critic neurons	512
discount factor	0.99
entropy loss coefficient	0.001
learning rate	$3 \times 10^{-4}$
target network learning rate $\alpha_{\phi'}$	$5 \times 10^{-3}$
<b>Data Collection</b>	
collect data every n iterations	100
env. steps per collection	200
replay buffer initial size	1,000
replay buffer capacity	60,000

we remotely control it from the workstation where the training procedure takes place. We sent observations and actions over TCP and ZeroMQ.

### A.3. Appendix for Chapter 4

#### A.3.1. Derivations

For model-free empowerment gain, we derive at the approximation of eq. (4.13) with

$$\begin{aligned}
 & \hat{\mathcal{E}}_{\theta'}^{\text{MF},T}(s) - \hat{\mathcal{E}}_{\theta}^{\text{MF},T}(s) \\
 = & \max_{\omega_{\phi'}(a_{1:T}|s)} \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi'}(a_{1:T} | s) p(s_{T+1} | s, a_{1:T}) \log \frac{p_{\psi'}(a_{1:T} | s, s_{T+1})}{\omega_{\phi'}(a_{1:T} | s)} \\
 & - \max_{\omega_{\phi}(a_{1:T}|s)} \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi}(a_{1:T} | s) p(s_{T+1} | s, a_{1:T}) \log \frac{p_{\psi}(a_{1:T} | s, s_{T+1})}{\omega_{\phi}(a_{1:T} | s)} \\
 & \text{(Assuming } \omega_{\phi'}^*(a_{1:T} | s) \text{ and } \omega_{\phi}^*(a_{1:T} | s) \text{ to be the resp. maximizing distribution)}
 \end{aligned}$$

$$\begin{aligned}
&= \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi'}^*(a_{1:T} | s) p(s_{T+1} | s, a_{1:T}) \log p_{\psi'}(a_{1:T} | s, s_{T+1}) \\
&\quad + \mathbb{H}\left(A_{1:T}^{\phi'} = \omega_{\phi'}(a_{1:T} | s) \mid s\right) \\
&\quad - \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi}^*(a_{1:T} | s) p(s_{T+1} | s, a_{1:T}) \log p_{\psi}(a_{1:T} | s, s_{T+1}) \\
&\quad - \mathbb{H}\left(A_{1:T}^{\phi} = \omega_{\phi}(a_{1:T} | s) \mid s\right) \\
&= \mathbb{H}\left(A_{1:T}^{\phi'} \mid s\right) - \mathbb{H}\left(A_{1:T}^{\phi} \mid s\right) \\
&\quad + \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi'}^*(a_{1:T} | s) p(s_{T+1} | s, a_{1:T}) \log p_{\psi'}(a_{1:T} | s, s_{T+1}) \\
&\quad - \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi'}^*(a_{1:T} | s) p(s_{T+1} | s, a_{1:T}) \left[ \frac{\omega_{\phi}^*(a_{1:T} | s)}{\omega_{\phi'}^*(a_{1:T} | s)} \log p_{\psi}(a_{1:T} | s, s_{T+1}) \right] \\
&= \mathbb{H}\left(A_{1:T}^{\phi'} \mid s\right) - \mathbb{H}\left(A_{1:T}^{\phi} \mid s\right) \\
&\quad + \mathbb{E}_{\omega_{\phi'}^*(a_{1:T}|s)p(s_{T+1}|s,a_{1:T})} [\log p_{\psi'}(a_{1:T} | s, s_{T+1}) - \log p_{\psi}(a_{1:T} | s, s_{T+1})] \\
&\quad - \mathbb{E}_{\omega_{\phi'}^*(a_{1:T}|s)p(s_{T+1}|s,a_{1:T})} \left[ \left( \frac{\omega_{\phi}^*(a_{1:T} | s)}{\omega_{\phi'}^*(a_{1:T} | s)} - 1 \right) \log p_{\psi}(a_{1:T} | s, s_{T+1}) \right] \\
&\approx \mathbb{H}\left(A_{1:T}^{\phi'} \mid s\right) - \mathbb{H}\left(A_{1:T}^{\phi} \mid s\right) \\
&\quad + \mathbb{E}_{\omega_{\phi'}^*(a_{1:T}|s)p(s_{T+1}|s,a_{1:T})} [\log p_{\psi'}(a_{1:T} | s, s_{T+1}) - \log p_{\psi}(a_{1:T} | s, s_{T+1})].
\end{aligned}$$

The approximation is exact if and only if  $\omega_{\phi'}^*(a_{1:T} | s)$  is equal to  $\omega_{\phi}^*(a_{1:T} | s)$ . What this approximation conceptually means is the following: if we look at the three lines before we introduce the approximation, the first line captures change in entropies of the source distribution, the second line measures the improvement of the (inverse) model (expectation is taken over the new source distribution  $\omega_{\phi'}^*(a_{1:T} | s)$ ), the third line accounts for the fact that the optimal source distribution (our distribution of relevant skills) may have changed given our new understanding of the world. Hence we should have computed the performance of our old (inverse) model over this old source distribution. This last term we drop to make our approximation. The approximation is exact if and only if the source distribution (set of skills) did not change at all and it becomes tighter as the two source distributions become closer to each other. It should be stated that we make this (and following approximations) mainly to gain and explain conceptual understanding of what constitutes EG. EG itself is still defined as the exact difference of the new and old empowerment estimate. We use the exact values in our experiments, not the approxi-

mations. However, we do hope that these approximations may also become useful when trying to derive more practical and efficient algorithms based on EG in future work.

### A.3.2. Model-Based Empowerment Gain

#### Model-Based Empowerment Gain

For model-based empowerment gain, we derive at the approximation of eq. (4.14) with

$$\begin{aligned}
& \hat{\mathcal{E}}_{\theta'}^T(s) - \hat{\mathcal{E}}_{\theta}^T(s) \\
&= \max_{\omega_{\phi'}(a_{1:T}|s)} \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi'}(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T}) \log \frac{p_{\xi'}(s_{T+1} | s, a_{1:T})}{p_{\phi', \xi'}(s_{T+1} | s)} \\
&\quad - \max_{\omega_{\phi}(a_{1:T}|s)} \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi}(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T}) \log \frac{p_{\xi}(s_{T+1} | s, a_{1:T})}{p_{\phi, \xi}(s_{T+1} | s)} \\
&\quad \text{(Assuming } \omega_{\phi'}^*(a_{1:T} | s) \text{ and } \omega_{\phi}^*(a_{1:T} | s) \text{ to be the resp. maximizing distribution)} \\
&= \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T}) \log p_{\xi'}(s_{T+1} | s, a_{1:T}) \\
&\quad + \mathbb{H}\left(S_{T+1}^{\phi', \xi'} = p_{\phi', \xi'}(s_{T+1} | s) \mid s\right) \\
&\quad - \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi}^*(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T}) \log p_{\xi}(s_{T+1} | s, a_{1:T}) \\
&\quad - \mathbb{H}\left(S_{T+1}^{\phi, \xi} = p_{\phi, \xi}(s_{T+1} | s) \mid s\right) \\
&= \mathbb{H}\left(S_{T+1}^{\phi', \xi'} \mid s\right) - \mathbb{H}\left(S_{T+1}^{\phi, \xi} \mid s\right) \\
&\quad + \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T}) \log p_{\xi'}(s_{T+1} | s, a_{1:T}) \\
&\quad - \int_{a_{1:T}} \int_{s_{T+1}} \omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T}) \\
&\quad \quad \left[ \frac{\omega_{\phi}^*(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T})}{\omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T})} \log p_{\xi}(s_{T+1} | s, a_{1:T}) \right] \\
&\quad \text{with } \alpha = \frac{\omega_{\phi}^*(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T})}{\omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T})} \\
&= \mathbb{H}\left(S_{T+1}^{\phi', \xi'} \mid s\right) - \mathbb{H}\left(S_{T+1}^{\phi, \xi} \mid s\right)
\end{aligned}$$

$$\begin{aligned}
& + \int_{a_{1:T}} \omega_{\phi'}^*(a_{1:T} \mid s) \int_{s_{T+1}} p_{\xi'}(s_{T+1} \mid s, a_{1:T}) \\
& \quad [\log p_{\xi'}(s_{T+1} \mid s, a_{1:T}) - \alpha \log p_{\xi}(s_{T+1} \mid s, a_{1:T})] \\
& = \mathbb{H}(S_{T+1}^{\phi', \xi'} \mid s) - \mathbb{H}(S_{T+1}^{\phi, \xi} \mid s) \\
& \quad + \mathbb{E}_{\omega_{\phi'}^*(a_{1:T} \mid s)} [\text{KL}(p_{\xi'}(s_{T+1} \mid s, a_{1:T}) \parallel p_{\xi}(s_{T+1} \mid s, a_{1:T}))] \\
& \quad - \mathbb{E}_{\omega_{\phi'}^*(a_{1:T} \mid s) p_{\xi'}(s_{T+1} \mid s, a_{1:T})} [(\alpha - 1) \log p_{\xi}(s_{T+1} \mid s, a_{1:T})] \\
& \approx \mathbb{H}(S_{T+1}^{\phi', \xi'} \mid s) - \mathbb{H}(S_{T+1}^{\phi, \xi} \mid s) \\
& \quad + \mathbb{E}_{\omega_{\phi'}^*(a_{1:T} \mid s)} [\text{KL}(p_{\xi'}(s_{T+1} \mid s, a_{1:T}) \parallel p_{\xi}(s_{T+1} \mid s, a_{1:T}))].
\end{aligned}$$

The approximation is relatively tight for small updates of  $\omega_{\phi'}^*(a_{1:T} \mid s)$  and  $p_{\xi}(s_{T+1} \mid s, a_{1:T})$ .

### A Different View on Empowerment Gain

As mutual information is inherently symmetric, we can rewrite empowerment in terms of the action distribution instead of final state distribution as we have done for the model-free case, that is

$$\hat{\mathcal{E}}_{\theta}^T(s) = \max_{\omega_{\phi}(a_{1:T} \mid s)} \mathbb{E}_{\omega_{\phi}(a_{1:T} \mid s) p_{\xi}(s_{T+1} \mid s, a_{1:T})} [\log p_{\psi}(a_{1:T} \mid s, s_{T+1}) - \log \omega_{\phi}(a_{1:T} \mid s)]. \quad (\text{A.1})$$

Similar to above, we can derive an approximation of the EG

$$\begin{aligned}
& \hat{\mathcal{E}}_{\theta'}^T(s) - \hat{\mathcal{E}}_{\theta}^T(s) \\
& \approx \mathbb{H}(A_{1:T}^{\phi'} \mid s) - \mathbb{H}(A_{1:T}^{\phi} \mid s) \\
& \quad + \mathbb{E}_{\omega_{\phi'}^*(a_{1:T} \mid s) p_{\xi'}(s_{T+1} \mid s, a_{1:T})} [\log p_{\psi'}(a_{1:T} \mid s, s_{T+1}) - \log p_{\psi}(a_{1:T} \mid s, s_{T+1})].
\end{aligned} \quad (\text{A.2})$$

Here, the focus shifts to the empowerment-realizing source distribution  $\omega_{\phi'}(a_{1:T} \mid s)$ , which should be a highly entropic action distribution of distinguishable (by their outcome, final state  $s'$ ) skills. Again, these two terms form a trade-off where more diverse actions are only helpful if they lead to distinguishable outcomes. However, a perfect inverse model alone is not helpful if we are limited to a small number of skills. We derive the

approximation of eq. (A.2) with

$$\begin{aligned}
& \hat{\mathcal{E}}_{\theta'}^T(s) - \hat{\mathcal{E}}_{\theta}^T(s) \\
&= \max_{\omega_{\phi'}(a_{1:T}|s)} \int_{a_{1:T}} \int_{s_{t+1}} \omega_{\phi'}(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T}) \log \frac{p_{\psi'}(a_{1:T} | s, s_{T+1})}{\omega_{\phi'}(a_{1:T} | s)} \\
&\quad - \max_{\omega_{\phi}(a_{1:T}|s)} \int_{a_{1:T}} \int_{s_{t+1}} \omega_{\phi}(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T}) \log \frac{p_{\psi}(a_{1:T} | s, s_{T+1})}{\omega_{\phi}(a_{1:T} | s)} \\
&\quad \text{(Assuming } \omega_{\phi'}^*(a_{1:T} | s) \text{ and } \omega_{\phi}^*(a_{1:T} | s) \text{ to be the resp. maximizing distribution)} \\
&= \int_{a_{1:T}} \int_{s_{t+1}} \omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T}) \log p_{\psi'}(a_{1:T} | s, s_{T+1}) \\
&\quad + \mathbb{H}\left(A_{1:T}^{\phi'} = \omega_{\phi'}(a_{1:T} | s) \mid s\right) \\
&\quad - \int_{a_{1:T}} \int_{s_{t+1}} \omega_{\phi}^*(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T}) \log p_{\psi}(a_{1:T} | s, s_{T+1}) \\
&\quad - \mathbb{H}\left(A_{1:T}^{\phi} = \omega_{\phi}(a_{1:T} | s) \mid s\right) \\
&= \mathbb{H}\left(A_{1:T}^{\phi'} \mid s\right) - \mathbb{H}\left(A_{1:T}^{\phi} \mid s\right) \\
&\quad + \int_{a_{1:T}} \int_{s_{t+1}} \omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T}) \log p_{\psi'}(a_{1:T} | s, s_{T+1}) \\
&\quad - \int_{a_{1:T}} \int_{s_{t+1}} \omega_{\phi}^*(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T}) \left[ \right. \\
&\quad \quad \left. \frac{\omega_{\phi}^*(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T})}{\omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T})} \log p_{\psi}(a_{1:T} | s, s_{T+1}) \right] \\
&= \mathbb{H}\left(A_{1:T}^{\phi'} \mid s\right) - \mathbb{H}\left(A_{1:T}^{\phi} \mid s\right) \\
&\quad + \mathbb{E}_{\omega_{\phi'}^*(a_{1:T}|s)p_{\xi'}(s_{T+1}|s,a_{1:T})} [\log p_{\psi'}(a_{1:T} | s, s_{T+1}) - \log p_{\psi}(a_{1:T} | s, s_{T+1})] \\
&\quad - \mathbb{E}_{\omega_{\phi'}^*(a_{1:T}|s)p_{\xi'}(s_{T+1}|s,a_{1:T})} \left[ \right. \\
&\quad \quad \left. \left( \frac{\omega_{\phi}^*(a_{1:T} | s) p_{\xi}(s_{T+1} | s, a_{1:T})}{\omega_{\phi'}^*(a_{1:T} | s) p_{\xi'}(s_{T+1} | s, a_{1:T})} - 1 \right) \log p_{\psi}(a_{1:T} | s, s_{T+1}) \right] \\
&\approx \mathbb{H}\left(A_{1:T}^{\phi'} \mid s\right) - \mathbb{H}\left(A_{1:T}^{\phi} \mid s\right) \\
&\quad + \mathbb{E}_{\omega_{\phi'}^*(a_{1:T}|s)p_{\xi'}(s_{T+1}|s,a_{1:T})} [\log p_{\psi'}(a_{1:T} | s, s_{T+1}) - \log p_{\psi}(a_{1:T} | s, s_{T+1})].
\end{aligned}$$

---

The approximation is relatively tight for small updates of  $\omega_\phi^*(a_{1:T} \mid s)$  and  $p_\xi(s_{T+1} \mid s, a_{1:T})$ .

### A.3.3. Experiments

For the environments in Section 4.3.3, the noise levels are: small noise  $p_a = 0.05$ , medium noise  $p_a = 0.1$  and large noise  $p_a = 0.2$ . There is noise when performing action "up" (with probability  $p_a$ ) and "left" (with probability  $2p_a$ ). We built our environments on top of MiniGrid (Chevalier-Boisvert et al. (2018), Apache License 2.0).

#### Model Details

The forward model of each cell is modeled by a Categorical distribution with a Dirichlet( $\alpha$ ) prior with  $\alpha = 0.01$ . For empowerment estimation, Blahut-Arimoto is run until the change of empowerment from one iteration to the next is less than  $1e-6$  or after at most 1000 iterations, whichever comes first.

#### Technical details

The implementation was done in PyTorch (Paszke et al. (2019), published under a BSD-style license). All individual experiments were run using a single CPU core each (on e. g. a Intel Xeon Gold 6252 or comparable) and took from less than an hour to up to a day to finish (for the longer horizon empowerment experiments). However, limited time was spent on optimizing the implementation for performance and huge improvements are likely possible. The computation was done on our internal compute cluster.

### A.3.4. An Alternative Empowerment Gain Formulation

As an alternative objective to eq. (4.12), we want to discuss a different formulation

$$a_{1:T}^* = \arg \max_{a_{1:T}} \mathbb{E}_{d=(s_1, a_1, s_2, a_2, \dots, s_T, a_T, s_{T+1})} \left[ \hat{\mathcal{E}}_{\theta'}^T(s_1) - \hat{\mathcal{E}}_{\theta}^T(s_1) \right] \quad (\text{A.3})$$

that focuses on action sequences instead of individual actions. Different to the original formulation, here we are comparing and choosing the best action sequence over an horizon  $T$  that matches the empowerment's horizon. Intuitively, as we compute the empowerment over a specific horizon, it might be helpful to consider how execution of a whole skill over the same horizon instead of just a single action impacts our empowerment estimate. This would allow us to understand the impact of exploring a whole skill and potentially

lead to temporally extended and more directed exploratory behavior. In particular when our forward model does not have the capacity to generalize over the state-action space, comparing EG after and before experiencing a single state action transition modifies empowerment only in so far as it changes the transition probabilities from the starting state. But while choosing among skills may be desirable, it problematically comes with greater computational cost. Naively computing the argmax becomes exponentially more expensive with the horizon  $T$  as we now need to compare all possible action sequences instead of individual actions.

### A.3.5. Empowerment in Partially Observable Environments

Having characterized empowerment gain, let us succinctly discuss its general applicability to partially observable settings. Instead of working with the true system state  $s$ , an agent typically observes only partial and noisy information about the world through its sensors. Indeed, this is how empowerment is typically defined and applied. Using observations  $o = f(s)$  of an arbitrary function  $f$ , one can show that empowerment conditioned on observations is upper bounded by the corresponding empowerment value conditioned on the true system state (Klyubin et al., 2008). Let  $b_t$  be a believe state given a history  $h_{\leq t} = (o_1, a_1, o_2, \dots, a_{t-1}, o_t)$ , then empowerment can be expressed as

$$\mathcal{E}^T(b) = \max_{A_{1:T}} \mathcal{I}(S', A_{1:T} \mid b) \leq \max_{A_{1:T}} \mathcal{I}(S', A_{1:T} \mid s) = \mathcal{E}^T(s). \quad (\text{A.4})$$

Intuitively, empowerment on a belief state is a lower bound as we can not condition on more information than the true system state. However, eq. (A.4) still relies on the true (final) state distribution to compute empowerment. To alleviate this, one can introduce another lower bound

$$\max_{A_{1:T}} \mathcal{I}(O', A_{1:T} \mid b) \leq \max_{A_{1:T}} \mathcal{I}(S', A_{1:T} \mid b) = \mathcal{E}^T(b) \quad (\text{A.5})$$

on  $\mathcal{E}^T(b)$  by using observations instead of system states, where the lower bound derives from the information processing inequality. Thus, the previously introduced formalism can readily be adopted to partially observable environments using common techniques for state estimation with e. g. latent dynamics models.





---

## Bibliography

---

- M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.
- P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, 2007.
- J. Achiam and S. Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.
- G. Ackerson and K. Fu. On state estimation in switching environments. *IEEE Transactions on Automatic Control*, 15(1):10–17, 1970.
- D. B. F. Agakov. The im algorithm: a variational approach to information maximization. *Advances in neural information processing systems*, 16:201, 2004.
- I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik’s cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.
- A. Alemi, B. Poole, I. Fischer, J. Dillon, R. A. Saurous, and K. Murphy. Fixing a broken elbo. In *International Conference on Machine Learning*, pages 159–168, 2018.
- O. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. doi: 10.1177/0278364919887447.
- S. Arimoto. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, 18(1):14–20, 1972.
- K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

- 
- 
- A. Aubret, L. Matignon, and S. Hassas. A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*, 2019.
- J. A. Bagnell and J. G. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE, 2001.
- D. Barber. *Bayesian Reasoning and Machine Learning*, pages 547–567. Cambridge University Press, 2012.
- A. Barto, M. Mirolli, and G. Baldassarre. Novelty or surprise? *Frontiers in psychology*, 4: 907, 2013.
- P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.
- J. Bayer and C. Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.
- P. Becker-Ehmck, J. Peters, and P. van der Smagt. Switching linear dynamics for variational bayes filtering. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.
- P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Learning to fly via deep model-based reinforcement learning. *ICRA 2020 Workshop on Machine Learning in Planning and Control of Robot Motion (MLPC)*, 2020a.
- P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Learning to control real robots via deep model-based reinforcement learning. *Submitted to IEEE Transactions on Neural Networks and Learning Systems*, 2020b.
- P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Exploration via empowerment gain: Combining novelty, surprise and learning progress. *ICML 2021 Workshop on Unsupervised Reinforcement Learning (URL)*, 2021.
- P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Exploration via empowerment gain: Combining novelty, surprise and learning progress. *Submitted to Frontiers in Robotics and AI*, 2022a.

- 
- P. Becker-Ehmck, J. Peters, and P. van der Smagt. Switching deep variational bayes filter for time-varying dynamics. *Submitted to Journal of Machine Learning Research (JMLR)*, 2022b.
- M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems 29*, pages 1471–1479, 2016.
- R. Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.
- D. E. Berlyne. Conflict, arousal, and curiosity. 1960.
- Betaflight. Betaflight – open source flight controller firmware. <https://github.com/betaflight/betaflight>, 2020.
- S. Blaes, M. Vlastelica Pogančić, J. Zhu, and G. Martius. Control what you can: Intrinsically motivated task-planning agent. *Advances in Neural Information Processing Systems*, 32, 2019.
- R. Blahut. Computation of channel capacity and rate-distortion functions. *IEEE transactions on Information Theory*, 18(4):460–473, 1972.
- G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- A. Byravan, J. T. Springenberg, A. Abdolmaleki, R. Hafner, M. Neunert, T. Lampe, N. Siegel, N. Heess, and M. Riedmiller. Imagined value gradients: Model-based policy optimization with transferable latent dynamics models. In *Proceedings of the 3rd Conference on Robot Learning*, 2019.
- C.-B. Chang and M. Athans. State estimation for discrete systems with switching parameters. *IEEE Transactions on Aerospace and Electronic Systems*, 1978.
- Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.
- M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic gridworld environment for openai gym. <https://github.com/maximecb/gym-minigrid>, 2018.

- 
- 
- J. Choi, A. Sharma, H. Lee, S. Levine, and S. S. Gu. Variational empowerment as representation learning for goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, pages 1953–1963. PMLR, 2021.
- J. Chu and L. E. Schulz. Play, curiosity, and cognition. *Annual Review of Developmental Psychology*, 2:317–343, 2020.
- K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.
- J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.
- E. Coumans and Y. Bai. PyBullet, a python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>, 2016.
- B. Dai, Y. Wang, J. Aston, G. Hua, and D. Wipf. Hidden talents of the variational autoencoder. *arXiv preprint arXiv:1706.05148*, 2017.
- F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pages 7178–7189, 2018.
- M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.
- J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.
- A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe. Probabilistic recurrent state-space models. *arXiv preprint arXiv:1801.10395*, 2018.
- Z. Dong, B. Seybold, K. Murphy, and H. Bui. Collapsed amortized variational inference for switching nonlinear dynamical systems. In *International Conference on Machine Learning*, pages 2638–2647. PMLR, 2020.
- G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*, 2020.

- 
- 
- S. Eleftheriadis, T. Nicholson, M. Deisenroth, and J. Hensman. Identification of gaussian process state space models. In *Advances in Neural Information Processing Systems*, pages 5309–5319, 2017.
- B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.
- V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-based value estimation for efficient model-free reinforcement learning. *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.
- C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.
- R. FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961.
- M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pages 2199–2207, 2016.
- M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems*, pages 3604–3613, 2017.
- F. Frank, A. Paraschos, and P. van der Smagt. Orc – a lightweight, lightning-fast middleware. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 337–343. IEEE, 2019.
- Z. Fu, A. Kumar, J. Malik, and D. Pathak. Minimizing energy consumption leads to the emergence of gaits in legged robots. *arXiv preprint arXiv:2111.01674*, 2021. URL <https://sites.google.com/view/energy-loco/>.
- S. Fujimoto, H. Van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.
- P. W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

- 
- 
- A. Goyal, A. Sordoni, M.-A. Côté, N. Ke, and Y. Bengio. Z-forcing: Training stochastic recurrent networks. In *Advances in Neural Information Processing Systems*, pages 6713–6723, 2017.
- K. Gregor, D. J. Rezende, and D. Wierstra. Variational Intrinsic Control. *arXiv preprint arXiv:1611.07507*, 2016.
- S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE, 2017.
- D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462. 2018.
- T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870, 2018.
- T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems (RSS)*, 2019.
- D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.
- E. Hazan, S. Kakade, K. Singh, and A. Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR, 2019.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2015.
- I. Higgins, L. Matthey, X. Glorot, A. Pal, B. Uria, C. Blundell, S. Mohamed, and A. Lerchner. Early visual concept learning with unsupervised deep learning. *arXiv preprint arXiv:1606.05579*, 2016.

- 
- 
- S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.
- R. Houthoofd, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29, 2016.
- J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.
- J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26): eaau5872, 2019.
- L. Itti and P. Baldi. A principled approach to detecting surprising events in video. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 631–637. IEEE, 2005.
- L. Itti and P. Baldi. Bayesian surprise attracts human attention. *Vision research*, 49(10): 1295–1306, 2009.
- L. Itti and P. F. Baldi. Bayesian surprise attracts human attention. In *Advances in neural information processing systems*, pages 547–554. Citeseer, 2006.
- E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.
- D. Jayaraman, F. Ebert, A. A. Efros, and S. Levine. Time-agnostic prediction: Predicting predictable video frames. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.
- M. Johnson, D. K. Duvenaud, A. Wiltschko, R. P. Adams, and S. R. Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.
- T. Jung, D. Polani, and P. Stone. Empowerment for continuous agent—environment systems. *Adaptive Behavior*, 19(1):16–39, 2011.
- D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proceedings of the 2nd Conference on Robot Learning*, pages 651–673, 2018.

- 
- D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.
- M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.
- M. Karl, P. Becker-Ehmck, M. Soelch, D. Benbouzid, P. van der Smagt, and J. Bayer. Unsupervised Real-Time Control through Variational Empowerment. In *International Symposium on Robotics Research*, 2019.
- E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *Proceedings of the 2nd Conference on Robot Learning*, 2018.
- D. Kingma and M. Welling. Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.
- D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.
- D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.
- T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.
- A. S. Klyubin, D. Polani, and C. L. Nehaniv. Empowerment: A universal agent-centric measure of control. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 128–135. IEEE, 2005a.
- A. S. Klyubin, D. Polani, and C. L. Nehaniv. All else being equal be empowered. In *European Conference on Artificial Life*, pages 744–753. Springer, 2005b.
- A. S. Klyubin, D. Polani, and C. L. Nehaniv. Keep your options open: An information-based driving principle for sensorimotor systems. *PLoS ONE*, 3(12):1–14, dec 2008. doi: 10.1371/journal.pone.0004018. Publisher: Public Library of Science.



- 
- 
- W. Koch, R. Mancuso, and A. Bestavros. Neuroflight: Next generation flight control firmware. *arXiv preprint arXiv:1901.06553*, 2019.
- R. G. Krishnan, U. Shalit, and D. Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.
- R. G. Krishnan, U. Shalit, and D. Sontag. Structured inference networks for nonlinear state space models. In *AAAI*, pages 2101–2109, 2017.
- A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.
- R. Kurlle, S. S. Rangapuram, E. de Bézenac, S. Günnemann, and J. Gasthaus. Deep rao-blackwellised particle filters for time series forecasting. *Advances in Neural Information Processing Systems*, 33:15371–15382, 2020.
- N. O. Lambert, D. S. Drew, J. Yaconelli, R. Calandra, S. Levine, and K. S. Pister. Low level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.
- A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33:741–752, 2020a.
- J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020b.
- K. Lee, Y. Seo, S. Lee, H. Lee, and J. Shin. Context-aware dynamics model for generalization in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 5757–5766. PMLR, 2020c.
- L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.
- S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.
- S. Li, T. Liu, C. Zhang, D.-Y. Yeung, and S. Shen. Learning unmanned aerial vehicle control for autonomous target following. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

- 
- 
- T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.
- J. Lin, L. Wang, F. Gao, S. Shen, and F. Zhang. Flying through a narrow gap using neural network: an end-to-end planning and control approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.
- S. Linderman, M. Johnson, A. Miller, R. Adams, D. Blei, and L. Paninski. Bayesian Learning and Inference in Recurrent Switching Linear Dynamical Systems. In A. Singh and J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 914–922, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.
- H. Liu, R. Socher, and C. Xiong. Taming maml: Efficient unbiased meta-reinforcement learning. In *International conference on machine learning*, pages 4061–4071. PMLR, 2019.
- M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Neural Information Processing Systems (NIPS)*, 2012.
- A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transaction on Robotics (T-RO)*, 2019.
- M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.
- D. J. MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.
- C. J. Maddison, A. Mnih, and Y. W. Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.
- J. Marino, M. Cvitkovic, and Y. Yue. A general method for amortizing variational filtering. In *Advances in Neural Information Processing Systems*, pages 7868–7879, 2018.

- 
- 
- V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
- V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.
- S. Mohamed and D. J. Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2125–2133, 2015.
- A. S. Morgan, D. Nandha, G. Chalvatzaki, C. D’Eramo, A. M. Dollar, and J. Peters. Model predictive actor-critic: Accelerating robot skill acquisition with deep reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6672–6678. IEEE, 2021.
- J. Nassar, S. Linderman, M. Bugallo, and I. M. Park. Tree-structured recurrent switching linear dynamical systems for multi-scale modeling. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.
- A. Neitz, G. Parascandolo, S. Bauer, and B. Schölkopf. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. In *Advances in Neural Information Processing Systems*, pages 9838–9848, 2018.
- A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX*, pages 363–372. Springer, 2006.
- OpenAI. Roboschool: open-source software for robot simulation. <https://github.com/openai/roboschool>, 2017.
- G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.
- P.-Y. Oudeyer and F. Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009.
- P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

- 
- 
- A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017.
- D. Pathak, D. Gandhi, and A. Gupta. Self-supervised exploration via disagreement. In *International Conference on Machine Learning*, pages 5062–5071. PMLR, 2019.
- X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.
- A. Piergiovanni, A. Wu, and M. S. Ryoo. Learning real-world robot policies by dreaming. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.
- M. Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.
- K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.
- S. Ramstedt and C. Pal. Real-time reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3067–3076, 2019.
- N. Ratzlaff, Q. Bai, L. Fuxin, and W. Xu. Implicit generative modeling for efficient exploration. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7985–7995. PMLR, 13–18 Jul 2020.
- S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

- 
- 
- D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.
- C. Salge, C. Glackin, and D. Polani. Approximation of empowerment in the continuous domain. *Advances in Complex Systems*, 16(02n03):1250079, 2013a.
- C. Salge, C. Glackin, and D. Polani. Empowerment and state-dependent noise: An intrinsic motivation for avoiding unpredictable agents. *Advances in Artificial Life, ECAL 2013*, 2013b.
- C. Salge, C. Glackin, and D. Polani. Empowerment—an introduction. In *Guided Self-Organization: Inception*, pages 67–114. Springer, 2014.
- A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.
- N. Savinov, A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. Lillicrap, and S. Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018.
- J. Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463, 1991a.
- J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991b.
- J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.
- R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.
- S. Shabanian, D. Arpit, A. Trischler, and Y. Bengio. Variational bi-lstms. *arXiv preprint arXiv:1711.05717*, 2017.
- A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

- 
- 
- G. Shi, X. Shi, M. O’Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural lander: Stable drone landing control using learned dynamics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.
- P. Shyam, W. Jaśkowski, and F. Gomez. Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788. PMLR, 2019.
- D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.
- X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan. Rapidly adaptable legged robots via evolutionary meta-learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3769–3776. IEEE, 2020.
- B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.
- J. Storck, S. Hochreiter, and J. Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the international conference on artificial neural networks, Paris*, volume 2, pages 159–164. Citeseer, 1995.
- R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.
- R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.
- R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.
- J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems (RSS)*, 2018.
- H. Tang, R. Houthoofd, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *31st Conference on Neural Information Processing Systems (NIPS)*, volume 30, pages 1–18, 2017.

- 
- 
- Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. *dm\_control: Software and tasks for continuous control*, 2020.
- S. B. Thrun. *Efficient exploration in reinforcement learning*. 1992.
- J. Tomczak and M. Welling. Vae with a vampprior. In *International Conference on Artificial Intelligence and Statistics*, pages 1214–1223, 2018.
- J. B. Travník, K. W. Mathewson, R. S. Sutton, and P. M. Pilarski. Reactive reinforcement learning in asynchronous environments. *Frontiers in Robotics and AI*, 5:79, 2018.
- H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.
- S. van Steenkiste, M. Chang, K. Greff, and J. Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
- T. J. Walsh, A. Nouri, L. Li, and M. L. Littman. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems*, 18(1):83, 2009.
- Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.
- C. J. C. H. Watkins. *Learning from delayed rewards*. 1989.
- M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.
- R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

- 
- 
- H. A. Xu, A. Modirshanechi, M. P. Lehmann, W. Gerstner, and M. H. Herzog. Novelty is not surprise: Human exploratory and adaptive behavior in sequential decision-making. *bioRxiv*, pages 2020–09, 2021.
- M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine. SOLAR: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, pages 7444–7453, 2019.
- R. Zhao, K. Lu, P. Abbeel, and S. Tiomkin. Efficient empowerment estimation for unsupervised stabilization. In *International Conference on Learning Representations*, 2021.
- T. Z. Zhao, A. Nagabandi, K. Rakelly, C. Finn, and S. Levine. Meld: Meta-reinforcement learning from images via latent state models. *arXiv preprint arXiv:2010.13957*, 2020.
- Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.



---

## List of Algorithms

---

1. Model-Based Actor-Critic . . . . . 41
2. Exploration Algorithm . . . . . 71



---

## List of Figures

---

1.1. Conceptual overview of Switching DVBF. . . . .	5
1.2. Conceptual drawing of Model-Based Actor-Critic. . . . .	6
1.3. Thesis outline . . . . .	7
2.1. SLDS and Switching DVBF generative model. . . . .	11
2.2. Switching DVBF inference model . . . . .	16
2.3. Multiple bouncing balls in a maze. . . . .	21
2.4. Learned latent state-space in the maze . . . . .	22
2.5. Learned latent state-space of the reacher. . . . .	24
2.6. Image ball-in-a-box reconstructions and predictions . . . . .	25
2.7. Prediction errors . . . . .	26
2.8. Prediction error influenced by time discretization. . . . .	26
2.9. Model predictions on the time-varying pendulum. . . . .	28
2.10. Encoding of time-varying properties governing system dynamics. . . . .	29
2.11. Reinforcement learning results in the time-varying pendulum environment. . . . .	31
3.1. Flying to goal marker . . . . .	36
3.2. Drone . . . . .	43
3.3. Communication graph . . . . .	45
3.4. Timing of the control loop. . . . .	46
3.5. Drone trajectory with policy quiver . . . . .	48
3.6. Control error over time (no yaw) . . . . .	50
3.7. Control error over time . . . . .	53
3.8. Model prediction quality . . . . .	54
3.9. Panda environment . . . . .	55
3.10. Panda policy trajectories . . . . .	58
4.1. State visitation frequency in deterministic environments. . . . .	73
4.2. State visitation frequency in prison state environment. . . . .	74
4.3. State visitation frequency in varying action noise environments. . . . .	75

---

---

4.4. State visitation frequency in state-dependent action noise environments. . . . .	77
4.5. Chain environment . . . . .	78

---

## List of Tables

---

2.1. Prediction error of reacher and maze. . . . .	20
2.2. Prediction error in time-varying dynamics. . . . .	27
2.3. Reinforcement learning results in the time-varying pendulum environment. . . . .	30
3.1. Drone hardware components . . . . .	44
3.2. Drone sensor streams . . . . .	47
3.3. Fly to marker: quantitative results (yaw disabled) . . . . .	51
3.4. Fly to marker: quantitative results (yaw enabled) . . . . .	52
3.5. Normalization ranges . . . . .	56
3.6. Panda task results . . . . .	57
4.1. State visitation frequency in the chain environment. . . . .	78
A.1. Dimensionality of environments. . . . .	89
A.2. Overview of hyperparameters. . . . .	90
A.3. Overview of hyperparameters for experiments with time-varying dynamics. . . . .	91
A.4. Hyperparameters for drone and arm experiments. . . . .	94
A.5. SAC hyperparameters for experiments on the Panda arm. . . . .	95



---

# Publications

---

## Journals

P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Learning to control real robots via deep model-based reinforcement learning. *Submitted to IEEE Transactions on Neural Networks and Learning Systems*, 2020b

P. Becker-Ehmck, J. Peters, and P. van der Smagt. Switching deep variational bayes filter for time-varying dynamics. *Submitted to Journal of Machine Learning Research (JMLR)*, 2022b

P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Exploration via empowerment gain: Combining novelty, surprise and learning progress. *Submitted to Frontiers in Robotics and AI*, 2022a

## Conferences

P. Becker-Ehmck, J. Peters, and P. van der Smagt. Switching linear dynamics for variational bayes filtering. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019

M. Karl, P. Becker-Ehmck, M. Soelch, D. Benbouzid, P. van der Smagt, and J. Bayer. Unsupervised Real-Time Control through Variational Empowerment. In *International Symposium on Robotics Research*, 2019

## Workshops

P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Learning to fly via deep model-based reinforcement learning. *ICRA 2020 Workshop on Machine Learning in Planning and*

---

*Control of Robot Motion (MLPC)*, 2020a

P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Exploration via empowerment gain: Combining novelty, surprise and learning progress. *ICML 2021 Workshop on Unsupervised Reinforcement Learning (URL)*, 2021



---

# Curriculum Vitae

---

## Experience

- Since 2020 **Research Scientist.** Volkswagen Group, Machine Learning Research Lab.  
Research on model-based reinforcement learning and their application in various domains.  
Implementation of a reinforcement learning framework with Python and PyTorch.  
Integration of real robotic platforms for Machine Learning using NVIDIA Isaac SDK.  
Creation and maintenance of an on-premise computational cluster: Kubernetes, Ansible, Docker.
- 2017 – 2020 **Ph.D. Student.** Volkswagen Group, Machine Learning Research Lab.  
Research on learned model-based control methods of real robots.  
Application of developed methods for prediction of battery temperature of a Formula E car (Audi Motorsport).
- 2013 – 2015 **Software Engineer (Working Student).** Devoss GmbH.  
Development and documentation of monitoring systems for the global backup operations of BMW with Python 3 and Shell scripting. Supported development of SAN (storage area network) administration platform (PHP, HTML, JavaScript).

---

---

## Education

- Since 2017 **Ph.D.** in Machine Learning & Robotics at Technical University of Darmstadt: Intelligent Autonomous Systems Group and Volkswagen Group: Machine Learning Research Lab.  
Title: Latent State-Space Models for Control  
Advisor: Prof. Dr. Jan Peters
- 2014 – 2017 **Master of Science** in Computer Science at Technical University of Munich.  
Grade – 1.1 (very good)  
Thesis: Learning of Latent Space Dynamics via Intrinsically Motivated Exploration
- 2011 – 2014 **Bachelor of Science** in Computer Science at Technical University of Munich.  
Grade – 1.6 (good)  
Thesis: Implementation of Optimal Algorithms for Energy-Efficient Scheduling on Parallel Processors
- 2011 **Higher Education Entrance Qualification (A-levels)** at Feodor-Lynen-Gymnasium, Planegg, Germany.  
Grade – 1.5 (good)

## Key Publications

- 2022 Exploration via Empowerment Gain: Combining Novelty, Surprise and Learning Progress. P. Becker-Ehmck, M. Karl, J. Peters, P. van der Smagt.
- 2020 Learning to Control Real Robots via Deep Model-Based Reinforcement Learning. P. Becker-Ehmck, M. Karl, J. Peters, P. van der Smagt.
- 2019 Switching Linear Dynamics for Variational Bayes Filtering. Proceedings of the 36th International Conference on Machine Learning (ICML). P. Becker-Ehmck, J. Peters, P. van der Smagt.

---

## Awards

2015 – 2017 **Member of best.in.tum.** Technical University of Munich.  
Consists of approximately the top 2% of students at the faculty of Informatics.

## Languages

German Native

English Fluent