Lifelong Learning in the Real World

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

M.Sc. Sarah Maria Elisabeth Bechtle

aus München

Tübingen

2021

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	09.05.2022
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Ludovic Righetti
2. Berichterstatter:	Prof. Dr. Philipp Hennig

For my mother - she was not granted her PhD because of her pregnancy.

Abstract

For robots to assist and support humans in their daily lives, they will have to learn from - and adapt to our world. But, applying machine learning to real world problems is still very challenging. The real world is full of uncertainties and always changing, shifting the distribution of the observed data. This means: a static dataset is not representative of the real world for long. Additionally, data-collection is expensive and only small amounts of data can be collected for a task. In this setting, the algorithms developed for the static domain using large datasets will not transfer well, even if they have taken part in the machine learning success stories of the recent past.

The work presented in this thesis is concerned with a specific type of real world machine learning application: to endow robots with the capability to learn in the real world. For this to happen the robot needs to gather the data necessary to learn through interactions with the environment. In this thesis the question of how representations can be learned that allow for quick generalization and adaptation to new tasks is of interested. The work presented investigates how the robot can build upon already learned representations to continue learning from experience over its lifetime. This would give the robot a capability that we humans have: to adapt quickly previously learned skills to new tasks. In this thesis we approach the question of how to learn representations that generalize quickly from two different but interconnected directions:

1. **Model Based Learning** in the real world, where a representation of the environment is learned iteratively from data collected on the robot through control.

   The human brain cannot afford to learn every task from scratch, that is why it builds models (Lake *et al.*, 2017). Models come with the promise of flexible adaptation to new tasks without having to re-learn everything each time. But, the learned models can be biased, wrong or old. In Chapter 2 (Bechtle *et al.*, 2020a) we showed how including the predictive uncertainty of learned dynamic models during policy optimization, facilitates exploration by resolving uncertainty in the models, and thus improves task performance, model predictions and generalization. Chapter 3 (Bechtle *et al.*, 2020c) presents an unbiased loss function for controller learning, that trades off predictive task performance and forward model quality. We analyse how this loss facilitates data collection that reduces model bias and as a consequence improves task

learning. Chapter 4 (Bechtle *et al.*, 2020b) tackles the problem of biased models from a different perspective: by leveraging prior analytical knowledge, and combining it with a data driven approach, a visual dynamics model is learned that generalizes well on manipulation tasks.

2. **Learning to Learn** in the real world, where the robot learns how to learn a task and thus a representation of the learning problem.

   Humans have the remarkable capability to continuously learn and adapt to new tasks - we are able to learn how to learn. Inspired by this, we propose on a fully differentiable learning to learn framework (Bechtle *et al.*, 2019) in Chapter 5, enabling robots to learn how to learn. In the learning to learn phase a loss function for a task is learned from experience. Later, after the robot learned how to learn, the loss can be used directly on new tasks. In (Das *et al.*, 2020a) we show how this learning to learn principle can also be used on an object manipulation task where the robot learned from human demonstrations. Learning to learn is a fundamental piece of human intelligence - endowing robots with such ability is a fundamental research question.

These directions fundamentally are concerned with learning representations that generalize quickly to new tasks and scenarios, which is a key capability that allows us humans to continuously learn.

Kurzfassung

Damit Roboter uns Menschen in unserem täglichen Leben unterstützen können, müssen sie von unserer Welt lernen und sich ihr anpassen. Die Anwendung des maschinellen Lernens auf reale Probleme stellt immer noch eine große Herausforderung dar. Die reale Welt ist voller Ungewissheiten und verändert sich ständig, wodurch sich die Verteilung der Beobachtungen ständig verschiebt. Dies bedeutet, dass ein statischer Datensatz die reale Welt nicht lange darstellt. Außerdem ist die Datenerfassung schwieriger und meistens kann immer nur eine kleine Menge an Daten für eine Aufgabe gesammelt werden. Aus diesem Grund lassen sich Algorithmen, die für einen statischen Problembereich und mithilfe große Datensätze entwickelt wurden, nicht gut auf diese Probleme übertragen, auch wenn sie an den jüngsten Erfolgsgeschichten des maschinellen Lernens beteiligt waren.

Diese Arbeit befasst sich mit einer speziellen Art von Anwendung des maschinellen Lernens in der realen Welt und zwar Roboter mit der Fähigkeit auszustatten, in der realen Welt zu lernen. Dazu muss der Roboter durch Interaktionen mit der Umgebung die zum Lernen notwendigen Daten sammeln. Diese Arbeit befasst sich mit der Frage, wie Repräsentationen erlernt werden können, die eine schnelle Generalisierung und Anpassung an neue Aufgaben ermöglichen. Wie kann der Roboter auf bereits Erlerntes aufbauen, um während seiner gesamten Lebenszeit weiter aus Erfahrungen zu lernen? Dies würde dem Roboter eine Fähigkeit verleihen, die auch wir Menschen haben: schnell zuvor erlernte Fähigkeiten an neue Aufgaben anzupassen. Ich möchte verstehen, wie lebenslanges Lernen in diesem Rahmen möglich ist: Der Roboter muss entscheiden, was er wann und wie lernen will.Insbesondere beschäftigt sich die Arbeit mit der Frage, wie man Repräsentationen lernt, die zu schneller Generalisierung führen und dadurch ein schnelles Erlernen neuer Aufgaben ermöglichen. Ich stütze meine Forschung auf Erkenntnisse aus den Neuro- und Kognitionswissenschaften sowie der Entwicklungspsychologie und basiere meine Ansätze auf diesen Erkenntnissen, um Lernalgorithmen für Roboter zu entwickeln.

Der Inhalt dieser Arbeit nähert sich dieser Frage aus zwei verschiedenen, aber miteinander verbundenen Richtungen:

1. **Modellbasiertes Lernen** in der realen Welt: Eine Repräsentation der Umgebung wird iterativ aus den durch den Roboter gesammelten Daten erlernt.

   Das menschliche Gehirn kann nicht jede Aufgabe von Grund auf neu zu erlernen, deshalb baut es Kognitive Modelle der Umgebung (Lake *et al.*, 2017).

Modelle versprechen eine flexible Anpassung an neue Aufgaben, ohne dass jedes Mal alles neu erlernt werden muss. Allerdings können die erlernten Modelle verzerrt, falsch oder alt sein. In Kapitel 2 (Bechtle *et al.*, 2020a) zeigen wir, wie die Einbeziehung der Unsicherheit von Vorhersagen der gelernter dynamischen Modelle während der Optimierung von Regelungsstrategien die Erkundung der Umgebung erleichtert und dadurch relevante Daten erfasst werden können. Das Ziel ist nun, nicht nur eine bestimmte Aufgabe zu erfüllen, aber auch die Unsicherheit in den Modellen aufzulösen. Dadurch verbessert sich nicht nur das Modell, welches dann für andere Aufgaben genutzt werden kann, sondern auch die Leistung der Regelungsstrategie. Kapitel 3 (Bechtle *et al.*, 2020c) stellt eine Verlustfunktion für das Lernen von Regelungsstrategien vor, die auch die Qualität des Modells berücksichtigt. In diesem Fall ist nicht nur von Bedeutung wie gut die gegebene Aufgabe erfüllt wurde, sondern auch wie genau das Modell Vorhersagen getroffen hat. Wir analysieren, wie diese Verlustfunktion die Datenerfassung erleichtert, um ein besseres Modell zu lernen und infolgedessen das Aufgabenlernen verbessert. Kapitel 4 (Bechtle *et al.*, 2020b) betrachtet das Problem voreingenommener Modelle von einer anderen Perspektive: Durch die Nutzung von analytischem Vorwissen und die Kombination mit einem datengesteuerten Ansatz wird ein visuelles Dynamikmodell erlernt, das bei Manipulationsaufgaben gute Ergebnisse erzielt, auch für neue Aufgaben.

2. **Lernen wie man lernt** in der realen Welt. Der Roboter lernt, wie er eine Aufgabe und damit eine Darstellung des Lernproblems erlernen kann.

   Menschen haben die bemerkenswerte Fähigkeit, kontinuierlich zu lernen und sich an neue Aufgaben anzupassen. Wir sind in der Lage zu lernen, wie man lernt. Davon inspiriert schlagen wir in Kapitel 5 ein vollständig differenzierbares Lernsystem vor, das Robotern ermöglicht, zu lernen, wie man lernt. In der Lernphase wird eine Verlustfunktion für eine Aufgabe aus Erfahrungen erlernt. Später, nachdem der Roboter gelernt hat, wie man lernt, kann diese Funktion direkt auf neue Aufgaben angewendet werden. In (Das *et al.*, 2020a) zeigen wir, wie dieses Lernprinzip auch bei einer Objektmanipulationsaufgabe angewendet werden kann, bei der der Roboter von menschlichen Demonstrationen gelernt hat. Lernen zu lernen ist ein grundlegender Bestandteil der menschlichen Intelligenz. Roboter mit dieser Fähigkeit auszustatten ist eine grundlegende Forschungsfrage.

Diese beiden Richtungen befassen sich im Wesentlichen mit dem Erlernen von Repräsentationen, die sich schnell für neue Aufgaben und Szenarien benutzen lassen. Diese Form von Generalisierung ist eine Schlüsselfähigkeit, die es uns Menschen ermöglicht, kontinuierlich zu lernen.

Acknowledgments

First and foremost I want to thank my PhD advisors, Franziska Meier, Ludovic Righetti and Stefan Schaal for really always being there for me and for your constant support throughout the journey of my PhD. Thank you Ludo and Franzi for meeting with me every week, talking through my ideas and always giving me invaluable feedback and suggestions, somehow you always knew how to put in words what I was trying to think about. Franzi thank you for guiding and motivating me from my first day as a PhD student, you always cared for my development as a scientist and as a person. I cannot count the things I have learned from you. Thank you Ludo for being so generous and welcoming me into your lab. I am incredibly thankful that you were willing to be my PhD advisor, to give me advise and to teach me so many things about robots, about life and what it means to be an academic. Thank you Stefan for always supporting and mentoring me in so many aspects of my life. Without your support and your ability to always find a solution many things would not have been possible for me.

Thank you Gaurav Sukhatme and the whole RESL Lab for making me feel so welcome as a visiting student at USC, the people I met there and the collaborations I had significantly shaped my research.

Thank you Akshi for being my friend, for spending so many hours in the robot dungeon together, for teaching me a lot about robots and how to make chai.

Thank you Bilal for our regular life updates over zoom, for really understanding the risk sensitive derivation and for dealing with solo on my behalf.

Thank you Yevgen and Artem for the times spent together at USC, the fun we had and all the work we did watching loss functions (not) being optimized. Thank you Giovanni for the great lunches and conversations we had together, you really always know how to find the best food no matter where we are. Thank you Paarth for reappearing in the Machines in Motion Lab.

Thank you Ahmad, Julian, Majid, Miroslav and the whole Machines in Motion lab for keeping me company on zoom during the pandemic. Thank you Alonso for being my friend and making my Tübingen times so much more enjoyable.

Thank you Todor for the great collaboration and organizing workshops with me, I just realized we never met in real life but it does not feel like that.

Thank you Jordie, Drea and Jenny for our Friday nights and Saturday hikes.

Thank you to my Los Angeles family Beaux and Shiro for being in my life.

Thank you to my family, my brothers Lucas and Marco, my mother and my father and Albi - I know you are always there for me and will always be.

Grazie Ame.

I would like to thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) and the Max Planck Society for supporting this work.

Contents

Chapter 1

Introduction

Humans continuously learn over their lifespan. We are able to acquire, extend and transfer skills easily from one task and context to a different one. We might need to adjust what we learned, re-learn parts or fully learn aspects of the new tasks, but we rarely learn completely from scratch in our adult life. The ability to transfer and re-use what was previously learned is a fundamental aspect of human intelligence. To endow robots with the capability to learn continuously in the real world is still an open research problem and of fundamental interest to this thesis.

In contrast to large scale neural network models that are trained on big datasets for some specific domain, applying machine learning to real world problems comprises different challenges: the real world is full of uncertainties and changing over time. Feedback systems, like robots, acting in the real world will be faced with an ever shifting data distributions of the collected data. For example, a robot acting in a kitchen environment will be faced with dishes placed always in a slightly different way, that nevertheless need to be loaded into the dishwasher. Additionally, large amounts of data might not always be available for learning. It is thus challenging for algorithms that were developed for a more static domain to transfer well since real world data is rarely static and a collected training set will not represent the real world for long.

The work in this thesis is concerned with a specific type of real world machine learning application: to endow robots with the capability to learn in the real world. Specifically learning within the action-perception-learning loop, where the robot gathers the data it needs to learn through interactions with - and perception of - the environment. For lifelong learning to happen within this loop, the robot must decide what to learn, when and how.

Being able to generalize quickly is a key capability that allows us humans to continuously learn. In the context of this thesis the question of how to learn representations that generalize quickly and enable fast learning of new tasks is of particular importance. The work in this thesis is centered around this fundamental research question and is organised around two interconnected directions: 1) **Model Based Learning** in the Action Perception Loop, where a model of the environment and a policy are learned iteratively from data collected on the robot and 2)

Learning to Learn in the Action Perception Loop, where the robot learns how to learn a task, often also referred to as meta-learning. Both of these directions are concerned with learning representations that generalize quickly to new tasks and scenarios.

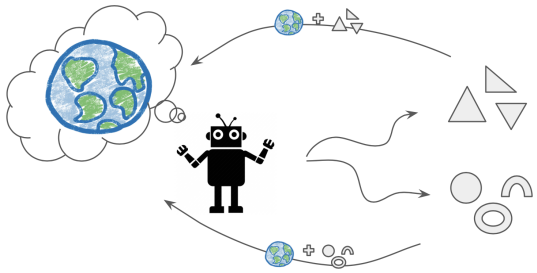


Figure 1.1: Lifelong learning in the real world. The robot acts in the world and collects data. The data is then used to learn more about the world. How to decide which actions to take and what to learn from the data? This is a key challenge in lifelong learning.

1.1 Background

Interacting with and perceiving the environment are fundamental sources of information that allow humans to learn and acquire a myriad of skills over a lifespan. (Parisi *et al.*, 2019) note that, since their moment of birth, children are immersed in a highly dynamical and multi-modal environment. This environment provides children with the opportunity to shape their perception, cognition and motor behaviour. Also robots need to be able to learn from these dynamic and rich environments if we want them to interact, assist and support humans in their daily lives. Just like humans, robots will have to continuously specialize their skills in an experience driven way: by building on top of the already existing, previously learned, sensorimotor contingencies (Tani, 2016). Thus, the fundamental question now is, how to learn from the real world?

Although there are many perspective on learning, in this thesis we align ourselves with the perspective presented in (Lake *et al.*, 2017), where the authors see building of representations, like models, as the hallmark of human level intelligence. Humans build 'cognitive maps' of the worlds that can be seen as models and use them to plan action sequences for complex tasks (Dolan and Dayan, 2013). Another key ingredient of human intelligence is the ability to learn how to learn (Lake *et al.*, 2017), allowing humans to rapidly learn new models from a rather limited amount of new experiences. This is possible because humans are able to learn a representation of how to learn that can quickly be reused in a new context (Lake *et al.*, 2017).

Humans learn from experience they collect throughout their lives. But how to make robots learn from the experience they gather throughout a lifetime? Lifelong learning in the robotics literature has been looked at from various angles. The most holistic view of trying to solve the question of how to continuously learn in an environment with ever changing data distribution can be found in the literature related to developmental robotics (Lungarella *et al.*, 2003). Developmental robotics is the interdisciplinary approach that views the emergence of motor and cognitive capabilities in the artificial agent as analogous to the developmental principles observed in children. In this context lifelong learning means improving skills and increasing the complexity of tasks to handle over time and experience starting from scratch. Even if the developmental approach inherently deals with all aspects of lifelong learning on robots, this endeavour extremely challenging. Breaking down the challenges, more concrete aspects of lifelong learning that have been studied in the literature and are relevant to the work in this thesis will be outlined next.

1.1.1 Intrinsic Motivation

Intrinsically motivated learning has been studied widely as a building block for autonomous lifelong learning (Santucci *et al.*, 2020). The main ambition of intrinsic motivation or curiosity driven leaning, is for the artificial agent to generate autonomously signals that lead to goal and skill discovery. Also in humans, curiosity has repeatedly been recognized as a fundamental building block of behaviour. Influencing our behaviour in both, positive and negative ways, curiosity is a driving force for our mental development in all life stages (Loewenstein, 1994). Some authors would go as far as to consider curiosity essential for the development of autonomous behaviour in humans (White, 1959). Nevertheless, up until today, there has not been a clear agreement on what curiosity exactly means and how it manifest itself in humans or in artificial agents. Authors generally agree that curiosity is a superficial affection - it can arise, diverge and end promptly and is therefore very easily satisfied (Burke, 1958). The definition of curiosity that has motivated the work in this thesis, was introduced by Kagan (Kagan, 1972) in a classic article on motivation. The authors identified four different basic human instincts among those the motive to resolve uncertainty, which was then identified as being synonymous with curiosity.

Thinking about a robotic setting, intrinsic motivation or curiosity should push the robot to detect new scenarios and parts of the environment that it did not experience before. When framing the goal of curiosity in this way, intrinsic motivation is tightly connected to the exploration-exploitation problem in Reinforcement Learning. Consequently the concept of curiosity has been explored within the reinforcement learning literature from various angles. For example, a first attempt towards intrinsically motivated agents consisted in rewarding agents to minimize prediction errors of sensory events (Barto, 2004; Singh *et al.*, 2004, 2010). The in-

trinsic rewards come from a novelty measurement. The agent has a built-in notion of the salience of stimuli. The intrinsic reward for each salient event is proportional to the error in the prediction of that salient event. This initial work was designed for low-dimensional and discrete state-and-action spaces. Alternatively (Forestier and Oudeyer, 2016) define intrinsic reward as a measure of learning progress. The sensorimotor space is subdivided into different sub-areas. The sub-area that expects the highest learning progress is chosen to be acted within. Recently, curiosity as a means to better explore was also investigated for high-dimensional continuous state spaces (Bellemare *et al.*, 2016; Pathak *et al.*, 2017). Most of this work, including recent efforts towards curiosity driven robot learning (Tanneberg *et al.*, 2019; Laversanne-Finot *et al.*, 2018), has defined curiosity as a function of model prediction error and within a model-free reinforcement learning framework. In Model Based Reinforcement Learning, (Shyam *et al.*, 2018) proposed a measure of disagreement as exploration signal. (Levine *et al.*, 2016) propose a maximum entropy exploration behaviour. While (Deisenroth and Rasmussen, 2011; Chua *et al.*, 2018) utilize model uncertainty to generate trajectory distributions, the uncertainty does not play an explicit role in the cost. All of these approaches do not explicitly try to resolve uncertainty in the current model of the dynamics or the environment, which is in contrast to the approach presented in this thesis where the uncertainty of the model is explicitly used to guide the selection of actions.

1.1.2 Model Based Approaches

Humans have impressive generalization capabilities when it comes to manipulating objects and tools. These capabilities are, at least partially, a result of humans having internal models of their bodies (Hoffmann *et al.*, 2010) which enables them to predict the consequences of their actions. Endowing robots with similar capabilities remains an important open research problem.

From a robotics perspective, forward and inverse models are representations of the physical properties of the robot. The forward model represents the causal relationship of the effect of an action, the inverse model represents a mapping from the current state to an action, given a desired goal. In the neuroscience and cognitive science literature (Miall and Wolpert, 1996; Ito, 1970), the presence of internal models, as representation of the body in the human brain (Ishikawa *et al.*, 2016) are believed to play an important role. Forward models are of potential use for more than only representing the causal relationship of a movement in the sensory domain, but also for solving other fundamental cognitive problems: faster adaptation through prediction ((Wolpert *et al.*, 1995; Hoffmann *et al.*, 2010)), anticipation of self caused movements, leading to a sense of agency (Blakemore and Frith, 2003), and enabling mental practice procedures through predictions, by only imagining the sequence of actions without actually performing them (Jackson *et al.*, 2003). All of these scenarios, involve a motor command as one of the inputs to the forward

model. This motor command could be the output of an inverse model or a policy.

Data driven approaches to learn control have been introduced as an alternative to overcome the limitations of imperfect analytical models of the robotic tasks (Bristow *et al.*, 2006). One particular form of data driven approaches for control is Model Based Reinforcement Learning (MBRL). The goal of MBRL is to solve a task through learning a model f of the true dynamics f_{real} of the system that is subsequently used to solve an optimal control problem. The dynamics are described through $x_{t+1} = f(x_t, u_t)$ where x_t and u_t are the state and action of the current time step, and x_{t+1} the state at the next time step. f represents the learned model of the dynamics. MBRL seeks to find a policy $u_t = \pi(x_t)$ that minimizes a cost $\mathcal{J}(x_t, u_t)$ describing the desired behavior. Policy optimization can be performed in various ways such as trajectory sampling approaches as summarized and evaluated in (Chua *et al.*, 2018), random shooting methods, where trajectories are randomly chosen and evaluated with the learned model, or iterative LQG approaches, as in (Levine and Koltun, 2013). Model learning also can be tackled with various methods. When learning dynamics models from data the literature offers a variety of choices of machine learning algorithms used for this learning task. From linear regression (Schaal *et al.*, 2002; Haruno *et al.*, 2001), to gaussian mixture (Khansari-Zadeh and Billard, 2011; Calinon *et al.*, 2010) or gaussian process regression (Deisenroth and Rasmussen, 2011; Kocijan *et al.*, 2004) as well as using feedfoward- or recurrent neural networks for fitting the models (Lenz *et al.*, 2015; Sanchez-Gonzalez *et al.*, 2018; Rueckert *et al.*, 2017). These approaches usually collect a dataset for supervised learning, and fit the models in an end to end fashion from input to output. Some other works however have considered more structured approaches for model learning, including analytical priors during the learning process. For example in (Calandra *et al.*, 2015) only the residual term of the external forces is learned for inverse dynamics model. In (Lutter *et al.*, 2019) the authors learn deep neural networks for each component of the equations of motion of a manipulator whereas in (Ledezma and Haddadin, 2017) the authors learn the dynamics parameters directly via gradient descent. Similar (Sutanto *et al.*, 2020) proposes a fully differentiable version of the recursive newton euler algorithm, allowing the inertial parameters to be learned using gradient descent and automatic-differentiation for gradient computation.

In model based learning for control, the learned model is used to simulate the robot behaviour when optimizing or learning a trajectory or control policy. The learned model and the optimizer are task independent; this independence promises sample efficiency and generalization capabilities, as an already learned model can be reused for new tasks. As a side effect, however, the learned models quality can drastically affect the computed solution, as pointed out in (Schaal, 1997; Atkeson and Santamaria, 1997; Deisenroth, 2010), since the policy is optimized given the current learned model and not by interacting with the robot. This effect is called model bias (Deisenroth, 2010) and can lead to a policy with drastically lower per-

formance on the real robot. The work presented in this thesis shows how model-bias could be alleviated in order to learn better models that generalize to new tasks. Specifically we argue that exploration can encourage visiting states which resolve ambiguities in the learned model and therefore lead to both better models and that by considering the forward model's quality during controller learning a less biased model can be learned.

1.1.3 Learning to Learn

Learning to learn refers to the process of learning over multiple learning processes. Usually in machine learning, the goal is to learn an input-output relationship, for example a function, from data. In the case of learning to learn or meta learning, the goal is to learn the process of learning (Hospedales *et al.*, 2020). In other words, the goal is to learn representations of the underlying learning mechanisms, with the hope of being able to re-use them later for similar but different tasks.

In humans learning to learn is an inherent part of our development. In machine learning, historically (Schmidhuber, 1987; Bengio and Bengio, 1990; Thrun and Pratt, 2012a) have been the first works presenting approaches on implementing learning to learn. (Schmidhuber, 1987) present in their work a framework for self-referential learning where the neural network takes as an input its own weights, and outputs the update for these respective weights. In (Bengio and Bengio, 1990) biologically plausible learning rules, like synaptic updates, are learned and used to update the neural network weights. And (Thrun and Pratt, 2012a) were the first to more clearly formalized the meaning of learning to learn in artificial agents and presented some theoretical and practical justifications.

More recently, as machine learning tools become more and more part of our daily lifes, there has a been a wide interest in finding ways to improve learning speeds and generalization to new tasks through meta-learning. The main directions of the research in this area can be divided into learning representations that can be easily adapted to new tasks (Finn *et al.*, 2017), learning unsupervised rules that can be transferred between tasks (Metz *et al.*, 2019; Hsu *et al.*, 2018), learning optimizer policies that transform policy updates with respect to known loss or reward functions (Maclaurin *et al.*, 2015; Andrychowicz *et al.*, 2016; Li and Malik, 2016; Franceschi *et al.*, 2017; Meier *et al.*, 2018; Duan *et al.*, 2016), or learning loss/reward landscapes (Sung *et al.*, 2017; **?**). The work presented in this thesis falls into the category of learning loss landscapes.

A range of recent works also demonstrate advantages of meta-learning for improving exploration strategies in RL settings, especially in the presence of sparse rewards. Mendonca *et al.* (2019) propose training an agent to mimic expert demonstrations while only having access to a sparse reward signal during test time. In the work of Hausman *et al.* (2018) and Gupta *et al.* (2018), a structured latent exploration space is learned from prior experience, which enables fast exploration

in novel tasks. Zou *et al.* (2019) propose a method for automatically learning potential-based reward shaping by learning the Q-function parameters during the training phase, such that after training the Q-function can adapt quickly to new tasks. In our work, we also demonstrate that we can significantly improve the RL sample efficiency by training our meta-loss to optimize an actor policy, even when providing only limited or no reward information to the learned loss function at test time.

Similar to work by Andrychowicz *et al.* (2016); Duan *et al.* (2016), we aim at learning a loss function that can be applied to various optimizee models. An optimizee is the function that we are trying to optimize. However, our framework does not require a specific recurrent architecture of the optimizer and can operate without an explicit external loss or reward function during test time. Furthermore, as our learned loss functions are independent of the models to be optimized, they can be easily transferred to other optimizee models.

Closest to our method are the works on *evolved policy gradients* (**?**), *teacher networks* (Wu *et al.*, 2018), *meta-critics* (Sung *et al.*, 2017) and *meta-gradient RL* (Xu *et al.*, 2018). In contrast to using an evolutionary approach (e.g. **?**), we design a differentiable framework and describe a way to optimize the loss function with gradient descent in both supervised and reinforcement learning settings. (Wu *et al.*, 2018) propose that instead of learning a differentiable loss function directly, a teacher network is trained to predict parameters of a manually designed loss function, whereas each new loss function class requires a new teacher network design and training. In (Xu *et al.*, 2018), discount and bootstrapping parameters are learned online to optimize a task-specific meta-objective. Our method does not require manual design of the loss function parameterization or choosing particular parameters that have to be optimized, as our loss functions are learned entirely from data. Finally, in work by (Sung *et al.*, 2017) a *meta-critic* is learned to provide a task-conditional value function, used to train an actor policy. Although training a meta-critic in the supervised setting reduces to learning a loss function as in our work, in the reinforcement learning setting we show that it is possible to use learned loss functions to optimize policies directly with gradient descent.

The idea of learning loss landscapes or reward functions in the reinforcement learning setting can be traced back to the field of inverse reinforcement learning (Ng *et al.*, 2000; Abbeel and Ng, 2004, IRL). However, in contrast to the original goal of IRL of inferring reward functions from expert demonstrations, in our work we aim at extending this idea and learning loss functions that can improve learning speeds and generalization for a wider range of applications. Furthermore, we design our framework to be fully differentiable, facilitating the training of both the learned meta-loss and optimizee models. Moreover we also show, how including expert demonstrations during meta-train time facilitiates loss learning for visual object manipulation that outperforms state of the art IRL algorithms.

In the following sections the two main research directions of the work presented

in this thesis are presented in more detail alongside with the main contributions and the key publications.

1.2 Model Based Learning in the Action Perception Loop

The human brain cannot afford the time and the resources to learn every task from scratch, for this reason it builds models that it can reuse (Lake *et al.*, 2017). Also in robotics, models come with the promise of flexible adaptation to new tasks without having to re-learn everything from scratch each time. Sample efficiency and fast generalization are key benefits of model based learning and essential for lifelong learning. But, the learned models can be biased, wrong or old and thus significantly hinder lifelong learning.

1.2.1 Thesis Contributions

Learning better Models

In model-based learning, the policy is learned given the model. When the policy is used on the robot, it affects the data distribution the robot experiences (see Fig. 1.1). The data is then used to learn and update the model. In this scenario a biased model will lead to a sub-optimal policy (and thus sub-optimal data collection) that might enforce the bias. However, even an unbiased model that is only covering some parts of the state space, will not allow a policy to explore unseen parts of the environment (since it does not know about them) and to discover new skills and goals. In (Bechtle *et al.*, 2020a) we show how including the predictive uncertainty of learned probabilistic dynamic models during policy optimization, facilitates exploration by resolving uncertainty in the models, and thus improves task performance, model predictions and generalization. The presented model-based reinforcement learning framework leverages results from stochastic optimal control to include, in a principled way, the predictive uncertainty of the learned model when optimizing a feedback policy. The resulting policy not only tries to optimize a task specific cost, but will also explore uncertainties in the states around the currently optimal trajectory. This leads to improved task performance and model quality. In Chapter 2 the approach and benefits of our work are explained in further detail. We also study model quality and its implications in (Bechtle *et al.*, 2020c), where we present an unbiased loss function for controller and policy learning, that trades off predictive task performance and forward model quality. We present an analysis of how this loss results in a controller that shifts the observed data distribution such that the collected data reduces model bias, improves model quality and, as a consequence, improves task learning. We present empirical and theoretical anal-

ysis. In particular, we also show how this unbiased loss can be successfully used on contact rich tasks on a quadruped dynamic walking task. Making and breaking contact is fundamental for acting in the real world and still a major challenge in robotics. Chapter 3 will explain in further detail this work.

Leveraging Structure for Model Learning

Another promising direction to allow for fast generalization and adaptation of models, is to leverage structured prior analytical knowledge, for example the analytical models of the robot kinematics and dynamics. Combining this structured information with data driven approaches will allow to leverage the generality of the analytical models while staying flexible to changes and unmodeled components. In (Bechtle *et al.*, 2020b) and Chapter 4 we present a self-supervised learning approach, that combines multi-modal sensory information to learn a visual predictive model used for object manipulation tasks. During learning we encode the structured information of the robot kinematics as a strong physical inductive bias to guide the visual detection of the manipulated object. We show how combining high dimensional sensory input, like images, with low dimensional structured proprioceptive information, allows for model learning that generalizes robustly to new and out of distribution tasks. In Chapter 3, we also show how combining structure and data driven learning accelerates learning of the robot- and contact-dynamics. This allows for self-supervised model-based policy learning of complex tasks, like walking, purely from observed data.

The work presented in this thesis on model based learning, investigates the question of how to learn better models from different perspectives. In (Bechtle *et al.*, 2020a) we show how using exploration allows for better model learning. The key distinction of our work is that we include the exploration signal already during policy optimization, considering the uncertainty of the model to when optimizing a policy. This is in contrast to existing work on exploration in Reinforcement Learning, where the exploration usually happens in the acting phase by adding noise to the policy output (Levine and Koltun, 2013; Deisenroth and Rasmussen, 2011). In general exploration in model based learning has not been studies extensively in the literature so far, even if learning a better model benefits the overall learning task. We also investigated how to formulate an objective function where the policy is incentivized to accomplish a task but also to learn a good forward model. Existing work (Jordan and Rumelhart, 1992) did not consider an objective function that trades off task and model learning, and thus struggles with generalizing the task learning to higher dimensional systems. Adding structure to the learning problem is a different avenue taken in the work presented in this thesis to learn better models that generalize beyond the training task. In (Bechtle *et al.*, 2020b) we show that fusing vision and structured proprioceptive information allows for better model learning. The learned models generalize well beyond training data,

this is in contrast to learning approaches that do not leverage the structure, that have difficulties generalizing to new tasks as we show in (Bechtle *et al.*, 2020b) and Chapter 4.

1.3 Learning to Learn in the Action Perception Loop

Humans have the remarkable capability to continuously learn and adapt to new tasks, in other words: we are able to learn how to learn. To endow robots with the capability of learning to learn, will allow them to make inference that goes beyond the training data and thus enable fast learning and adaptation of skills.

When humans learn a new skill, trial and error plays an important role. Each trial gives us more information about the task and we are able to learn from the collected experiences through interactions.

1.3.1 Thesis Contributions

Inspired by this, we proposed on a fully differentiable learning to learn framework (Bechtle *et al.*, 2019) presented in Chapter 5. This framework enables robots to learn how to learn. In the learning to learn phase a loss function for a task is learned from experience: The robot tries to accomplish a task, collects data-points by interacting with the environment, and thereby learns a loss function to increase learning progress. Later, after the robot learned to learn, the loss can be used directly on new tasks.

Another way for humans to learn how to learn, is by having a teacher, an example or some other form of extra information, like a manual. The learning to learn framework of (Bechtle *et al.*, 2019) can be extended to include this notion of learning to learn, namely learning from expert knowledge. In (Bechtle *et al.*, 2019) we show how this extra information can be used to encode exploratory behaviour in the learned policies, or how analytical structure can be used to shape the learned loss. In (Das *et al.*, 2020a) we show how this learning to learn principle can also be used on tasks where the robot learned to learn from human demonstrations. The human demonstrates how to accomplish a task. The robot observes the demonstration through its camera and learns how to learn from them. In (Davchev *et al.*, 2021), we show how our learning to learn framework can encode different speeds of demonstrations. A time invariant loss is learned from misaligned demonstrations and can be used for task execution at different speeds. This allows for example for slower manipulation of fragile objects and faster manipulation of sturdy ones - another axis of generalisation that is crucial for lifelong learning.

Our work distinguished itself from the related learning to learn literature presented in section 1.1 in two key aspects. Firstly, in our framework, we learn a loss

function. Learning a loss function is fundamentally different from learning a representation of the policy parameters as in (Finn *et al.*, 2017) or a gradient update rule as in (Meier *et al.*, 2018). It is different since a loss is a more general representation of the learning task, that has potential to generalize to new, also out of distribution, tasks. The same loss function, for example the mean squared error loss, is widely used in machine learning for fitting data that follows very different distributions. Thus, the key idea behind this framework is to learn a representation of the learning mechanisms that can be reused in a lifelong learning setting. Secondly, the way we learn the loss function is different from what has been presented so far. In (Houthooft *et al.*, 2018)a loss function is also learned, but the optimal parameters of the loss are found with evolutionary learning strategies. This takes a lot of time and is very sample inefficient. In contrast to this, our framework is fully differentiable and can be learned completely with gradient-based methods. How the loss function is learned, distinguishes itself from the inverse reinforcement learning literature (Ng *et al.*, 2000; Abbeel and Ng, 2004): Besides of not needing demonstrations, and thus being able to learn in a self-supervised way, our framework given its differentiability, learns a loss that is optimized to increase learning progress in the optimizee. It can evaluate the learning progress of the optimizee and thus it can evaluate how well the current loss was at optimizing the optimizee.

Part I

Model Based Learning in the Action Perception Loop

Chapter 2

Curious iLQR: Resolving Uncertainty in Model-based RL

2.1 Introduction

Model-based reinforcement learning holds promise for sample-efficient learning on real robots (Atkeson and Santamaria, 1997). The hope is that a model learned on a set of tasks can be used to learn to achieve new tasks faster. A challenge is then to ensure that the learned model generalizes beyond the specific tasks used to learn it. We believe that curiosity, as means of exploration, can help with this challenge. Though curiosity has been defined in various ways, it is generally considered a fundamental building block of human behaviour (Loewenstein, 1994) and essential for the development of autonomous behaviour (White, 1959).

In this work, we take inspiration from (Kagan, 1972), which defines curiosity as motivation to resolve uncertainty in the environment. Following this definition, we postulate that by seeking out uncertainties, a robot is able to learn a model faster and therefore achieve lower costs more quickly compared to a non-curious robot. Keeping real robot experiments in mind, our goal is to develop a model-based reinforcement learning (MBRL) algorithm that optimizes action sequences to not only minimize a task cost but also to reduce model uncertainty.

Specifically, our MBRL algorithm iterates between learning a probabilistic model of the robot dynamics and using that model to optimize local control policies (i.e. desired joint trajectories and feedback gains) via a *curious* version of the iterative Linear Quadratic Regulator (iLQR) (Tassa *et al.*, 2014). These policies are executed on the robot to gather new data to improve the dynamics model, closing the loop, as summarized in Figure 2.1.

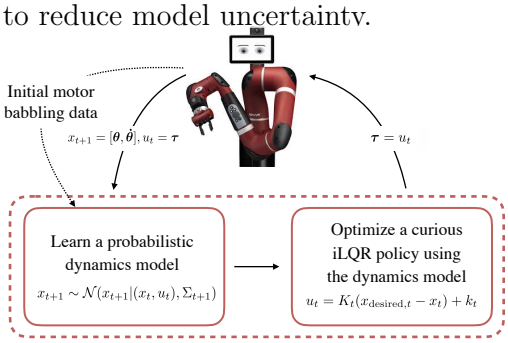


Figure 2.1: Approach overview: motor babbling data initializes the dynamic model, the main loop then alternates between model learning and policy updates.

In a nutshell, our curious iLQR aims
at optimizing local policies that minimize the cost *and* explore parts of the model with high uncertainty. In order to encourage actions that explore states for which the dynamics model is uncertain, we incorporate the variance of the model predictions into the cost function evaluation. We propose a computationally efficient approach to incorporate this uncertainty by leveraging results on risk-sensitive optimal control (Jacobson, 1973; Farshidian and Buchli, 2015). (Jacobson, 1973) showed that optimizing actions with respect to the expected exponentiated cost directly takes into account higher order moments of the cost distribution while affording the explicit computation of the optimal control through Riccati equations. A risk-sensitive version of iLQR was recently proposed in (Farshidian and Buchli, 2015). While in these approaches the dynamic model is typically considered known and uncertainty comes from external disturbances, we propose to instead explicitly incorporate model uncertainty in the algorithm to favor the exploration of uncertain parts of the model. The proposed coupling between model learning and risk-sensitive control explicitly favours actions that resolve the uncertainty in the model while minimizing a task-related cost.

The contributions of this work are as follows: 1) We present a MBRL algorithm that learns a global probabilistic model of the dynamics of the robot from data and show how to utilize the uncertainty of the model for exploration through our curious iLQR optimization. 2) We demonstrate that our MBRL algorithm can scale to seven degree of freedom (DoF) manipulation platform in the real world without requiring demonstrations to initialize the MBRL loop. 3) The results show that using curiosity not only learns a better model faster on the initial task, but also that this model generalizes to new tasks more reliably. We perform an extensive evaluation in both simulation and on hardware.

2.2 Background

The goal of MBRL is to solve a task through learning a model f of the true dynamics f_{real} of the system that is subsequently used to solve an optimal control problem. The dynamics are described through $x_{t+1} = f(x_t, u_t)$ where x_t and u_t are the state and action of the current time step, and x_{t+1} the state at the next time step. f represents the learned model of the dynamics. MBRL seeks to find a policy $u_t = \pi(x_t)$ that minimizes a cost $\mathcal{J}(x_t, u_t)$ describing the desired behavior. Policy optimization can be performed in various ways such as trajectory sampling approaches as summarized and evaluated in (Chua *et al.*, 2018), random shooting methods, where trajectories are randomly chosen and evaluated with the learned model, or iterative LQG approaches, as in (Levine and Koltun, 2013). Model learning also can be tackled with various methods. (Levine and Abbeel, 2014) proposes learning linear models of the forward dynamics. In (Chua *et al.*, 2018)

the dynamics are learned with an ensemble of neural networks. In general, the learned model of dynamics can be deterministic as in (Levine and Abbeel, 2014) or probabilistic as in (Deisenroth and Rasmussen, 2011; Chua *et al.*, 2018).

In MBRL, the learned model is used to simulate the robot behaviour when optimizing a trajectory or control policy. The learned model and the optimizer are task independent; this independence promises sample efficiency and generalization capabilities, as an already learned model can be reused for new tasks. As a side effect, however, the learned models quality can drastically affect the computed solution, as pointed out in (Schaal, 1997; Atkeson and Santamaria, 1997; Deisenroth, 2010), since the policy is optimized given the current learned model and not by interacting with the robot. This effect is called model bias (Deisenroth, 2010) and can lead to a policy with drastically lower performance on the real robot. We argue that exploration can alleviate this model-bias. Resolving model uncertainty while optimizing for a task can encourage visiting states which resolve ambiguities in the learned model and therefore lead to both better models and control policies.

2.2.1 Intrinsic motivation for RL

The concept of curiosity has also been explored within the reinforcement learning literature from various angles. For example, a first attempt towards intrinsically motivated agents consisted in rewarding agents to minimize prediction errors of sensory events (Barto, 2004; Singh *et al.*, 2004, 2010). This initial work was designed for low-dimensional and discrete state-and-action spaces. Recently, curiosity as a means to better explore was also investigated for high-dimensional continuous state spaces (Bellemare *et al.*, 2016; Pathak *et al.*, 2017). Most of this work, including recent efforts towards curiosity driven robot learning (Tanneberg *et al.*, 2019; Laversanne-Finot *et al.*, 2018), has defined curiosity as a function of model prediction error and within a model-free reinforcement learning framework. In MBRL, (Shyam *et al.*, 2018) recently proposed a measure of disagreement as exploration signal. (Levine *et al.*, 2016) propose a maximum entropy exploration behaviour. Other algorithms which take uncertainty into account have been presented as well (Deisenroth and Rasmussen, 2011; Williams *et al.*, 2017; Chua *et al.*, 2018; Boedecker *et al.*, 2014). They differ in their choice of policy optimization, dynamics model representation and how they incorporate uncertainty. While (Deisenroth and Rasmussen, 2011; Chua *et al.*, 2018) utilize model uncertainty to generate trajectory distributions, the uncertainty does not play an explicit role in the cost. Thus, these approaches do not explicitly optimize actions that resolve uncertainty in the current model of the dynamics, which is in contrast to the approach we propose in this paper.

2.2.2 Risk Sensitive stochastic optimal control

Risk-sensitive optimal control has a long history (Jacobson, 1973; Whittle, 1981). The central idea is to not only minimize the expectation of the performance objective under the stochastic dynamics but to also take into account higher-order moments of the cost distribution. The objective function takes the form of an exponential transformation of the performance criteria $J = \min_\pi \mathbb{E}\{\exp[\sigma \mathcal{J}(\pi)]\}$ (Jacobson, 1973). Here, $\mathcal{J}(\pi)$ is the performance index, which is a random variable, and a functional of the policy π. $\mathbb{E}$ is the expected value of $\mathcal{J}$ over stochastic trajectories induced by the policy π. $\sigma \in \mathbb{R}$ accounts for the sensitivity of the cost to higher order moments (variance, skewness, etc.). Notably, from (Farshidian and Buchli, 2015), the cost is $\frac{1}{\sigma}\log(J) = \mathbb{E}(\mathcal{J}^*) + \frac{\sigma}{2}\mathrm{var}(\mathcal{J}^*) + \frac{\sigma^2}{6}\mathrm{sk}(\mathcal{J}^*) + \cdots$, where var and sk stand for variance and skewness and $\mathcal{J}^*$ is the optimal task cost. When $\sigma > 0$ the optimal control will be risk-averse, favoring low costs with low variance but when $\sigma < 0$ the optimal control will be risk-seeking, favoring low costs with high variance. $\sigma = 0$, reduces to the standard, risk-neutral, optimal control problem. Jacobson (Jacobson, 1973) originally demonstrated that for linear dynamics and quadratic costs the optimal control could be computed as the solution of a Riccati equation. Leveraging this result, (Farshidian and Buchli, 2015) recently proposed a risk-sensitive extension of iLQR and (Ponton *et al.*, 2016) further extended the approach to explicitly incorporate measurement noise.

2.3 MBRL via Curious iLQR

We present our approach to incorporate curious behaviour into a robot's learning control loop. We are interested in minimizing a performance objective $\mathcal{J}$ to achieve a desired robot behavior and approximate the true dynamics of the system with a discrete-time dynamical system

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)\,\Delta t \tag{2.1}$$

where $\mathbf{x}_t$ denotes the state of the system at time step t and $\mathbf{f}$ represents the unknown model of the dynamics of the system and needs to be learned to achieve the desired task. The hypothesis we seek to confirm is that, by trying to explore uncertain parts of the model, our MBRL algorithm can learn a good dynamics model more quickly and find behaviors with higher performance. Our algorithm learns a probabilistic model of the system dynamics while concurrently optimizing a desired cost objective (Figure 2.1). It combines i) a risk-seeking iLQR algorithm and ii) a probabilistic model of the dynamics. We describe the algorithm in the following. In particular, we show how to incorporate model uncertainty in risk-sensitive optimal control. Algorithm 1 shows the complete algorithm.

Algorithm 1 MBRL Algorithm

$\mathcal{D} \leftarrow$ motor babbling data
train model $f$on D
while $i <$ iter **do**
  $\pi \leftarrow$ optimize policy via Alg 3
  $D_{\text{new}} \leftarrow$ rollout π on system
  $\mathcal{D} = \mathcal{D} \cup D_{\text{new}}$
  train model $f$on $\mathcal{D}$
end while

Algorithm 2 simulate-policy(x, τ, k, K, α)

$x_0^{\text{new}} \leftarrow x_0$
while $t < T$ **do**
  $\tau_t^{\text{new}} \leftarrow \tau_t + \alpha k_t + K_t(x_t - x_t^{\text{new}})$
  $x_{t+1}^{\text{new}} \leftarrow f(x_t^{\text{new}}, \tau_t)$
end while
return $\tau^{\text{new}}, x^{\text{new}}$

Algorithm 3 curious-iLQR

$\tau \leftarrow$ Initial random torque trajectory
$x^* \leftarrow$ unroll τ using f
$\mathbb{A} \leftarrow$ Line search parameters, $[0,\ldots,1]$
$J^* \leftarrow$ Optimal iLQR cost so far
while $i <$ opt iter **do**
  $k, K \leftarrow$ backward pass, see 2.3.2
  for $\alpha \in \mathbb{A}$ **do**
    $\tau^{new}, x^{new} \leftarrow$ simulate-policy(x, τ, k, K, α)
    $J_{new} \leftarrow$ Compute cost of $\tau^{\text{new}}, x^{\text{new}}$
    if $J_{new} < J^*$ **then**
      $\tau, x^* \leftarrow x^{\text{new}}, \tau^{new}$
    end if
    if converged **then**
      return $\pi(x) = \tau + K(x - x^*)$
    end if
  end for
end while

2.3.1 Risk-sensitive iLQR

Consider the following general nonlinear stochastic difference equation

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)\,\Delta t + \mathbf{g}(\mathbf{x}_t, \mathbf{u}_t)\,\Delta\omega \tag{2.2}$$

where $\mathbf{g}$ maps a Brownian motion $\Delta\omega$, with 0 mean and covariance $(\mathbf{\Sigma} \cdot \Delta t)$, to system states. $\Delta\omega$ and the nonlinear map $\mathbf{g}$, typically model an unknown physical disturbance, while assuming a known model $\mathbf{f}$ of the dynamics. When considering the exponentiated performance criteria $J = \min_\pi \mathbb{E}\{\exp[\sigma \mathcal{J}(\pi)]\}$ (see 2.2 for more details), it has been shown that iLQR (Tassa *et al.*, 2014) can be extended to risk-sensitive stochastic nonlinear optimal control problems (Farshidian and Buchli, 2015). The algorithm begins with a nominal state and control input trajectory $\mathbf{x^n}$ and $\mathbf{u^n}$. The dynamics and cost are approximated to first and second order respectively along the nominal trajectories $\mathbf{u_t^n}$, $\mathbf{x_t^n}$ in terms of state and control deviations $\delta\mathbf{x_t} = \mathbf{x_t} - \mathbf{x_t^n}$, $\delta\mathbf{u_t} = \mathbf{u_t} - \mathbf{u_t^n}$. Given a quadratic control cost, the locally optimal control law will be of the form $\delta\mathbf{u_t} = \mathbf{k_t} + \mathbf{K_t}\delta\mathbf{x_t}$. The underlying optimal control problem can be solved by using Bellman equation

$$\Psi_\sigma(\delta\mathbf{x_t}, \mathbf{t}) = \min_{\boldsymbol{u}}\{l(\mathbf{x}, \mathbf{u}, \mathbf{t}) + \mathbb{E}[\Psi_\sigma(\delta\mathbf{x_{t+1}}, \mathbf{t+1})]\} \tag{2.3}$$

where l is the quadratic cost, and by making the following quadratic approximation of the value function $\Psi(\delta\mathbf{x_t}, \mathbf{t}) = \frac{1}{2}\delta\mathbf{x_t^T}\mathbf{S_t}\delta\mathbf{x} + \delta\mathbf{x_t^T}\mathbf{s_t} + s_t$ where $\mathbf{S_t} = \nabla_{\delta x \delta x}\Psi$ and $\mathbf{s_t} = \nabla_{\delta x}\Psi - \mathbf{S_t}\delta\mathbf{x_t}$ are functions of the partial derivatives of the value function.

Using the (time-varying) linear dynamics, the quadratic cost and the quadratic approximation of Ψ, and solving for the optimal control, we get

$$\delta\mathbf{u_t} = \mathbf{k_t} + \mathbf{K_t}\delta\mathbf{x_t}, \quad \mathbf{k_t} = -\mathbf{H_t^{-1}}\mathbf{g_t}, \quad \text{and} \quad \mathbf{K_t} = -\mathbf{H_t^{-1}}\mathbf{G_t} \tag{2.4}$$

where $\mathbf{H_t}$, $\mathbf{g_t}$, $\mathbf{G_t}$ are given by

$$\begin{aligned}
\mathbf{H_t} &= \mathbf{R_t} + \mathbf{B_t^T}\mathbf{S_{t+1}}\mathbf{B_t} + \sigma\mathbf{B_t^T}\mathbf{S_{t+1}^T}\mathbf{C}\mathbf{W_{t+1}^{-1}}\mathbf{C^T}\mathbf{S_{t+1}}\mathbf{B_t} \\
\mathbf{g_t} &= \mathbf{r_t} + \mathbf{B_t^T}\mathbf{s_{t+1}} + \sigma\mathbf{B_t^T}\mathbf{S_{t+1}^T}\mathbf{C}\mathbf{W_{t+1}^{-1}}\mathbf{C^T}\mathbf{s_{t+1}} \\
\mathbf{G_t} &= \mathbf{P_t^T} + \mathbf{B_t^T}\mathbf{S_{t+1}}\mathbf{A_t} + \sigma\mathbf{B_t^T}\mathbf{S_{t+1}^T}\mathbf{C}\mathbf{W_{t+1}^{-1}}\mathbf{C^T}\mathbf{S_{t+1}}\mathbf{A_t}
\end{aligned} \tag{2.5}$$

where $\mathbf{W_{t+1}} = \mathbf{\Sigma_t^{-1}} - \sigma\mathbf{C_t^T}\mathbf{S_{t+1}}\mathbf{C_t}$ represents how the uncertainty is propagated through the trajectory, $\mathbf{A_t} = \Delta t\frac{\partial \mathbf{f}}{\partial \mathbf{x_t}}$, $\mathbf{B_t} = \Delta t\frac{\partial \mathbf{f}}{\partial \mathbf{u_t}}$ and $\mathbf{q_t}$, $\mathbf{r_t}$, $\mathbf{Q_t}$, $\mathbf{R_t}$ and $\mathbf{P_t}$ are the coefficients of the Taylor expansion of the cost function around the nominal trajectory. The corresponding backward recursions are

$$\mathbf{s_t} = \mathbf{q_t} + \mathbf{A_t^T}\mathbf{s_{t+1}} + \mathbf{G_t^T}\mathbf{k_t} + \mathbf{K_t^T}\mathbf{H_t}\mathbf{k_t} + \sigma\mathbf{A_t^T}\mathbf{S_{t+1}^T}\mathbf{C}\mathbf{W_{t+1}^{-1}}\mathbf{C^T}\mathbf{s_{t+1}} \tag{2.6}$$

$$\mathbf{S_t} = \mathbf{Q_t} + \mathbf{A_t^T}\mathbf{S_{t+1}}\mathbf{A_t} + \mathbf{K_t^T}\mathbf{H_t}\mathbf{K_t} + \mathbf{G_t^T}\mathbf{K_t} + \mathbf{K_t^T}\mathbf{G_t} + \sigma\mathbf{A_t^T}\mathbf{S_{t+1}^T}\mathbf{C}\mathbf{W_{t+1}^{-1}}\mathbf{C^T}\mathbf{S_{t+1}}\mathbf{A_t} \tag{2.7}$$

We note that this Riccati recursion is different from usual iLQR (Tassa *et al.*, 2014)) due to the presence of the covariance $\mathbf{\Sigma}$: the locally optimal control law explicitly depends on the noise uncertainty. The derivation of the algorithm is presented in detail in Appendix A.1.

2.3.2 Curious iLQR: seeking out uncertainties

We use Gaussian Process (GP) regression to learn a probabilistic model of the dynamics in order to include the predictive variance from the model into the risk-sensitive iLQR algorithm. This predictive variance will then capture both model as well as measurement uncertainty. Specifically, we set $\mathbf{x_t} = [\boldsymbol{\theta_t}, \dot{\boldsymbol{\theta}}_\mathbf{t}]$ where $\boldsymbol{\theta_t}$, $\dot{\boldsymbol{\theta}}_\mathbf{t}$ are joint position and velocity vectors respectively. We let $\mathbf{u_t}$ denote the vector of commanded torques. After each system rollout, we get a new set of tuples of states and actions $(\mathbf{x_t}, \mathbf{u_t})$ as inputs and $\boldsymbol{\theta_{t+1}}$, joint accelerations at the next time step, as outputs which we add to our dataset $\mathcal{D}$ on which we re-train the probabilistic dynamics model (see Algorithm 1). Once trained, the model produces a one step prediction of the joint accelerations of the robot as a probability distribution of the form

$$p(\boldsymbol{\theta_{t+1}}|\mathbf{x_t}, \mathbf{u_t}) = \mathcal{N}(\boldsymbol{\theta_{t+1}}|\mathbf{h}\left(\mathbf{x_t}, \mathbf{u_t}\right)\Delta t, \boldsymbol{\Sigma_{t+1}}) \tag{2.8}$$

where $\mathbf{h}$ is the mean vector and $\boldsymbol{\Sigma_{t+1}}$ the covariance matrix of the predictive distribution evaluated at $(\mathbf{x_t}, \mathbf{u_t})$. The outputs is the acceleration at the next time step $\boldsymbol{\theta_{t+1}}$ which is numerically integrated to velocity $\boldsymbol{\theta_{t+1}}\Delta t + \boldsymbol{\theta_t} = \boldsymbol{\theta_{t+1}}$ and position $\boldsymbol{\theta_{t+1}}\Delta t + \boldsymbol{\theta_t} = \boldsymbol{\theta_{t+1}}$. This results in a Gaussian predictive distribution of the system dynamics $\mathbf{f}$

$$\mathbf{x_{t+1}} \sim \mathcal{N}(\mathbf{x_{t+1}}|\mathbf{x_t} + \mathbf{h}\left(\mathbf{x_t}, \mathbf{u_t}\right)\Delta t, \boldsymbol{\Sigma_{t+1}}) \tag{2.9}$$

It is the covariance matrix Σ_{t+1} of this distribution that is incorporated into the Riccati equations from above. Specifically, during each MBRL iteration we optimize a new local feedback policy under the current dynamics model $\mathbf{f}$, via Algorithm 3. Each outer loop of the optimization, re-linearizes $\mathbf{f}$ with respect to the current nominal trajectories $\mathbf{u_t^n}$, $\mathbf{x_t^n}$ in the backward-pass:

$$\delta\mathbf{x_{t+1}} = \mathbf{A_t}\delta\mathbf{x_t} + \mathbf{B_t}\delta\mathbf{u_t} + \mathbf{C_t}\omega_t \tag{2.10}$$

with $\mathbf{A_t} = \Delta t\frac{\partial \mathbf{f}}{\partial \mathbf{x_t^n}}$, $\mathbf{B_t} = \Delta t\frac{\partial \mathbf{f}}{\partial \mathbf{u_t^n}}$ and $\omega_t \sim \mathcal{N}(\omega_t|0, \boldsymbol{\Sigma_{t+1}})$, where $\mathbf{A_t}$ and $\mathbf{B_t}$ are the analytical gradients of the probabilistic model prediction at each time step and $\mathbf{C_t}$ weights how the uncertainty is propagated trough the system. We utilize the Riccati equations from Section 2.3.1, Equations ((2.5)) and ((2.6)), to optimize a new local feedback policy that utilizes the models predictive covariance Σ_{t+1}. During the shooting phase of the algorithm, we integrate the nonlinear model from the GP and, to guarantee convergence to lower costs, we use a line search approach during the optimization. We leverage the risk-seeking capabilities of the optimization by setting $\sigma < 0$. The algorithm then favors costs with higher variance which is related to exploring regions of the state space with higher uncertainty in the dynamics. As a result, the agent is encouraged to select actions that explore uncertain regions of the dynamic model while still trying to reduce the task specific error. With $\sigma = 0$ the agent will ignore any uncertainty in the environment and therefore not

explore. This is equivalent to standard iLQR optimization which ignores higher order statistics of the cost function. An overview of *curious iLQR* is given in Algorithm 3.

2.4 Illustration: Curious iLQR

In this section, we want to illustrate the advantages of using the motivation to resolve model uncertainty as an exploration tool. The objectives of this section is to give an intuitive example of the effect of our MBRL loop. In the following, and throughout the paper, we will refer to the agent that tries to resolve the uncertainty in its environment as curious and the one that is not following the uncertainty but only optimizes for the task related cost as normal.

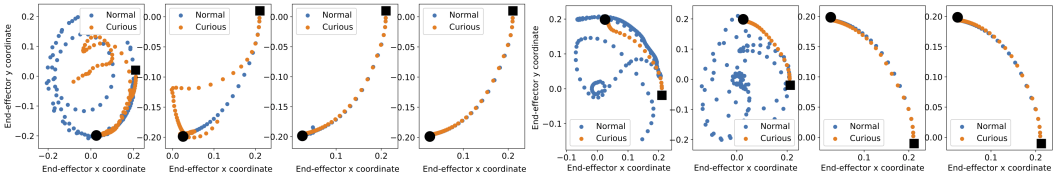


Figure 2.2: End-effector position of curious and normal agent for 4 learning iterations on 2 different targets. The targets are represented by the black dots, the starting position by the black squares.

The experimental platform is the OpenAI Gym Reacher environment (Brockman *et al.*, 2016), a two degrees of freedom arm attached at the center of the scene. The goal of the task is to reach a target placed in the environment. In the experiments presented here, actions were optimized as described in section 2.3. The probabilistic model was learned with Gaussian Process (GP) regression using the GPy library (GPy, 2012). The intuition behind this experiment is that, if an agent is driven to resolve uncertainty in its model, a better model of the system dynamics can be learned and therefore used to optimize a control sequence more reliably. Our hypothesis is that, the model learned by the curious agent is better by the end of learning and therefore we expect it to perform better when using it to solve new reaching tasks. In Figure 2.2 we show the resulting end-effector trajectories of 8 consecutive MBRL iterations when optimizing to reach 2 different targets in sequence. We compare the behavior of the curious and normal agent in orange and blue, respectively. The targets are represented by the black dot. The curious agent tries to resolve the uncertainty within the model; the normal agent optimizes only for the task related

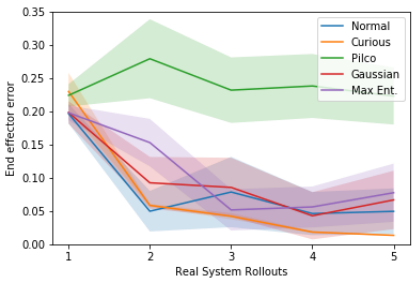


Figure 2.3: Reacher performance /10 trials.

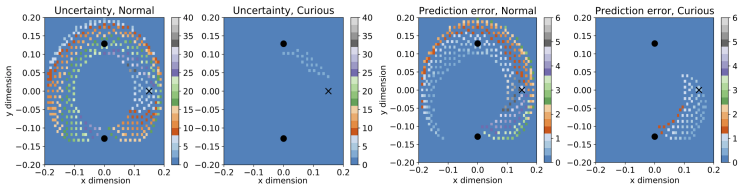


Figure 2.4: The uncertainty and the prediction error in end-effector space after training, for the normal and curious agent. The cross is the initial position. Regions that are not by the arm reachable are shown in blue.

cost. The normal agent seemingly reaches the first target after the second learning iteration; the curious agent only manages to reach the target during the third iteration. Interestingly, the exploration of the curious agent leverages the arm to reach the second target immediately and continues to reach it consistently thereafter. Figure 2.4 confirms the intuition that the curious agent has learned a better model than the normal agent. The figure shows the uncertainty and the prediction error (in end-effector space) of the model learned by the normal and the curious agent respectively. With curiosity, the learned model has overall lower uncertainty and prediction error values over the whole state space. We also compare our MBRL loop via *curious iLQR* optimization to: normal iLQR, a random exploration controller that adds Gaussian noise to the actions with mean 0 and variance 0.2, a maximum entropy exploration behaviour following the approach proposed in (Levine *et al.*, 2016) and PILCO (Deisenroth and Rasmussen, 2011), in Figure 2.3. For these experiments, we initialize the model with only two data points collected randomly during motor babbling. We report the mean and the standard deviation across 10 trials, where each trial starts from a different initial joint configuration and is initialized with a different initial torque trajectory for optimization. In this scenario, with a very poor initial model quality, PILCO could not perform comparably to our MBRL loop. MBRL via *curious iLQR* outperforms all the other approaches. Furthermore it converges to solutions more reliably, as the variance between trials is lowest.

2.5 Experiments on high-dimensional problems

Finally, the goal of this work is to learn motor skills on a torque-controlled manipulator. Our experimental platform is the Sawyer robot from Rethink Robotics (Sawyer, 2012), a 7 degrees of freedom manipulator. We start with experiments performed in the PyBullet physics simulator (Pybullet, 2012). In the next Section, we present results on the Sawyer robot arm hardware. Previous work such as (Farshidian and Buchli, 2015) and (Ponton *et al.*, 2016), which use risk-sensitive control variations of iLQR, primarily deal with simplified, low dimensional problems. Our experiments are conducted on a 7 degree of freedom robot, and the higher dimensional system adds some complexities to the approach: the gradients

in Section 2.3.1 of the value function (Equations ((2.6)), ((2.7))) tend to suffer from numerical ill-conditioning in high-dimensions. We account for this issue with Tikhonov regularization: before inversion for calculating the optimal control we add a diagonal matrix to $\mathbf{H_k}$ from Equation ((2.5)). The regularization parameter and the line search parameter α are adapted following the Levenberg Marquardt heuristic (Tassa *et al.*, 2014).

The goal of these experiments is to reach a desired target joint configuration θ. We show results for dynamics learned with GP regression (GPR), as well as initial results on ensemble of probabilistic neural networks (EPNN) following the approach presented in (Lakshminarayanan *et al.*, 2017). When using GPs, a separate GP is trained for each output dimension.

We perform two sets of experiments, both in simulation and on hardware, to analyze the effect of using curiosity. Specifically, we believe that curiosity helps to find better solutions faster, because it guides exploration within the MBRL loop. Intuitively, curiosity helps to observe more diverse data samples during each rollout such that the model learns more about the dynamics.

We start with evaluation in simulation. Throughout all of the simulation experiments the optimization horizon was 150 time steps long at a sampling rate of 240 Hz. Motor babbling was performed at the beginning for $0.5s$ by commanding random torques in each joint.

2.5.1 Reaching task from scratch

During the first set of experiments, we compare the performance when learning to reach a given target configuration from scratch. We compare our MBRL loop, as before, when using our *curious iLQR* optimization, regular iLQR, a random exploration controller and a maximum entropy exploration behaviour as described previously. PILCO was not able to learn the reaching movement on the 7-DoF manipulator, so we exclude the results from the analysis. We perform this experiments for each kind of controller 5 times. Each run slightly perturbs the initial joint positions, and uses a random initial torque trajectories for the optimizer. For a given target joint configuration, 5 iterations of optimizing the trajectory, running the trajectory on the system and updating the dynamics model, were performed. We perform this experiment for 3 different target joint configurations. The following results are averaged across the $5x3$ runs (5 runs per target). The left most plot in Figure 2.5 compares performance of *curious iLQR* when using EPNN vs GPR for dynamics model learning, with and without curiosity. Our analysis shows that MBRL via *curious iLQR* improves performance over regular iLQR, for both model architectures. While the EPNN is more promising in scaling our approach, it currently requires more data to train. For this reason we will focus on the GP model for the remainder of our experimental section. In the 2nd to 4th plot of Figure 2.5, we compare the performance of *curious iLQR* against the above mentioned baselines

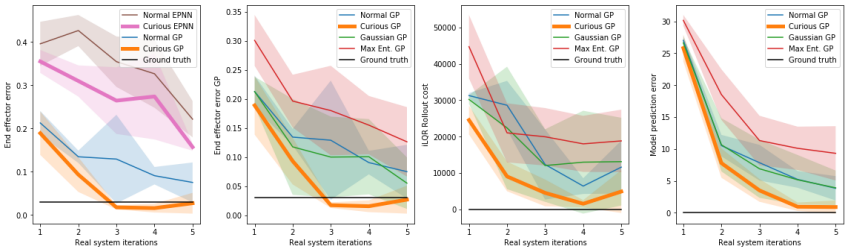


Figure 2.5: Distance in end effector space for EPNN vs. GP in m (1). Distance in end effector space in m (2), iLQR rollout cost (3) and model prediction error (4) with the GP model, compared to our baselines.

for exploration during policy optimization, when using GPR for model learning. We compare the methods with respect to 3 metrics: final Euclidean end-effector distance (plot 2), iLQR cost (plot 3) and the predictive performance of the model on each rollout (plot 4). We can consistently see that, on average, MBRL via *curious iLQR* outperforms the other approaches: the error/cost is smaller and the solutions are more consistent across trials as the standard deviation is lower. This shows that curiosity can lead to faster learning of a new task, when learning from scratch. The results on the predictive performance of the model suggest that the quality of the model learned via *curious iLQR* might be better in terms of generalization. In the next section we present results that investigate this assumption.

2.5.2 Optimizing towards new targets after model learning

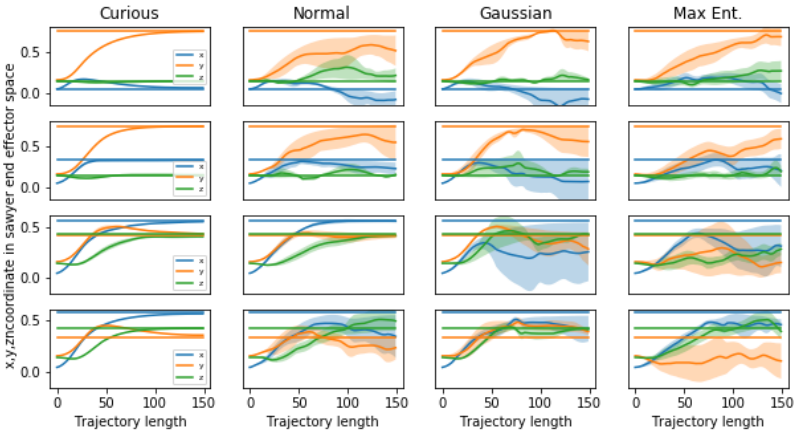


Figure 2.6: Optimizing to reach new targets with regular iLQR after models were learned. 4 different targets (one per row) are evaluated and the final end-effector trajectories presented. Constant lines are targets for x/y/z.

To confirm the hypothesis that the models learned by MBRL with curious iLQR generalize better, because they have explored the state space better, we decided to evaluate the learned dynamics models on a second set of experiments in which the robot tries to reach new, unseen targets. In this experiment we take the GP models

learned during experiment 1 in Section 2.5.1 and use them to optimize trajectories to reach new targets that were not seen during training of the model. The results are shown in Figure 2.6, where four randomly chosen targets were set and the trajectory was optimized with regular iLQR. Note, that here we use regular iLQR to optimize for the trajectory so that we can better compare the models learned with/without curiosity in the previous set of experiments. Figure 2.6 shows the trajectory in end effector space for each coordinate dimension, together with the target end effector position as a solid horizontal line. The results are averaged across 5 trials. The trials correspond to using one of the 5 dynamics models at the end of Experiment 1 in Section 2.5.1. For each trial, the initial torque trajectory was initialized randomly, and the initial joint configuration slightly perturbed. The mean and the standard deviation of the optimized trajectories are computed across the 5 models learned via MBRL with curious iLQR (first col), MBRL with normal iLQR (second col), iLQR with random exploration (third col) and iLQR with maximum entropy exploration bonus (fourth col.). We see that MBRL with curious iLQR results in a model that performs better when presented with a new target. The new targets are reached more reliably and precisely.

2.6 Real hardware experiments

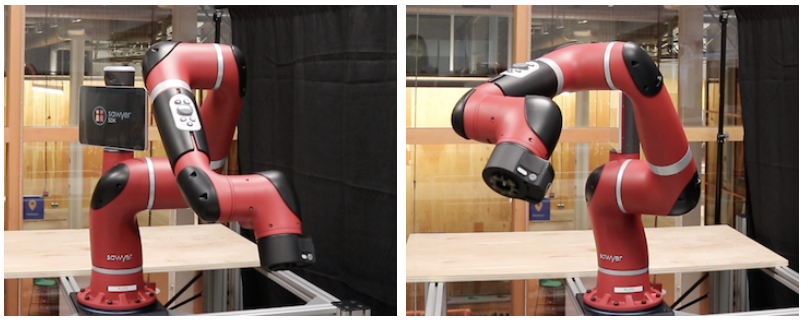
Target	**Distance to Target in m (Learning Iteration)**							
	Curious				**Normal**			
1	0.05 (6)	0.09 (2)	0.09 (3)	**0.07 (3.67)**	0.37 (8)	0.08 (2)	0.18 (8)	**0.21 (7.0)**
2	0.05 (3)	0.09 (4)	0.09 (4)	**0.07 (3.67)**	0.20 (8)	0.08 (3)	0.09 (5)	**0.12 (5.3)**
3	0.09 (6)	0.09 (4)	0.09 (3)	**0.09 (4.33)**	0.17 (8)	0.16 (8)	0.11 (8)	**0.15 (8.0)**
4	0.04 (2)	0.07 (2)	0.07 (2)	**0.06 (2.33)**	0.04 (3)	0.08 (3)	0.05 (3)	**0.06 (3.0)**
				0.07 (3.5)				**0.14 (5.9)**

Table 2.1: Results on a reaching task. Each task (target) was repeated three times. The mean values are reported in bold font.

	Reaching Precision (m)	
Target	**Curious**	**Normal**
1	0.20	0.67
2	0.26	0.61
3	0.25	1.06
4	0.24	0.67
5	0.37	0.49
	0.26	**0.7**

Table 2.2: Reaching a new target not seen during training.

The experimental platform for our hardware experiments is the Sawyer Robot (Sawyer, 2012). The purpose of the experiments was to demonstrate the applicability and the benefits of our algorithm on real hardware. We perform reaching experiments for 4 different target locations. Each experiment is started from scratch with no prior data, and the number of hardware experiments needed to reach the target are compared. The results are summarized in Table 2.1 and show the number of learning iterations needed in order to reach the target together with the precision in end-effector space. If the target was reached with a precision of below 10 cm, we would consider the task as achieved; if the target was not reached after the 8th learning iteration we would stop the experiment and consider the last end-effector position. We decided to terminate our experiments after the eight iteration as running the experiment on hardware was a lengthy process, as the GP training and the rollout would happen iteratively and GP training time increases with growing amount of data. Also, the reaching precision that we were able to achieve on hardware was significantly lower, compared to the simulation experiments. We believe this is due to the data collected from the Sawyer robot, as we could only control the robot at $100Hz$ which introduces inaccuracies when reading the effects of the sent torque command. We repeated each experiment three times to demonstrate the repeatability of our method as we expected measurement noise to affect solutions. From the table we can see that MBRL with curious iLQR would reach a target on average after 3.5 iterations with an average



(a) start configuration (b) target configuration

Figure 2.7: Joint configuration of Sawyer.

precision of 7 cm, compared to MBRL with regular iLQR that needed 5.9 iterations (often not ever reaching the target after eight iterations with the desired precision), with a precision of 14cm on average. As in simulation, similar to Experiment 2.5.2 we wanted to evaluate the quality of the learned models on new target positions. The results are summarized in Table 2.2 and are similar to what we observe in simulation: the models learned with curiosity, when used to optimize for new targets, can achieve higher precision than when using the models learned without curiosity.

2.7 Conclusion and future work

In this work, we presented a model-based reinforcement learning algorithm that uses an optimal control framework to trade-off between optimizing for a task specific cost and exploring around a locally optimal trajectory. Our algorithm explicitly encourages actions that seek out uncertainties in our model by incorporating them into the cost. By doing so, we are able to learn a model of the dynamics that achieves the task faster than MBRL with standard iLQR, and also transfers well to other tasks. We present experiments on a Sawyer robot in simulation and on hardware. In both sets of experiments, MBRL with *curious iLQR* (our approach) not only learns to achieve the specified task faster, but also generalizes to new tasks and initial conditions. All this points towards the conclusion that resolving dynamics uncertainty during model-based reinforcement learning is indeed a powerful tool. As (Loewenstein, 1994) states, curiosity is a superficial affection: it can arise, diverge and end promptly. We were able to observe similar behaviour in our experiments as well, as can be seen in Figure 2.5: towards the end of learning, the exploration signal around the trajectory decreases and the robot would explore, deviate from the task slightly, before going back to exploiting once it is fairly certain about the dynamics. In the future, we would like to explore this direction by considering how to maintain exploration strategies. This could be helpful if the robot is still certain about a task, even though the environment or task has changed.

Chapter 3

Tackling Model Bias: Leveraging Forward Model Prediction Error for Learning Control

3.1 Introduction

Data driven approaches to learn control have been introduced as an alternative to overcome the limitations of imperfect analytical models of the robotic tasks (Bristow *et al.*, 2006). In this work, we consider iterative model based learning, where we iterate between learning a forward dynamics model of the robot, using the forward model to learn a controller and using the controller to collect data on the robot. The controller essentially inverts the forward model, computing an action given a current and desired next or goal state. Two challenges arise here: first the quality of the learned forward model is decisive for the success of learning the controller and second, the learned controller is used to collect data on the robot used for learning the forward model. Both models thus influence each other and need to converge to a solution that is good in practice in order to perform a task successfully. We present an approach that couples forward model and controller learning and connects them by leveraging forward model prediction error during controller learning. The controller predicts the motor command required to achieve a desired state. The forward model predicts the next state, from the current measured state and motor command predicted by the controller, thus representing the causal relationship of the movement (Wolpert *et al.*, 1995). This couples the models, as the predicted action is used as input to the forward model.

From a robotics perspective, forward and inverse models are representations of the physical properties of the robot. In the neuroscience and cognitive science literature (Miall and Wolpert, 1996; Ito, 1970), the presence of internal models, as representation of the body in the human brain (Ishikawa *et al.*, 2016) are believed to play an important role. Forward models are of potential use for more than only representing the causal relationship of a movement in the sensory domain, but also for solving other fundamental cognitive problems: faster adaptation through

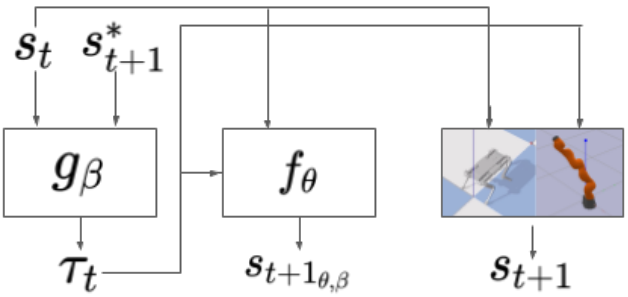


Figure 3.1: Overview of connecting inverse and forward models: the motor command, that is the output of the controller, is fed to the forward model used to predict the next state. The motor command is also run on the robot in order to observer the real next state. The learned controller is then updated by taking the gradient of either ((3.2)) or ((3.4)).

prediction ((Wolpert *et al.*, 1995; Hoffmann *et al.*, 2010)), anticipation of self caused movements, leading to a sense of agency (Blakemore and Frith, 2003), and enabling mental practice procedures through predictions, by only imagining the sequence of actions without actually performing them (Jackson *et al.*, 2003). All of these scenarios, involve a motor command as one of the inputs to the forward model. This motor command could be the output of an inverse model. In the cognitive literature there is believe for a combined workflow between forward and inverse models in the brain for motor learning and control (Wolpert *et al.*, 1998; Wolpert and Kawato, 1998). In (Wolpert *et al.*, 1998) the authors explain the necessity for a connection between forward and inverse models in the cerebellum by pointing out that acquiring an inverse model purely from motor learning is difficult, since the optimal motor command is not available during learning (otherwise the learning would not be necessary). From a robotics perspective, this argument holds as well, since a desired trajectory is usually defined in the state space and not in the action space.

Following this observation, and unlike the more common approach in the robotics literature, where the forward or the inverse model are trained separately using supervised learning from data (as for example in (Nguyen-Tuong *et al.*, 2008; Camoriano *et al.*, 2016; Atkeson and Reinkensmeyer, 1988; Miller, 1987)), we show how connecting the models, and formulating a loss in the state space, improves the performance when learning control. In contrast to other work (Jordan and Rumelhart, 1992) that considers learning these models together, we show how including the prediction error of the forward model during controller learning creates an unbiased loss signal, that leads to a significant improvement in performance.

In a nutshell, the primary contributions of this work are:

1 We propose a method for connecting controller and forward model learn-

ing during learning control in an iterative fashion on a manipulator and a quadruped.

2 We show, with theoretical and empirical results, how including forward model prediction error during controller learning significantly improves learning a motor control task on a robot by creating an unbiased learning objective.

3 We present manipulation and locomotion experiments in simulation, specifically we also show learning of a walking controller that can inherently handle contact switching.

This work is an extension of our previously published conference paper (Bechtle *et al.*, 2020c). In this work we extend our previous work:

4 We present experiments where we learn more general form of controllers like a policy, that does not have access to a desired full body trajectory.

5 we present a structured way of learning the forward model, that speeds up learning by including analytical knowledge of the robot's forward model.

6 we further extend our theoretical analysis of the proposed loss functions.

7 and we present hardware experiments evaluating our approach on a real quadruped.

3.2 Related Work

3.2.1 Coupling forward and inverse models

Wolpert et al. present in (Wolpert and Kawato, 1998) an architecture for multiple paired inverse and forward models, the pairs are coupled and trained jointly. The predictions of the forward models determine which inverse model to use. (Koert *et al.*, 2018) extends this for a manipulator. (Schillaci *et al.*, 2012) present results on coupled learning of kinematic models for tool use. In (Lutter *et al.*, 2019) the authors present a deep neural network that structures the learning of a manipulator's dynamics model following Lagrangian mechanics. The trained model can be used for forward as well as inverse dynamics computation, but does not directly connect the models. Most similar to our work is (Jordan and Rumelhart, 1992), where the authors show the benefits of using a 'distal teacher' for training the inverse model on a 2 link 2D arm. Their approach is based on a stochastic gradient, computed by comparing the observed states with the desired state. In contrast to these approaches, we present an iterative method to train the models jointly. Our experiments are conducted on two different robots, in 3D, and present a loss function that considers the forward model prediction error during controller learning.

We show in Section 5.3 how our approach mathematically differs from (Jordan and Rumelhart, 1992), and in Section 4.7 that it achieves significantly better results on higher dimensional systems. In particular, our approach can easily include contact interactions.

3.2.2 Using model prediction error for learning

The idea of using model prediction error during learning has been explored within the reinforcement learning literature mostly form the perspective of intrinsically motivated agents. For example, (Barto, 2004; Singh *et al.*, 2004, 2010) propose rewarding agents to minimize prediction errors of sensory events to explore the state space. This work is limited to low-dimensional and discrete state-and-action spaces. More recently (Bellemare *et al.*, 2016; Pathak *et al.*, 2017; Tanneberg *et al.*, 2019; Laversanne-Finot *et al.*, 2018) present results on higher dimensional systems, however this work focuses on model free reinforcement learning where the learned models are purely used to provide an additional learning signal to train a policy. In contrast to this work, our approach uses forward model prediction error during learning in a setting where the learned model is actually used to learn a motor control task.

3.2.3 Improving model learning in model based approaches

Fewer works have included additional learning signals during model based learning. (Shyam *et al.*, 2018) proposes a measure of disagreement in an ensemble of forward models as an exploration signal. (Bechtle *et al.*, 2020a) shows that including the predictive uncertainty of the forward model during controller optimization could improve forward model learning. In (Lopes *et al.*, 2012), an empirical measure of learning progress in included on a low dimensional discrete MDP. Similarly, self correcting forward models were proposed in (Talvitie, 2014, 2016) but the considered problem remains low dimensional. While it is widely acknowledged that model quality is of crucial importance in model based approaches, to the best of our knowledge this problem is seldom tackled for high dimensional systems.

3.2.4 Learning models including force measurements

Learning models that include non-trivial contact interactions is especially challenging as contacts create discontinuous force measurements and control actions. In (Zhang *et al.*, 2019) the authors use force measurements as an additional input to their model for a manipulator. However, the measurements are not used for controller learning but only to discriminate between different tasks. In (Lee *et al.*, 2019) multimodal input signals, including forces, are used to train an embedding for a downstream model free reinforcement learning task that takes as input the learned

embedding but does not use the learned model during policy learning. Even with accurate physical models, the conception of inverse dynamics controllers is challenging with changing contacts (Herzog *et al.*, 2016) as special care is necessary at each contact transitions, i.e. typically involving manual design of switching events or advanced constraint switching strategies (Jarquín *et al.*, 2013). We show in Section 4.7 how our approach enables to learn a walking controller for a quadruped by including measured contact forces not only as inputs, but also as predictions during controller learning. Importantly, the learned controller seamlessly handles contact switches without any additional assumptions as it learns to predict contact switches using the forward model.

3.2.5 Learning models with structure

When learning forward or inverse dynamics models from data the literature offers a variety of choices of machine learning algorithms used for this learning task. From linear regression (Schaal *et al.*, 2002; Haruno *et al.*, 2001), to gaussian mixture (Khansari-Zadeh and Billard, 2011; Calinon *et al.*, 2010) or gaussian process regression (Deisenroth and Rasmussen, 2011; Kocijan *et al.*, 2004) as well as using feedfoward- or recurrent neural networks for fitting the models (Lenz *et al.*, 2015; Sanchez-Gonzalez *et al.*, 2018; Rueckert *et al.*, 2017). These approaches usually collect a dataset for supervised learning, and fit the models in an end to end fashion from input to output. Some other works however have considered more structured approaches for model learning, including analytical priors during the learning process. For example in (Calandra *et al.*, 2015) the authors learn only the residual term of the external forces, with for inverse dynamics. In (Lutter *et al.*, 2019) the authors learn deep neural networks for each component of the equations of motion of a manipulator whereas in (Ledezma and Haddadin, 2017) the authors learn the dynamics parameters directly via gradient descent. Similar in (Sutanto *et al.*, 2020) the authors propose a fully differentiable version of the recursive newton euler algorithm, allowing the inertial parameters to be learned using gradient descent and automatic-differentiation for gradient computation. In this work we propose a way to include structure for one step prediction of the forward model, by using the analytical dynamics parameters and only learning the contact dynamics from data with a neural network.

3.3 Problem Formulation and Approach

The goal of model based learning control is to learn a forward model f of the dynamics of the robot and a controller, or inverse model, g. In general, g can be learned from data but can also be optimized using trajectory optimization algorithms. See (Levine and Koltun, 2013; Deisenroth and Rasmussen, 2011; Bechtle *et al.*, 2020a)

for a variety of approaches of iteratively learning a model and a controller.

In this work, we propose an algorithm inspired by the concept of connected forward and inverse models, while still being able to iteratively collect data and update the models. We learn a forward model f_θ that performs one step prediction of the form $s_{t+1} = f_\theta(s_t, \tau_t)$, where θ are the parameters of the forward model, s_t and τ_t the state and action at time t. We also learn a controller g_β that predicts $\tau_t = g_\beta(s_t, s^*)$, given the current state s_t and the desired state s^*. β are the parameters of the controller and s^* can be the immediate desired next state s^*_{t+1}, or a final goal state s^*_T, where $t = 0...T$ and T is the time horizon of the task. We learn both models from data collected on the robot,while alternating between model learning and data collection. Algorithm 4 shows the training procedure. We create a direct connection between f_θ and g_β by using the action predicted by g as an input to f. Since $s_{t+1\theta,\beta} = f_\theta(s_t, g_\beta(s_t, s^*))$, the next state is a function not only of the parameters of f but also of g. We can then use the prediction of the forward model to compute an error signal for inverse model training by creating a direct connection from the prediction of the forward model to the output of the inverse model. This means that, using $s_{t+1\theta,\beta}$ we can formulate a loss that enables us to compute a gradient to update the parameters β of g.

In Figure 3.1 the coupling of the forward and the inverse model is illustrated. Using $s_{t+1\theta,\beta}$ has the advantage of representing the actual effect that the action, that was predicted by g has. In contrast to learning g in a supervised fashion from collected data, this approach is conceptually more sound as the correct or desired supervision signal for the action itself is usually not available. However the goal of the task, s^*, is available in the state space, since it defines the task.

In model based approaches, forward models and controllers are inherently intertwined: during the training phase, the forward model predicts the possible next state, and the controller is learned based on this prediction. The controller is the acting component of the loop, facilitating data collection on the robot. The data is then used to update the models. It becomes clear here, that if the forward model prediction is inaccurate, controller training will fail and converge to a solution that, when used on the robot, collects data that might not be meaningful for the current task and eventually bias the models. This brings us back to one of the major challenges of model based learning, which is to learn models that are accurate enough to use for motor control on a robot.

In the next section, we introduce a new loss function as well as other, more standard, losses used as comparison. We propose a loss function for controller learning that ultimately reduces model bias, by including forward model prediction error for learning control. As a result, this improves model prediction and as a consequence task performance.

3.3.1 Learning control via coupled models with *joint loss*

Our approach (Algorithm 4) alternates between model learning and data collection. g and f are randomly initialized at the beginning of the learning loop. Each iteration collects data using the controller g for the duration of a predefined horizon T. After the roll-out, the collected data is used to update both the forward model f and the controller g.

Algorithm 4 Learning control with Coupled Models

1: $\mathcal{D} \leftarrow$ motor babbling data(s_t, τ_t, s_{t+1})
2: $f_\theta \leftarrow$ initialize forward model
3: $g_\beta \leftarrow$ initialize inverse model
4: train model f_θ on $\mathcal{D}$
5: train model g_β on $\mathcal{D}$
6: **while** $i <$ iter **do**
7: $\quad D_{\text{new}} \leftarrow$ rollout g_θ on system(s_t, τ_t, s_{t+1}), $t = 0...T$
8: $\quad \mathcal{D} = \mathcal{D} \cup D_{\text{new}}$
9: $\quad$ train model f_θ on $\mathcal{D}$ with Loss from ((3.1))
10: $\quad$ train model g_β on $\mathcal{D}$ with Loss from ((3.2)) or ((3.4))
11: **end while**

To update the forward model, we use a regular supervised learning objective representing the model prediction error

$$\mathcal{L}_{sup}(\theta) = (f_\theta(s_t, \tau_t) - s_{t+1})^2 \tag{3.1}$$

where s_{t+1} is the next state observed on the robot and $f_\theta(s_t, \tau_t)$ is the next state predicted by f.

To learn g_β, we propose a loss function *joint loss* that trades-off actual robot behavior and control performance prediction using the forward model. We compare it with two other, simpler, approaches: one, *task loss* that only improves control performance prediction using the forward model and a supervised approach that does not use the forward model.

Comparison - updating g with *task loss*

The *task loss* computes a learning objective by comparing the prediction of the forward model (that was coupled with the output of g_β): $s_{t+1\theta,\beta} = f_\theta(s_t, g_\beta(s_t, s^*_{t+1}))$ with the desired next state s^*_{t+1}

$$\mathcal{L}_{task\ loss}\ (\beta) = (f_\theta(s_t, g_\beta(s_t, s^*_{t+1})) - s^*_{t+1})^2 \tag{3.2}$$

This loss evaluates how well the action of g will be able to achieve the desired state s^*_{t+1} by using f to predict the next state. Intuitively, this will lead to the desired

behaviour only if the prediction of the forward model is accurate enough, making the learned controller susceptible to model-bias and inaccuracies.

Comparison - updating g with supervised loss

Alternatively, a general supervised learning loss can be used, of the form

$$\mathcal{L}_{inverse\ sup}(\beta) = (g_\beta(s_t, s_{t+1}) - \tau_t^{run})^2 \tag{3.3}$$

where s_{t+1} is the observed next state when executing τ_t^{run} on the robot, and τ_t^{run} is the output of $g_\beta(s_t, s_{t+1}^*)$. This loss is the most common in the literature, especially for inverse dynamics learning (Camoriano *et al.*, 2016; Pathak *et al.*, 2017). $\mathcal{L}_{inverse\ sup}(\beta)$ uses the observed data to update the controller. In contrast to the *task loss* and also our *joint loss* , this loss is not goal oriented, but purely tries to learn the state-control relationship by fitting observed data.

Updating g with *joint loss*

Our proposed *joint loss* accounts for the quality of the dynamics model, by adding a term that compares the predicted next state with the actual next state.

$$\begin{aligned}\mathcal{L}_{joint\ loss}\ (\beta) = (f_\theta(s_t, g_\beta(s_t, s_{t+1}^*)) - s_{t+1}^*)^2 \\ +(f_\theta(s_t, g_\beta(s_t, s_{t+1}^*)) - s_{t+1})^2\end{aligned} \tag{3.4}$$

where s_{t+1} is the next state observed on the robot. The *joint loss* thus evaluates not only how well τ_β was able to achieve the desired next state (as predicted by the forward model), but also how good the predictive performance of the forward model actually is. This essentially creates a trade-off between controller and forward model performance, shifting the data distribution seen during roll-out towards a solution that is desirable in reality. In all cases, the parameters of g_β are then optimized with gradient descent by taking the gradient $\nabla_\beta \mathcal{L}(\beta)$.

In the next section, we analyse in details the *task loss* and the *joint loss* . We show why adding the forward model prediction error benefits the controller, and as a consequence, also forward model learning. We then experimentally compare in Section 4.7 these losses with the supervised loss, and show the benefits of our *joint loss* .

3.3.2 Theoretical analysis of loss functions

To show the benefit of including the forward model prediction error during inverse model learning let's consider a simplified 1D example: $s_{t+1_{\theta,\beta}} = f_\theta(s_t, g_\beta(s_t, s_{t+1}^*))$. Where $s_{t+1_{\theta,\beta}}$ is the prediction of the forward model f_θ, s_t is the current state, s_{t+1} is the actual next state observed on the robot and s_{t+1}^* is the desired next state. g_β

computes the action for given s_t and s^*_{t+1}. In order to update parameters β of g the gradients that have to be computed are

$$\nabla_\beta \mathcal{L}_{task\ loss} = 2\frac{\delta f_\theta}{\delta g_\beta}\frac{\delta g_\beta}{\delta \beta}(s_{t+1_{\theta,\beta}} - s^*_{t+1}) \tag{3.5}$$

and

$$\nabla_\beta \mathcal{L}_{joint\ loss} = 2\frac{\delta f_\theta}{\delta g_\beta}\frac{\delta g_\beta}{\delta \beta}(2s_{t+1_{\theta,\beta}} - s^*_{t+1} - s_{t+1}) \tag{3.6}$$

When looking at ((3.5)) it becomes clear, that $\nabla_\beta \mathcal{L}_{task\ loss} = 0$ when $s^*_{t+1} = s_{t+1_{\theta,\beta}}$ which means, when the predicted next state is equal to the desired next state. This is a desirable equilibrium, if the forward model prediction is accurate enough, meaning that the predictions of f are not biased. However, if this is not the case, g reaches its equilibrium given a biased model and converges to the wrong solution. We are going to show in section 4.7 how this model bias can affect negatively the learning performance, even if the forward model keeps being improved.

In the case of ((3.6)), the general solution for equilibrium is $s_{t+1_{\theta,\beta}} = \frac{s^*_{t+1}+s_{t+1}}{2}$, which is the average between the desired next state and the measured next state. The special solution $s_{t+1_{\theta,\beta}} = s^*_{t+1} = s_{t+1}$ would be desired. However, since we optimize in an iterative way, if $s_{t+1_{\theta,\beta}} = \frac{s^*_{t+1}+s_{t+1}}{2}$ and we continue optimizing, we can plug the general solution back into $\mathcal{L}_{joint\ loss}$ and we get

$$\begin{aligned} \mathcal{L}_{joint\ loss} &= (\frac{s_{t+1} + s^*_{t+1} - 2s^*_{t+1}}{2})^2_{\beta,\theta} \\ +(\frac{s_{t+1} + s^*_{t+1} - 2s_{t+1}}{2})^2_{\beta,\theta} &= \frac{1}{2}(s_{t+1} - s^*_{t+1})^2_{\beta,\theta} \end{aligned} \tag{3.7}$$

This means, the loss will reach its global minimum when $s^*_{t+1} = s_{t+1}$ which becomes an unbiased loss function. It is worthwhile noting that this loss still carries gradient information for β to further improve the inverse model, and is directly affected, through the forward model, by changes of the inverse model. The general solution, $s_{t+1_{\theta,\beta}} = \frac{s^*_{t+1}+s_{t+1}}{2}$, is a local minimum, that the optimization could get stuck in. However we observe that, because our approach alternates between learning the models and collecting new data, the *joint loss* and its trade-off between controller performance and forward model prediction error, facilitates data collection that allows to reach the global minimum $s^*_{t+1} = s_{t+1}$. We show empirical evidence for this hypothesis in section 4.7. In addition to being an unbiased loss, this loss also now reflects a kind of feedback controller loss, trying to push the inverse model to match the observed data with the desired data. Once $s_{t+1} = s^*_{t+1}$ then also the special solution $s_{t+1_{\theta,\beta}} = s^*_{t+1} = s_{t+1}$ holds and in particular also $s_{t+1_{\theta,\beta}} = s_{t+1}$. Another observation is that, if we only train the inverse model on $\mathcal{L} = (s_{t+1_{\theta,\beta}} - s_{t+1})^2$, the equilibrium of the inverse model is reached when $s_{t+1_{\theta,\beta}} = s_{t+1}$ which means when

the forward model is predicting the actual next state. Potentially this could also help to learn forward models from scratch: instead of random motor babbling as initial data collection, this loss could provide a self supervised way to collect data.

3.3.3 Comparison distal teacher loss proposed by (Jordan and Rumelhart, 1992)

In (Jordan and Rumelhart, 1992) the authors propose a stochastic gradient of the form

$$\nabla_\beta \mathcal{L}_{\text{(Jordan and Rumelhart, 1992)}} = \frac{\delta f_\theta}{\delta g_\beta} \frac{\delta g_\beta}{\delta \beta} (s^*_{t+1} - s_{t+1}) \tag{3.8}$$

Here the current gradient of the forward model w.r.t. β is used, but the loss does not carry gradient information, as it is purely specified in terms of the observed and desired data. This is equivalent of formulating a loss of the form

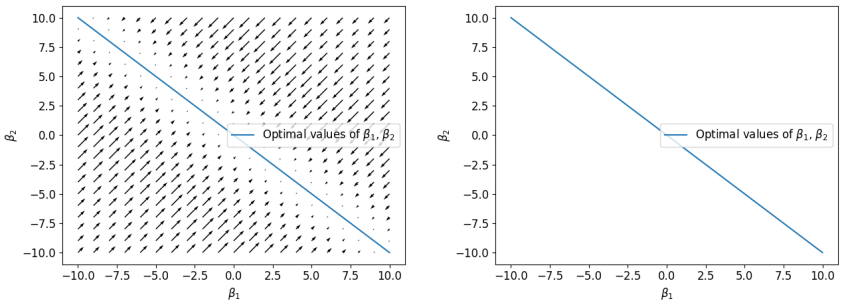
$$\mathcal{L}_{\text{(Jordan and Rumelhart, 1992)}} = (s_{t+1_{\theta,\beta}} - s^*_{t+1})^2 - (s_{t+1_{\theta,\beta}} - s_{t+1})^2 \tag{3.9}$$

which effectively subtracts the forward model prediction error from the *task loss* error and eventually does not care about the quality of the forward model, as long as the loss between actual and desired next state is decreasing. In simpler scenarios this can have the effect that goal oriented behaviour is achieved even if the forward model is not perfect, as stated in (Jordan and Rumelhart, 1992). On the other hand, this loss does not account for wrong gradients taken through the forward model, that way biasing the solution because of an inaccurate forward model. In practice this seems to be a significant drawback for higher dimensional systems as we show in Sec. 4.7.

3.3.4 Illustration of model bias problem with point mass

To further illustrate the effect of model bias when optimizing g, and how our *joint loss* can help with this problem, we use a simple linear point mass example with linear forward dynamics $s_{t+1_\theta} = s_t\theta_1 + \tau_t\theta_2$ and inverse dynamics $\tau_t = s_t\beta_1 + s^*_{t+1}\beta_2$. Coupling the two models, as described earlier gives a prediction of the next state of the form $s_{t+1_{\theta,\beta}} = s_t\theta_1 + s_t\beta_1\theta_2 + s^*_{t+1}\beta_2\theta_2$. In this simple scenario it is possible to enforce an extreme case of model bias when $s_{t+1_{\theta,\beta}} = s^*_{t+1}$, by setting $\theta_1 = \frac{(s^*_{t+1} - \tau_t)}{s_t}$ and $\theta_2 = \frac{(s^*_{t+1} - s_t\theta_1)}{(\beta_1 s_t - \beta_2 s^*_{t+1})}$. In this case $\mathcal{L}_{task\ loss} = 0$ and thus the *task loss* converged. This simple example allows for gradient analysis by plotting the gradient fields of *task loss* and *joint loss* . In Fig. 3.2 the gradient field is shown for the parameters β of g. The figure shows how the gradients behave as a function of the values of β_1 and β_2. In the case of the *task loss* (right), the gradients are zero everywhere, since when $s_{t+1_{\theta,\beta}} = s^*_{t+1}$ also $\nabla_\beta \mathcal{L}_{\text{task}} = 0$, thus g converged even if the desired valued of β

have not been recovered. In the case of the *joint loss* (left), it is possible to see that even if model bias was introduced, still the gradients will facilitate convergence to the desired values of β. This shows how our *joint loss* is superior to the *task loss* in this case.



(a) Gradient field for β when using *joint loss* with model bias.

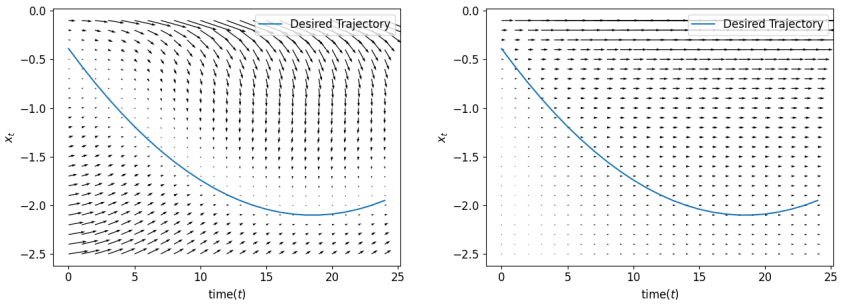
(b) Gradient field for β when using *task loss* with model bias.

Figure 3.2: Gradient fields of controller parameters β with introduced model bias. The line represents the desired values of β. Using the *joint loss* still allows for gradients that converge to the optimal values for β.

Another, more intuitive way of comparing these the losses and they gradient behaviour, is by looking at Fig. 3.3. The figure shows the resulting gradient field in state space, showing how changes in parameters β will affect the resulting trajectory. Concretely this means computing $\nabla_{s_t} s_{t+1_{\theta,\beta}}$ and $\nabla_{s^*_{t+1}} s_{t+1_{\theta,\beta}}$, where β has been updated using either $\nabla_\beta \mathcal{L}_{\text{joint}}$ or $\nabla_\beta \mathcal{L}_{\text{task}}$. From Fig. 3.3 it is possible to see that when using the *joint loss* the trajectory will converge to the desired trajectory even if model bias is present. Which is not the case when using *task loss* where the gradients will not facilitate the convergence to the desired solution.

3.3.5 Forward Model learning including contacts with Structured Priors

In this work we also learn a forward dynamics model f that includes contact interactions in its prediction. This means $s_{t+1} = f(s_t, \tau_t)$ where $s_t = [q, \dot{q}, f_{[x,y,z]}]$, where q and $\dot{q}$ are joint positions and velocities and $f_{[x,y,z]}$ are the contact forces at the end effector. Besides of learning the forward model f end to end with a highly parametrized function approximator like a neural network, we now present an alternative avenue using prior structured information for forward model learning. For this we consider the following general formulation of a robot's dynamics



(a) Gradient field as a function of the state s_t and the respective desired state s^*_{t+1} when using *joint loss* with model bias.

(b) Gradient field as a function of the state s_t and the respective desired state s^*_{t+1} when using *task loss* with model bias.

Figure 3.3: Gradient fields of the system states, when using optimized parameters β with introduced model bias. The line represents the desired trajectory in state space. Using the *joint loss* still allows for gradients that converge to the optimal desired trajectory, which is not the case when using *task loss* .

following the lagrangian formulation of rigid body dynamics (Siciliano *et al.*, 2010).

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{J}^T\boldsymbol{F} \tag{3.10}$$

Where $\boldsymbol{M}$ is the inertia matrix, $\boldsymbol{G}$ are the gravitational, centripedal and coriolis forces, $\boldsymbol{J}$ is the Jacobian $\boldsymbol{F}$ are the contact forces and $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ are the joint positions, velocities and accelerations. The forward dynamics relationship, to recover $\ddot{\boldsymbol{q}}$ is given by

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{\tau} - \boldsymbol{G}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{J}^T\boldsymbol{F}) \tag{3.11}$$

For a rigid body $\boldsymbol{M}(\boldsymbol{q})$ and $\boldsymbol{G}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ can be computed accurately analytically. We want to leverage this prior analytical knowledge when learning the full forward dynamics including the contact dynamics. To this end we divide equation (3.11) into a learnable and a non learnable part. The learnable part being the part of the equation involving contact interaction, since modelling contacts is still very challenging and the analytical models often not accurate when compared to the real measurements.

$$\hat{\ddot{\boldsymbol{q}}}_{t+1} = \underbrace{\boldsymbol{M}^{-1}(\boldsymbol{q}_t)(\boldsymbol{\tau} - \boldsymbol{G}(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t))}_{\text{not learned}} - \underbrace{\boldsymbol{M}^{-1}(\boldsymbol{q}_t)(\boldsymbol{J}^T\boldsymbol{F})}_{\text{learned}} \tag{3.12}$$

$\hat{\ddot{\boldsymbol{q}}}_{t+1}$ is thus composed of a learned part and a part that is computed analytically. We model $\boldsymbol{M}^{-1}(\boldsymbol{q}_t)(\boldsymbol{J}^T\boldsymbol{F})$ as a neural network and learn this part from data by

computing a supervised learning loss between the prediction $\hat{\ddot{\boldsymbol{q}}}_{t+1}$ and the observed $\ddot{\boldsymbol{q}}_{t+1}$ acceleration.

$$\mathcal{L} = (\hat{\ddot{\boldsymbol{q}}}_{t+1} - \ddot{\boldsymbol{q}}_{t+1})^2 \tag{3.13}$$

3.4 Experiments on inverse dynamics learning

In this section, we present experiments to show empirically the benefits of learning control with coupled models and our *joint loss* . We show how our method of including the forward model prediction error during inverse dynamics model learning outperforms all the other methods. We show evidence that including the forward model prediction error leads to a robot behaviour that favours data collection to improve forward model learning and ultimately less biased models.In this section we present experiments in simulation with a 7DoF iiwa Kuka arm (AG, 2020) and the 12 DoF quadruped robot Solo (Grimminger *et al.*, 2020) (Fig.3.1). All robots are simulated with PyBullet (Pybullet, 2012).

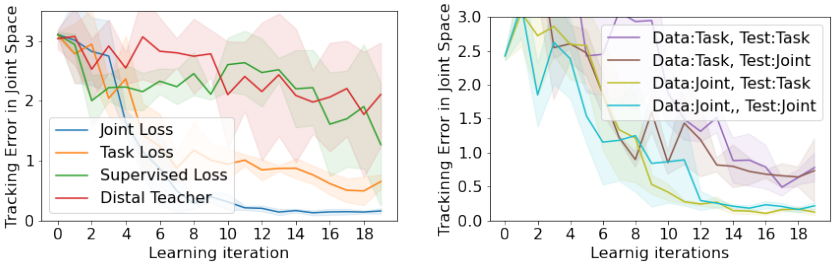
3.4.1 Experiments with Kuka iiwa

In these experiments, we learn the forward model with an ensemble of probabilistic neural networks, similar to (Chua *et al.*, 2018). For the forward model we use three hidden layers with 400 neurons each and ReLU activation functions and an ensemble size of 3. The controller is a neural network with three hidden layers, with 300, 200 and 100 neurons each and ReLU activation function. In the experiments we learn the inverse dynamics model of the Kuka arm and we compare performance for reaching tasks for five different target positions. Each time, the controller needs to track a desired reaching trajectory in joint space that is computed in advance, i.e. we learn a tracking controller.

The inverse dynamics model $\tau_t = g_\beta(x_t, \ddot{q}^*_{t+1})$ takes as an input the state $s_t = [q_t, \dot{q}_t]$, where q_t are the joint angles and $\dot{q}_t$ the joint velocities at time t, and the desired joint acceleration at the next time step $\ddot{q}^*_{t+1}$ and outputs the torque τ_t. The forward model takes as an input the current state s_t and action τ_t coming from the inverse model. In Fig.3.4a we can see that training the inverse model with the *joint loss* leads to faster and more stable convergence when compared to the other methods. Notably, it can consistently learn a very good tracking controller in less than 10 iterations.

Is the *joint loss* improving data collection?

In model based learning the controller g significantly influences the data seen during learning, since it is the acting component which enables the robot to move and



(a) Inverse dynamics learning (b) Losses on collected datasets

Figure 3.4: (a) Experiments on 7 DoF Kuka arm, the MSE tracking errors (mean and standard deviation over all learning experiments) are reported over learning iterations. (b) We compare the performance of the *joint loss* and *task loss* when used on a pre-collected dataset. We pre-collect two datasets, one when using *joint loss* for learning control, and the other when using *task loss* . We see here that *joint loss* and *task loss* performs similiar when deployed on the same dataset. We also see that both perform well on the dataset pre-collected with the *joint loss* suggesting that *joint loss* leads to better data distrubutions in the learning problem.

collect more data. Therefore, it is natural to ask whether the performance observed with the *joint loss* is only due to the quality of the data collected during learning. To test this hypothesis, we collect two dataset while running our learning loop with *task loss* and *joint loss* for inverse dynamics learning. After collecting the two datasets, we re-train the inverse and forward models from scratch with both losses on the datasets collected. The results of this experiment are shown in Fig.3.4b, where we can see that the *task loss* and *joint loss* perform similarly when they are deployed on the exact same data. In particular, both losses perform better when trained with the data collected using the controller optimized using the *joint loss* . This suggests that the data collected while learning with the *joint loss* contains more useful information to accomplish a given motor control task and confirms our hypothesis of less biased model learning with *joint loss* .

3.4.2 Experiments on Solo

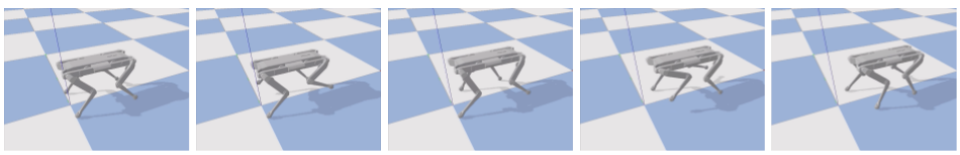
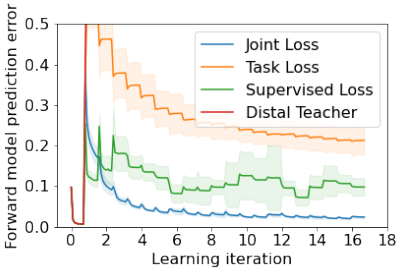


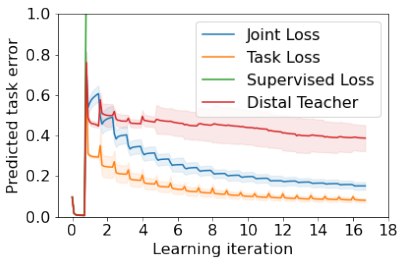
Figure 3.5: Image sequence of solo successfully walking

In this section we present experiments performed on the Solo quadruped robot

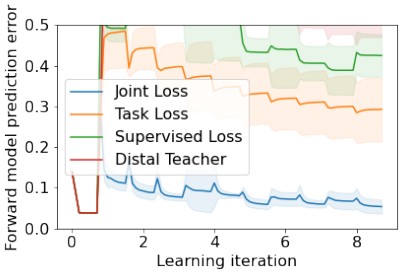
(Grimminger *et al.*, 2020). The forward model is learned with a neural network with three hidden layers of 1000, 500, 500 neurons each and Relu activation function. The input to the forward model is the current torque τ_t and the current state $s_t = [x_{bb_t}, q_t, f_{[x,y,z]_t}, \dot{x}_{bb_t}, \dot{q}_t]$ where x_{bb_t} is the current base pose (position and orientation), $\dot{x}_{bb_t}$ the current base velocity and $f_{[x,y,z]_t}$ are measured contact forces at the four end effectors. The forward model predicts $\Delta s = [\ddot{x}_{bb_{t+1}}\Delta t, f_{[x,y,z]_{t+1}}, \ddot{q}_t\Delta t]$, which is the change in acceleration for the base and joints together with the expected contact forces at $t+1$ when applying τ_t in s_t. The inverse model's architecture is of three hidden layers with 300 neurons each, the input is s_t and s^*_{t+1} which is the desired accelerations of the joints and base together with the desired contact forces at the next time step. We compute the desired trajectories (walking and jumping) using the kino-dynamic planner presented in (Ponton *et al.*, 2018). The goal of these experiments is to show that our approach can also use force measurements and handle hard contact switching for unstable underactuated systems. This is a significantly more challenging task compared to inverse model learning of a fixed base manipulator.



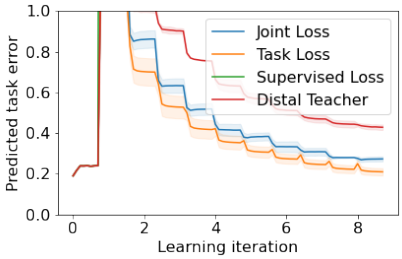
(a) Forward model prediction error during learning (walking)



(b) Predicted task error during learning (walking)

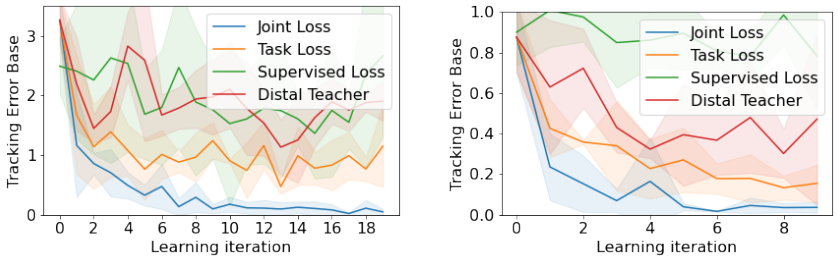


(c) Forward model prediction error during learning (jumping)



(d) Predicted task error during learning (jumping)

Figure 3.6: Comparing forward model prediction error and predicted task error during inverse model optimization for walking and jumping



(a) Solo walking (b) Solo jumping

Figure 3.7: Inverse model learning on the quadruped, the tracking error in base position and orientation is reported over iterations. Only the *joint loss* is successful at the task.

Learning to walk

In this set of experiments we show how we can learn to control walking. We average our results over 5 different walking horizons of different length and report the mean and the standard deviation of the experiments. When walking, the robot has to make and break contact with the floor multiple times during the trajectory. Making and breaking contact, and transitioning between these two states, is a challenging task that requires careful control at the moment of contact to avoid slipping and preventing the robot from falling. Contact forces are explicitly included in the state as well during model learning and controller optimization. Thus, the *joint loss* also includes the error on the contact forces, between predicted, desired and observed contact forces.

In Fig. 3.7a we show the tracking error of the base over learning iterations. We can see clearly here that our *joint loss* outperforms all the other approaches. Qualitatively the controller trained with *joint loss* is the only one that was able to generate stable walking (cf. Fig.3.5). This also becomes evident when looking at Table 3.1, where we report the tracking error of the ground reaction forces, together with the tracking performance visualized in Fig.3.8, where we show the predicted (by the forward model), desired and observed ground reaction forces at the front right foot. The controller trained with the *joint loss* (left), tracks the desired forces more accurately than the one trained with *task loss* (right). Also the forward model's prediction of the contact forces, is more accurate in the *joint loss* case. Importantly, the controller is again learned in less than 10 iterations, which makes it amenable to real robot applications.

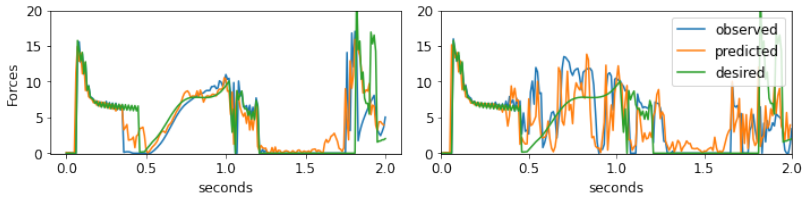


Figure 3.8: Tracking of ground reaction forces for *joint loss* (left) and *task loss* (right)

	MSE Tracking Error: mean(std) in N			
	Joint Loss	**Task Loss**	**Distal Teacher**	**Supervised**
forces (walking)	**0.09 (0.1)**	0.16 (0.18)	0.34 (0.35)	0.16 (0.26)
forces (jumping)	**0.02 (0.01)**	0.3 (0.28)	0.38 (0.4)	0.28 (4.3)

Table 3.1: Tracking error in ground reaction forces

Learning to jump

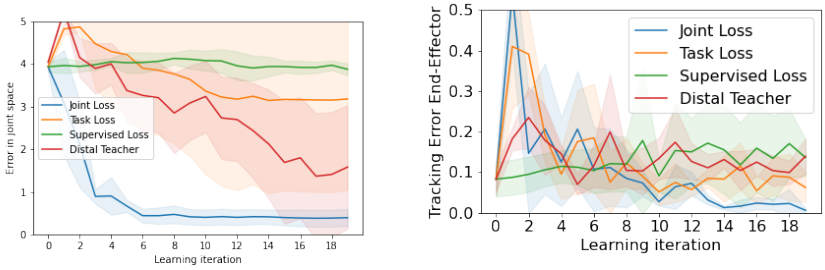
This experiment shows how Solo can learn to control a jump, which is a task with high impact dynamics and requires precise control especially during take-off (to create the right amount of momentum) and landing (to dissipate the impact). We show the results over 5 different jumping heights in Fig.3.7b where we can again see that the *joint loss* learns how to successfully accomplish the task. We show again, in Table 3.1 the tracking error for the ground reaction forces.

3.4.3 Model bias and its effect on performance

When looking at Fig. 3.6 we can see how the optimization with *task loss* is prone to find sub-optimal solutions due to a biased forward model. In Fig.3.6a and 3.6c the prediction error of the forward model during learning is shown for both experiments. We see that the prediction error of the forward model trained while running the experiment using the *joint loss* is lower than the prediction error of the forward model when using the *task loss* . On the other side in Fig.3.6b and 3.6d the predicted task error ($(s_{t+1_{\theta,\beta}} - s^*_{t+1})^2$) is shown. For the experiment trained with the *task loss* , the predicted task error is the lowest, however the prediction error of the forward model is high. This means that the model trained with *task loss* is predicting s^*_{t+1}, but the prediction is not correct. This leads to a biased solution, which explains the higher tracking error. Training with the *joint loss* does not result in this scenario, since the prediction of the forward model is accurate, in turn, leading to a better performing controller.

3.5 Experiments on general controller learning

So far we have only considered inverse dynamics model learning for motor control. In this section we are going to present experiments that learn a motor control task without having information about the desired trajectory for each joint, but only more general goal information regarding the given task, like a desired final state or a desired trajectory in task space. Again we present experiments on the 7 DoF kuka iiwa 7 and solo in simulation.



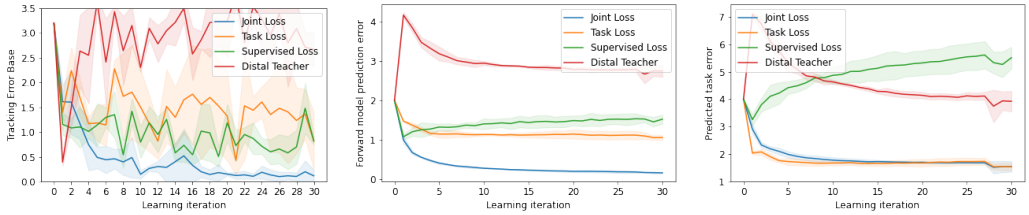
(a) Policy learning (b) Task-space controller learning

Figure 3.9: Results for general controller learning on Kuka arm, our *joint loss* outperforms all other approaches.

3.5.1 Learning operational space controller with kuka iiwa 7: the forward model as a disambiguator

In this set of experiments we show how the coupled learning with *joint loss* can also be used to learn a task-space controller (Khatib, 1987). g_β takes as an input the current state $s_t = [q_t, \dot{q}_t]$ and a desired acceleration in end-effector space $\ddot{x}^*_{ee_{t+1}}$. The forward model learns a combination of forward dynamics and kinematics, it takes as an input s_t and τ_t and outputs $\ddot{q}_t$ and $\ddot{x}_{ee_{t+1}}$. The problem of learning an operational space controller is more challenging than learning an inverse dynamics model, since joint redundancy implies that an infinite number of controllers can lead to $\ddot{x}^*_{ee_{t+1}}$. The non-uniqueness of a perfect tracking controller renders learning difficult when done in a supervised way from collected data, since the same input to g can have different output values. In this scenario, using the coupled models approach, we can disambiguate the problem since the forward model's mapping is unique. In Fig.3.9b, we can see experimentally how our approach outperforms others enabling to consistently learn an operational space controller in a few iterations. It becomes evident here that learning an operational space controller from data in a supervised learning fashion does not perform satisfactory because of the redundancy in mapping torques to end-effector accelerations. Previous approaches

to learn operational space controllers have been proposed (Peters and Schaal, 2006), however they only consider learning in the vicinity of a local model, to force the selection of only one of the many redundancy resolution strategy.



(a) Base tracking error during learning

(b) Forward model prediction error during learning

(c) Predicted task error during learning

Figure 3.10: Results for operational space controller learning for a walking controller on solo. The *joint loss* is the only loss that will lead to desired controller convergence. The *task loss* does not converge since the predictions of the forward model are biased.

3.5.2 Learning a policy with kuka iiwa 7

So far we have only considered learning controllers g_β that had access to a desired acceleration trajectory, either in joint space or in task space. Now we are going to present results of learning a controller that only takes as an input the current state $s_t = [q_t, \dot{q}_t]$ and a desired last state q_T^* in joint space. This is equivalent to learning a goal conditioned policy, which is a more challenging task since the goal information is not available at every time step but only for the final state at the end of the trajectory. During optimization we compute a loss by comparing every predicted next state to the final goal state $(s_{t+1_{\beta,\theta}} - s_T)^2$. When using the *joint loss* we also add the forward model prediction error at each time step, as described before. In Fig.3.9a we can see the results. As expected, learning a policy in a supervised way from data performs poorly, since mapping the current state and the desired last state to an action, is a non unique mapping making it impossible to train it in a supervised way. On the other hand clearly the *joint loss* performs the best in this more challenging task, learning a policy that achieves the desired goal position. Using the *task loss* leads to a very high variance in performance and bad mean behaviour, since controller learning converges to a biased local minima.

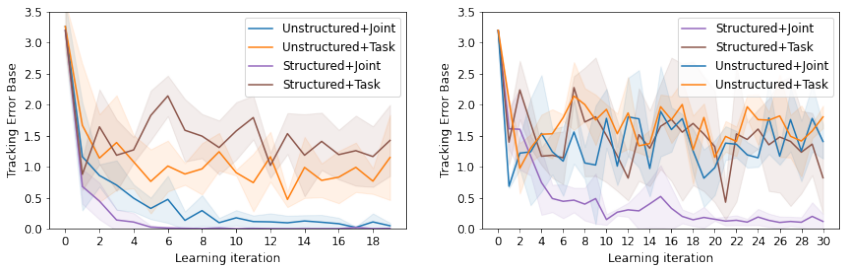
3.5.3 Learning a operational space controller with solo for walking

We now show how it is possible to use our approach to learn an operational space controller for walking on solo. In this experiments the forward model was learned with structured priors as presented in section 3.3.5. In this experiment we do not have a desired trajectory in joint space as previously, but have only a desired trajectory of the forward movement of the base and desired foot displacement of solo. We show how our approach with *joint loss* is able to learn a controller for full-body control for 5 different walking horizons. We model the contact dynamics with a feed-forward neural network with three layers with 1000, 500, 500 neurons each and ReLU activation functions. The input to the forward model is the current torque τ_t and the current state $s_t = [x_{bb_t}, q_t, f_{[x,y,z]_t}, \dot{x}_{bb_t}, \dot{q}_t]$ where x_{bb_t} is the current base pose (position and orientation), $\dot{x}_{bb_t}$ the current base velocity and $f_{[x,y,z]_t}$ are measured contact forces at the four end effectors. The forward model predicts $\Delta s = [\ddot{x}_{bb_{t+1}}\Delta t, \dot{x}_{ee_{t+1}}, \ddot{q}_t\Delta t]$, which is the change in acceleration for the base and joints together with the expected foot displacement $\dot{x}_{ee_{t+1}}$ at $t+1$ when applying τ_t in s_t. We compute $\dot{x}_{ee_{t+1}} = J^T\dot{q}_{t+1}$, where $\dot{q}_{t+1}$ is the predicted joint velocity and J^T is computed analytically. The controller model's neural network architecture is of three hidden layers with 300 neurons each, the input is the current states_t and $\ddot{x}^*_{bb_{t+1}}$, $\dot{x}^*_{ee_{t+1}}$ which are the desired accelerations of the base and desired foot displacement respectively. The controller then outputs a torque τ.

In Fig. 3.10a we compare again our *joint loss* with *task loss* , distal teacher loss and the supervised learning loss. Also here only controller learning with *joint loss* converges to the desired solution and the quadruped walks. In Fig. 3.10b and Fig. 3.10c the forward model prediction error and the predicted task error are shown. Even if the predicted task error is similar between *joint loss* and *task loss* the forward model prediction error is significantly higher when training with *task loss* , signalizing that the forward model is biased to predict states that are favourable to the task but not accurate to the real world observations.

3.6 Experiments on comparing structured and unstructured forward models

Finally in this last set of simulation experiments we want to show how learning forward models with structured analytical priors compares to learning forward models end to end on a contact rich task like walking. We show the results of inverse dynamics and operational space controller learning on solo. In Fig. 3.11 it becomes evident that adding structured priors improves learning the motor control task for both kinds of controller. In Fig. 3.11a the results for inverse dynamics learning for the walking task are shown. Here we can see that using a structured prior improves



(a) Tracking error on base, comparing models for inverse dynamics learning

(b) Tracking error on base, comparing models for task space controller learning

Figure 3.11: Tracking error on base, comparing learning performance when learning the forward model with structured analytical priors and without priors using *joint loss* and *task loss* .

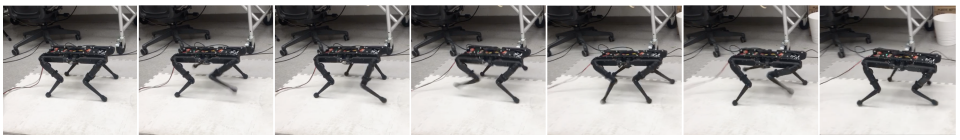


Figure 3.12: solo robot walking for inverse dynamics learning experiments on hardware

convergence speed compared to learning the forward model without structure. It becomes also evident however, that even with a structured prior the *task loss* does not perform well. Meaning that also when only learning the contact dynamics and using the analytical solution for the rest, learning with *task loss* leads to biased solutions. In Fig. 3.11b we show the same comparison but when learning an operational space controller on solo. Here we see that learning with a structured model is essential for successful task performance. We think this is due to the complexity of the task: the structured model alleviates the complexity of forward model learning, significantly allowing for faster convergence.

3.7 Experiments on Hardware

Finally we evaluate our method in the real world on the solo robot. We learn an inverse dynamics controller as presented in section 4.7 and an operational space controller as presented in section 3.5 for walking. The parametrisation of the controllers and the forward model is the same as in simulation. On the hardware we control the robot at a frequency of 1000Hz, however we learn our models at only 250Hz, since in practice learning from the data collected in the real-time loop is

difficult. This means that we only predict, with our learned controller, at 250Hz. To hold the 1000Hz we hold the predicted torque for 4 time steps and compute a feedback PD torque on the desired trajectory for the intermediate steps. The desired trajectories we recover from our simulation experiments and describe the desired joint positions and velocities of the 12 leg joints. We pre-train the controller and forward model in simulation and keep learning them on the hardware.

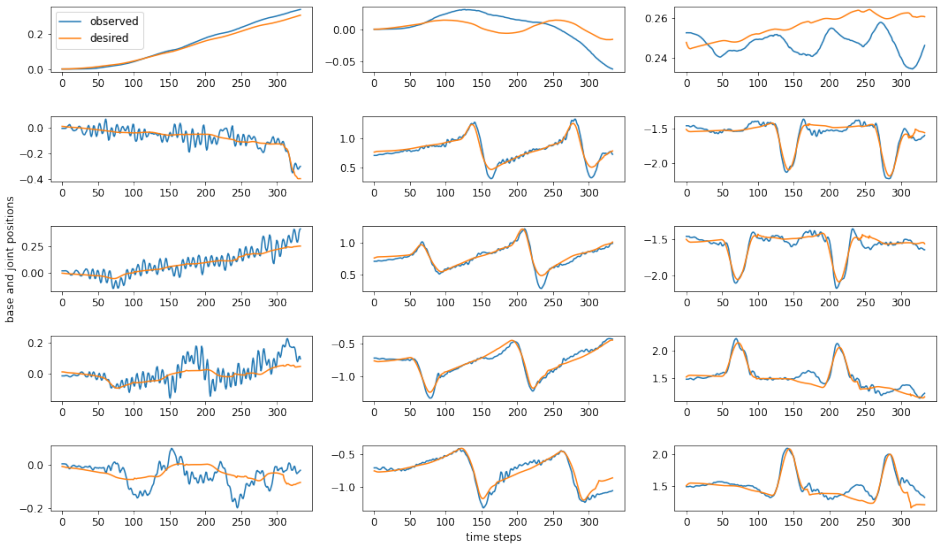


Figure 3.13: Joint position tracking. The first three images show the tracking of the base positions, followed by the 12 leg joints. Overall our learned inverse dynamics controller achieved good tracking performance on hardware.

3.7.1 Results for inverse dynamics model learning on hardware

In this section we present our results for inverse dynamics model learning on hardware. On hardware we only use our *joint loss* since it was the only one that performed well in simulation. In Fig. 3.13 we show the tracking behaviour when running the inverse dynamics controller learned on hardware on the robot. This behaviour is achieved after 4 learning iterations when the model was pre-trained in simulation. Similarly in Fig. 3.14 we show the desired and observed force profiles for each leg of the ground reaction forces. In Table 3.2 we show the errors after 4 iterations for three different walking experiments. Overall we see good results and

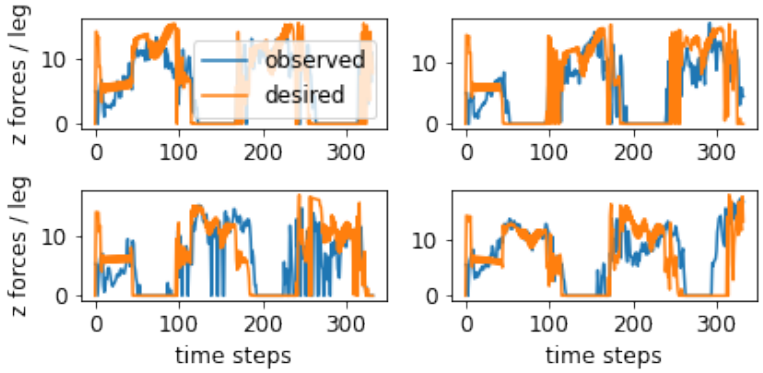


Figure 3.14: Tracking of the forces for learned inverse dynamics controller on the solo robot. The desired and the actual forces are shown.

	Tracking Error Base (m)	**Tracking Error Forces (N)**
Walk 1	0.02	0.07
Walk 2	0.05	0.09
Walk 3	0.03	0.10

Table 3.2: Base tracking error and force tracking error for inverse dynamics learning on hardware.

transferability of our model based learning framework on hardware. A visualization of the walking behaviour of the robot can be seen in Fig. 3.12

3.8 Conclusions

In this work, we show how to leverage forward model prediction error for learning control in an iterative way. Our approach connects controller and forward model learning by using the predicted control signal as an input to the learned forward model. We show how using forward model prediction error during controller learning results in a learned controller that enables the robot to successfully accomplish non-trivial motor control tasks. We present theoretical and empirical evidence that the improved performance is due to an unbiased loss function, that reduces bias in the learning problem. We also show empirical evidence that when using the controller trained with our *joint loss* on the robot, the collected data is more meaningful for the current task and thus improves model learning. This could explain the reduced model bias of the forward model for the learning task. In simulation, our approach systematically outperforms other approaches also for contact rich tasks on underactuated, unstable systems and enables learning controllers in

a few iterations. We also show the applicability of our approach on hardware.

Chapter 4

Multi-Modal Learning of Keypoint Predictive Models for Visual Object Manipulation

4.1 Introduction

In this work, we consider the problem of learning predictive models for visual control of grasped objects. Specifically, we consider the setting of visual model-predictive control for object manipulation as visualized in Figure 4.1. In such settings, a low-dimensional representation of the object is extracted and then a predictive model is learned in that low-dimensional state-representation. This model is then used to optimize action sequences that accomplish a desired (visual) goal state. We built on the recent success of learned keypoint representations which have been shown to capture task-independent visual landmarks for various applications (Minderer *et al.*, 2019; Kulkarni *et al.*, 2019). Benefits of such learned representations over more traditional object representations (such as $6D$ pose), are the ability to represent non-rigid objects, and not requiring object models. In contrast to other learned latent-state representations, keypoints are interpretable, and are less prone to become task-dependent. Given this learned state representation, current state-of-the-art then learns action-conditioned predictive models over keypoints (Minderer *et al.*, 2019; Das *et al.*, 2020b; Manuelli *et al.*, 2020). Once such a dynamics model is trained, the robot can optimize actions to move observed keypoints into a desired goal keypoint configuration.

While this overall framework is very promising, in the context of object manipulation at least two major challenges remain: 1) The self-supervised training of visual keypoints does not necessarily lead to keypoints that capture the object of interest. And even if it does, those keypoints may not consistently track the same part of the object throughout motion sequences; 2) In current state of the art methods (Minderer *et al.*, 2019; Das *et al.*, 2020b; Manuelli *et al.*, 2020), keypoint predictive models are represented by unstructured neural networks, which tend to not extrapolate well to observations outside of the training distribution.

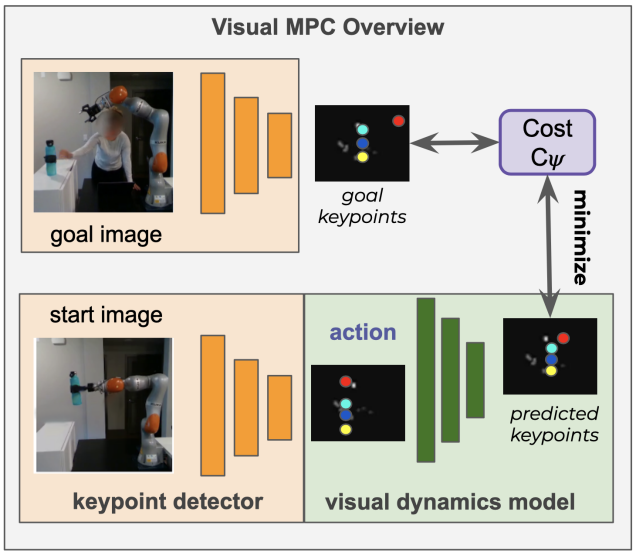


Figure 4.1: Overview of visual MPC via keypoints. Our work improves this framework in two places. (1) we present a multimodal version of the keypoint detector that merges vision and proprioceptive information, and (2) we show how we can use the visual keypoints to learn an extended kinematic chain that serves as reliable visual dynamics model for model based control.

As a result, such predictive models often do not perform well when used for object manipulation tasks.

We address these challenges as follows: First, during keypoint training, we encourage keypoint learning around the end-effector location, which leads to more consistent on-object keypoint detection. We achieve this by merging proprioceptive information, in form of joint positions, with visual information gathered from the camera. Then, instead of learning unstructured predictive models, we extend an existing kinematic model of the robots arm with virtual links and estimate the translation parameters of the links from visual keypoint predictions (see Figure 4.2). This leaves us with an extended kinematic chain or body schema that includes the object in the hand. Once the parameters of the virtual joints have been learned, we have a fully differentiable forward kinematics model that can be utilized for model-based control. A key-feature of this approach is that we can estimate parameters to adapt to various grasp variations of the object. We will show experimental evaluation of these benefits in section 4.7.

To summarize, in this work we propose a fully self-supervised framework to automatically learn extended body schemas from multi-modal data to successfully perform object manipulation tasks. Towards this our contributions are as follows:

1. We propose a multi-modal keypoint learning approach that merges proprioceptive state, RGB and depth measurements to spatially bias the visual

keypoints towards the end-effector. This leads to better on-object keypoint predictions as compared to keypoint methods (Minderer *et al.*, 2019) that only take RGB measurements into account.

2. We use a fully differentiable kinematic representation of the manipulator that we extended with a priori unknown virtual links and joints representing the object. To recover the virtual joints parameters, we propose a gradient based learning approach that learns the parameters given the visual keypoint predictions as targets.

3. We evaluate the learned extended kinematic chain on a downstream object manipulation task and show that our self-supervised approach achieves good performance in simulation and on hardware.

4. We evaluate our multimodal keypoint detector and extended kinematic chain on a 7DoF iiwa Kuka arm that has various objects in hand, in simulation and on hardware.

Our results show that when trained with proprioceptive information, the learned keypoints represent the manipulated objects more reliably. After regressing the virtual joints from the visual keypoint information, we compare the resulting extended kinematic model to dynamics models learned with neural networks, on a model-based control task: placing a grasped object. Our method outperforms the learned dynamics models by an order of magnitude on the downstream task.

4.2 Related Work

4.2.1 Internal Representations of Body in the Brain

It has been well established that distinct representations of the body are created, adapted and utilized by the brain while performing sensorimotor tasks and throughout one's lifetime (See (Hoffmann *et al.*, 2010) for a review). Research in the domain of cognition (Limanowski and Friston, 2020; Van Beers *et al.*, 1999; Sober and Sabes, 2005) has additionally inferred that the "internal state" of a limb is informed by both vision and proprioception, and that both signals are combined in a weighted fashion to update this internal representation, with the weights attributed to each signal depending on their reliability. Several works ((Cardinali *et al.*, 2009, 2012; Baccarini *et al.*, 2014; Martel *et al.*, 2021)) also consider the plasticity or extension of body schemas during (or in anticipation of) tool use in human and primate subjects. (Martel *et al.*, 2021) shows for instance that the use of a 40 cm long tool in a study causes the participants to move as though their arm is longer, indicating a change in their body schema. Taking inspiration from biological cognition, we combine the above notions in our approach. We extend the existing kinematic chain

(analogous to body schema in humans) of our robotic arm to include a grasped object. Furthermore, we fuse signals from proprioception and vision for estimating the state of the extended parts of the chain more accurately. We expand more on related works in robotics in these contexts in sections 4.2.3 and 4.2.4.

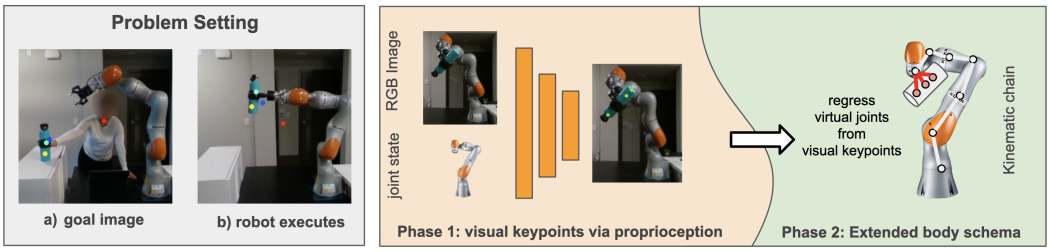


Figure 4.2: Overview of self-supervised learning of extended body schemas. Our approach comprises two phases, (1) in the first phase we learn an autoencoder architecture to detect visual keypoints on the object in the manipulator's hand by merging proprioceptive and visual information. The forward kinematic model of the robot is used, to create a kinematic features map. The visual and the kinematic feature maps are then combined to train the keypoint detector. In the (2) second phase we use the predicted visual keypoints, that ideally are detected on the object, to learn an extended kinematic chain of the manipulator that inherently includes the object in the robot's hand. The extended kinematic chain is then used to accomplish a manipulation task. Learning happens fully from visual and proprioceptive information and can adapt to different rigid bodies and grasps.

4.2.2 Learning Visual Representations for Model Predictive Control (MPC)

Since the core emphasis of the article is learning representations for robotic object manipulation, we contrast our framework to other relevant approaches with respect to latent space representation, dynamics model learning and action optimization. Approaches that employ model-predictive control in visual space can be distinguished by how much, and what kind of structure is infused into the predictive model. Approaches such as (Ebert *et al.*, 2018), utilize no structure and learn a predictive model f directly in pixel space. Both, (Watter *et al.*, 2015) and (Byravan *et al.*, 2018) learn a latent representations z and a predictive model f in the latent-space. While (Watter *et al.*, 2015) assumes no structure for learning z, they assume locally linear dynamics for learning the latent space transition model f. (Byravan *et al.*, 2018) on the other hands learns a structured SE3 latent representation that is more interpretable, and learns unstructured predictive models f. Our method builds upon an self-supervised keypoint representation learning approach in (Minderer *et al.*, 2019; Kulkarni *et al.*, 2019), which (Das *et al.*, 2020b; Lambeta *et al.*, 2020) utilize for object manipulation tasks. It relies on an auto-encoding

scheme with a structural bottleneck that can extract 2D keypoints representing the object(s) of interest. However, as observed by (Das *et al.*, 2020b; Lambeta *et al.*, 2020). the keypoints encoded by this method are not always consistently on the object of interest. We illustrate this further in our experiments that compare this keypoint detector (Minderer *et al.*, 2019) with our multimodal keypoint detector.

(Manuelli *et al.*, 2020) is another recent work that takes a structured approach towards learning keypoints and utilizes them for model-predictive control. The keypoints learned are, by design, always located on the object and are consistent across frames. This is achieved by first learning a dense visual object descriptor (Florence *et al.*, 2020) for the object of interest. which has two requirements: (a) pixel to pixel correspondences between image sets, and (b) an object mask learned by utilizing results from (Finman *et al.*, 2013). Following the training of a dense visual descriptor, k descriptors are sampled as keypoints. An unstructured predictive model f is then learned in keypoint space (Manuelli *et al.*, 2020). In contrast to this approach for keypoint detection, our multimodal detector does not need pixel-wise corresponding images for supervision or an object mask, while still being able to produce keypoints that are reliably present on an object (4.2.3).

The above mentioned approaches, also differ in how action sequences $\boldsymbol{u}$ are optimized: via the cross-entropy method CEM (Rubinstein and Kroese, 2004; Ebert *et al.*, 2018; Lambeta *et al.*, 2020; Manuelli *et al.*, 2020), gradient based optimization (Byravan *et al.*, 2018; Das *et al.*, 2020b), Model Predictive Path Integrals (Manuelli *et al.*, 2020) or by using optimal control methods (Watter *et al.*, 2015). Our framework is agnostic to the specific action optimization method. We present results using gradient based action optimization.

4.2.3 Multi-Modal Learning: Fusing Vision and Proprioception

In the previous subsection we discussed model-predictive control that utilize visual data only. However, there is more and more evidence that utilizing multi-modal sensor streams improves perception and manipulation (Bohg *et al.*, 2017; Edelman, 1987; Lacey and Sathian, 2016). This has been explored in various robotics applications. For instance, fusing vision and proprioception (Garcia Cifuentes *et al.*, 2017; Kappler *et al.*, 2018; Martín-Martín and Brock, 2017) or combining visual and tactile information (Martín-Martín and Brock, 2017; Lambert *et al.*, 2019; Yu and Rodriguez, 2018) has been explored for better state-estimation in applications such as object-tracking. These approaches are mostly concerned with understanding state-estimation from multi-modal sensor data and assume expert-designed low-dimensional features are extracted from each modality. Manually designing features for heterogeneous data is extremely challenging, time-consuming and thus not scalable. Because of this reason, there has been a recent push towards learning

state representations from multi-modal data for a wide range of applications. For example, there have been many works that have explored the correlation between auditory and visual data for tasks such as speech or material recognition or for sound source localization (Ngiam *et al.*, 2011; Owens and Efros, 2018; Owens *et al.*, 2016; Yang *et al.*, 2017). Several work (Bekiroglu *et al.*, 2011; Calandra *et al.*, 2018; Gao *et al.*, 2016; Sinapov *et al.*, 2014) fuse visual and haptic data for various applications such as grasp stability assessment, manipulation, material recognition, or object categorization.

As discussed above, fusing multiple sensor modalities for better state-estimation is common in the rigid-body tracking literature, and recently it has also been shown by (Lee *et al.*, 2019) that it leads to better learned state representations for robotic manipulation tasks. In this work we show how fusing proprioception and vision (rgb images and depth) significantly improves the learning of keypoint representations.

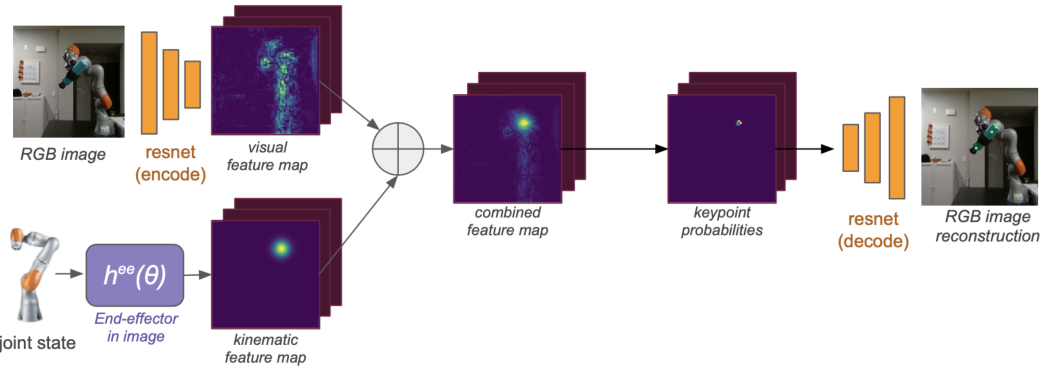


Figure 4.3: Phase 1: Multimodal keypoint detector, fusing vision and proprioception to predict visual keypoints. The forward kinematic model of the robot is used, to create a kinematic features map. The visual and the kinematic feature maps are then combined to train the keypoint detector.

4.2.4 Learning body schemas

A promising alternative to learning unstructured predictive models are approaches that use structure for body schema learning. A body schema (Hoffmann *et al.*, 2010) is a representation of a robot's body and its extensions such as grasped objects or tools, which can then be used for control. Body schema approaches can be classified into approaches that estimate parameters of structured representations of a body and tool (i.e kinematic representation) (Sturm *et al.*, 2009; Martinez-Cantin *et al.*, 2010; Ulbrich *et al.*, 2009; Hersch *et al.*, 2008; Gothoskar *et al.*, 2020; Stepanova *et al.*, 2019), or of unstructured models, such as neural network representations (Boots *et al.*, 2014; Hikita *et al.*, 2008; Rolf *et al.*, 2010; Nabeshima *et al.*, 2006; Yoshikawa *et al.*, 2003; Schillaci *et al.*, 2012). To address the challenge of learning

models that generalize to differet parts of the state space, we follow approaches that learn parametrized kinematic models. Prior work in this category typically utilizes markers or ground truth knowledge about the end-effector or tool-tip location in the robots workspace (Sturm *et al.*, 2009; Martinez-Cantin *et al.*, 2010; Ulbrich *et al.*, 2009; Gothoskar *et al.*, 2020) or simplified visual signals (Hersch *et al.*, 2008), and instead focus on learning kinematic parameters. In contrast to this, our work extends an existing kinematic chain to include a grasped object purely from learned visual latent representations in a self supervised way.

4.3 Problem Setting and Method Overview

In this work, we address deterministic, fixed-horizon and discrete-time control problems with continuous states $\mathbf{s} = (s_1, \ldots, s_T)$ and continuous actions $\mathbf{u} = (u_1, ..., u_T)$. Each state $s_t = [\theta_t, z_t]$ is the concatenation of the measured joint angles θ_t and a learned visual latent state z_t at time step t. To solve the control problem we use a learned visual predictive dynamics model $\hat{s}_{t+1} = f(s_t, u_t)$ and a cost function $C(z_t, z_{\text{goal}})$ that measures the distance between current and desired goal state in the visual latent space.

The learned predictive model f predicts the change in object state, as perceived from the camera, given the current joint displacements. f is learned in a self supervised fashion by merging proprioceptive and visual information and we do not assume any additional model knowledge of the external object.

Learning f involves two phases. In the first phase a keypoint detector is trained to detect keypoints z on the object in the manipulator's hand from the visual data. The keypoint detector follows an autoencoder architecture and we present a novel approach to merge visual and proprioceptive information for keypoint detector training, when encoding the visual information. In the second phase of our approach, we use the learned keypoint detector to learn the parameters of an extended kinematic chain of the robot. The extended kinematic chain, extends the standard kinematics of the robot arm to include also the object in the manipulator's hand. The extended kinematic chain can then be used to optimize a control policy for a manipulation task. The control tasks are characterized by a desired goal state for an object in the manipulators hand. The goal state is provided in the visual latent space i.e. the visual keypoits. An overview of our self-supervised learning approach can be seen in Fig. 4.2.

4.4 Phase 1: Fusing Proprioception and Vision for Multi-Modal Keypoint Learning

In this section we introduce a novel multimodal keypoint learning framework that leverages RGB, depth and proprioceptive measurements for improved keypoint training and prediction. Intuitively, proprioception is the sense of self movement and body position. In the context of this work we use robot joint positions and forward kinematics as proprioceptive information during learning.

4.4.1 Learning Visual Keypoints for Object Manipulation

We base our multi-modal keypoint learning framework on the visual keypoint detector presented in (Minderer *et al.*, 2019). To learn keypoints (Minderer *et al.*, 2019) uses an autoencoder with a structural bottleneck to detect 2D visual keypoints that correspond to pixel positions with maximum variability in the input data. For keypoint prediction (Minderer *et al.*, 2019) extract K 2-D visual feature maps o_k^{visual} through a mini-RESNET 18, where K is the number of keypoints. The autoencoder architecture is trained with RGB image sequences collected from videos. The maximum variability can intuitively be thought of as areas of biggest movement in the image. The autoencoder architecture is then trained with a combination of losses. The loss is composed of a reconstruction error ($\mathcal{L}_{\text{rec}}$), a keypoint sparsity error ($\mathcal{L}_{\text{spa}}$), and a keypoint separation loss ($\mathcal{L}_{\text{sep}}$), that encourages the separation of keypoints in pixel-space.

$$\mathcal{L}_{\text{(Minderer } et\ al.\text{, 2019)}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{spa}} \cdot \mathcal{L}_{\text{spa}} + \lambda_{\text{sep}} \cdot \mathcal{L}_{\text{sep}} \tag{4.1}$$

where λ_{spa} and λ_{sep} represent the scale parameters for various losses.

In the next sections we are going to present how we extend the keypoint detector by (Minderer *et al.*, 2019) to a multimodal keypoint detector that merges vision and proprioception.

4.4.2 Fusing visual and kinematic feature maps

We now present how we include proprioceptive information when training our multimodal keypoint detector. Our method extends the one presented by (Minderer *et al.*, 2019) in two places. We (1) create, additionally to the visual feature map, a kinematic feature map, representing the end effector position of the robot in image space and (2) we extend the training loss with a kinematic consistency loss. In the following we are going to explain both extensions in detail.

To create the kinematic feature map we specify $s_n^{\text{img}} = h^n(\theta)$ as a forward kinematic function, that directly projects the 3D position of link n into image space

when the robot is in configuration θ. To retrieve s_n^{img}, first, the robot's forward kinematic model is used to deliver x_n, the 3D location of link n in the robot's coordinate frame. Then x_n is projected into image and depth space. The full transformation is given by

$$s_n^{\text{img,depth}} = h^n(\theta) = T_{\text{proj}} T_{\text{cam}} x_n \text{ where } x_n = \prod_{i=1}^{n} T_i(\theta_i; \phi_i)$$

where T_i is the transformation matrix from the coordinate frame of link i to link $i-1$ and ϕ_i are the parameters (rotation and translation) of link i; T_{cam} transforms x_n from robot to camera coordinate frame, and T_{proj} performs the projection to image and depth space. In the following, we will use $s_n^{\text{img, depth}}$ to denote the use of both image and depth value, and s_n^{img} when we only use image pixel predictions. In the following, we assume that the parameters ϕ_i up to the end-effector index $n = \text{ee}$ are known, and we will use $h^{\text{ee}}(\theta)$ to combine visual and proprioceptive information to train the keypoint detector.

In order to spatially bias keypoint learning to be close to end-effector location, we propose to include proprioceptive information during keypoint detector training in the encoding phase. We do this by computing a second feature map, which we call the kinematic feature-map o^{kin}, visible in Fig. 4.3. The kinematic feature-map is generated by first computing the end-effector position in image space, $s_{\text{ee}}^{\text{img}} = h^{\text{ee}}(\theta)$, as described in section 4.4.2, followed by placing a Gaussian blob g^{kin} over the location of the projected end-effector position in image space:

$$o^{\text{kin}} = g^{\text{kin}}(s_{\text{ee}}^{\text{img}})$$

where g^{kin} converts the pixel coordinates of $s_{\text{ee}}^{\text{img}}$ into a heatmap with a Gaussian shaped blob $\mathcal{N}(s_{\text{ee}}^{\text{img}}, \Sigma)$ centered at x, y pixel locations of $s_{\text{ee}}^{\text{img}}$.

We then combine the kinematic with the visual feature map, creating a joint feature-map $o^{\text{joint}} = o^{\text{visual}} + o^{\text{kin}}$, which fuses visual and proprioceptive information. Given o^{joint}, keypoints are trained through a reconstruction objective, similar to the method presented in (Minderer *et al.*, 2019). Figure 4.3 provides an overview of our adapted encoder-decoder architecture.

4.4.3 Utilizing proprioception and depth to augment loss

We extend the loss proposed in (Minderer *et al.*, 2019) $\mathcal{L}_{\text{(Minderer et al., 2019)}}$ by including a term that penalizes the distance between keypoints and end-effector. This *kinematic consistency loss*, together with the kinematic information during feature map creation, spatially biases the keypoint learning towards the end-effector location to incentivize the detector to place keypoints on the object. The kinematic

consistency loss, is given by

$$\mathcal{L}_{\text{kin}} = \sum_{k} (z_k^{[x,y,depth]} - s_{\text{ee}}^{\text{img,depth}})^2 \tag{4.2}$$

where $z_k^{[x,y,depth]}$ is the x, y pixel locations and depth value of the predicted keypoint z_k. The depth value of z_k is retrieved, by querying the depth image at pixel locations x, y.

The complete loss that is minimized during Phase I is thus given by:

$$\mathcal{L} = \mathcal{L}_{\text{(Minderer et al., 2019)}} + \lambda_{\text{kin}} \cdot \mathcal{L}_{\text{kin}} \tag{4.3}$$

where λ_{kin} again is a scaling parameter for kinematic consistency loss.

To train our keypoint detector we collect visual and proprioceptive data $\mathcal{D}_{\text{key-train}}$ for self-supervised keypoint training. After this training phase, we have a keypoint detector that predicts keypoints z of dimensionality $K \times 3$. Here K is the number of keypoints, and each keypoint is given by $z_k = (z_k^x, z_k^y, z_k^{\text{depth}})$, where z_k^x, z_k^y are pixel locations of the $k-$th keypoint, and z_k^{depth} is the value of the depth image at location z_k^x, z_k^y.

4.4.4 Inference at test time

It is important to note that the proprioceptive information is only added during keypoint detector training. Also during training, we do not need to necessarily include proprioceptive information for each datapoint, but can also train from mixed datasets that contain only visual information and visual and proprioceptive information. After training, the proprioceptive information is no longer needed for keypoint prediction at test time since the proprioceptive information is now inherently part of the leared keypoint detector. The keypoint detector can be used with purely visual input. This allows extraction of keypoints from a goal image that is produced, for instance, by recording human demonstrations. At inference time, the keypoint detector predicts the x, y locations of the keypoint z_k, the depth value is then retrieved by querying the depth image at pixel location x, y.

4.5 Phase 2: Learning Body Schema Extension from Visual Keypoints

The multimodal keypoint detector presented in section 4.4 merges visual and proprioceptive information during training, facilitating better on-object keypoint predictions during test time. Since the keypoints predicted by the detector are more likely placed on the object in the manipulator's hand, they will enable us to learn

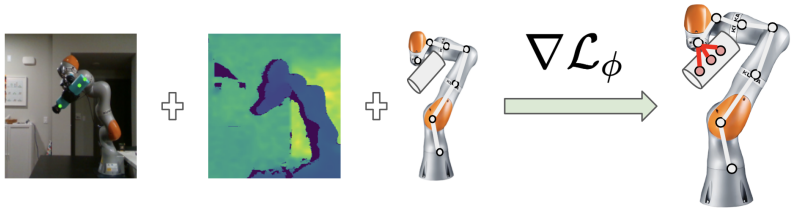


Figure 4.4: Phase 2: The translation parameters of the virtual joints representing the object in the manipulator's hand (red dots) are estimated from the x,y pixel location of the visual keypoints together with the depth information of the pixels. The virtual joints extend the kinematic chain of the robot to also include the object, as is illustrated in the figure.

an extended kinematic chain of the robot arm that includes the object in the manipulator's hand. The extended kinematic chain can then be used for model-based control on a downstream manipulation task. In this section we are going to explain how we learn the extended kinematic chain and how it can be used for a downstream manipulation task.

As already pointed out in section 4.2, an object or a tool in the gripper of a manipulator can be seen as an extension of the kinematic chain of the robot. In Fig. 4.4, we visually show how the extension of the kinematic chain looks like for the kuka manipulator: additional virtual links and joints are added, representing the object in the manipulator's hand, to create a full chain that includes the object. The extended kinematics of the robot can then be controlled.

In order to learn this extended body schema from vision we use the keypoint predictions of the multimodal keypoint detector presented in section 4.4. The predicted keypoints on the object are then used to regress the kinematic parameters of the extended body schema.

4.5.1 Estimating virtual joints from visual keypoints

During the second phase of our approach, the extended kinematic chain of the manipulator should be learned to include the object in the manipulator's hand. The self-supervised nature of our approach allows the kinematic chain to adapt when the object changes or the grasp around the object shifts, making the learning of the extended kinematic chain flexible to changes in the experimental setup. Once the keypoint detector has been trained, we use its predictions to learn an extension of the kinematic chain to include the object into the body schema. We extend the kinematics model from Section 4.4.2, to include virtual joints, one per visual keypoint, such that $s_{\mathrm{k}}^{\mathrm{img,depth}} = h_\phi^k(\theta)$ is the projection of the virtual joint in image space. Here, ϕ are the learnable translation parameters of the virtual joints, that we aim to learn such that they represent the object (see Fig. 4.2 and Fig. 4.4, where

the red circles represent the virtual joints with the virtual links connecting them to the robot end effector, we assume the extension of the kinematic chain starts at the end effector). To regress ϕ, a dataset $\mathcal{D} = \{(x_t = \theta_t, y_t = z_t)\}_{t=1}^{T}$ is collected where the trained keypoint detector is used to predict visual keypoints $\mathbf{z}^{[x,y,depth]}$ on images of the object in the robot's hand while being in joint configuration θ. The goal of this learning problem is to regress the translations ϕ, such that the output of $h_\phi^k(\theta)$ matches the keypoints detected by the detector. In other words, to optimize ϕ we want to minimize the loss $\mathcal{L}_{\text{trans}}$ via gradient descent.

$$\mathcal{L}_{\text{trans}} = (\mathbf{s}_{\text{k}}^{\text{img,depth}} - \mathbf{z}_{\text{k}}^{[x,y,depth]})^2 \tag{4.4}$$

where $\mathbf{s}_{\text{k}}^{\text{img,depth}} = h_\phi^k(\theta)$. This loss is a function of the learnable kinematic parameters ϕ, making it possible to update ϕ via gradient descent. Learning the translation parameters of the virtual joints, results in a new extended kinematic model, which includes the object. This new kinematic chain does not require visual information anymore and can be used for action optimization. As will be presented in section 4.7 using our version of the keypoint detector, that merges visual and proprioceptive information provides more accurate on-object keypoint predictions and thus makes learning the extended kinematic chain possible.

4.6 Gradient-Based Control for Object Manipulation

The goal is to successfully perform a manipulation task in the visual domain. This means that the desired goal position of the object in the manipulator's hand is given in image space instead of in joint space. Specifying a desired goal position in image space is more intuitive compared to specifying it in joint positions, where the relationship between joint position and real world object position is not readily available. In contrast to other visual MPC work (Ebert *et al.*, 2018; Byravan *et al.*, 2018; Das *et al.*, 2020b), that utilize learned visual dynamics models to optimize actions $\mathbf{u}$, we make use of our learned extended kinematic chain. Specifically, we define a visual predictive model $s_{t+1} = f(s_t, u_t)$ where $s_{t+1} = [\theta_{t+1}, z_{t+1}]$, and

$$\theta_{t+1} = \theta_t + u_t \tag{4.5}$$

$$z_{t+1} = h_\phi^k(\theta_{t+1}) \tag{4.6}$$

where actions $\mathbf{u_t}$ are desired changes in joint positions and h is, as defined in section 4.4, the forward kinematic call of the learned extended kinematic model projected in image space. To optimize actions, we follow the gradient based action optimization presented in (Byravan *et al.*, 2018; Das *et al.*, 2020b) and minimize a task specific

cost function C. Specifically, to optimize a sequence of action parameters $\mathbf{u} = (u_0, u_1, \ldots, u_T)$ for a horizon of T time steps, we first predict the trajectory $\hat{\tau}$, that is created using an initial $\mathbf{u}$ from starting configuration s_0: $\hat{s}_1 = f(s_0, u_0)$, $\hat{s}_2 = f(s_1, u_1), \ldots$ $\hat{s}_T = f(s_{t-1}, u_{t-1})$, which generates a predicted (or planned) trajectory $\hat{\tau}$.

Practically, this step uses the extended kinematics model $h^n_\phi(\theta)$ to simulate forward what would happen if we applied action sequence $\mathbf{u}$ to the initial state s_0. We then measure the cost achieved $C(\hat{\tau}, z_{\text{goal}})$, where z_{goal} is a goal location in the visual domain. Since h is differentiable, the cost of the planned trajectory can be minimized via gradient descent by taking the gradient of the cost with respect to the action sequence and performing a gradient update step.

$$\mathbf{u}_{\text{new}} = \mathbf{u} - \eta \nabla_u C(\hat{\tau}, z_{\text{goal}}) \tag{4.7}$$

Algorithm 5 shows the details of our visual MPC algorithm.

Algorithm 5 Gradient Based Control

$f(s_t, u_t) = [\theta_t + u_t, h^k_\phi(\theta_{t+1})]$
initial state $s_0 = [\theta_0, z_0]$
for each *epoch* **do**
    $u_t = 0, \forall t = 1, \ldots, T$
    // rollout $\hat{\tau}$ from initial state s_0 and actions u
    $\hat{\tau} \leftarrow \text{rollout}(s_0, u, f)$
    // Gradient descent on u
    $u_{new} \leftarrow u - \eta \nabla_u C(\hat{\tau}, z_{goal})$
end for

4.7 Experiments: Self-Supervised Learning of Body Schemas

In this section, we present the experimental evaluation of our approach. We train two versions of the keypoint detector: our multimodal keypoint detector, merging vision and proprioception, following the approach presented in section 5.3 and the detector presented in (Manuelli *et al.*, 2019),(Minderer *et al.*, 2019), as a baseline comparison. All of our experiments are performed on a 7 DoF iiwa Kuka arm (AG, 2020), with an object attached to the Kuka's gripper. We present experiments in simulation and on hardware. For our simulation experiments, we use the Habitat simulator (Manolis Savva* *et al.*, 2019) with a pybullet integration (Pybullet, 2012). In simulation, we use three different objects to show the adaptability of our approach. The experimental setup together with the objects used, is shown in Fig. 4.5 for simulation and hardware.

4.7.1 Data Collection

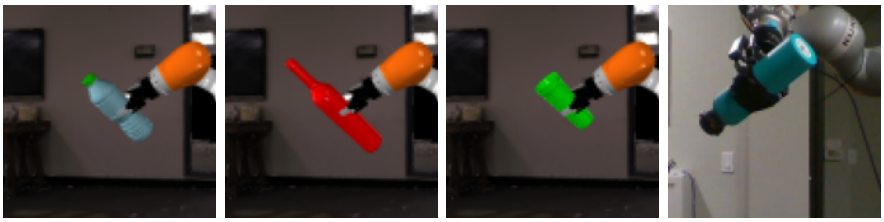


Figure 4.5: The 3 objects used for simulation experiments, and one object on hardware.

The keypoint architecture as presented in (Minderer *et al.*, 2019) is built to learn keypoints that capture motion in video-sequences. Because we aim to learn keypoints on the object, we collect data that only has object motion, similar to (Das *et al.*, 2020b). Specifically, we first move the manipulator to a random joint configuration, then we keep the manipulator in its position and only move the end-effector for 6 seconds. In total we move the robot to 60 initial random joint configurations, and from there collect data at a frequency of 5Hz for a total of 5 seconds, for each object. We collect image data, which includes RGB and depth data, alongside with proprioceptive information about the joint positions. Data collection on hardware follows the same procedure as presented in (Das *et al.*, 2020b): we collect 50 sequences of motion data in which only the end-effector and object move, starting from a random joint configuration. Each sequence is 3 seconds long, and contains 10 data frames. In simulation we perform this data collection for 3 objects and for 1 object on hardware, as shown in Figure 4.5.

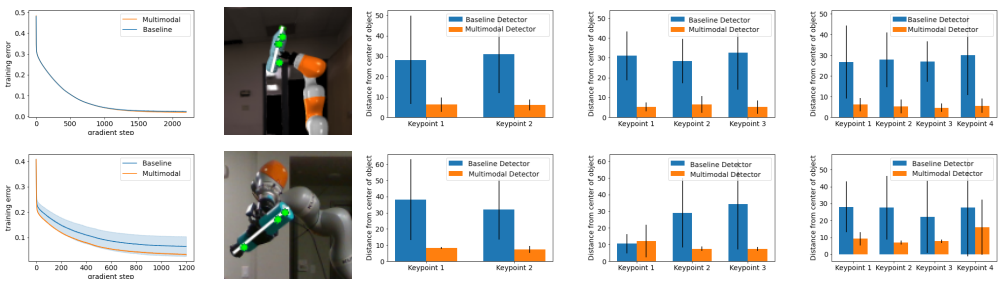


Figure 4.6: **(top)** simulation results; **(bottom)** hardware results. **(1st col)** training loss for detector training, both detectors converge during training. **(2nd col)** illustration of evaluation metric. **(3rd to 5th col)** Results for training keypoint detectors with $K = 2, 3, 4$ keypoints, respectively. The bar plots show the average distance from the center line for each keypoint in pixel space. The multimodal keypoint detector (orange) predicts keypoints that are closer to the center line, and the average pixel distance suggests that they are placed on the object.

4.7.2 Does proprioception help train better keypoints?

In this section we want to analyse how merging proprioceptive information and visual information benefits on-object keypoint detection. For this purpose we train two kinds of keypoint detectors, one that includes proprioceptive and visual information which we call the multimodal keypoint detector and one that only uses visual information which is our baseline and equivalent to the keypoint detector presented in (Minderer *et al.*, 2019). After training, we compare the performance of the multimodal and baseline keypoint detectors of achieving on-object keypoint detection. For our simulation experiments, we present results averaged over five seeds and 3 different objects, where we train a detector per object. On hardware, the experiments are also averaged over five seeds and we present results for two to four keypoints on a single object. First, we show the training loss curves in Figure 4.6 (left), averaged across seeds, objects and number of keypoints. In simulation, the loss curves are very similar, we believe that this is due to the reconstruction loss being the biggest component of the total training loss. Since only the gripper and object moves, the reconstruction for most of the image works well, reducing the loss and resulting in low training error even if the keypoints are not placed on the object.

Next, we evaluate how well the learned keypoints capture moving objects. To this end, we define an imaginary line, that cuts through the object in the middle and connects its extremities. A visualization of it is visible in Fig. 4.6 (2nd column). Computing the shortest distance of a keypoint $z_k = (x_k, y_k)$ to this line, will give us a good intuition of whether this keypoint lies on the object or not. We perform this evaluation, for training detectors with $K = 2, 3, 4$ number of keypoints. We evaluate the detectors on a test dataset with 250 datapoints, that was held out during training. The distance is computed in pixel space and we report the mean and the standard deviation of that distance.

In Figure 4.6, we show the averaged distance of each trained keypoint to the line, with error bars, after training. Results for the simulation experiments (top row) and hardware experiments (bottom row) are shown. The bar plots illustrate the distance of the detected keypoints to the center line. We see that the multimodel keypoint detector, that was trained with proprioceptive information, outperforms the baseline detector. The low distance to the center line suggests that the multimodal detector places the keypoints on the object. This is confirmed by the qualitative analysis in Fig. 4.7 where three examples are shown. The multimodal detector detects keypoints that are on the object (top row). This is not true for the baseline detector, which often fails at detecting keypoints that lie on the object (bottom row).

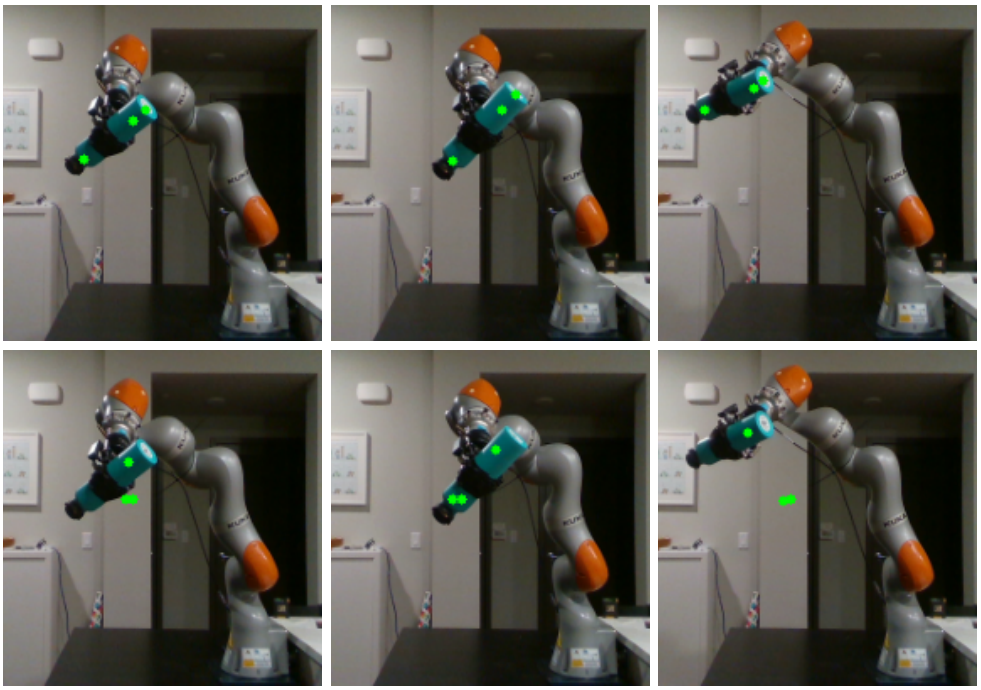


Figure 4.7: Qualitative performance of multimodal (top) and baseline (bottom) keypoint detector on the hardware dataset. The multimodal keypoint detector, always detect keypoints on the object, this is not the case for the baseline detector.

4.7.3 How much proprioception is needed?

In this experiment, we want to evaluate how combining multi-modal observations, that contain both visual and proprioceptive measurements, with visual only observations affects results. This experiment will give us insights into whether a robot could combine visual observations obtained in a passive way (watching humans in action or videos) with observations gained in an active way (by moving the object itself) to learn keypoint detectors. To perform this experiment, we pretend that for a fraction of observations, the proprioceptive state measurement was not observed, and evaluate how stable keypoint training is as a function of that fraction. As can be seen in Fig. 4.8 while we can achieve already good performance when only including 50% of the proprioceptive information during training, the consistency of on-object keypoint detection significantly improves when we include all the proprioceptive information, as can be seen from the smaller standard deviation in the plot. In the plot we can also see that including or not including proprioceptive information during test time has no effect on the performance of detecting keypoints on the object. During test time the keypoint detector can be used only from visual information. This means that our multimodal keypoint detector learned to

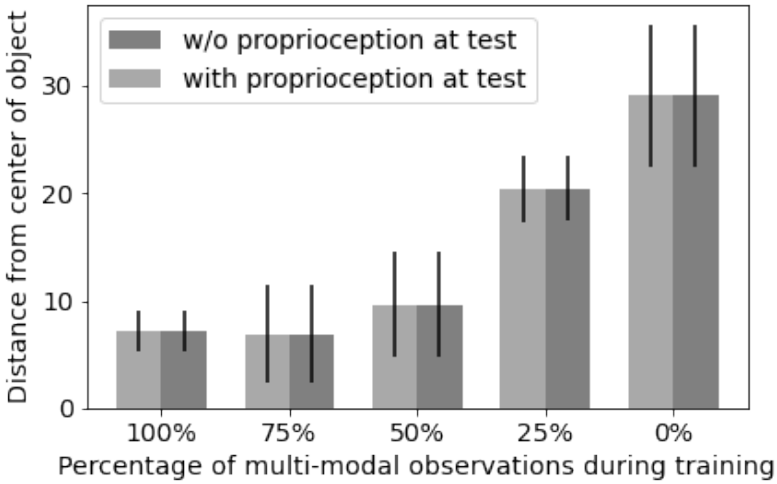


Figure 4.8: Share of multimodel data (including proprioceptive information) in the dataset used to train the keypoint detector, we compare the performance of the keypoint detectors during test time when including or excluding proprioceptive information. We see that using multimodal data during training shows a significant performance improvement, however during test time the proprioceptive information has no effect on on-object keypoint detection performance.

incorporate the proprioceptive information and has inherently access to it when predicting, even when the proprioceptive information is not available.

4.7.4 Virtual link/joint regression

In this section we present the experiments showing how we can learn an extended body schema from vision. The purpose of these experiments is to show that we can use our learned visual latent space to regress the translation parameters of some virtual joints that extend the kinematic chain of the robot to also include an object in the manipulator's hand. For this to be successful it is a prerequisite that the visual keypoints are detected on the object, since otherwise the extended kinematic chain would not represent the object in the robot's hand. As we already evaluated in the previous section only our multimodal keypoint detector successfully detects keypoints on the object, therefore we are using our multimodal keypoint detector for these experiments.

We perform two set of experiments for the regression of virtual joint parameters: 1) to evaluate our virtual joint estimation procedure quantitatively, we test whether we can identify ground truth parameters $\bar{\phi}$, if we knew them; 2) We evaluate regressing virtual links from visual keypoints detected on 4 set of grasps.

To collect data for the first experiment, we pick a set of three ground truth virtual joint parameters $\bar{\phi}$ such that their projections $h^k_{\bar{\phi}}$ lie on the object. We collect a dataset $\mathcal{D} = \{(\theta_t, h^k_{\bar{\phi}}(\theta_t)\}_{t=1}^{15}$ for 15 random joint configurations θ. Then, we pretend to not know $\bar{\phi}$, initialize ϕ to be zero, and aim to estimate ϕ from the observations in $\mathcal{D}$ via gradient descent, as described in Section 4.5.1. In Fig. 4.9, we show the MSE error between ground truth and estimated parameters $|||\phi - \bar{\phi}|||^2$ as a function of number of gradient steps, both from simulation and hardware data. We observe that after approximately 50 gradient steps ϕ converges towards their ground truth values, showing that the regression of virtual joint parameters from projected image and depth values is successful.

For our second set of experiments, we use the trained keypoint detectors to create a dataset $\mathcal{D} = \{(\theta_t, z_t)\}_{t=1}^{15}$, where z_t are the keypoints predicted by the detector, and learn virtual joints from these visual keypoint observations. The first image of Fig. 4.10 shows the visual keypoints predicted by the multimodal detector (in green), together with the projected virtual links h^k_{ϕ} (in red) after learning ϕ. The following two images show how the projection of the kinematic chain remains consistent, also when the pose θ of the manipulator changes. Results are presented for simulation (top) and on hardware (bottom) experiments. The results show that we can successfully learn ϕ from visual features.

Finally, we also evaluate our kinematic parameter regression when the robot re-grasps the object. Re-grasping changes the kinematic chain, by for example shifting it. In Fig. 4.11, we show how our method can successfully learn kinematic

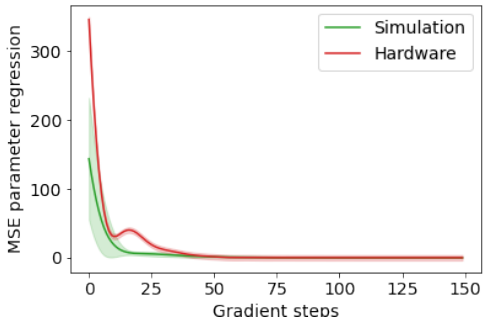


Figure 4.9: Regression of kinematic parameters, MSE to ground truth virtual joint values over gradient steps. Our method is able to regress the virtual joint parameters successfully, on simulation and on hardware. Simulation results are averaged over five seeds and three object with manually chosen ground truth parameters for each object. Hardware results are averaged over five seeds.

parameters ϕ that reflect this change. We show results for three new grasps. In the next section we present results for a downstream placing task, where our learned extended kinematic model is used, with different grasps, to place the object on a table.

4.8 Down-stream Task Experiments: Model-based Control for Object Manipulation

Finally, we want to use our approach of learning extended body schemas from multimodal data on a object manipulation task, specifically a placing task (see Figure 4.13). With these experiments we want to show the practical applicability of our approach on a robotic task, and present results in simulation as well as on hardware. We compare our approach to various baselines.

During the placing task, the robot is required to manipulate the object in the gripper such that it successfully places it on a table. The task is defined in the visual domain, with a desired goal image and associated keypoint positions in pixel. The optimization also takes place in keypoint-space.

Performing a motor control task, such as a placing task, in a model based fashion, requires optimizing controls using a model of the robot and the task. In the case of the placing task the model needs to capture information about the arm movement, when an action is applied, and the corresponding movement of the object in the manipulator's hand. The actions are optimized for a time-horizon of $T = 10$ time

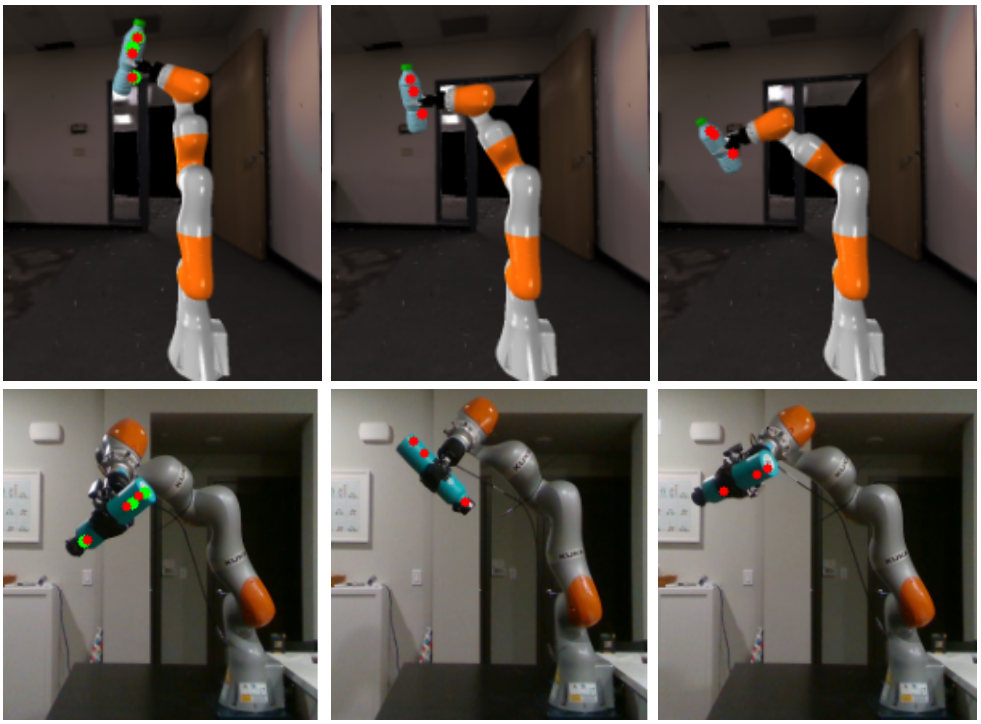


Figure 4.10: Learned kinematic extension for three poses for simulation (top) and hardware (bottom). The first images shows, in green, the keypoints detected by the keypoint detector. The red dots are the projections in image given the learned virtual joints. Our method is able to successfully learn parameters ϕ, that generalize across poses.

steps, with the gradient based approach introduced in section 4.6. We define a cost function, which penalizes the distance between predicted and goal keypoint locations. We compare our learned extended kinematic chain to various baselines that have previously been presented in (Das *et al.*, 2020b; Manuelli *et al.*, 2020). The baselines also use visual keypoint detectors for the latent visual representation z but, instead of learning an extended kinematic chain, learn a neural network black-box dynamics model g_β, $s_{t+1} = g_\beta(s_t, u_t)$ that maps the current state s_t and action u_t to the next state s_{t+1}. As previously $s_t = [\theta_t, z_t]$.

We show results for experiments in simulation and on hardware. Our approach learns an extended kinematic chain from the multimodal visual keypoints detector, that uses proprioceptive information during training (see section 5.3). The virtual joints, that represent the object, are regressed from a few measurements, as we described previously. Specifically we used three keypoints and thus, regress the parameters of three virtual joints. We compare our approach to the following four baselines that learn a neural network dynamics model in the keypoint latent space:

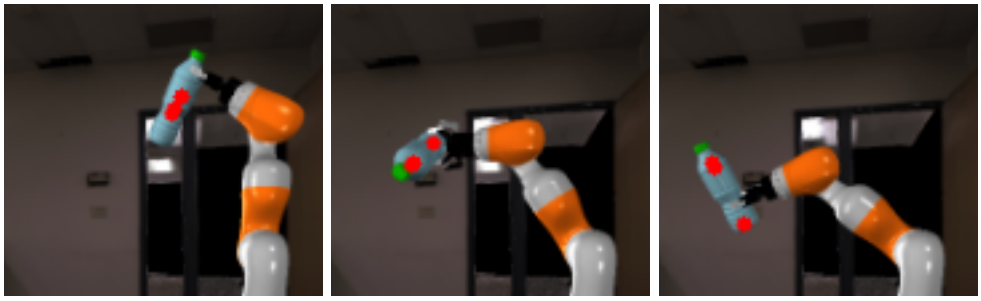


Figure 4.11: Re-grasping for three different grasps. After each new grasp we regress the kinematic parameters successfully.

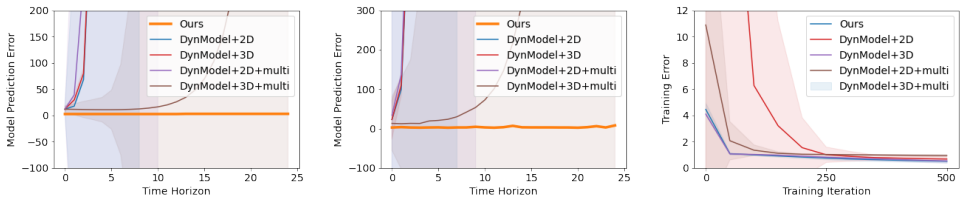


Figure 4.12: On the two plots on the left the long horizon prediction performance of our approach compared with neural networks dynamics models is shown. The plot on the left shows the error when using the action sequence that performs a placing task, the plot in the middle shows the prediction error for a random action sequence. The prediction error of the neural network increases exponentially over time or is high to begin with, while the error for our method stays small. The plot on the right shows the training error of the neural network models, to show that they were trained until convergence. We report the mean and the standard deviation of the error.

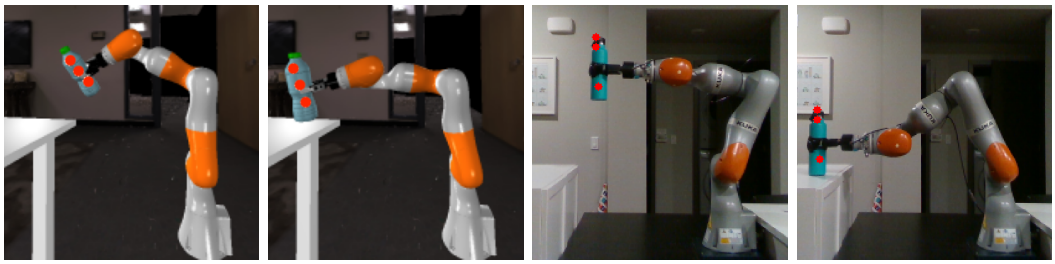


Figure 4.13: Placing task in simulation and hardware

DynModel in 2D Keypoint space (DynModel+2D) We train a keypoint detector as described in (Minderer *et al.*, 2019; Das *et al.*, 2020b) without depth information. This keypoint detector does not use any proprioceptive information during training. We use the keypoint detector to train a dynamics model $s_{t+1} = g_\beta(s_t, u_t)$ from a collected dataset, this reimplements the approach presented in (Das *et al.*, 2020b).

DynModel in 3D Keypoint space (DynModel+3D) This baseline extends the baseline presented in a) by including also depth information during keypoint detector and dynamics model training. The latent space now is three dimensional as it includes (x, y, d) - pixel locations and depth information. With this baseline we want to test whether adding depth information improves task performance.

DynModel in 2D-multimodal keypoint space (DynModel+2D+multi) We train a structured keypoint detector as described in section 4.4.3, the structured keypoint detector is trained using proprioceptive information. After keypoint detector t raining we train a black box neural network dynamics model $s_{t+1} = g_\beta(s_t, u_t)$, instead of regressing virtual joint parameter. This baseline will allow us to see whether using the structured keypoint detector also improves performance on the manipulation task, even if the visual dynamics model is learned with a neural network.

DynModel in 3D-multimodal keypoint space DynModel+3D+multi) same as above the baseline in c) but here again we include depth information for keypoint detector training and neural network dynamics model training. As in the baseline before we want to test whether using proprioceptive information during keypoint detector training improves performance on the task, but we also want to test whether adding depth information, and thus a third dimension to our visual latent space, improves performance.

For all our baselines we train a visual dynamics model in the keypoint space represented by a neural network. We use dataset $\mathcal{D} = \{(\theta_t, u_t, z_t))\}_{t=0}^{2000}$ collected on sine motions of the robot, where z are the keypoints predicted by the detectors, to train a feedforward neural network. All the dynamics models are trained to convergences on the training data, and achieve a normalized mean squared error below 0.1 on the test data. Training a visual dynamics model in the latent space has been a popular choice in the literature so far, (Das *et al.*, 2020b; Manuelli *et al.*, 2020). It has the advantage that neural networks have a lot of flexibility to represent the problems at hand, however a downside of neural network based approaches is the unreliable extrapolation behaviour for out of distribution data and long horizon prediction performance, that degenerates fast and is detrimental for model-based control. In Fig. 4.12 we show the predictive behaviour of our

Method	**Task 1** Mean (Std)	**Task 2** Mean (Std)	**Task 3** Mean (Std)
Ours	**4.85(2.91)**	**2.04(1.26)**	**2.06(1.18)**
a	155.62(343.84)	60.78(73.67)	70.00(67.73)
b	135.16(119.7)	69.60(59.69)	85.32(83.27)
c	103.35(85.07)	29.37(48.09)	70.38(88.42)
d	120.21(107.39)	40.14(42.95)	96.58(258.14)
Hardware	**4.06 (3.83)**	**2.12 (0.04)**	**5.60 (4.43)**

Table 4.1: The performance of three placing tasks. We compare our method to the baselines introduced before. We report the final distance of the object in the hand from the desired target position of the object as root mean squared error. We average our results over five seeds and four different re-grasps of the object in simulation. On hardware we average over five seeds. The errors are reported in pixel space.

baseline neural network models. In the image on the right, the training error of the models is shown and it is clearly possible to see that the models were trained until convergence. Nevertheless the two plots on the left show the long horizon prediction behaviour of the neural networks compared with out extended kinematic chain. On the x axis the time horizon is shown and on the y axis the prediction error. From the plots it becomes evident that predicting for longer horizons through the neural network dynamics models, which is necessary when optimizing actions for a motor control task like placing, leads to exponential prediction error. The plot on the left shows long horizon prediction for the placing task action sequence, the plot in the middle for a random action sequence. In all cases the predictive performance of the neural networks deteriorates exponentially over time for all baselines which also explains the poor results presented in Table 4.1 where we report the results for our experiments. Only the neural network model that is trained using our multimodal keypoint detector with depth information (baseline d), can achieve a satisfactory predictive behaviour over 10 time steps, showing that improved on-object keypoint detection can increase also the performance of neural network visual dynamics models.

In Table 4.1 the simulation experiments are averaged over five seeds, the three objects introduced earlier and four different grasp positions. The results show the final distance of the object keypoints in the manipulators hand to the desired goal keypoints on the table. We perform a total of three tasks that vary in start-goal configurations. All the distances are reported in pixel space, and represents the root mean squared error (RMSE). Between all the baselines, the baselines c) and d) that use our proposed keypoint detector, perform best. This indicates, that the keypoints trained with our detector are better suited for the object manipulation task. Yet overall, compared to our extended kinematic chain, all of them perform fairly poorly in the model-based control task. Our approach outperforms all baseline

variations and is the only one to successfully place the object on the table. This gap in performance can be explained by the fact that the extended kinematic is able to generalize throughout the state-space, while the trained dynamics models prediction quality deteriorates quickly outside of the training data distribution.

It is worthwhile highlighting, that our approach is successful also when used on the real robot, where we achieve an average RMSE of 3.92 pixels on the placing task. This is remarkable not only because on hardware noise and controller inaccuracies often make the task harder, but also because we run our optimized action sequence in a feedforward fashion, without having to replan at each time step.

4.9 Discussion and Future Work

We present a method for learning extended body schemas from vision. Our method inherently incorporates an external object in the manipulator's hand. To this end we show how merging two sensor modalities, precisely proprioceptive information, in form of joint positions, and vision, enables us to learn better visual latent representation of the object. The latent representation is then used to successfully learn an extension of a robot's kinematics model. As a result the learned extended kinematic chain incorporates the object into the kinematic model of the robot. We show how we can use the learned kinematic model for object manipulation on a placing task, where the task is defined in the visual domain. We also show how we can adapt the kinematic chain for different grasps from only few datapoints. Our experiments, on a 7 DoF iiwa Kuka arm in simulation and on hardware, show the generality of our approach and the good performance to out of distribution tasks and on the real system. In contrast to current state of the art, we leverage structured analytical models, i.e the kinematic model, and combine this structure with a data driven approach to learn visual latent keypoint representations for consistent on-object keypoint placing. When learning the extended kinematic chain, we again combine structure and learning to recover a representation on the robot's arm with the object that transfers easily to new tasks. We believe the combination of data driven learning and structured models are an interesting avenue for further research. A remaining challenge of our approach is how to decide when to add or remove an object from the extended kinematic chain. This depends on the grasp and release behaviour of the desired task. In the future we would also like to explore more complex, contact rich manipulation tasks, that use the learned extended body schema to control the robot during contact rich manipulation. We also believe that combining vision only data with robot experience in form of proprioception is an interesting avenue for future work. This would allow to leverage larger visual dataset, not collected directly on the robot, and combine this visual data with a smaller but informative amount of robot experience.

Part II

Learning To Learn in the Action Perception Loop

Chapter 5

Meta Learning via Learned Loss

5.1 Introduction

Inspired by the remarkable capability of humans to quickly learn and adapt to new tasks, the concept of learning to learn, or *meta-learning*, recently became popular within the machine learning community (Andrychowicz *et al.*, 2016; Duan *et al.*, 2016; Finn *et al.*, 2017). We can classify *learning to learn* methods into roughly two categories: approaches that learn representations that can generalize and are easily adaptable to new tasks (Finn *et al.*, 2017), and approaches that learn how to optimize models (Andrychowicz *et al.*, 2016; Duan *et al.*, 2016).

In this paper we investigate the second type of approach. We propose a learning framework that is able to learn any parametric loss function—as long as its output is differentiable with respect to its parameters. Such learned functions can be used to efficiently optimize models for new tasks.

Specifically, the purpose of this work is to encode learning strategies into a parametric loss function, or a *meta-loss*, which generalizes across multiple training contexts or tasks. Inspired by *inverse reinforcement learning* (Ng *et al.*, 2000), our work combines the *learning to learn* paradigm of meta-learning with the generality of learning loss landscapes. We construct a unified, fully differentiable framework that can learn optimizee-independent loss functions to provide a strong learning signal for a variety of learning problems, such as classification, regression or reinforcement learning. Our framework involves an inner and an outer optimization loops. In the inner loop, a model or an *optimizee* is trained with gradient descent using the loss coming from our learned meta-loss function. fig. 5.1 shows the pipeline for updating the optimizee with the meta-loss. The outer loop optimizes the meta-loss function by minimizing a *task-loss*, such as a standard regression or reinforcement-learning loss, that is induced by the updated optimizee.

The contributions of this work are as follows: i) we present a framework for learning adaptive, high-dimensional loss functions through back-propagation that create the loss landscapes for efficient optimization with gradient descent. We show that our learned meta-loss functions improve over directly learning via the task-loss itself while maintaining the generality of the task-loss. ii) We present several

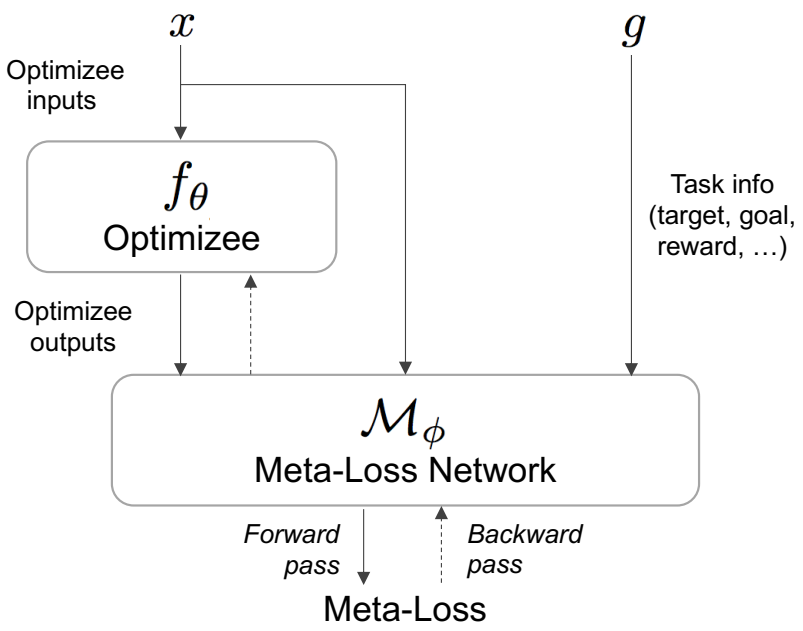


Figure 5.1: Framework overview: The learned meta-loss is used as a learning signal to optimize the optimizee f_θ, which can be a regressor, a classifier or a control policy.

ways our framework can incorporate extra information that helps shape the loss landscapes at *meta-train* time. This extra information can take on various forms, such as exploratory signals or expert demonstrations for RL tasks. After training the meta-loss function, the task-specific losses are no longer required since the training of optimizees can be performed entirely by using the meta-loss function alone, without requiring the extra information given at *meta-train* time. In this way, our meta-loss can find more efficient ways to optimize the original task loss.

We apply our meta-learning approach to a diverse set of problems demonstrating our framework's flexibility and generality. The problems include regression problems, image classification, behavior cloning, model-based and model-free reinforcement learning. Our experiments include empirical evaluation for each of the aforementioned problems.

5.2 Related Work

Meta-learning originates from the concept of learning to learn (Schmidhuber, 1987; Bengio and Bengio, 1990; Thrun and Pratt, 2012a). Recently, there has been a wide interest in finding ways to improve learning speeds and generalization to new tasks through meta-learning. Let us consider gradient based learning approaches, that update the parameters of an *optimizee* $f_\theta(x)$, with model parameters θ and

inputs x as follows:

$$\theta_{\text{new}} = h_\psi(\theta, \nabla_\theta \mathcal{L}_\phi(y, f_\theta(x)); \tag{5.1}$$

where we take the gradient of a loss function $\mathcal{L}$, parametrized by ϕ, with respect to the optimizee's parameters θ and use a gradient transform h, parametrized by ψ, to compute new model parameters θ_{new}[1]. In this context, we can divide related work on meta-learning into learning model parameters θ that can be easily adapted to new tasks (Finn *et al.*, 2017; Mendonca *et al.*, 2019; Gupta *et al.*, 2018; Yu *et al.*, 2018), learning optimizer policies h that transform parameters updates with respect to known loss or reward functions (Maclaurin *et al.*, 2015; Andrychowicz *et al.*, 2016; Li and Malik, 2016; Franceschi *et al.*, 2017; Meier *et al.*, 2018; Duan *et al.*, 2016), or learning loss/reward function representations ϕ (Sung *et al.*, 2017; Houthooft *et al.*, 2018; Zou *et al.*, 2019). Alternatively, in unsupervised learning settings, meta-learning has been used to learn unsupervised rules that can be transferred between tasks (Metz *et al.*, 2019; Hsu *et al.*, 2018).

Our framework falls into the category of learning loss landscapes. Similar to works by (Sung *et al.*, 2017) and (Houthooft *et al.*, 2018), we aim at learning loss function parameters ϕ that can be applied to various optimizee models, e.g. regressors, classifiers or agent policies. Our learned loss functions are independent of the model parameters θ that are to be optimized, thus they can be easily transferred to other optimizee models. This is in contrast to methods that meta-learn model-parameters θ directly (e.g. Finn *et al.*, 2017; Mendonca *et al.*, 2019), which are orthogonal and complementary to ours, where the learned representation θ cannot be separated from the original model of the optimizee. The idea of learning loss landscapes or reward functions in the reinforcement learning (RL) setting can be traced back to the field of inverse reinforcement learning (Ng *et al.*, 2000; Abbeel and Ng, 2004, IRL). However, in contrast to IRL we do not require expert demonstrations (however we can incorporate them). Instead we use task losses as a measure of the effectiveness of our loss function when using it to update an optimizee.

Closest to our method are the works on *evolved policy gradients* (Houthooft *et al.*, 2018), *teacher networks* (Wu *et al.*, 2018), *meta-critics* (Sung *et al.*, 2017) and *meta-gradient RL* (Xu *et al.*, 2018). In contrast to using an evolutionary approach (e.g. Houthooft *et al.*, 2018), we design a differentiable framework and describe a way to optimize the loss function with gradient descent in both supervised and reinforcement learning settings. (Wu *et al.*, 2018) propose that instead of learning a differentiable loss function directly, a teacher network is trained to predict parameters of a manually designed loss function, whereas each new loss function class requires a new teacher network design and training. In (Xu *et al.*, 2018),

[1] For simple gradient descent: $h(\theta, \nabla_\theta \mathcal{L}(y, f_\theta(x)) = \theta - \psi \nabla_\theta \mathcal{L}(y, f_\theta(x))$

discount and bootstrapping parameters are learned online to optimize a task-specific meta-objective. Our method does not require manual design of the loss function parameterization or choosing particular parameters that have to be optimized, as our loss functions are learned entirely from data. Finally, in work by (Sung *et al.*, 2017) a *meta-critic* is learned to provide a task-conditional value function, used to train an actor policy. Although training a meta-critic in the supervised setting reduces to learning a loss function as in our work, in the reinforcement learning setting we show that it is possible to use learned loss functions to optimize policies directly with gradient descent.

5.3 Meta-Learning via Learned Loss

In this work, we aim to learn a loss function, which we call *meta-loss*, that is subsequently used to train an *optimizee*, e.g. a classifier, a regressor or a control policy. More concretely, we aim to learn a meta-loss function $\mathcal{M}_\phi$ with parameters ϕ, that outputs the loss value $\mathcal{L}_{\text{learned}}$ which is used to train an optimizee f_θ with parameters θ via gradient descent:

$$\theta_{\text{new}} = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{learned}}, \tag{5.2}$$

$$\text{where } \mathcal{L}_{\text{learned}} = \mathcal{M}_\phi(y, f_\theta(x)) \tag{5.3}$$

where y can be ground truth target information in supervised learning settings or goal and state information for reinforcement learning settings. In short, we aim to learn a loss function that can be used as depicted in Algorithm 7. Towards this goal, we propose an algorithm to learn meta-loss function parameters ϕ via gradient descent.

The key challenge is to derive a training signal for learning the loss parameters ϕ. In the following, we describe our approach to addressing this challenge, which we call **M**eta-**L**earning via **L**earned **L**oss (ML^3).

5.3.1 ML^3 for Supervised Learning

We start with supervised learning settings, in which our framework aims at learning a meta-loss function $\mathcal{M}_\phi(y, f_\theta(x))$ that produces the loss value given the ground truth target y and the predicted target $f_\theta(x)$. For clarity purposes we constrain the following presentation to learning a meta-loss network that produces the loss value for training a regressor f_θ via gradient descent, however the methodology trivially generalizes to classification tasks.

Our meta-learning framework starts with randomly initialized model parameters θ and loss parameters ϕ. The current loss parameters are then used to produce loss value $\mathcal{L}_{\text{learned}} = \mathcal{M}_\phi(y, f_\theta(x))$. To optimize model parameters θ we need to compute the gradient of the loss value with respect to θ, $\nabla_\theta \mathcal{L} = \nabla_\theta \mathcal{M}_\phi(y, f_\theta(x))$.

Using the chain rule, we can decompose the gradient computation into the gradient of the loss network with respect to predictions of model $f_\theta(x)$ times the gradient of model f with respect to model parameters[2].

$$\nabla_\theta \mathcal{M}_\phi(y, f_\theta(x)) = \nabla_f \mathcal{M}_\phi(y, f_\theta(x)) \nabla_\theta f_\theta(x). \tag{5.4}$$

Once we have updated the model parameters $\theta_{\text{new}} = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{learned}}$ using the current meta-loss network parameters ϕ, we want to measure how much learning progress has been made with loss-parameters ϕ and optimize ϕ via gradient descent. Note, that the new model parameters θ_{new} are implicitly a function of loss-parameters ϕ, because changing ϕ would lead to different θ_{new}. In order to evaluate θ_{new}, and through that loss-parameters ϕ, we introduce the notion of a *task-loss* during *meta-train* time. For instance, we use the mean-squared-error (MSE) loss, which is typically used for regression tasks, as a task-loss $\mathcal{L}_\mathcal{T} = (y - f_{\theta_{\text{new}}}(x))^2$. We now optimize loss parameters ϕ by taking the gradient of $\mathcal{L}_\mathcal{T}$ with respect to ϕ as follows[2]:

$$\nabla_\phi \mathcal{L}_\mathcal{T} = \nabla_f \mathcal{L}_\mathcal{T} \nabla_{\theta_{\text{new}}} f_{\theta_{\text{new}}} \nabla_\phi \theta_{\text{new}} \tag{5.5}$$

$$= \nabla_f \mathcal{L}_\mathcal{T} \nabla_{\theta_{\text{new}}} f_{\theta_{\text{new}}} \nabla_\phi [\theta - \alpha \nabla_\theta \mathbb{E}\left[\mathcal{M}_\phi(y, f_\theta(x))\right] \tag{5.6}$$

where we first apply the chain rule and show that the gradient with respect to the meta-loss parameters ϕ requires the new model parameters θ_{new}. We expand θ_{new} as one gradient step on θ based on meta-loss $\mathcal{M}_\phi$, making the dependence on ϕ explicit.

Optimization of the loss-parameters can either happen after *each inner gradient* step (where inner refers to using the current loss parameters to update θ), or after M *inner gradient steps* with the current meta-loss network $\mathcal{M}_\phi$.

The latter option requires back-propagation through a chain of all optimizee update steps. In practice we notice that updating the meta-parameters ϕ after each inner gradient update step works better. We reset θ after M *inner gradient steps*. We summarize the *meta-train* phase in Algorithm 6, with one *inner* gradient step.

Algorithm 6 ML^3 at (*meta-train*)

$\phi \leftarrow$ randomly initialize
while not done **do**
  $\theta \leftarrow$ randomly initialize
  $x, y \leftarrow$ Sample task samples from $\mathcal{T}$
  $\mathcal{L}_{\text{learned}} = \mathcal{M}(y, f_\theta(x))$
  $\theta_{\text{new}} \leftarrow \theta - \alpha \nabla_\theta \mathbb{E}_x [\mathcal{L}_{\text{learned}}]$
  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_\mathcal{T}(y, f_{\theta_{\text{new}}})$
end while

[2]Alternatively this gradient computation can be performed using automatic differentiation

Algorithm 7 ML^3 at (*meta-test*)

$M \leftarrow$ # of optimizee updates
$\theta \leftarrow$ randomly initialize
for $j \in \{0, \ldots, M\}$ **do**
  $x, y \leftarrow$ Sample task samples from $\mathcal{T}$
  $\mathcal{L}_{\text{learned}} = \mathcal{M}(y, f_\theta(x))$
  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathbb{E}_x \left[\mathcal{L}_{\text{learned}}\right]$
end for

5.3.2 ML^3 Reinforcement Learning

In this section, we introduce several modifications that allow us to apply the ML^3 framework to reinforcement learning problems. Let $\mathcal{M} = (S, A, P, R, p_0, \gamma, T)$ be a finite-horizon Markov Decision Process (MDP), where S and A are state and action spaces, $P : S \times A \times S \rightarrow \mathbb{R}_+$ is a state-transition probability function or system dynamics, $R : S \times A \rightarrow \mathbb{R}$ a reward function, $p_0 : S \rightarrow \mathbb{R}_+$ an initial state distribution, γ a reward discount factor, and T a horizon. Let $\tau = (s_0, a_0, \ldots, s_T, a_T)$ be a trajectory of states and actions and $R(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$ the trajectory return. The goal of reinforcement learning is to find parameters θ of a policy $\pi_\theta(a|s)$ that maximizes the expected discounted reward over trajectories induced by the policy: $\mathbb{E}_{\pi_\theta}[R(\tau)]$ where $s_0 \sim p_0, s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ and $a_t \sim \pi_\theta(a_t|s_t)$. In what follows, we show how to train a meta-loss network to perform effective policy updates in a reinforcement learning scenario. To apply our ML^3 framework, we replace the optimizee f_θ from the previous section with a stochastic policy $\pi_\theta(a|s)$. We present two applications of ML^3 to RL.

ML^3 for Model-Based Reinforcement Learning

Model-based RL (MBRL) attempts to learn a policy π_θ by first learning a dynamic model P. Intuitively, if the model P is accurate, we can use it to optimize the policy parameters θ. As we typically do not know the dynamics model a-priori, MBRL algorithms iterate between using the current approximate dynamics model P, to optimize the policy π_θ such that it maximizes the reward R under P, then use the optimized policy π_θ to collect more data which is used to update the model P. In this context, we aim to learn a loss function that is used to optimize policy parameters through our meta-network $\mathcal{M}$.

Similar to the supervised learning setting we use current meta-parameters ϕ to optimize policy parameters θ under the current dynamics model P: $\theta_{\text{new}} = \theta - \alpha \nabla_\theta \left[\mathcal{M}_\phi(\tau, g)\right]$,

where $\tau = (s_0, a_0, \ldots, s_T, a_T)$ is the sampled trajectory and the variable g captures some task-specific information, such as the goal state of the agent. To optimize ϕ we again need to define a task loss, which in the MBRL setting can be defined as

$\mathcal{L}_{\mathcal{T}}(g, \pi_{\theta_{\text{new}}}) = -\mathbb{E}_{\pi_{\theta_{\text{new}}}, P}[R_g(\tau_{\text{new}})]$, denoting the reward that is achieved under the current dynamics model P. To update ϕ, we compute the gradient of the task loss $\mathcal{L}_{\mathcal{T}}$ wrt. ϕ, which involves differentiating all the way through the reward function, dynamics model and the policy that was updated using the meta-loss $\mathcal{M}_\phi$. The pseudo-code in Algorithm 8 (Appendix A.3) illustrates the MBRL learning loop. In Algorithm 10 (Appendix A.3), we show the policy optimization procedure during meta-test time. Notably, we have found that in practice, the model of the dynamics P is not needed anymore for policy optimization at meta-test time. The meta-network learns to implicitly represent the gradients of the dynamics model and can produce a loss to optimize the policy directly.

ML^3 for Model-Free Reinforcement Learning

Finally, we consider the model-free reinforcement learning (MFRL) case, where we learn a policy without learning a dynamics model. In this case, we can define a surrogate objective, which is independent of the dynamics model, as our task-specific loss (Williams, 1992; Sutton *et al.*, 2000; Schulman *et al.*, 2015):

$$\mathcal{L}_{\mathcal{T}}(g, \pi_{\theta_{\text{new}}}) = -\mathbb{E}_{\pi_{\theta_{\text{new}}}}\left[R_g(\tau_{\text{new}}) \log \pi_{\theta_{\text{new}}}(\tau_{\text{new}})\right] \tag{5.7}$$

$$= -\mathbb{E}_{\pi_{\theta_{\text{new}}}}\left[R_g(\tau_{\text{new}}) \sum_{t=0}^{T-1} \log \pi_{\theta_{\text{new}}}(a_t|s_t)\right] \tag{5.8}$$

Similar to the MBRL case, the task loss is indirectly a function of the meta-parameters ϕ that are used to update the policy parameters. Although we are evaluating the task loss on full trajectory rewards, we perform policy updates from Eq. (5.2) using stochastic gradient descent (SGD) on the meta-loss with mini-batches of experience (s_i, a_i, r_i) for $i \in \{0, \ldots, B-1\}$ with batch size B, similar to (Houthooft *et al.*, 2018). The inputs of the meta-loss network are the sampled states, sampled actions, task information g and policy probabilities of the sampled actions: $\mathcal{M}_\phi(s, a, \pi_\theta(a|s), g)$. In this way, we enable efficient optimization of very high-dimensional policies with SGD provided only with trajectory-based rewards. In contrast to the above MBRL setting, the rollouts used for task-loss evaluation are real system rollouts, instead of simulated rollouts. At test time, we use the same policy update procedure as in the MBRL setting, see Algorithm 10 (Appendix A.3).

5.3.3 Shaping ML^3 loss by adding extra loss information during *meta-train*

So far, we have discussed using standard task losses, such as MSE-loss for regression or reward functions for RL settings. However, it is possible to provide more information about the task at *meta-train* time, which can influence the learning of

the loss-landscape. We can design our task-losses to incorporate extra penalties; for instance we can extend the MSE-loss with $\mathcal{L}_{\text{extra}}$ and weight the terms with β and γ:

$$\mathcal{L}_{\mathcal{T}} = \beta(y - f_\theta(x))^2 + \gamma\mathcal{L}_{\text{extra}} \tag{5.9}$$

In our work, we experiment with 4 different types of extra loss information at *meta-train* time: for supervised learning we show that adding extra information through $\mathcal{L}_{\text{extra}} = (\theta - \theta^*)^2$, where θ^* are the optimal regression parameters, can help shape a convex loss-landscape for otherwise non-convex optimization problems; we also show how we can use $\mathcal{L}_{\text{extra}}$ to induce a physics prior in robot model learning. For reinforcement learning tasks we demonstrate that by providing additional rewards in the task loss during meta-train time, we can encourage the trained meta-loss to learn exploratory behaviors; and finally also for reinforcement learning tasks, we show how expert demonstrations can be incorporated to learn loss functions which can generalize to new tasks. In all settings, the additional information shapes the learned loss function such that the environment does not need to provide this information during meta-test time.

5.4 Experiments

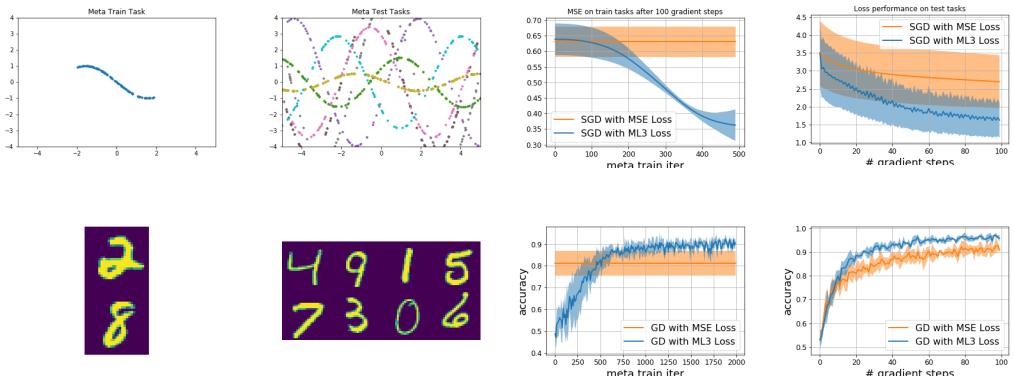
In this section we evaluate the applicability and the benefits of the learned meta-loss from two different view points. First, we study the benefits of using standard task losses, such as the mean-squared error loss for regression, to train the meta-loss in Section 5.4.1. We analyze how a learned meta-loss compares to using a standard task-loss in terms of generalization properties and convergence speed. Second, we study the benefit of adding extra information at *meta-train* time to shape the loss landscape in Section 5.4.2.

5.4.1 Learning to mimic and improve over known task losses

First, we analyze how well our meta-learning framework can learn to mimic and improve over standard task losses for both supervised and reinforcement learning settings. For these experiments, the meta-network is parameterized by a neural network with two hidden layers of 40 neurons each.

Meta-Loss for Supervised Learning

In this set of experiments, we evaluate how well our meta-learning framework can learn loss functions $\mathcal{M}_\phi$ for regression and classification tasks. In particular, we perform experiments on sine function regression and binary classification of digits



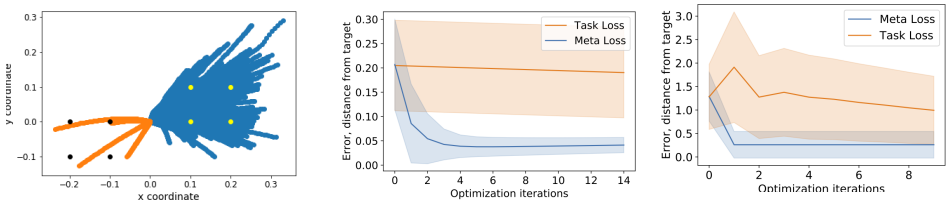
(a) Meta-Train Tasks (b) Meta-Test Tasks (c) Meta-Train (d) Meta-Test

Figure 5.2: Meta-learning for regression (top) and binary classification (bottom) tasks. (a) meta-train task, (b) meta-test tasks, (c) performance of the meta-network on the meta-train task as a function of (outer) meta-train iterations in blue, as compared to SGD using the task-loss directly in orange, (d) average performance of meta-loss on meta-test tasks as a function of the number of gradient update steps

(see details in Appendix A.6). At meta-train time, we randomly draw one task for meta-training (see fig. 5.2 (a)), and at meta-test time we randomly draw 10 test tasks for regression, and 4 test tasks for classification (fig. 5.2(b)). For the sine regression, tasks are drawn according to details in Appendix A.6, and we initialize our model f_θ to a simple feedforward NN with 2 hidden layers and 40 hidden units each, for the binary classification task f_θ is initialized via the *LeNet* architecture (LeCun *et al.*, 1998). For both experiments we use a fixed learning rate $\alpha = \eta = 0.001$ for both inner (α) and outer (η) gradient update steps. We average results across 5 random seeds, where each seed controls the initialization of both initial model and meta-network parameters, as well as the the random choice of meta-train/test task(s), and visualize them in fig. 5.2. We compare the performance of using SGD with the task-loss $\mathcal{L}$ directly (in orange) to SGD using the learned meta-network $\mathcal{M}_\phi$ (in blue), both using a learning rate $\alpha = 0.001$. In fig. 5.2 (c) we show the average performance of the meta-network $\mathcal{M}_\phi$ as it is being learned, as a function of (outer) meta-train iterations in blue. In both regression and classification tasks, the meta-loss eventually leads to a better performance on the meta-train task as compared to the task loss. In fig. 5.2 (d) we evaluate SGD using $\mathcal{M}_\phi$ vs SGD using $\mathcal{L}$ on previously unseen (and out-of-distribution) meta-test tasks as a function of the number of gradient steps. Even on these novel test tasks, our learned $\mathcal{M}_\phi$ leads to improved performance as compared to the task-loss.

Learning Reward functions for Model-based Reinforcement Learning

In the MBRL example, the tasks consist of a free movement task of a point mass in a 2D space, we call this environment PointmassGoal, and a reaching task with a 2-link



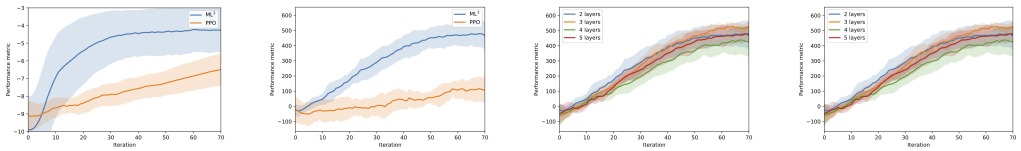
(a) train (blue), test (orange) tasks (b) Meta vs Task Loss Pointmass (c) Meta vs Task Loss Reacher

Figure 5.3: ML^3 for MBRL: results are averaged across 10 runs. We can see in (a) that the ML^3 loss generalizes well, the loss was trained on the blue trajectories and tested on the orange ones for the PointmassGoal task. ML^3 loss also significantly speeds up learning when compared to the task loss at meta-test time on the PointmassGoal (b) and the ReacherGoal (c) environments.

2D manipulator, which we call the ReacherGoal environment (see Appendix A.4 for details). The task distribution $p(\mathcal{T})$ consists of different target positions that either the point mass or the arm should reach. During meta-train time, a model of the system dynamics, represented by a neural network, is learned from samples of the currently optimal policy. The task loss during meta-train time is $\mathcal{L}_{\mathcal{T}}(\theta) = \mathbb{E}_{\pi_\theta, P}[R(\tau)]$, where $R(\tau)$ is the final distance from the goal g, when rolling out $\pi_{\theta_{\text{new}}}$ in the dynamics model P. Taking the gradient $\nabla_\phi \mathbb{E}_{\pi_{\theta_{\text{new}}}, P}[R(\tau)]$ requires the differentiation through the learned model P (see Appendix 8). The input to the meta-network is the state-action trajectory of the current roll-out and the desired target position. The meta-network outputs a loss signal together with the learning rate to optimize the policy. fig. 5.3a shows the qualitative reaching performance of a policy optimized with the meta loss during meta-test on PointmassGoal. The meta-loss network was trained only on tasks in the right quadrant (blue trajectories) and tested on the tasks in the left quadrant (orange trajectories) of the x, y plane, showing the generalization capability of the meta loss. Figure 5.3b and 5.3c show a comparison in terms of final distance to the target position at test time. The performance of policies trained with the meta-loss is compared to policies trained with the task loss, in this case final distance to the target. The curves show results for 10 different goal positions (including goal positions where the meta-loss needs to generalize). When optimizing with the task loss, we use the dynamics model learned during the meta-train time, as in this case the differentiation through the model is required during test time. As mentioned in Section 5.3.2, this is not needed when using the meta-loss.

Learning Reward functions for Model-free Reinforcement Learning

In the following, we move to evaluating on model-free RL tasks. fig. 5.4 shows results when using two continuous control tasks based on OpenAI Gym MuJoCo environments (OpenAI Gym, 2019): ReacherGoal and AntGoal (see Appendix A.5 for details).[3] fig. 5.4a and fig. 5.4b show the results of the meta-test time performance



(a) ReacherGoal (b) AntGoal (c) ReacherGoal (d) AntGoal

Figure 5.4: ML^3 for model-free RL: results are averaged across 10 tasks. (a+b) Policy learning on new task with ML^3 loss compared to PPO objective performance during *meta-test* time. The learned loss leads to faster learning at meta-test time. (c+d) Using the same ML^3 loss, we can optimize policies of different architectures, showing that our learned loss maintains generality.

for the ReacherGoal and the AntGoal environments respectively. We can see that ML^3 loss significantly improves optimization speed in both scenarios compared to PPO. In our experiments, we observed that on average ML^3 requires 5 times fewer samples to reach 80% of task performance in terms of our metrics for the model-free tasks.

To test the capability of the meta-loss to generalize across different architectures, we first meta-train $\mathcal{M}_\phi$ on an architecture with two layers and meta-test the same meta-loss on architectures with varied number of layers. fig. 5.4 (c+d) show meta-test time comparison for the ReacherGoal and the AntGoal environments in a model-free setting for four different model architectures. Each curve shows the average and the standard deviation over ten different tasks in each environment. Our comparison clearly indicates that the meta-loss can be effectively re-used across multiple architectures with a mild variation in performance compare to the overall variance of the corresponding task optimization.

5.4.2 Shaping loss landscapes by adding extra information at meta-train time

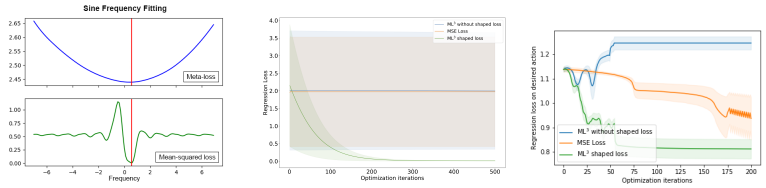
This set of experiments shows that our meta-learner is able to learn loss functions that incorporate extra information available only during meta-train time. The

[3] Our framework is implemented using open-source libraries *Higher* (Grefenstette *et al.*, 2019) for convenient second-order derivative computations and *Hydra* (Yadan, 2019) for simplified handling of experiment configurations.

learned loss will be shaped such that optimization is faster when using the meta-loss compared to using a standard loss.

Illustration: Shaping loss

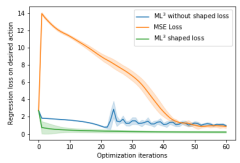
We start by illustrating the loss shaping on an example of sine frequency regression where we fit a single parameter for the purpose of visualization simplicity.



(a) Sine: learned vs task loss

(b) Sine: meta-test time

(c) Reacher: inverse dynamics



(d) Sawyer: inverse dynamics

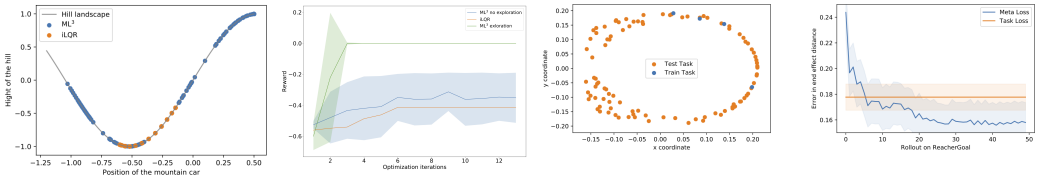
Figure 5.5: Meta-test time evaluation of the shaped meta-loss (ML^3), i.e. trained with shaping ground-truth (extra) information at meta-train time: a) Comparison of learned ML^3 loss (top) and MSE loss (bottom) landscapes for fitting the frequency of a sine function. The red lines indicate the ground-truth values of the frequency. b) Comparing optimization performance of: ML^3 loss trained with (green), and without (blue) ground-truth frequency values; MSE loss (orange). The ML^3 loss learned with the ground-truth values outperforms both the non-shaped ML^3 loss and the MSE loss. c-d) Comparing performance of inverse dynamics model learning for ReacherGoal (c) and Sawyer arm (d). ML^3 loss trained with (green) and without (blue) ground-truth inertia matrix is compared to MSE loss (orange). The shaped ML^3 loss outperforms the MSE loss in all cases.

For this illustration we generate training data $\mathcal{D} = \{x_n, y_n\}^N, N = 1000$, by drawing data samples from the ground truth function $y = sin(\nu x)$, for $x = [-1, 1]$. We create a model $f_\omega(x) = \sin(\omega x)$, and aim to optimize parameter ω on $\mathcal{D}$, with the goal of recovering value ν. fig. 5.5a (bottom) shows the loss landscape for optimizing ω, when using the MSE loss. The target frequency ν is indicated by a vertical red line. As noted by Parascandolo *et al.* (2017), the landscape of this loss is highly non-convex and difficult to optimize with conventional gradient descent.

Here, we show that by utilizing additional information about the ground truth

value of the frequency at meta-train time, we can learn a better shaped loss. Specifically, during meta-train time, our task-specific loss is the squared distance to the ground truth frequency: $(\omega - \nu)^2$ that we later call the *shaping loss*. The inputs of the meta-network $\mathcal{M}_\phi(y, \hat{y})$ are the training targets y and predicted function values $\hat{y} = f_\omega(x)$, similar to the inputs to the mean-squared loss. After meta-train time commences our learned loss function $\mathcal{M}_\phi$ produces a convex loss landscapes as depicted in fig. 5.5a(top).

To analyze how the shaping loss impacts model optimization at meta-test time, we compare 3 loss functions: 1) directly using standard MSE loss (orange), 2) ML^3 loss that was trained via the MSE loss as task loss (blue), and 3) ML^3 loss trained via the shaping loss, fig. 5.5b. When comparing the performance of these 3 losses, it becomes evident that without shaping the loss landscape, the optimization is prone to getting stuck in a local optimum.



(a) Trajectory ML^3 vs. iLQR (b) MountainCar: meta-test time (c) Train and test targets (d) ML^3 vs. Task loss at test

Figure 5.6: (a) MountainCar trajectory for policy optimized with iLQR compared to ML^3 loss with extra information. (b) optimization performance during meta-test time for policies optimized with iLQR compared to ML^3 with and without extra information. (c+d) ReacherGoal with expert demonstrations available during meta-train time. (c) shows the targets in end-effector space. The four blue dots show the training targets for which expert demonstrations are available, the orange dots show the meta-test targets. In (d) we show the reaching performance of a policy trained with the shaped ML^3 loss at meta-test time, compared to the performance of training simply on the behavioral cloning objective and testing on test targets.

Shaping loss via physics prior for inverse dynamics learning

Next, we show the benefits of shaping our ML^3 loss via ground truth parameter information for a robotics application. Specifically, we aim to learn and shape a meta-loss that improves sample efficiency for learning (inverse) dynamics models, i.e. a mapping $\boldsymbol{u} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_{\text{des}})$, where: $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}_{\text{des}}$ are vectors of joint angular positions, velocities and desired accelerations; $\boldsymbol{u}$ is a vector of joint torques.

Rigid body dynamics (RBD) provides an analytical solution to computing the

(inverse) dynamics and can generally be written as:

$$M(q)\ddot{q} + F(q, \dot{q}) = u \tag{5.10}$$

where the inertia matrix $M(q)$, and $F(q, \dot{q})$ are computed analytically (Featherstone, 2014). Learning an inverse dynamics model using neural networks can increase the expressiveness compared to RBD but requires many data samples that are expensive to collect. Here we follow the approach in (Lutter *et al.*, 2019), and attempt to learn the inverse dynamics via a neural network that predicts the inertia matrix $M_\theta(q)$. To improve upon sample efficiency we apply our method by shaping the loss landscape during meta-train time using the ground truth inertia matrix $M(q)$ provided by a simulator. Specifically, we use the task loss $\mathcal{L}_\mathcal{T} = (M_\theta(q) - M(q))^2$ to optimize our meta-loss network. During meta-test time we use our trained meta-loss shaped with the physics prior (the inertia matrix exposed by the simulator) to optimize the inverse dynamics neural network. In fig. 5.5-c we show the prediction performance of the inverse dynamics model during meta-test time on new trajectories of the ReacherGoal environment. We compare the optimization performance during meta-test time when using the meta-loss trained with physics prior, the meta loss trained without physics prior (i.e via MSE loss) to the optimization with MSE loss. fig. 5.5-d shows a similar comparison for the Sawyer environment - a simulator of the 7 degrees-of-freedom Sawyer anthropomorphic robot arm. Inverse dynamics learning using the meta loss with physics prior achieves the best prediction performance on both robots. ML^3 without physics prior performs worst on the ReacherGoal environment, in this case the task loss formulated only in the action space did not provide enough information to learn a $\mathcal{L}_{\text{learned}}$ useful for optimization. For the Sawyer training with MSE loss leads to a slower optimization, however the asymptotic performance of MSE and ML^3 is the same. Only ML^3 with shaped loss outperforms both.

Shaping Loss via intermediate goal states for RL

We analyze loss landscape shaping on the MountainCar environment (Moore, 1990), a classical control problem where an under-actuated car has to drive up a steep hill. The propulsion force generated by the car does not allow steady climbing of the hill, thus greedy minimization of the distance to the goal often results in a failure to solve the task. The state space is two-dimensional consisting of the position and velocity of the car, the action space consists of a one-dimensional torque. In our experiments, we provide intermediate goal positions during meta-train time, which are not available during the meta-test time. The meta-network incorporates this behavior into its loss leading to an improved exploration during the meta-test time as can be seen in fig. 5.6-a, when compared to a classical iLQR-based trajectory optimization (Tassa *et al.*, 2014). fig. 5.6-b shows the average distance between the

car and the goal at last rollout time step over several iterations of policy updates with ML^3 with and without extra information and iLQR. As we observe, ML^3 with extra information can successfully bring the car to the goal in a small amount of updates, whereas iLQR and ML^3 without extra information is not able to solve this task.

Shaping loss via expert information during meta-train time

Expert information, like demonstrations for a task, is another way of adding relevant information during meta-train time, and thus shaping the loss landscape. In *learning from demonstration (LfD)* (Pomerleau, 1991; Ng *et al.*, 2000; Billard *et al.*, 2008), expert demonstrations are used for initializing robotic policies. In our experiments, we aim to mimic the availability of an expert at meta-test time by training our meta-network to optimize a behavioral cloning objective at meta-train time. We provide the meta-network with expert state-action trajectories during train time, which could be human demonstrations or, as in our experiments, trajectories optimized using iLQR. During meta-train time, the task loss is the behavioral cloning objective $\mathcal{L}_{\mathcal{T}}(\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1}[\pi_{\theta_{\text{new}}}(a_t|s_t) - \pi_{\text{expert}}(a_t|s_t)]^2\right]$. Fig. 5.6d shows the results of our experiments in the ReacherGoal environment.

5.5 Conclusions

In this work we presented a framework to meta-learn a loss function entirely from data. We showed how the meta-learned loss can become well-conditioned and suitable for an efficient optimization with gradient descent. When using the learned meta-loss we observe significant speed improvements in regression, classification and benchmark reinforcement learning tasks. Furthermore, we showed that by introducing additional guiding information during training time we can train our meta-loss to develop exploratory strategies that can significantly improve performance during the meta-test time.

We believe that the ML^3 framework is a powerful tool to incorporate prior experience and transfer learning strategies to new tasks. In future work, we plan to look at combining multiple learned meta-loss functions in order to generalize over different families of tasks. We also plan to further develop the idea of introducing additional curiosity rewards during training time to improve the exploration strategies learned by the meta-loss.

Chapter 6

Conclusion

The work in this thesis is centered around the fundamental research question of how to learn in the action perception loop without having to re-learn everything from scratch each time. One way of thinking about answering this research question, and the perspective taken in this thesis, is to learn representations that generalize quickly to new tasks and scenarios. A representation means an abstractions that maintains an underlying structure of the learning problem for multiple tasks. For example, learning a model of the environment or learning a loss for a family of tasks. Both these things represent aspects of the world that can be learned to be shared, an re-used later to enable fast learning in new situations that potentially share aspects with the already learned experience. In this thesis we show approaches for learning such representations and presents results that indeed improve learning in the real world in terms of task performance and sample efficiency.

The first part of the thesis is concerned with how to learn better representations of the world, specifically how to learn better models of the robot dynamics (Chapter 2), contact dynamics (Chapter 3) and the dynamics of a grasped object for manipulation (Chapter 4). In Chapter 2 we show how including the learned dynamic model's uncertainty during policy optimization leads to exploratory behaviour of the policy and thus to data collection that allows for better model learning and as a consequence also improved model quality. In Chapter 3 we present a method that improves data collection by specifically considering the forward model's quality during policy learning. The presented learning objective trades off predictive task performance with forward model quality and leads to a policy that facilitates controller learning on a contact rich task. In Chapter 4 structured proprioceptive information is included when learning from images and we show how this results in more accurate and stable predictions and thus also allows for better model learning. While these approaches were used to learn representations that could be used for learning to control a robot, they are always focused on specific aspects, mostly around the body of the robot. Also, the sensory inputs considered are limited since at most two sensor modality are used. This limits how expressive the models are and, as a consequence, how well they could generalize to new situations - commonalities between different contexts might only become clear when considering multiple

sensory inputs. For example when making contact with an object, only vision can ultimately tell us (in absence of a model of the object) if we caused the contact or the contact was caused by an external force. Using more sensor modalities would allow to contextualized the observations and could help generalize learning to similar but unseen contexts. This would be a natural extension of the work presented in this thesis and would allow learning properties of the environment that go beyond the body of the robot.

In the second part of the thesis we present a framework for learning to learn, or put differently: learn a representation of the learning problem. In the current setting we can show generalization capabilities of the learned loss function. This supports our assumption that a loss function is a more general representation of the task, that can be learned from interaction with the environment and reused later for generalization purposes. However in the current state, we add minimal structure or prior knowledge to our learning problems. This allows for maximal flexibility but also makes learning in the real world more difficult since we do not consider information that we have readily available - like for example the analytical models of the robots. This information, or prior knowledge, can be used as a form of inductive bias to the learning problem, adding structure and helping with generalization and learning in the real world. We did explore some form on additional expert information in the current experimental evaluation, but we did not explore its full potential yet.

In the next section specifically these two extensions to the current work are present, alongside with further interests and future direction for research in the lifelong learning on robots.

Future Directions

A key aspect of lifelong learning in humans is the ability to learn representations that generalize quickly to new tasks (Lake *et al.*, 2017; Thrun and Pratt, 2012b). Leveraging inductive biases coming from structured prior knowledge and using multi-modal sensory information are important next steps to extend the work presented in this thesis.

Leveraging Structure as Inductive Priors:

I want to investigate how to combine structured and data driven approaches to induce prior knowledge in the learning problem. Inductive priors are essential to quickly learn and generalize (Silver, 2011). The robot offers a unique possibility - given its embodiment - to leverage strong structured priors that are shared across tasks, for learning in the action perception loop. When learning dynamics models from data the literature offers a variety of choices of machine learning algorithms used for this learning task. From linear regression (Schaal *et al.*, 2002; Haruno *et al.*,

2001), to gaussian mixture (Khansari-Zadeh and Billard, 2011; Calinon *et al.*, 2010) or gaussian process regression (Deisenroth and Rasmussen, 2011; Kocijan *et al.*, 2004) as well as using feedfoward- or recurrent neural networks for fitting the models (Lenz *et al.*, 2015; Sanchez-Gonzalez *et al.*, 2018; Rueckert *et al.*, 2017).Other works, including my work (Bechtle *et al.*, 2020b), have considered more structured approaches for model learning, including analytical priors during the learning process. For example in (Calandra *et al.*, 2015) only the residual term of the external forces is learned for inverse dynamics. In (Lutter *et al.*, 2019) the authors learn deep neural networks for each component of the equations of motion of a manipulator whereas in (Ledezma and Haddadin, 2017) the authors learn the dynamics parameters directly via gradient descent. Similar (Sutanto *et al.*, 2020) proposes a fully differentiable version of the recursive newton euler algorithm, allowing the inertial parameters to be learned using gradient descent.

In contrast to these approaches, I want to investigate how to use prior knowledge when learning to learn (l2l). In l2l the goal is to learn a representation of the learning problem. Using structured prior knowledge during meta learning will allow to leverage this additional physical information. As a result, the learned representation will inherently incorporate those aspects of the structure that are most necessary for successful learning. This is in contrast to directly using the structured information for regression. When the input-output relationship changes, which is always happening in the real world, the regressed models would need to be relearned from scratch. If a representation of the learning problem was learned, this representation might be general enough to be used in the new changed environment. For example, once we learned how to lift an object, we can easily adapt to new, heavier objects. If we cannot estimate the weight of the object, we will need a coupled of trials to learn, but the underlying mechanisms of learning to lift an object will transfer. This thinking is in contrast to my previous work, where the structure was used directly to learn the models. This can potentially make the models less adaptable since the structure provides the framing for learning. Using structure while l2l allows to encode the structured information needed for learning a representation of learning a task. Besides of speeding up the l2l process, the learning mechanisms will transfer, or at least provide a good initial guess, also when the environment changed. This is because the embodiment of the robot does not change or only changes slowly over time and will always be a good initial guess of how to approach the problem.

Multi-Modal Sensory Information:

Exploiting the multi-modality of sensory data will allow to identify, anticipate and contextualize the experience and to build models that can be shared across tasks (Lungarella *et al.*, 2003). For example when colliding with an object, combining vision proprioception and touch will allow us to understand whether we are re-

sponsible for the interaction or the object was pushed towards us. With this in mind, I want to continue my work on model based learning and move beyond low-level sensorimotor learning, towards higher level representations of the real world: where the states are not joint positions and velocities but higher level multi-modal representations of the environment. Learning from multi-modal sensor streams allows for a more consistent and robust representation of the world and the body (Lungarella *et al.*, 2003). Only when considering multiple sensory streams, cause and effect can be clearly attributed to each other. In the current literature, there is more evidence that utilizing multi-modal sensor streams improves perception (Bohg *et al.*, 2017; Edelman, 1987; Lacey and Sathian, 2016). This has been explored in various robotics applications. For instance, fusing vision and proprioception (Garcia Cifuentes *et al.*, 2017; Kappler *et al.*, 2018; Martín-Martín and Brock, 2017) or combining visual and tactile information (Martín-Martín and Brock, 2017; Lambert *et al.*, 2019; Yu and Rodriguez, 2018) has been explored for better state-estimation in applications such as object-tracking. These approaches are mostly concerned with understanding state-estimation from multi-modal sensor data and assume expert-designed low-dimensional features are extracted from each modality. Recently, there has been a recent push towards learning state representations from multi-modal data for a wide range of applications. For example, there have been many works that have explored the correlation between auditory and visual data for tasks such as speech or material recognition or for sound source localization (Ngiam *et al.*, 2011; Owens and Efros, 2018; Owens *et al.*, 2016; Yang *et al.*, 2017). Several work (Bekiroglu *et al.*, 2011; Calandra *et al.*, 2018; Gao *et al.*, 2016; Sinapov *et al.*, 2014) fuse visual and haptic data for various applications such as grasp stability assessment, manipulation, material recognition, or object categorization. In essence, fusing multiple sensor modalities for better state-estimation is common in the rigid-body tracking literature, and recently it has also been shown by (Lee *et al.*, 2019) that it leads to better learned state representations for robotic manipulation tasks. In summary, considering multi-modal data is important.

I want to work on investigating further the multi-modality of the environment when learning representations of it. How to uncover the connection between sensor streams and how to make sense of them in order to improve learning a robotic task. For example how to decide which sensor stream to trust more in a multi-modal setting, or which sensor stream should be favoured in a specific context. When manipulating an object, vision can give us a good intuition of when contact will happen. After contact with the environment happened, the tactile or force information will be important to perform a manipulation tasks. But there is no need to optimize for a force profile while the robot is not in contact. Learning representations of the environment from multi-modal data will allow to make such distinctions in the learning problem. Additionally, learning multi modal representations of the robotic task can help extrapolate similar sensory streams to different tasks. Already in my previous work (Bechtle *et al.*, 2020c; b) I showed how models learned

from multi-modal data can generalize better, however these models were learned for one specific family of tasks and from maximal two sensor streams. Generalizing the learned models across tasks is another aspect of lifelong learning that I want to investigate in the future.

Further Interests

There are many other opportunities and open questions when it comes to lifelong learning in the real world that I am excited to work on in the future:

Reinforcement Learning and Goal Discovery: Exploration and Causal Inference Reinforcement learning has shown great promise for task learning. I want to complement the model-based reinforcement learning approaches I have worked on so far, with model-free approaches for task learning and switch between these strategies depending on the context. I believe that model based and model free approaches should complement each other, and there is evidence for a combined workflow in the human brain as well. For example a model can help 'warm start' policies that are then fine-tuned from experience in a model-free fashion. The other way around, the reactive and flexible learning approach of model-free algorithms can help uncover parts of the state-space that are useful for model learning. For this I want to formulate artificial intrinsic motivation and curious behaviour for exploration and goal discovery, two important problems in reinforcement and lifelong learning. To this end I want to leverage results from causality and decision making research to guide policy and model learning. The robot acts in the real world, following the laws of physics. Causal claims are an integral part of physics and I believe that causality can be an enabler for curiosity driven and thus lifelong learning.

Active Learning in Reinforcement Learning Lifelong learning also means to decide when not to learn. If a task is sufficiently learned the experienced data might not be interesting for learning. But, if a task is to difficult to learn, it might be best to stop trying to learn and potentially revisit the task later on. How to detect these moments of change in the learning problem is still an open important question. I want to formulate active learning principles for lifelong learning. By considering learning signals available to the robot, like prediction error and uncertainty, I want trying to answer the question of what is useful to learn when and what data to keep or discard.

Algorithmic Transfer to other Domains: The robot is a feedback system, that acts in the environment and experiences the effect of its actions. There are other feedback systems in our world, for example college admission or credit score systems as well as recommender systems. In the case of college admissions, once a year new data will be available to the system and update its believe given the data. Modelling inaccuracies and biases will have a direct effect on the observed data distribution of the next cycle which can enforce bias and wrong predictions. These systems can be formulated as learning loops just like robots acting in the

world, the difference being that they often receive data at a different time scale. In the long term, I want to investigate these connections and contribute to algorithms that have a positive impact on humans lives.

Appendix A

Appendix

A.1 Background: Uncertainty in Optimal Control

In this appendix we review the fundamentals on which our work is built on. An approach enabling the inclusion of higher order statistics in the performance measure while keeping computations tractable, at least in the linear case, is to use exponential costs, as introduced by Jacobson (Jacobson, 1973). (Farshidian and Buchli, 2015) extended this work by deriving an iterative algorithm for continuous-time stochastic nonlinear optimal control problems called iterative Linear Exponential-Quadratic Optimal Control under Gaussian Process Noise (iLEG). Ponton et al. (Ponton *et al.*, 2016) extended the work from (Farshidian and Buchli, 2015) to cases where not only process noise is present but also measurement noise has to be taken into consideration. Next, we briefly present the details of the risk-sensitive iLQR algorithm, following (Farshidian and Buchli, 2015), (Ponton *et al.*, 2016) and (Jacobson, 1973).

A.1.1 Risk-sensitive iLQR

To include stochastic processes when optimizing a trajectory, it is necessary to consider a nonlinear optimal control problem where the system dynamics are defined by the following stochastic differential equation

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \mathbf{f}\left(\mathbf{x_k}, \mathbf{u_k}\right) \Delta t + \mathbf{g}\left(\mathbf{x_k}, \mathbf{u_k}\right) \Delta \omega \tag{A.1}$$

where $\mathbf{f}$ represents the dynamics of the system and $\mathbf{g}$ the stochasticity of the problem. $\Delta\omega$ is a Brownian motion with zero mean and covariance $(\mathbf{\Sigma} \cdot \Delta t)$. Following the idea of (Jacobson, 1973) to include higher order momenta of the cost function, the objective function takes the form of an exponential transformation of the performance criteria J:

$$J = \min_{\pi} \mathbb{E}\left\{\exp[\sigma \mathcal{J}\left(\pi\right)]\right\} \tag{A.2}$$

where $\mathcal{J}(\pi)$ is the performance index, which is a random variable, and a functional of the policy π. $\mathbb{E}$ is the expected value of $\mathcal{J}$ over stochastic trajectories induced by the policy π. $\sigma \in \mathbb{R}$ accounts for the sensitivity of the cost to higher order moments (variance, skewness, etc). Notably from (Farshidian and Buchli, 2015), the cost is

$$\frac{1}{\sigma}\log(J) = \mathbb{E}(\mathcal{J}^*) + \frac{\sigma}{2}\text{var}(\mathcal{J}^*) + \frac{\sigma^2}{6}\text{sk}(\mathcal{J}^*) + \cdots \tag{A.3}$$

where var and sk stand for variance and skewness and $\mathcal{J}^*$ is the optimal task cost. When $\sigma > 0$ the optimal control will be risk-averse, favoring low costs with low variance but when $\sigma < 0$ the optimal control will be risk-seeking, favoring low costs with high variance. $\sigma = 0$ reduces to the standard, risk-neutral, optimal control problem. We will exploit this property to create policies that explore regions with high uncertainty in the next sections.

A.1.2 Algorithm derivation

The algorithm begins with a nominal state and control input trajectory $\mathbf{x^n}$ and $\mathbf{u^n}$. The dynamics are linearized and the cost is quadratized along $\mathbf{u_t^n}$, $\mathbf{x_t^n}$ in terms of state and control deviations $\delta\mathbf{x_t} = \mathbf{x_t} - \mathbf{x_t^n}$, $\delta\mathbf{u_t} = \mathbf{u_t} - \mathbf{u_t^n}$ leading to the linear dynamics approximation and cost function as:

$$\delta x_{t+1} = A_t \delta x_t + B_t \delta u_t + C_t \omega_t \tag{A.4}$$

$$\mathcal{J}^* = \min_{u_t} \mathbb{E}\left[\exp\left(\sigma\left[l_T(\delta x_T) + \sum_0^{T-1} l_t(\delta x_t, \delta u_t)\right]\right)\right] \tag{A.5}$$

where $\mathbf{A_t} = \Delta t \frac{\partial \mathbf{f}}{\partial \mathbf{x_t}}$ and $\mathbf{B_t} = \Delta t \frac{\partial \mathbf{f}}{\partial \mathbf{u_t}}$
$\mathbf{C_t}$ represents how the uncertainty propagates through the system.
l is approximated as a quadratic function

$$l_t(\delta x_t, \delta u_t) = \frac{1}{2}\delta x_t^T Q_t \delta x_t + \delta x_t^T q_t + \bar{q}_t + \frac{1}{2}\delta u_t^T R_t \delta u_t + \delta u_t^T r_t + \bar{r}_t + \delta x_t^T P_t \delta u_t \tag{A.6}$$

where $\mathbf{q_t}$, $\mathbf{r_t}$, $\mathbf{Q_t}$, $\mathbf{R_t}$ and $\mathbf{P_t}$ are the coefficients of the Taylor expansion of the cost function around the nominal trajectory

The value function is also approximated by a quadratic function, and following (Jacobson, 1973) is defined by:

$$V(t, \delta x_t) = \sigma F_t \exp\left\{\sigma\left(\frac{1}{2}\delta x_t^T S_t \delta x_t + \delta x_t^T s_t + \bar{s}_t\right)\right\} \tag{A.7}$$

By substituting ((A.4)) into ((A.7)) at the next time step

$$V(t+1,\delta x_{t+1}) = \sigma F_{t+1} \exp\left\{\frac{\sigma}{2}\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right)^T S_{t+1}\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right) + \sigma\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right)^T s_{t+1} + \sigma\bar{s}_{t+1}\right\}$$

We assume gaussian noise $w_t \sim \mathcal{N}(0,\Sigma_t)$ thus we can write the pdf as: ($|\Sigma|$ denotes the determinant).

$$pdf = \frac{1}{\sqrt{|2\pi\Sigma_t|}} \exp\left\{-\frac{1}{2}\left(\omega_t - \mu_t\right)^T \Sigma_t^{-1}\left(\omega_t - \mu_t\right)\right\} \tag{A.8}$$

and the expectation can be computed as

$$\mathbb{E}\left[f(\omega)\right] = \int_{-\infty}^{+\infty} pdf(\omega).f(\omega).d\omega \tag{A.9}$$

Since we assume a zero mean random variable, the pdf reduces to:

$$\mathbb{E}\left[V(t+1,\delta x_{t+1})\right] = \int_{-\infty}^{+\infty} \frac{\sigma F_{t+1}}{\sqrt{|2\pi\Sigma_t|}} \exp\left\{-\frac{1}{2}\omega_t^T\Sigma_t^{-1}\omega_t + \frac{\sigma}{2}\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right)^T S_{t+1}\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right) + \sigma\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right)^T s_{t+1} + \sigma\bar{s}_{t+1}\right\} d\omega_t$$

(Jacobson, 1973) provides a lemma on how to integrate this, start by inspecting argument of the exponential

$$-\frac{1}{2}\omega_t^T\Sigma_t^{-1}\omega_t + \frac{\sigma}{2}\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right)^T S_{t+1}\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right) + \sigma\left(A_t\delta x_t + B_t\delta u_t + C_t\omega_t\right)^T s_{t+1} + \sigma\bar{s}_{t+1} \tag{A.10}$$

keep in mind the value function hessian S_t is symmetric positive definite, all the terms that do not contain ω_t can be expanded as,

$$N_t = \frac{\sigma}{2}\delta x_t^T A_t^T S_{t+1} A_t \delta x_t + \sigma \delta u_t^T B_t^T S_{t+1} A_t \delta x_t + \frac{\sigma}{2}\delta u_t^T B_t^T S_{t+1} B_t \delta u_t + \sigma \delta x_t^T A_t^T s_{t+1} + \sigma \delta u_t^T B_t^T s_{t+1} + \sigma \bar{s}_{t+1} \tag{A.11}$$

all the terms containing ω_t

$$Z_t = -\frac{1}{2}\omega_t^T \Sigma_t^{-1} \omega_t + \sigma \delta x_t^T A_t^T S_{t+1} C_t \omega_t + \sigma \delta u_t^T B_t^T S_{t+1} C_t \omega_t + \sigma \frac{1}{2}\omega_t^T C_t^T S_{t+1} C_t \omega_t + \sigma \omega_t^T C_t^T s_{t+1} \tag{A.12}$$

Grouping the terms in Z_t

$$Z_t = -\frac{1}{2}\omega_t^T \left(\Sigma_t^{-1} - \sigma C_t^T S_{t+1} C_t\right) \omega_t + \underbrace{\sigma \left(\delta x_t^T A_t^T S_{t+1} C_t + \delta u_t^T B_t^T S_{t+1} C_t + s_{t+1}^T C_t\right)}_{M_t^T} \omega_t \tag{A.13}$$

Now we want to create the perfect square in ω_t in the argument of the exponential. For his we write (A.12) as the following

$$Z_t = -\frac{1}{2}\left(\omega_t - \bar{\omega}_t\right)^T W_t \left(\omega_t - \bar{\omega}_t\right) \tag{A.14}$$

$$Z_t = -\frac{1}{2}\omega_t^T W_t \omega_t - \frac{1}{2}\bar{\omega}_t^T W_t \bar{\omega}_t + \bar{\omega}_t^T W_t \omega_t \tag{A.15}$$

by comparison we have

$$W_t = \Sigma_t^{-1} - \sigma C_t^T S_{t+1} C_t \tag{A.16}$$

$$\bar{\omega}_t^T W_t = M_t^T \tag{A.17}$$

$$\bar{\omega}_t = W_t^{-1} M_t \tag{A.18}$$

and to complete the square

$$-\frac{1}{2}\bar{\omega}_t^T W_t \bar{\omega}_t = -\frac{1}{2} M_t^T W_t^{-1} M_t \tag{A.19}$$

then we can go back to the terms of the exponential argument containing ω_t to

get the overall expectation

$$\mathbb{E}\left[V\left(t+1,\delta x_{t+1}\right)\right]=\int_{-\infty}^{+\infty}\frac{\sigma F_{t+1}}{\sqrt{|2\pi\Sigma_t|}}\exp\left\{N_t+\frac{1}{2}M_t^T W_t^{-1}M_t - \frac{1}{2}\left(\omega_t-\bar{\omega}_t\right)^T W_t\left(\omega_t-\bar{\omega}_t\right)\right\}d\omega_t$$

but we know that

$$\int_{-\infty}^{+\infty}\frac{1}{\sqrt{|2\pi W_t^{-1}|}}\exp\left\{-\frac{1}{2}\left(\omega_t-\bar{\omega}_t\right)^T W_t\left(\omega_t-\bar{\omega}_t\right)\right\}d\omega_t=1 \tag{A.20}$$

so we can multiply by

$$\frac{\sqrt{|2\pi W_t^{-1}|}}{\sqrt{|2\pi W_t^{-1}|}} \tag{A.21}$$

and then the expectation of the value function can be written as

$$\begin{aligned}\mathbb{E}\left[V\left(t+1,\delta x_{t+1}\right)\right]=&\frac{\sigma F_{t+1}\sqrt{|2\pi W_t^{-1}|}}{\sqrt{|2\pi\Sigma_t|}}\exp\left\{N_t+\frac{1}{2}M_t^T W_t^{-1}M_t\right\}\cdot\\ &\int_{-\infty}^{+\infty}\frac{1}{\sqrt{|2\pi W_t^{-1}|}}\exp\left\{-\frac{1}{2}\left(\omega_t-\bar{\omega}_t\right)^T W_t\left(\omega_t-\bar{\omega}_t\right)\right\}d\omega_t\end{aligned} \tag{A.22}$$

and finally we arrive at

$$\mathbb{E}\left[V\left(t+1,\delta x_{t+1}\right)\right]=\frac{\sigma F_{t+1}\sqrt{|2\pi W_t^{-1}|}}{\sqrt{|2\pi\Sigma_t|}}\exp\left\{N_t+\frac{1}{2}M_t^T W_t^{-1}M_t\right\} \tag{A.23}$$

what remains now is matching terms. For this, the recursive value function then becomes

$$V(t, \delta x_t) = \min_{\delta u_t} \left\{ \frac{\sigma F_{t+1} \sqrt{|2\pi W_t^{-1}|}}{\sqrt{|2\pi \Sigma_t|}} \exp \left\{ \sigma l_t (\delta x_t, \delta u_t) + N_t + \frac{1}{2} M_t^T W_t^{-1} M_t \right\} \right\} \quad \text{(A.24)}$$

clearly it is sufficient to minimize the argument of the exponential: expand, group, and obtain the recursions.

Start by differentiating and setting derivative to equal zero. Two easy terms are:

$$\partial_{\delta u_t} \sigma l_t (\delta x_t, \delta u_t) = \sigma \delta u_t^T R_t + \sigma r_t + \sigma \delta x_t^T P_t \quad \text{(A.25)}$$

$$\partial_{\delta u_t} N_t = \sigma \delta x_t^T A_t^T S_{t+1} B_t + \sigma \delta u_t^T B_t^T S_{t+1} B_t + \sigma s_{t+1}^T B_t \quad \text{(A.26)}$$

The third term needs some expansions before hand:

$$\partial_{\delta u_t} \left\{ \sigma^2 \delta x_t^T A_t^T S_{t+1} C_t W_t^{-1} C_t^T S_{t+1} B_t \delta u_t + \frac{\sigma^2}{2} \delta u_t^T B_t^T S_{t+1} C_t W_t^{-1} C_t^T S_{t+1} B_t \delta u_t \right.$$
$$\left. + \sigma^2 \delta u_t^T B_t^T S_{t+1} C_t W_t^{-1} C_t^T s_{t+1} \right\} \quad \text{(A.27)}$$

then the partial derivative will look like

$$\partial_{\delta u_t} \frac{1}{2} M_t^T W_t^{-1} M_t = \sigma^2 \delta x_t^T A_t^T S_{t+1} C_t W_t^{-1} C_t^T S_{t+1} B_t + \sigma^2 \delta u_t^T B_t^T S_{t+1} C_t W_t^{-1} C_t^T S_{t+1} B_t$$
$$+ \sigma^2 s_{t+1}^T C_t W_t^{-1} C_t^T S_{t+1} B_t \quad \text{(A.28)}$$

Set the derivatives to zero and after grouping things we get:

$$\begin{aligned} \mathbf{H_t} &= \mathbf{R_t} + \mathbf{B_t^T S_{t+1} B_t} + \sigma \mathbf{B_k^T S^T C W_t^{-1} C^T S_{t+1} B_t} \\ \mathbf{g_t} &= \mathbf{r_t} + \mathbf{B_t^T s_{t+1}} + \sigma \mathbf{B_t^T S_{t+1}^T C W_t^{-1} C^T s_{t+1}} \\ \mathbf{G_t} &= \mathbf{P_t^T} + \mathbf{B_t^T S_{t+1} A_t} + \sigma \mathbf{B_t^T S_t^T C W_t^{-1} C^T S_{t+1} A_t} \end{aligned} \quad \text{(A.29)}$$

Solving for the optimal control in

$$\partial_{\delta u_t} \sigma l_t (\delta x_t, \delta u_t) + \partial_{\delta u_t} N_t + \partial_{\delta u_t} \frac{1}{2} M_t^T W_t^{-1} M_t = \delta u_t^T H + \delta x_t^T G + g \quad \text{(A.30)}$$

we get:

$$\begin{aligned}\delta\mathbf{u_t} &= \mathbf{k_t} + \mathbf{K_t}\delta\mathbf{x_k} \\ \mathbf{k_t} &= -\mathbf{H_t^{-1}}\mathbf{g_t} \\ \mathbf{K_t} &= -\mathbf{H_t^{-1}}\mathbf{G_t}\end{aligned} \tag{A.31}$$

substitute δu_t back to the exponential argument and match, The three terms in the exponential argument become

$$\begin{aligned}l(\delta x_t, \delta u_t) =& \frac{1}{2}\delta x_t^T Q_t \delta x_t + \delta x_t^T q_t + \bar{q}_t + \\ &+ \frac{1}{2}\left(k_t + K_t\delta x_t\right)^T R_t \left(k_t + K_t\delta x_t\right) \\ &+ \left(k_t + K_t\delta x_t\right)^T r_t + \bar{r}_t \\ &+ \delta x_t^T P_t \left(k_t + K_t\delta x_t\right)\end{aligned} \tag{A.32}$$

$$\begin{aligned}N_t =& \frac{\sigma}{2}\delta x_t^T A_t^T S_{t+1} A_t \delta x_t + \sigma\left(k_t + K_t\delta x_t\right)^T B_t^T S_{t+1} A_t \delta x_t \\ &+ \frac{\sigma}{2}\left(k_t + K_t\delta x_t\right)^T B_t^T S_{t+1} B_t \left(k_t + K_t\delta x_t\right) \\ &+ \sigma\delta x_t^T A_t^T s_{t+1} + \sigma\left(k_t + K_t\delta x_t\right)^T B_t^T s_{t+1} + \sigma\bar{s}_{t+1}\end{aligned} \tag{A.33}$$

we can rewrite M_t as

$$M_t = \sigma\left(\left(C_t^T S_{t+1} A_t + C_t^T S_{t+1} B_t K_t\right)\delta x_t + C_t^T S_{t+1} B_t k_t + C_t^T s_{t+1}\right) \tag{A.34}$$

then, regrouping and matching the terms to construct the value function approximation, the corresponding backward recursions are

$$\mathbf{s_t} = \mathbf{q_t} + \mathbf{A_t^T s_{t+1}} + \mathbf{G_t^T k_t} + \mathbf{K_t^T H_t k_t} + \sigma\mathbf{A_t^T S_{t+1}^T C W_t^{-1} C^T s_{t+1}} \tag{A.35}$$

$$\mathbf{S_{t+1}} = \mathbf{Q_t} + \mathbf{A_t^T S_{t+1} A_t} + \mathbf{K_t^T H_t K_t} + \mathbf{G_t^T K_t} + \mathbf{K_t^T G_t} + \sigma\mathbf{A_t^T S_{t+1}^T C}\ \mathbf{W_t^{-1} C^T S_{t+1} A_t} \tag{A.36}$$

With $\sigma = 0$ the recursions revert to the usual Ricatti recursions for iLQR (Tassa *et al.*, 2014).

A.2 Details for Experiments

Throughout the experiments we chose $\sigma = -0.05$ for the curious robot, and $\sigma = 0.0$ for the normal robot. The iLQR position error weight $Q_{pos} = 5.0$ the velocity weight $Q_{vel} = 0.1$ and the torque error weight $R = 10^{-7}$. The regularization parameter λ was initialized with 1, the scaling factor for λ was 10 and the maximum λ value allowed was 1000.

A.3 MFRL and MBRL algorithms details

Algorithm 8 ML^3 for MBRL *(meta-train)*

$\phi, \leftarrow$ randomly initialize parameters
Randomly initialize dynamics model P
while not done **do**
  $\theta \leftarrow$ randomly initialize parameters
  $\tau \leftarrow$ forward unroll π_θ using P
  $\pi_{\theta_{\text{new}}} \leftarrow$ optimize$(\tau, \mathcal{M}_\phi, g, R)$
  $\tau_{\text{new}} \leftarrow$ forward unroll $\pi_{\theta_{\text{new}}}$ using P
  Update ϕ to maximize reward under τ_{new}
  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_\mathcal{T}(\tau_{\text{new}})$
  $\tau_{real} \leftarrow$ roll out $\pi_{\theta_{\text{new}}}$ on real system
  $P \leftarrow$ update dynamics model with τ_{real}
end while

Algorithm 9 ML^3 for MFRL (*meta-train*)

$I \leftarrow$ # of inner steps
$\phi \leftarrow$ randomly initialize parameters
while not done **do**
  $\theta_0 \leftarrow$ randomly initialize policy
  $\mathcal{T} \leftarrow$ sample training tasks
  $\tau_0, R_0 \leftarrow$ roll out policy π_{θ_0}
  for $i \in \{0, \ldots, I\}$ **do**
    $\pi_{\theta_{i+1}} \leftarrow$ optimize$(\pi_{\theta_i}, \mathcal{M}_\phi, \tau_i, R_i)$
    $\tau_{i+1}, R_{i+1} \leftarrow$ roll out policy $\pi_{\theta_{i+1}}$
    $\mathcal{L}^i_\mathcal{T} \leftarrow$ compute task-loss $\mathcal{L}^i_\mathcal{T}(\tau_{i+1}, R_{i+1})$
  end for
  $\mathcal{L}_\mathcal{T} \leftarrow \mathbb{E}\left[\mathcal{L}^i_\mathcal{T}\right]$
  $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_\mathcal{T}$
end while

Algorithm 10 ML^3 for RL *(meta-test)*

$\theta \leftarrow$ randomly initialize policy
for $j \in \{0, \ldots, M\}$ **do**
  $\tau, R \leftarrow$ roll out π_θ
  $\pi_\theta \leftarrow$ optimize$(\pi_\theta, \mathcal{M}_\phi, \tau, R)$
end for

We notice that in practice, including the policy's distribution parameters directly in the meta-loss inputs, e.g. mean μ and standard deviation σ of a Gaussian policy, works better than including the probability estimate $\pi_\theta(a|s)$, as it provides a direct way to update the distribution parameters using back-propagation through the meta-loss.

A.4 Experiments: MBRL

The forward model of the dynamics is represented in both cases by a neural network, the input to the network is the current state and action, the output is the next state of the environment.

The Pointmass state space is four-dimensional. For PointmassGoal $(x, y, \dot{x}, \dot{y})$ are the 2D positions and velocities, and the actions are accelerations $(\ddot{x}, \ddot{y})$.

The ReacherGoal environment for the MBRL experiments is a lower-dimensional variant of the MFRL environment. It has a four dimensional state, consisting of position and angular velocity of the joints $[\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2}]$ the torque is two dimensional $[\tau_1, \tau_2]$ The dynamics model P is updated once every 100 outer iterations with the samples collected by the policy from the last inner optimization step of that outer optimization step, i.e. the latest policy.

A.5 Experiments: MFRL

The ReacherGoal environment is a 2-link 2D manipulator that has to reach a specified goal location with its end-effector. The task distribution (at meta-train and meta-test time) consists of an initial link configuration and random goal locations within the reach of the manipulator. The performance metric for this environment is the mean trajectory sum of negative distances to the goal, averaged over 10 tasks. As a trajectory reward $R_g(\tau)$ for the task-loss (see Eq. (5.7)) we use $R_g(\tau) = -d + 1/(d + 0.001) - |a_t|$, where d is the distance of the end-effector to the goal g specified as a 2-d Cartesian position. The environment has eleven dimensions specifying angles of each link, direction from the end-effector to the goal, Cartesian coordinates of the target and Cartesian velocities of the end-effector.

The AntGoal environment requires a four-legged agent to run to a goal location. The task distribution consists of random goals initialized on a circle around the initial position. The performance metric for this environment is the mean trajectory sum of differences between the initial and the current distances to the goal, averaged over 10 tasks. Similar to the previous environment we use $R_g(\tau) = -d + 5/(d + 0.25) - |a_t|$, where d is the distance from the center of the creature's torso to the goal g specified as a 2D Cartesian position. In contrast to the ReacherGoal this environment has 33 [1] dimensional state space that describes Cartesian position, velocity and orientation of the torso as well as angles and angular velocities of all eight joints. Note that in both environments, the meta-network receives the goal information g as part of the state s in the corresponding environments. Also, in practice, including the policy's distribution parameters directly in the meta-loss inputs, e.g. mean μ and standard deviation σ of a Gaussian policy, works better

[1] In contrast to the original Ant environment we remove external forces from the state.

than including the probability estimate $\pi_\theta(a|s)$, as it provides a more direct way to update θ using back-propagation through the meta-loss.

A.6 Experiments: Regression and Classification Details

For the sine task at meta-train time, we draw 100 data points from function $y = \sin(x - \pi)$, with $x \in [-2.0, 2.0]$. For meta-test time we draw 100 data points from function $y = A \sin(x - \omega)$, with $A \sim [0.2, 5.0]$, $\omega \sim [-\pi, pi]$ and $x \in [-2.0, 2.0]$. We initialize our model f_θ to a simple feedforward NN with 2 hidden layers and 40 hidden units each, for the binary classification task f_θ is initialized via the *LeNet* architecture. For both regression and classification experiments we use a fixed learning rate $\alpha = \eta = 0.001$ for both inner (α) and outer (η) gradient update steps. We average results across 5 random seeds, where each seed controls the initialization of both initial model and meta-network parameters, as well as the the random choice of meta-train/test task(s), and visualize them in fig. 5.2. Task losses are $\mathcal{L}_{\text{Regression}} = (y - f_\theta(x))^2$ and $\mathcal{L}_{\text{BinClass}} = \mathit{CrossEntropyLoss}(y, f_\theta(x))$ for regression and classification meta-learning respectively.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *ICML*.

AG, K. (2020). Kuka ag.

Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In *NeurIPS*, pages 3981–3989.

Atkeson, C. G. and Reinkensmeyer, D. J. (1988). Using associative content-addressable memories to control robots. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 792–797. IEEE.

Atkeson, C. G. and Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 3557–3564. IEEE.

Baccarini, M., Martel, M., Cardinali, L., Sillan, O., Farnè, A., and Roy, A. C. (2014). Tool use imagery triggers tool incorporation in the body schema. *Frontiers in psychology*, **5**, 492.

Barto, A. G. (2004). Intrinsically motivated learning of hierarchical collections of skills. *International Conference on Developmental Learning and Epigenetic Robotic*, pages 112–119.

Bechtle, S., Molchanov, A., Chebotar, Y., Grefenstette, E., Righetti, L., Sukhatme, G. S., and Meier, F. (2019). Meta-learning via learned loss. *CoRR*, **abs/1906.05374**.

Bechtle, S., Lin, Y., Rai, A., Righetti, L., and Meier, F. (2020a). Curious ilqr: Resolving uncertainty in model-based rl. In *Conference on Robot Learning*, pages 162–171.

Bechtle, S., Das, N., and Meier, F. (2020b). Learning extended body schemas from visual keypoints for object manipulation. *arXiv preprint arXiv:2011.03882*.

Bechtle, S., Hammoud, B., Rai, A., Meier, F., and Righetti, L. (2020c). Leveraging forward model prediction error for learning control. *2021 IEEE International Conference on Robotics and Automation (ICRA)*.

Bekiroglu, Y., Detry, R., and Kragic, D. (2011). Learning tactile characterizations of object-and pose-specific grasps. In *2011 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 1554–1560. IEEE.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479.

Bengio, Y. and Bengio, S. (1990). Learning a synaptic learning rule. Technical Report 751, Département d'Informatique et de Recherche Opérationelle, Université de Montréal, Montreal, Canada.

Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008). Robot programming by demonstration. In *Springer Handbook of Robotics*, pages 1371–1394. Springer.

Blakemore, S.-J. and Frith, C. (2003). Self-awareness and action. *Current opinion in neurobiology*, **13**(2), 219–224.

Boedecker, J., Springenberg, J. T., Wülfing, J., and Riedmiller, M. (2014). Approximate real-time optimal control based on sparse gaussian process models. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 1–8. IEEE.

Bohg, J., Hausman, K., Sankaran, B., Brock, O., Kragic, D., Schaal, S., and Sukhatme, G. S. (2017). Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, **33**(6), 1273–1291.

Boots, B., Byravan, A., and Fox, D. (2014). Learning predictive models of a depth camera & manipulator from raw execution traces. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

Bristow, D. A., Tharayil, M., and Alleyne, A. G. (2006). A survey of iterative learning control. *IEEE control systems magazine*, **26**(3), 96–114.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Burke, E. (1958). *A Philosophical Enquiry into the Origin of our Ideas of the Sublime and Beautiful*. Columbia University Press.

Byravan, A., Leeb, F., Meier, F., and Fox, D. (2018). Se3-pose-nets: Structured deep dynamics models for visuomotor control. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*.

Calandra, R., Ivaldi, S., Deisenroth, M. P., Rueckert, E., and Peters, J. (2015). Learning inverse dynamics models with contacts. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3186–3191. IEEE.

Calandra, R., Owens, A., Jayaraman, D., Lin, J., Yuan, W., Malik, J., Adelson, E. H., and Levine, S. (2018). More than a feeling: Learning to grasp and regrasp using vision and touch. *IEEE Robotics and Automation Letters*, **3**(4), 3300–3307.

Calinon, S., D'halluin, F., Sauser, E. L., Caldwell, D. G., and Billard, A. G. (2010). Learning and reproduction of gestures by imitation. *IEEE Robotics & Automation Magazine*, **17**(2), 44–54.

Camoriano, R., Traversaro, S., Rosasco, L., Metta, G., and Nori, F. (2016). Incremental semiparametric inverse dynamics learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 544–550. IEEE.

Cardinali, L., Frassinetti, F., Brozzoli, C., Urquizar, C., Roy, A. C., and Farnè, A. (2009). Tool-use induces morphological updating of the body schema. *Current biology*, **19**(12), R478–R479.

Cardinali, L., Jacobs, S., Brozzoli, C., Frassinetti, F., Roy, A. C., and Farnè, A. (2012). Grab an object with a tool and change your body: tool-use-dependent changes of body representation for action. *Experimental Brain Research*, **218**(2), 259–271.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*.

Das, N., Bechtle, S., Davchev, T., Jayaraman, D., Rai, A., and Meier, F. (2020a). Model-based inverse reinforcement learning from visual demonstrations. *Conference on Robot Learning*.

Das, N., Bechtle, S., Davchev, T., Jayaraman, D., Rai, A., and Meier, F. (2020b). Model-based inverse reinforcement learning fromvisual demonstrations. In *4th Conference on Robot Learning (CoRL)*. IEEE.

Davchev, T., Bechtle, S., Ramamoorthy, S., and Meier, F. (2021). Learning time-invariant reward functions through model-based inverse reinforcement learning.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*.

Deisenroth, M. P. (2010). *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing.

Dolan, R. J. and Dayan, P. (2013). Goals and habits in the brain. *Neuron*, **80**(2), 312–325.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). RI^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, **abs/1611.02779**.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A. X., and Levine, S. (2018). Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, **abs/1812.00568**.

Edelman, G. M. (1987). *Neural Darwinism: The theory of neuronal group selection.* Basic books.

Farshidian, F. and Buchli, J. (2015). Risk sensitive, nonlinear optimal control: Iterative linear exponential-quadratic optimal control with gaussian noise. *arXiv preprint arXiv:1512.07173*.

Featherstone, R. (2014). *Rigid body dynamics algorithms*. Springer.

Finman, R., Whelan, T., Kaess, M., and Leonard, J. J. (2013). Toward lifelong object segmentation from change detection in dense rgb-d maps. In *2013 European Conference on Mobile Robots*, pages 178–185. IEEE.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

Florence, P., Manuelli, L., and Tedrake, R. (2020). Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, **5**(2), 492–499.

Forestier, S. and Oudeyer, P. Y. (2016). Modular active curiosity-driven discovery of tool use. *IEEE International Conference on Intelligent Robots and Systems*, **2016-Novem**, 3965–3972.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017). Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1165–1173. JMLR. org.

Gao, Y., Hendricks, L. A., Kuchenbecker, K. J., and Darrell, T. (2016). Deep learning for tactile understanding from visual and haptic data. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 536–543. IEEE.

Garcia Cifuentes, C., Issac, J., Wüthrich, M., Schaal, S., and Bohg, J. (2017). Probabilistic articulated real-time tracking for robot manipulation. *IEEE Robotics and Automation Letters*.

Gothoskar, N., Lázaro-Gredilla, M., Agarwal, A., Bekiroglu, Y., and George, D. (2020). Learning a generative model for robot control using visual feedback. *arXiv preprint arXiv:2003.04474*.

GPy (since 2012). GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

Grefenstette, E., Amos, B., Yarats, D., Htut, P. M., Molchanov, A., Meier, F., Kiela, D., Cho, K., and Chintala, S. (2019). Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*.

Grimminger, F., Meduri, A., Khadiv, M., Viereck, J., Wüthrich, M., Naveau, M., Berenz, V., Heim, S., Widmaier, F., Flayols, T., Fiene, J., Badri-Spröwitz, A., and Righetti, L. (2020). An open torque-controlled modular robot architecture for legged locomotion research. *IEEE Robotics and Automation Letters*, **5**(2), 3650–3657.

Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. (2018). Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pages 5302–5311.

Haruno, M., Wolpert, D. M., and Kawato, M. (2001). Mosaic model for sensorimotor learning and control. *Neural computation*, **13**(10), 2201–2220.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*.

Hersch, M., Sauser, E., and Billard, A. (2008). Online learning of the body schema. *International Journal of Humanoid Robotics*.

Herzog, A., Rotella, N., Mason, S., Grimminger, F., Schaal, S., and Righetti, L. (2016). Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots*, **40**(3), 473–491.

Hikita, M., Fuke, S., Ogino, M., Minato, T., and Asada, M. (2008). Visual attention by saliency leads cross-modal body representation. In *2008 7th IEEE International Conference on Development and Learning*, pages 157–162. IEEE.

Hoffmann, M., Marques, H., Arieta, A., Sumioka, H., Lungarella, M., and Pfeifer, R. (2010). Body schema in robotics: a review. *IEEE Transactions on Autonomous Mental Development*, **2**(4), 304–324.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2020). Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*.

Houthooft, R., Chen, Y., Isola, P., Stadie, B. C., Wolski, F., Ho, J., and Abbeel, P. (2018). Evolved policy gradients. In *NeurIPS*, pages 5405–5414.

Hsu, K., Levine, S., and Finn, C. (2018). Unsupervised learning via meta-learning. *CoRR*, **abs/1810.02334**.

Ishikawa, T., Tomatsu, S., Izawa, J., and Kakei, S. (2016). The cerebro-cerebellum: Could it be loci of forward models? *Neuroscience research*, **104**, 72–79.

Ito, M. (1970). Neurophysiological aspects of the cerebellar motor control system. *Int. J. Neurol.*, **7**, 126–179.

Jackson, P. L., Lafleur, M. F., Malouin, F., Richards, C. L., and Doyon, J. (2003). Functional cerebral reorganization following motor sequence learning through mental practice with motor imagery. *Neuroimage*, **20**(2), 1171–1180.

Jacobson, D. H. (1973). Optimal Stochastic Linear Systems with Exponential Performance Criteria and Their Relation to Deterministic Differential Games. *IEEE Transaction on Automatic Control*, **18**.

Jarquín, G., Escande, A., Arechavaleta, G., Moulard, T., Yoshida, E., and Parra-Vega, V. (2013). Real-time smooth task transitions for hierarchical inverse kinematics. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 528–533.

Jordan, M. I. and Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive science*, **16**(3), 307–354.

Kagan, J. (1972). Motives and development. *Journal of Personality and Social Psychology*, **22**(1), 51–66.

Kappler, D., Meier, F., Issac, J., Mainprice, J., Cifuentes, C. G., Wüthrich, M., Berenz, V., Schaal, S., Ratliff, N., and Bohg, J. (2018). Real-time perception meets reactive motion generation. *IEEE Robotics and Automation Letters*.

Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, **27**(5), 943–957.

Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, **3**(1), 43–53.

Kocijan, J., Murray-Smith, R., Rasmussen, C. E., and Girard, A. (2004). Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE.

Koert, D., Maeda, G., Neumann, G., and Pcters, J. (2018). Learning coupled forward-inverse models with combined prediction errors. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2433–2439. IEEE.

Kulkarni, T. D., Gupta, A., Ionescu, C., Borgeaud, S., Reynolds, M., Zisserman, A., and Mnih, V. (2019). Unsupervised learning of object keypoints for perception and control. In *Advances in neural information processing systems*, pages 10724–10734.

Lacey, S. and Sathian, K. (2016). Crossmodal and multisensory interactions between vision and touch. In *Scholarpedia of Touch*, pages 301–315. Springer.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, **40**.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413.

Lambert, A. S., Mukadam, M., Sundaralingam, B., Ratliff, N., Boots, B., and Fox, D. (2019). Joint inference of kinematic and force trajectories with visuo-tactile sensing. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3165–3171. IEEE.

Lambeta, M., Chou, P., Tian, S., Yang, B., Maloon, B., Most, V. R., Stroud, D., Santos, R., Byagowi, A., Kammerer, G., Jayaraman, D., and Calandra, R. (2020). Digit: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation. *IEEE Robotics and Automation Letters*.

Laversanne-Finot, A., Péré, A., and Oudeyer, P.-Y. (2018). Curiosity driven exploration of learned disentangled goal spaces. *arXiv preprint arXiv:1807.01521*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

Ledezma, F. D. and Haddadin, S. (2017). First-order-principles-based constructive network topologies: An application to robot inverse dynamics. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 438–445. IEEE.

Lee, M. A., Zhu, Y., Srinivasan, K., Shah, P., Savarese, S., Fei-Fei, L., Garg, A., and Bohg, J. (2019). Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE.

Lenz, I., Knepper, R. A., and Saxena, A. (2015). Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy.

Levine, S. and Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079.

Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on Machine Learning*, pages 1–9.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, **17**(1), 1334–1373.

Li, K. and Malik, J. (2016). Learning to optimize. *arXiv preprint arXiv:1606.01885*.

Limanowski, J. and Friston, K. (2020). Active inference under visuo-proprioceptive conflict: Simulation and empirical results. *Scientific reports*, **10**(1), 1–14.

Loewenstein, G. (1994). The psychology of curiosity: A review and reinterpretation. *Psychological bulletin*, **116**(1), 75.

Lopes, M., Lang, T., Toussaint, M., and Oudeyer, P.-Y. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in neural information processing systems*, pages 206–214.

Lungarella, M., Metta, G., Pfeifer, R., and Sandini, G. (2003). Developmental robotics: a survey. *Connection science*, **15**(4), 151–190.

Lutter, M., Ritter, C., and Peters, J. (2019). Deep lagrangian networks: Using physics as model prior for deep learning. *arXiv preprint arXiv:1907.04490*.

Maclaurin, D., Duvenaud, D., and Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122.

Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., and Batra, D. (2019). Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Manuelli, L., Gao, W., Florence, P., and Tedrake, R. (2019). kpam: Keypoint affordances for category-level robotic manipulation. *International Symposium on Robotics Research (ISRR)*.

Manuelli, L., Li, Y., Florence, P., and Tedrake, R. (2020). Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning.

Martel, M., Finos, L., Koun, E., Farnè, A., and Roy, A. C. (2021). The long developmental trajectory of body representation plasticity following tool use. *Scientific Reports*, **11**(1), 1–15.

Martín-Martín, R. and Brock, O. (2017). Cross-modal interpretation of multi-modal sensor streams in interactive perception based on coupled recursion. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3289–3295. IEEE.

Martinez-Cantin, R., Lopes, M., and Montesano, L. (2010). Body schema acquisition through active learning. In *2010 IEEE international conference on robotics and automation*, pages 1860–1866. IEEE.

Meier, F., Kappler, D., and Schaal, S. (2018). Online learning of a memory for learning rates. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2425–2432. IEEE.

Mendonca, R., Gupta, A., Kralev, R., Abbeel, P., Levine, S., and Finn, C. (2019). Guided meta-policy search. *arXiv preprint arXiv:1904.00956*.

Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. (2019). Learning unsupervised learning rules. In *International Conference on Learning Representations*.

Miall, R. C. and Wolpert, D. M. (1996). Forward models for physiological motor control. *Neural networks*, **9**(8), 1265–1279.

Miller, W. (1987). Sensor-based control of robotic manipulators using a general learning algorithm. *IEEE Journal on Robotics and Automation*, **3**(2), 157–165.

Minderer, M., Sun, C., Villegas, R., Cole, F., Murphy, K. P., and Lee, H. (2019). Unsupervised learning of object structure and dynamics from videos. In *Advances in Neural Information Processing Systems*, pages 92–102.

Moore, A. (1990). Efficient memory-based learning for robot control. *PhD thesis, University of Cambridge*.

Nabeshima, C., Kuniyoshi, Y., and Lungarella, M. (2006). Adaptive body schema for robotic tool-use. *Advanced Robotics*, **20**(10), 1105–1126.

Ng, A. Y., Russell, S. J., *et al.* (2000). Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670.

Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. Y. (2011). Multimodal deep learning. In *ICML*.

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2008). Computed torque control with nonparametric regression models. In *2008 American Control Conference*, pages 212–217. IEEE.

OpenAI Gym (2019).

Owens, A. and Efros, A. A. (2018). Audio-visual scene analysis with self-supervised multisensory features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–648.

Owens, A., Isola, P., McDermott, J., Torralba, A., Adelson, E. H., and Freeman, W. T. (2016). Visually indicated sounds. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2405–2413.

Parascandolo, G., Huttunen, H., and Virtanen, T. (2017). Taming the waves: sine as activation function in deep neural networks.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, **113**, 54–71.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-Driven Exploration by Self-Supervised Prediction. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, **2017-July**, 488–489.

Peters, J. and Schaal, S. (2006). Learning operational space control. In *Robotics: Science and Systems*.

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, **3**(1), 88–97.

Ponton, B., Schaal, S., and Righetti, L. (2016). Risk sensitive nonlinear optimal control with measurement uncertainty. *CoRR*.

Ponton, B., Herzog, A., Del Prete, A., Schaal, S., and Righetti, L. (2018). On time optimization of centroidal momentum dynamics. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5776–5782, Brisbane, Australia. IEEE.

Pybullet (since 2012). a python module for physics simulation in robotics, games and machine learning. `http://pybullet.org/`.

Rolf, M., Steil, J. J., and Gienger, M. (2010). Learning flexible full body kinematics for humanoid tool use. In *2010 International Conference on Emerging Security Technologies*, pages 171–176. IEEE.

Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-Carlo Simulation (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Rueckert, E., Nakatenus, M., Tosatto, S., and Peters, J. (2017). Learning inverse dynamics models in o (n) time with lstm networks. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 811–816. IEEE.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. (2018). Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR.

Santucci, V. G., Oudeyer, P.-Y., Barto, A., and Baldassarre, G. (2020). Editorial: Intrinsically motivated open-ended learning in autonomous robots. *Frontiers in Neurorobotics*, **13**, 115.

Sawyer (since 2012). Rethink robotics. `https://www.rethinkrobotics.com/sawyer/`.

Schaal, S. (1997). Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046.

Schaal, S., Atkeson, C. G., and Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, **17**(1), 49–60.

Schillaci, G., Hafner, V. V., and Lara, B. (2012). Coupled inverse-forward models for action execution leading to tool-use in a humanoid robot. In *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 231–232. IEEE.

Schmidhuber, J. (1987). Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. Institut für Informatik, Technische Universität München.

Schulman, J., Heess, N., Weber, T., and Abbeel, P. (2015). Gradient estimation using stochastic computation graphs. In *NeurIPS*, pages 3528–3536.

Shyam, P., Jaskowski, W., and Gomez, F. (2018). Model-based active exploration. *arXiv preprint arXiv:1810.12162*.

Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2010). *Robotics: modelling, planning and control*. Springer Science & Business Media.

Silver, D. L. (2011). Machine lifelong learning: Challenges and benefits for artificial general intelligence. In *International conference on artificial general intelligence*, pages 370–375. Springer.

Sinapov, J., Schenck, C., and Stoytchev, A. (2014). Learning relational object categories using behavioral exploration and multimodal perception. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5691–5698. IEEE.

Singh, S., Barto, A., and Chentanez, N. (2004). Intrinsically motivated reinforcement learning. *18th Annual Conference on Neural Information Processing Systems (NIPS)*.

Singh, S., Lewis, R. L., Barto, A. G., and Sorg, J. (2010). Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective. *IEEE Transactions on Autonomous Mental Development*, **2**(2), 70–82.

Sober, S. J. and Sabes, P. N. (2005). Flexible strategies for sensory integration during motor planning. *Nature neuroscience*, **8**(4), 490–497.

Stepanova, K., Pajdla, T., and Hoffmann, M. (2019). Robot self-calibration using multiple kinematic chains—a simulation study on the icub humanoid robot. *IEEE Robotics and Automation Letters*, **4**(2), 1900–1907.

Sturm, J., Plagemann, C., and Burgard, W. (2009). Body schema learning for robotic manipulators from visual self-perception. *Journal of Physiology-Paris*, **103**(3-5), 220–231.

Sung, F., Zhang, L., Xiang, T., Hospedales, T., and Yang, Y. (2017). Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*.

Sutanto, G., Wang, A., Lin, Y., Mukadam, M., Sukhatme, G., Rai, A., and Meier, F. (2020). Encoding physical constraints in differentiable newton-euler algorithm. In *Learning for Dynamics and Control*, pages 804–813. PMLR.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*.

Talvitie, E. (2014). Model regularization for stable sample rollouts. In *UAI*, pages 780–789.

Talvitie, E. (2016). Self-correcting models for model-based reinforcement learning. *arXiv preprint arXiv:1612.06018*.

Tani, J. (2016). *Exploring robotic minds: actions, symbols, and consciousness as self-organizing dynamic phenomena*. Oxford University Press.

Tanneberg, D., Peters, J., and Rueckert, E. (2019). Intrinsic motivation and mental replay enable efficient online adaptation in stochastic recurrent networks. *Neural Networks*, **109**, 67–80.

Tassa, Y., Mansard, N., and Todorov, E. (2014). Control-limited differential dynamic programming. *IEEE International Conference on Robotics and Automation, ICRA*.

Thrun, S. and Pratt, L. (2012a). *Learning to learn*. Springer Science & Business Media.

Thrun, S. and Pratt, L. (2012b). *Learning to learn*. Springer Science & Business Media.

Ulbrich, S., de Angulo, V. R., Asfour, T., Torras, C., and Dillmann, R. (2009). Rapid learning of humanoid body schemas with kinematic bézier maps. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pages 431–438. IEEE.

Van Beers, R. J., Sittig, A. C., and Gon, J. J. D. v. d. (1999). Integration of proprioceptive and visual position-information: An experimentally supported model. *Journal of neurophysiology*, **81**(3), 1355–1364.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754.

White, R. W. (1959). Motivation reconsidered: The concept of competence. *Psychological review*, **66**(5), 297.

Whittle, P. (1981). Risk-Sensitive Linear-Quadratic-Gaussian Control. *Advances in Applied Probability*, **13**(4), 764–777.

Williams, G., Wagener, N., Goldfain, B., Drews, P., Rehg, J., Boots, B., and Theodorou, E. (2017). Information theoretic MPC for model-based reinforcement learning. In *ICRA*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, **8**, 229–256.

Wolpert, D. M. and Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural networks*, **11**(7-8), 1317–1329.

Wolpert, D. M., Ghahramani, Z., and Jordan, M. I. (1995). An internal model for sensorimotor integration. *Science*, **269**(5232), 1880–1882.

Wolpert, D. M., Miall, R. C., and Kawato, M. (1998). Internal models in the cerebellum. *Trends in cognitive sciences*, **2**(9), 338–347.

Wu, L., Tian, F., Xia, Y., Fan, Y., Qin, T., Lai, J.-H., and Liu, T.-Y. (2018). Learning to teach with dynamic loss functions. In *NeurIPS*, pages 6467–6478.

Xu, Z., van Hasselt, H., and Silver, D. (2018). Meta-gradient reinforcement learning. In *NeurIPS*, pages 2402–2413.

Yadan, O. (2019). A framework for elegantly configuring complex applications. Github.

Yang, X., Ramesh, P., Chitta, R., Madhvanath, S., Bernal, E. A., and Luo, J. (2017). Deep multimodal representation learning from temporal data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5447–5455.

Yoshikawa, Y., Hosoda, K., and Asada, M. (2003). Does the invariance in multimodalities represent the body scheme?-a case study with vision and proprioception. In *2nd Intelligent Symposium on Adaptive Motion of Animals and Machines*. Citeseer.

Yu, K.-T. and Rodriguez, A. (2018). Realtime state estimation with tactile and visual sensing for inserting a suction-held object. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1628–1635. IEEE.

Yu, T., Finn, C., Xie, A., Dasari, S., Zhang, T., Abbeel, P., and Levine, S. (2018). One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*.

Zhang, K., Sharma, M., Veloso, M., and Kroemer, O. (2019). Leveraging multimodal haptic sensory data for robust cutting. In *Proceedings of IEEE-RAS International Conference on Humanoid Robots*.

Zou, H., Ren, T., Yan, D., Su, H., and Zhu, J. (2019). Reward shaping via meta-learning. *arXiv preprint arXiv:1901.09330*.