Combining Learning and Structure for Robotic Manipulation

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. Alina Kloss

aus Bietigheim-Bissingen

Tübingen

2020

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

Tag der mündlichen Qualifikation:	16.12.2020
Stellvertretender Dekan:	Prof. Dr. József Fortágh
1. Berichterstatter:	Assist. Prof. Dr. Jeannette Bohg
2. Berichterstatter:	Prof. Dr. Hendrik P. A. Lensch

Abstract

Household robots have been a long standing promise of robotics. But despite of decades of robotic research and progress in the field, we still do not encounter many robots in our everyday life. Creating the robotic helpers envisioned in science-fiction movies would of course require advances towards more general, human-like artificial intelligence. However, today's robots also already struggle with much simpler tasks, like accurately and reliably manipulating novel objects based only on sensory data. Such *sensory-motor skills* often have challenging dynamics and are further complicated by the fact that the state information that can be extracted from raw sensory input is noisy or even incomplete.

Over the last decade, a perceived dichotomy between *model-based* and *data-driven* approaches has often shaped the discussion about how to implement such robotic skills: In the "traditional", model-based approach, engineers manually define a skill as a combination of models, state representations and algorithms. The data-driven approach, in contrast, relies on powerful function approximators like deep neural networks to learn robot skills from large amounts of training data. Both approaches have advantages and limitations that are in many aspects complementary.

The model-based approach requires no training data, is transparent and relies on models that are grounded in the laws of physics and are thus universally applicable. However, formulating accurate and efficient models can be difficult for many tasks, especially when it comes to interpreting sensory signals. For example, manually modeling the appearance of all objects that a robot might encounter in a typical household is clearly impossible.

The data-driven approach, on the other hand, requires little prior knowledge about the task. Deep neural networks can directly predict robot actions from raw sensory inputs and have shown impressive results especially in domains like computer vision, for which specifying analytical models is difficult. The disadvantages of learning are the large amounts of required training data, which can be difficult to obtain when robots are involved, and the black-box nature of the resulting learned solutions. In particular, guaranteeing safety and correct performance for data not seen during training (generalization) is currently an open problem.

In this thesis, we propose to combine learning and structure to get the best of both worlds. We hypothesize that including learning into a model-based approach could fill the gaps where no analytical model could be found, speed up approaches when the analytical solution is too slow or improve existing models with a learned component. Furthermore, introducing more structure into learning methods could help to reduce the amount of necessary training data and improve the generalization performance and interpretability of the learned models.

We evaluate this hypothesis on the sensory-motor skill of planar pushing. This task requires the robot to translate and rotate an object on a flat surface using a cylindrical pusher given only visual observations of the system. While the task is simple to describe, the associated dynamics are already quite challenging with different contact modes and complex friction forces.

Our evaluation starts at the level of the individual models and representations required for perception and prediction. We compare a purely learned approach for predicting the outcome of pushing actions from sensory data to a combined one, that trains a neural network for perception end-to-end through an analytical model for prediction. While the purely learned approach reaches the best prediction accuracy when training and test data are similar, the combined method generalizes better to pushes and objects not seen during training and is more data-efficient.

Analytical dynamics models and physically meaningful state representations are of course not the only way to provide structure to learning approaches. While models describe certain aspects of the system, algorithms contain knowledge about how to solve a given task. On the example of state estimation with Bayesian filters, we demonstrate that embedding model-learning into differentiable algorithms facilitates learning in comparison to unstructured models. In addition, it allows to optimize the learned models specifically for their respective algorithm and can be advantageous when designing the models manually is difficult.

Finally, we address the complete task of planning pushing actions based on visual input. Its main challenges are the computational cost of optimizing pushing motions and contact locations simultaneously and the uncertainty induced by imperfect perception. We present an approach that combines learning and structure to address both of these challenges. First, we improve the planning efficiency by decomposing the problem into contact point selection and pushing motion optimization. This allows focusing the expensive motion optimization on few promising contact locations. We compare using a learned or an analytical model for both planning steps. Second, we combine learning-based perception with a physically meaningful state representation and an explicit state estimation algorithm to increase the accuracy of our approach as compared to a previously published, purely learned method.

In summary, this thesis shows that the model-based and the data-driven approach do not have to be understood as exclusive choices but can rather be viewed as extreme points in a trade-off between providing prior knowledge and leaving flexibility for learning new solutions. On the example of planar pushing, we demonstrate how combining learning and structure can make sensory-motor skills more robust, general and data-efficient.

Kurzfassung

Haushaltsroboter gehören schon seit Langem zu den Versprechungen der Robotik. Doch trotz jahrzehntelanger Forschung und großen Fortschritten auf dem Gebiet treffen wir in unserem Alltag nach wie vor nicht viele Roboter an. Um die Roboter-Helfer zu entwickeln, die sich das Science-Fiction Kino ausmalt, bedürfte es natürlich großer Fortschritte hin zu einer menschenähnlicheren, allgemeinen künstlichen Intelligenz. Jedoch scheitern unsere heutigen Roboter auch noch häufig an viel einfacheren Aufgaben, etwa daran, nur anhand von Sensor-Daten unbekannte Gegenstände genau und verlässlich zu handhaben. Die physikalischen Prozesse hinter solchen *sensorisch-motorischen Fähigkeit* sind in vielen Fällen komplex und schwierig mathemathisch zu modellieren. Erschwerend hinzu kommt zudem, dass aus den sensorischen Daten oftmals nur fehlerbehaftete oder unvollständige Informationen über den aktuellen Zustand des Systems gewonnen werden können.

In den letzten zehn Jahren hat eine gefühlte Zweiteilung zwischen modellbasierten und datenbasierten Methoden häufig die Diskussion darüber bestimmt, wie solche Roboter-Fähigkeiten implementiert werden sollten. Bei der 'traditionellen' modellbasierten Herangehensweise definieren Ingenieure die Fähigkeit von Hand, indem sie Modelle, Repräsentationen des Systemzustands und Algorithmen miteinander kombinieren. Die datenbasierte Herangehensweise hingegen versucht mit Hilfe von Lernverfahren wie künstlichen neuronalen Netzen die gewünschte Fähigkeit direkt anhand großer Mengen von Trainingsbeispielen zu erlernen. Beide Methoden haben sowohl Vor- als auch Nachteile, die jedoch in vielerlei Hinsicht komplementär zueinander sind.

Die modellbasierte Herangehensweise kommt größtenteils ohne Trainingsdaten aus, ist verständlich und nutzt Modelle, die auf den Gesetzen der Physik basieren und somit universell gültig sind. Jedoch kann es oft schwierig sein, exakte und effiziente Modelle zu formulieren, besonders wenn es darum geht, sensorische Signale zu interpretieren. Es ist zum Beispiel offensichtlich unmöglich, manuell Modelle für das Aussehen jeglicher Objekte zu erstellen, denen ein Roboter in einem durchschnittlichen Haushalt begegnen könnte.

Dagegen braucht es kaum vorheriges Wissen über die Aufgabe um datenbasierte Methoden zu nutzen. Künstliche neuronale Netze mit vielen Schichten von Neuronen können direkt sensorische Daten verarbeiten, um die nächsten Aktionen eines Roboters zu planen. Besonders in Bereichen wie dem maschinellen Sehen, wo es schwer ist analytische Modelle zu formulieren, erreichen neuronale Netze beeindruckende Erfolge. Ein Nachteil neuronaler Netze ist ihr hoher Bedarf an Trainingsdaten. Diese sind zum Teil schwierig zu beschaffen, insbesondere wenn dazu reale Roboter genutzt werden müssen. Zusätzlich sind die gelernten Lösungen intransparent und es ist daher schwer zu garantieren, dass sie auch dann sicher und zuverlässig funktionieren, wenn die Eingabedaten nicht den im Training Gesehen entsprechen.

In dieser Dissertation schlagen wir daher vor, maschinelles Lernen mit analytischen Strukturen zu kombinieren um von den Vorteilen beider Herangehensweisen zu profitieren. Unser Hypothese ist, dass Lernverfahren im Rahmen der modellbasierten Herangehensweise genutzt werden können, wenn kein analytisches Modell gefunden werden konnte oder die analytische Lösung zu rechenaufwändig ist. Auch könnten Komponenten gelernt werden, um existierende Modelle zu verbessern. Im Gegenzug könnte die bessere Strukturierung von Lernverfahren die Menge an benötigten Trainingsdaten reduzieren, die Übertragbarkeit gelernter Funktionen auf neue Situationen verbessern und die gelernten Lösungen transparenter machen.

Wir überprüfen diese Hypothese anhand einer sensorisch-motorischen Fähigkeit, bei der der Roboter ein Objekt mit Hilfe eines zylindrischen Fingers auf einer flachen Oberfläche verschieben und drehen muss. Dazu erhält er ausschließlich unverarbeitete visuelle Informationen über das System. Diese Aufgabe ist zwar an sich einfach zu beschreiben, die zugehörigen physikalischen Abläufe sind jedoch auf Grund von komplexen Reibungsinteraktionen und wechselnden Kontakt-Modi schwer zu modellieren.

Wir beginnen unsere Untersuchung auf der Ebene der einzelnen Modelle und Systembeschreibungen, die für die visuelle Wahrnehmung und die Vorhersage der Konsequenzen einer Schiebebewegung notwendig sind. Dazu vergleichen wir ein reines Lernverfahren mit einer kombinierten Methode, die ein neuronales Netz für die Verarbeitung der sensorischen Daten direkt durch ein analytisches Modell der physikalischen Prozesse trainiert. Wir stellen fest, dass die gelernte Lösung die genauesten Vorhersagen trifft, solange auf Daten getestet wird, die den Trainingsdaten ähneln. Die kombinierte Methode hingegen lässt sich besser auf Bewegungen und Objekte übertragen, die nicht in den Trainingsdaten enthalten waren und benötigt darüber hinaus weniger Trainingsdaten.

Physikalische Modelle und Repräsentationen des Systemzustands sind natürlich nicht die einzige Möglichkeit um Lernverfahren mit Struktur zu versehen. Neben Modellen, die einzelne Aspekte des Systems beschreiben, enthalten Algorithmen Wissen darüber, wie ein gegebenes Problem gelöst werden kann. Am Beispiel von Bayesschen Filtern demonstrieren wir, dass das Einbetten von neuronalen Netzen in differenzierbare Algorithmen das Lernen vereinfacht. Darüber hinaus erlaubt das Verfahren es, die gelernten Modelle speziell für den jeweiligen Algorithmus zu optimieren.

Abschließend betrachten wir die Aufgabe, anhand visueller Daten Schiebeaktionen zu planen, in ihrer Gesamtheit. Die größten Herausforderungen dabei sind es, gleichzeitig die Schiebebewegungen und den Kontaktpunkt dafür zu optimieren, sowie mit der Unsicherheit durch die unvollkommene Wahrnehmung umzugehen. Wir präsentieren eine Methode, die Lernverfahren mit modellbasierten Herangehensweisen kombiniert um beiden Herausforderungen zu begegnen. Zuerst verbessern wir die Effizienz bei der Planung, indem wir die Wahl des Kontaktpunktes und die Optimierung der Schiebebewegung dort getrennt betrachten. So kann die aufwändige Optimierung auf wenige, vielversprechende Punkte beschränkt werden. Für beide Schritte vergleichen wir ein analytisches und ein gelerntes Modell. Darüber hinaus erreicht unsere Methode eine höhere Genauigkeit als eine bisher veröffentlichte Methode, welche ausschließlich auf Lernverfahren basiert, indem wir neuronale Netze für die Wahrnehmung mit einer physikalisch bedeutsamen Beschreibung des Systemzustands und einem Bayesschen Filter Algorithmus zur Schätzung dieses Systemzustands kombinieren.

Zusammenfassend zeigt diese Dissertation, dass die modellbasierte und die datenbasierte Herangehensweise sich nicht gegenseitig ausschließen müssen. Stattdessen können sie als Extreme in einer Abwägung dazwischen verstanden werden, wie viel Vorwissen in einem Ansatz bereit gestellt wird und wie viel Flexibilität für das Erlernen neuer Lösungen gelassen wird. Am Beispiel der Schiebeaufgabe zeigen wir, wie sensorisch-motorische Fähigkeiten durch die Kombination von Lernverfahren und analytischen Strukturen robuster, allgemeingültiger und dateneffizienter werden können.

Acknowledgments

This thesis marks the end of a five year journey full of ups and downs, with parts I thoroughly enjoyed but also parts that brought me to my limits. If there is one advice that I would like to give to students at the beginning of their journey, it would be to take care of yourself and not let other people's pace put pressure on you. Sometimes, good research needs time and in my case, the only person that was not supportive when things took longer than expected turned out to be myself. In the end, the best and most important part about this journey were the people that I was lucky enough to share it with.

First and foremost, this is my supervisor Jeannette Bohg, who did a great job and was always there when I needed her. Even when her path took her to Stanford, she stayed firmly at my side and was maybe more present remotely than some supervisors who have their office one corridor away from yours. Thank you for all your great advice, your patience and for all the times you got up early to meet with your student at the other side of the ocean.

I also want to thank Georg Martius, who took me in when the Autonomous Motion Department (AMD) was shut down and I suddenly found myself without an official group. Thank you for your help and advice and all the additional work you faced to support me, even though we did not have much time to do research together. Professor Hendrik Lensch and Jörg Stückler completed my Thesis Advisory Committee and supported me in those last two years that turned out to be so different from what I had expected when I started.

The official end of AMD of course did not mean that I ever had to spent my time at the institute alone. I want to thank all the great folks at AMD and later at the Intelligent Control Systems and the Movement Generation and Control groups for making the every-day research fun, for their helpful advice, the funny and weird discussions over lunch and the wonderful sushi-evenings. Alonso Marco-Valle especially has been a great friend right from the start. I am so glad we could go on this adventure called PhD together.

Lidia Pavel, Kara Loehr, Leila Masri, Vincent Berenz and Felix Grimminger are the people that keep the research running at AMD and IMPRS. Thank you all for your kind, quick and competent help with administrative and technical issues.

I also encountered lots of great people during my visit at MIT. Both Josh Tenebaum's and Alberto Rodriguez' group made me feel at home right away. Special thanks go to Alberto, Josh and Jiajun Wu, who made this stay possible for me and gave great input to our joint research project. Amir Soltani helped me find my way around MIT and has been a great friend and office mate. Finally, I want to thank Maria Bauza Villalonga for all the effort she put into the project to make sure that I got all the real-robot time I needed.

But even for PhD students, there is life outside of the institute, and mine was made so much better by the people close to me. Felix Widmaier was always there for me, had my back when things got stressful and sometimes overwhelming and is simply the best thing that ever happened to me. My parents also always supported me and made sure that I did not lose track of the nicer things in life. Thank you for everything and sorry that I made you worry sometimes.

I also want to thank Sarah Borner and all my friends outside of the institute for always cheering me up when I needed it, for long board-game nights, for cake, for hikes, festivals, COVID-19 Discord calls and all the other great activities, and especially for having patience with me when another upcoming deadline made me vanish into the lab again.

And finally, I want to thank a bunch of people who do not even know I exist and still helped me countless times during this journey with their art. To name a few, Neil Gaiman and Brandon Sanderson could always cheer me up with their stories and provided a sometimes much-needed contrast to the mathematical world of science. Insomnium, Alcest, Parkway Drive and all the others made me feel alive with their music, calmed me down when I felt stressed or gave me energy when I needed to get things done. Thanks to you, this has been no *Slow Surrender...*

This work was supported by the Max Planck Society and the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

Contents

Notation and Symbols

Throughout this thesis, we use the following notation conventions:

- Functions are denoted by italic lowercase symbols followed by their arguments in parentheses, e.g. $f(\mathbf{x})$. If the arguments are not specified, we use $\cdot$ as a placeholder, e.g. $p(\cdot)$

- Scalars are denote by italic lowercase symbols, e.g. a, μ

- Vectors are denoted by bold lowercase symbols, e.g. $\mathbf{x} = \big(x, y\big)^T$

- Matrices are denoted by bold uppercase symbols, e.g. $\mathbf{Q}, \boldsymbol{\Sigma}$

- Sets are denoted by calligraphic uppercase symbols, e.g. $\mathcal{X}$

- Predictions of quantities are indicated with a hat, e.g. $\hat{\mathbf{x}}$

- $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$

- $\mathbf{I}_n$ denotes an identity matrix with n rows and columns

- Where relevant, the timestep t at which a quantity is evaluated is denoted by a subscript, e.g. $\mathbf{x}_t, \boldsymbol{\Sigma}_t$

Chapter 1

Introduction

1.1 Motivation

Household robots have been a long standing promise of robotics. But despite decades of robotic research and progress in the field, we still do not encounter many robots in our everyday life. The only notable exceptions are robots that mow lawns, vacuum floors or clean pools - all tasks that do not require complex interactions with objects or humans in the environment but can essentially be solved by driving around in an enclosed area.

So what are the challenges that prevent us from developing robots for tasks that require more interaction? One problem is surely that we are still far from developing the kind of general *artificial intelligence* (AI) that would be necessary for building capable robot helpers like Wall-E (Stanton *et al.*, 2008). But even tasks where the high-level planning can be solved by today's AI agents can often not be executed reliably by our robots.

Let's for example think about a robot that plays board-games with its owner. The recent game of Go that Google Deepminds's AI AlphaGo (Silver *et al.*, 2016) played and won against the human Go champion Lee Sedol demonstrated impressively that AIs are able to compete with and win against the best human players in such games. However, for moving and placing the Go pieces on the board, AlphaGo still relied on a human. Of course, building a Go-playing robot was not the aim of the project and a team of good engineers could undoubtedly build a system that could have physically played against Sedol.

But the core of the problem still becomes evident if we imagine trying to build a general board-game playing robot: Even if the robot knows how to play all possible games well, it would also need to be able to manipulate all possible variants of pieces, from beautifully carved wooden chess figures to the small magnetic stones found in portable game sets (see Figure 1.1). And the users will not only expect their robot to deal with boards and pieces in all shapes, sizes and colors. It should also be able to play in different places with vastly different appearances and lighting-conditions. Furthermore, the robot should still be able to play reliably and safely if the user's cat decides to walk over the board or if a child tries to interfere with

Figure 1.1: Left: “White” Queen pieces from three different chess sets. Right: A portable Reversi game with small, magnetic stones. Board-game pieces come in a huge variety of shapes, colors and materials, which would make it hard for a robot to recognize and manipulate them reliably.

the robot’s sensors or movements.

The robotic systems that we build for manipulation tasks today do not display this level of robustness and generalization ability. Seemingly simple tasks like grasping (Du *et al.*, 2020) or pushing (Stüber *et al.*, 2020) previously unseen objects based only on sensory data still remain active areas of research.

1.2 Sensory-Motor Skills: An Example

In this thesis, we will mostly focus on what we call *sensory-motor* skills: The robot interacts with an object based only on raw sensory information. Specifically, we study *planar pushing* as an exemplary task: The robot is equipped with a single “finger” that it can use to push an object to a desired position and into a desired orientation. As sensory information, it receives RGBD camera images such as the view show in Figure 1.2.

Although the state-space of the object is rather low-dimensional (2D position plus orientation), pushing is already a quite complex manipulation problem: The system is under-actuated and the relationship between the push and the object movement is highly non-linear as well as non-smooth. The pusher can for example slide along the object during pushing and the dynamics change drastically when it transitions between sticking and sliding contact or makes and breaks contact. We will discuss one analytical model for describing the dynamics of pushing in detail in Section 2.2.

But not only the dynamics of the pushing task are challenging: We attempt to solve the task using the raw image data as input, which means that crucial information like the current object pose or its shape has to be extracted from this high-dimensional and potentially noisy sensory signal. This is a challenging problem and will usually result in inaccurate state estimates - which has to be taken into

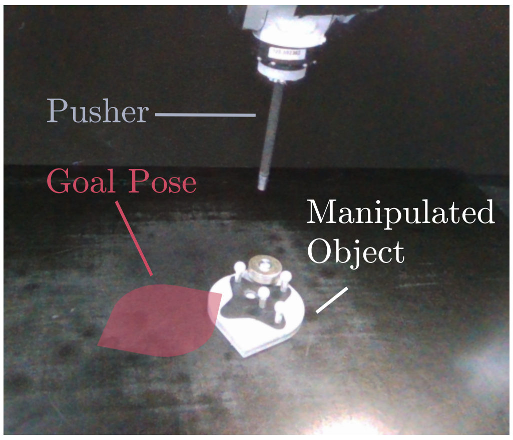


Figure 1.2: The exemplary task of *planar pushing* that we study in this thesis. The robot is equipped with a cylindrical, vertical pusher that it can use to make point contact with the manipulated object. The task is to move the object into a desired position and orientation as indicated by the red overlay. In this thesis, we only make use of visual sensor data in the form of RGB images and depth information.

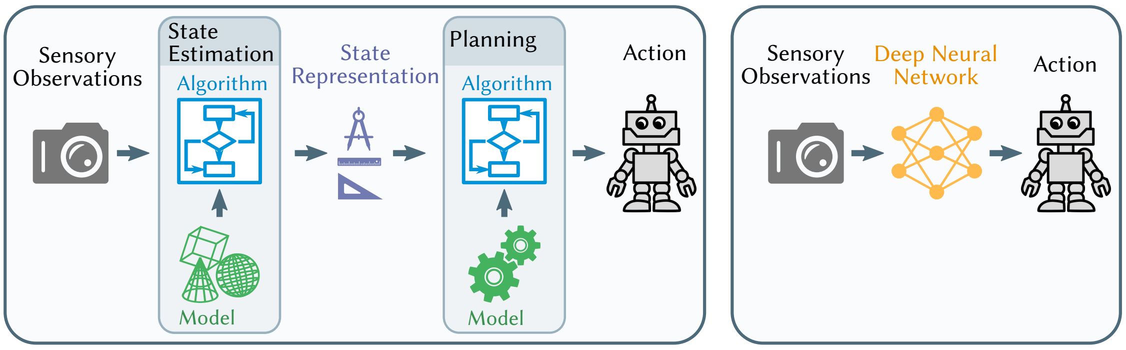
account when planning actions. In addition, we might not even be able to extract all necessary information for solving the task from the sensory input directly. For example, attributes like friction coefficients or the mass of an object cannot be determined from single images. And even aspects like the shape of an object that are generally observable will not be recovered reliably when the object is occluded in the current view.

1.3 Learning and Structure

On a very abstract level, we can view a sensory-motor skill like planar pushing as a function $s(\cdot)$ that accepts the current goal $\mathbf{g}_t$ and the current sensory observations $\mathbf{D}_t$ and outputs robot actions $\mathbf{u}_t$ to achieve the goal: $s(\mathbf{D}_t, \mathbf{g}_t) = \mathbf{u}_t$. Our task is then to implement this function such that some quality metric, for example the distance to the goal after applying the actions, is optimized over all possible observations and goals.

Over the last decade, a perceived dichotomy between *model-based* and *data-driven* approaches has often shaped the discussion about how to implement robotic skills: In the “traditional”, model-based approach, robotic engineers manually define the function $s(\cdot)$ as a combination of models, state representations and algorithms. The data-driven approaches, in contrast, seek to learn $s(\cdot)$ using general function approximators like deep neural networks (DNN) and large amounts of training data.

We can see these approaches as extreme cases on a trade-off curve between the amount of prior *structure* that we impose on $s(\cdot)$ and the flexibility that we leave for *learning* the shape of $s(\cdot)$ by fitting its response to the training data. In the following, we discuss the advantages and limitations of these two extreme cases on the example of planar pushing and motivate what could be gained by combining both, *learning* and *structure*.



(a) Model-based approach (b) Data-driven approach

Figure 1.3: Schematic comparison between the purely model-based and the purely data-driven approach to robotics. In the model-based approach, the task is decomposed into sub-problems, e.g. state estimation and planning. Each sub-problem is solved by an **algorithm** which relies on hand-designed **models**. The different modules communicate through predefined intermediate **representations**. In the data-driven approach, a **deep neural network** or a comparable function approximator learns how to solve the task *end-to-end* from labeled training data.

1.3.1 Model-Based Approach

Under the purely model-based approach, the first step to solving a task like planar pushing is to decompose the problem into smaller sub-problems that can be addressed individually. For sensory-motor skills like pushing, this decomposition will usually result in modules for *state estimation*, *planning* and *control*. The state estimation module processes the raw sensory observations $\mathbf{D}$ and outputs a representation of the system state $\mathbf{x}$. In the pushing example, $\mathbf{x}$ could correspond to the position and orientation of the object while $\mathbf{D}$ would be camera and depth images. The planning module then takes the estimated state $\mathbf{x}$ and the goal $\mathbf{g}$ as input to plan a sequence of pushing actions that will move the object towards the goal pose. Finally, the control module is responsible for translating these desired pusher motions into motor-commands that are executed to move the robot arm. In the remainder of this thesis, we will assume that the control module is given and focus on state-estimation and planning.

As depicted in Figure 1.3a, each module consists of an algorithm that uses knowledge about the system in the form of models. For example, the state-estimation module could use a 3D model of the pushed object to determine its position and orientation from an observed point-cloud using the RANSAC algorithm (Fischler and Bolles, 1981). For planning, an analytical model that describes how the object will move in response to a push could be used to compute the action necessary to push the object from its estimated pose into the desired pose.

Advantages

One big advantage of fully specifying the structure of function $s(\cdot)$ is that the resulting pipeline is interpretable for humans. The intermediate representations usually correspond to physical quantities that we can easily relate to the real world, which greatly facilitates the development and debugging process. In addition, we can understand why and how each of the algorithms and models produced the results they did and what assumptions they make. In many cases, we can give estimates of the uncertainty and noise in the system or even formulate theoretical guarantees like error bounds or optimality guarantees for some or all of the components of $s(\cdot)$.

A further advantage of this approach is that it requires little to no training data, since $s(\cdot)$ only has very few degrees of freedom[1]. That also means that we do not need to worry much about generalization - in some sense, the generalization ability of a model- based solution is mostly determined by our ability to formulate models that hold for the complete task domain.

Disadvantages

On the other hand, the performance of a robotic skill implemented with the fully model-based approach is always limited by how well its structure explains the real process.

Decomposing the problem into a sequence of modules can easily lead to a loss of information when the intermediate state representations are not expressive enough to capture all available information from the input data. The state representation that one module outputs might also not be the ideal input for the next module or, vice versa, the ideal input for one module might not be easy to produce with the previous one.

In addition, it can be difficult to formulate accurate models and optimal algorithms. For example, when modeling the dynamics of a system, it is often necessary to make simplifying assumptions for the computations to become tractable and fast enough for execution on a robot. These assumptions rarely hold in practice and the resulting models will thus not be accurate. The same holds true for many algorithms we use. The Kalman filter for state estimation is, for example, provably optimal, but only for linear systems with Gaussian additive noise (Kalman, 1960).

But things get even worse for perception: While providing 3D models of objects that a robot should be able to detect and localize is a viable solution for laboratory experiments with a limited set of objects, it is clearly impossible to model every object a robot could ever encounter in the real world. Until now, it remains unclear how a “general” perception algorithm would work and what kind of models it would require.

[1]Some algorithms for example have parameters that might require data for tuning

Summary

In summary, the traditional approach to robotics mainly relies on prior knowledge about the system, which is supplied by the engineer in the form of (i) a decomposition of the task into sub-problems, (ii) their intermediate representations, (iii) algorithms for addressing the sub-problems and (iv) the models used by these algorithms.

It requires little to no training data and the resulting implementation of $s(\cdot)$ is transparent and interpretable for humans. With the exemption of few tuning parameters, the performance of a system that executes $s(\cdot)$ is fully determined by the accuracy of the used models and the correctness and optimality of the algorithms.

The biggest challenge and limitation of this approach is formulating models and algorithms that are at the same time accurate and efficient enough to be executed on a robot and general enough to apply to every situation the robot may encounter in the real world.

1.3.2 Data-Driven Approach

In its most extreme form, the data-driven approach implements the function $s(\cdot)$ with a single trainable model that learns $s(\cdot)$ *end-to-end.* As shown in Figure 1.3b, this means it takes the raw sensory observations as input and learns to directly output motor commands for the robot.

The use of learning techniques however does not mean that $s(\cdot)$ has zero prior structure: For example, the user still has to decide which type of trainable model to fit against the training data. While there are several possibilities, in this thesis we will focus on *deep neural networks* (DNN). Since the immensely successful application of DNNs for image classification by Krizhevsky *et al.* (2012) less than a decade ago, *deep learning* methods have rapidly become the state-of-the-art for many different problems - from computer vision (Russakovsky *et al.*, 2015; Zendel *et al.*, 2018) over natural language processing (Otter *et al.*, 2020) to playing games like Go or StarCraft II (Vinyals *et al.*, 2019). In the context of robotics, vision-based sensory-motor skills have been successfully addressed with Reinforcement Learning techniques, for example in the seminal work by Levine *et al.* (2016). Such approaches are sometimes referred to as *pixel-to-torque* methods.

One key ingredient for this success is large amounts of labeled training data. These labels can take different forms, for example manually annotated images for classification or segmentation, ground truth trajectories for state estimation or rewards for the outcome of robot trials to complete a given task.

Another important characteristic of today's deep models is a very high number of trainable parameters (or *weights*). The main idea is that the larger the number of weights, the larger the family of functions that the network can represent. Specifically, the Universal Approximation Theorem (Cybenko, 1989; Leshno *et al.*, 1993) states that a large enough neural network with most standard activation functions can fit any function arbitrarily well within a closed domain.

Advantages

The main appeal of the data-driven approach is that it does not require any prior knowledge about the function one tries to model. This is, of course, especially useful for problems where applying the model-based approach is difficult - be it because we struggle to find the right structure or because the required models and algorithms are not computationally tractable.

For example, sensory-motor sills such as vision-based grasping require detecting and localizing objects in images and extracting relevant object properties. While model-based approaches can solve these problems using predefined descriptions of known objects, it is not clear how they could generalize to previously unseen objects. Vision-based Reinforcement Learning methods like QT-Opt (Kalashnikov *et al.*, 2018) can instead learn to directly predict actions from images and have demonstrated remarkable generalization performance to novel objects.

Disadvantages

While the Universal Approximation Theorem promises that for every function $s(\cdot)$ a network exists that can learn it, finding this network and the correct set of weights can be challenging. In practice, designing and training DNN is often a trial-and-error process which requires experience and has been compared to the methods of medieval alchemists by Rahimi and Recht (2017).

While too small networks will not be able to fit $s(\cdot)$ well (*underfitting*), having too many trainable parameters can result in *overfitting*. In this case, the network can perfectly reproduce the training data but does not perform well on previously unseen input values. A particularly illustrative example for this problem are *adversarial examples*: Tiny perturbations of the network input that are not even perceptible for humans but alter the response of a trained DNN dramatically (Szegedy *et al.*, 2014). *Generalization* to data not seen during training is thus one of the main issues that prevents DNNs from being used in consumer robots today. We will discuss the function approximation abilities and limitations of DNNs in more detail in Chapter 2.1.

The limited ability of DNN to generalize also means that the larger the domain over which we want to learn $s(\cdot)$, the more training data will be necessary to achieve a good network performance. However, producing the required amount of data can be challenging and costly when real robots are involved. And while simulation offers a wealth of easily accessible data, the gap between simulation and the real world is often too large to successfully transfer models without additional finetuning on real-world data (James *et al.*, 2019).

Another criticism of the data-driven approach is that the learned models and internal representations are not easily interpretable for humans. This makes it difficult to understand failures of the trained networks or to guarantee their safety - which is, of course, especially important for physical robots. Furthermore, the lack of an explicit problem decomposition with interpretable intermediate representations also makes it difficult to reuse parts of an existing solution for new but related tasks.

Summary

To summarize, the data-driven approach to robotics relies on powerful function approximators like deep neural networks and large amounts of labeled training data. It has shown impressive results and excels especially in domains like computer vision, for which specifying analytical models can be difficult.

The main challenges when applying deep learning to robotics is the large amount of required training data that may be hard to come by, and the black-box nature of the learned function. In particular, guaranteeing safety and correct performance for data not seen during training (generalization) is currently an open problem (Corso *et al.*, 2020).

1.3.3 Combining Learning and Structure in Robotics

If we look at the advantages and disadvantages of the purely model-based and the purely data-driven approach for implementing robotic skills, it becomes clear that the two approaches are in many aspects complementary: Including learning into a model-based approach could fill the gaps where no analytical model could be found, speed up approaches when the analytical solution is too slow or improve existing models with a learned component. And introducing more structure into learning methods could help to reduce the amount of necessary training data and improve the generalization performance and interpretability of the learned models. In this thesis, we will thus investigate different ways to combine learning and structure for robotic sensory-motor skills.

1.4 Outline and Contributions

The next chapter reviews the foundations for this thesis. In Chapters 3 - 5, we will present different ways of combining learning approaches with structure from the model-based approach.

Chapter 3: Models for Perception and Prediction In Chapter 3, we start by investigating the models and representations required for perception and prediction in terms of accuracy, data-efficiency and generalization to novel input data. For this, we look at the task of predicting the effects of a pushing action based on sensory input.

We propose to combine an analytical model for describing the dynamics of pushing with a DNN for perception. The network is trained end-to-end through the dynamics model and can thus learn to extract an optimal state representation from the raw sensory data. At the same time, the analytical model may be able to regularize the learned component to prevent overfitting. To further increase the accuracy of the predictions, we also investigate augmenting the analytical dynamics model with a learned error-correction term.

We compare these combined approaches to using one DNN for both, perception and predictions. A systematic evaluation on a large real-world dataset shows two main advantages of the combined approach: Compared to a pure neural network, it significantly (i) reduces the required training data and (ii) improves generalization to novel pushing actions and object shapes. This chapter is based on (Kloss *et al.*, 2020b).

Chapter 4: State Estimation and Uncertainty Chapter 4 takes the focus from the individual models to the algorithms that use them. While models describe certain aspects of the system, algorithms contain knowledge about how to solve tasks. Discovering this higher-level logic on top of learning the required models can be challenging for DNNs. By embedding the model-learning into the given structure of algorithms, we can thus facilitate learning and optimize the models for their respective algorithm.

Specifically, we look at the problem of state estimation: Many robotic applications require maintaining a probabilistic belief about the system state over a series of robot actions. These state estimates serve as input for planning and decision making and provide feedback during task execution. Having information about how certain the system is about its predictions enables identifying failures and thus makes the system more safe.

Recursive Bayesian filtering algorithms address the state estimation problem, but they require models of process dynamics and sensory observations as well as the noise characteristics of these models. Recently, multiple works have demonstrated that these models can be learned by training DNN components end-to-end through differentiable versions of Recursive filtering algorithms.

The aim of Chapter 4 is to to highlight the advantages of such *differentiable filters* (DF) over both, unstructured learning approaches and Bayesian filtering algorithms with manually designed models, while also providing concrete practical advice on how to train differentiable filters.

We implement DFs with four different underlying filtering algorithms and compare them in extensive experiments. Specifically, we (i) evaluate different implementation choices and training approaches, (ii) investigate the advantages of learning complex models of uncertainty in DFs and (iii) compare the DFs among each other and to unstructured LSTM models (Hochreiter and Schmidhuber, 1997).

Chapter 5: Planning Contact Interactions In Chapter 5, we finally address the full task of planning pushing actions based on raw sensory input. Planning such contact interactions is one of the core challenges of many robotic tasks. The main problems are the computational cost of optimizing actions and contact points simultaneously and the uncertainty induced by imperfect perception and state estimation.

We present an approach that combines learning and structure to address both of these challenges. First, we improve planning efficiency by further decomposing the planning problem into one step for selecting contact points and one for optimizing the direction and magnitude of the pushing motion taken there. Explicitly reasoning over contact locations allows us to focus the expensive motion optimizations on few promising contact locations. Second, we compare using a learned or an analytical model for proposing contact points as well as for optimizing the pushing actions. And finally, as proposed in Chapters 3 and 4, we combine learning-based perception with a physically meaningful state representation and explicit state estimation algorithms to increase the accuracy and generalizability of our approach under partial observability.

In simulation and real-world experiments on the task of planar pushing, we show that our method is efficient and achieves a higher manipulation accuracy than a previous vision-based method that relies entirely on learning. This chapter is based on (Kloss *et al.*, 2020a).

Chapter 2

Foundations

In this chapter, we introduce the foundations for this thesis. We start by reviewing some concepts from deep learning that are important for understanding the capability and limitations of DNNs to learn different functions.

Then we turn to the model-based approach: Here, we start with introducing the notation that we will use throughout this thesis for describing dynamic systems. Afterwards, we discuss different Bayesian filtering algorithms for state estimation that will serve as an example for combining algorithmic structure with learning in Chapter 4. Finally, we take a closer look at the planar pushing task that serves as experimental test bed for many of the concepts introduced in this thesis.

2.1 Approximating Functions with Deep Learning

Deep Learning is a huge and rapidly growing field and for a detailed introduction, we refer the reader to books like Goodfellow *et al.* (2016). In this section, we want to focus on understanding how DNN approximate their target functions and what that means for their ability to accurately model the training data and to generalize to input data not seen during training.

2.1.1 Universal Approximation Theorem

The Universal Approximation Theorem for Neural Networks as first formulated by Cybenko (1989) and Hornik *et al.* (1989) states that a single-layer network with a large enough number of neurons (with sigmoidal activation functions) can approximate any (Borel) measurable function to any desired accuracy. It was later extended to multiple layers and broader classes of activation functions (e.g. by Leshno *et al.* (1993)). This theorem clearly explains the appeal of DNNs for addressing any task that requires modeling an unknown function. However, for understanding the potential but also the limitations of DNNs, it is worthwhile looking at how the theorem can be intuitively visualized.

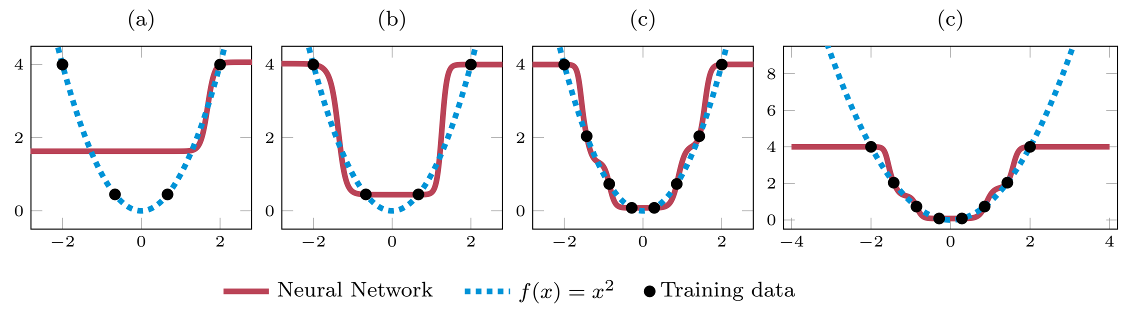


Figure 2.1: Fitting the function $f(x) = x^2$ given observations from the interval $[-2, 2]$. (a) With a single sigmoid neuron in the hidden layer and four training examples, the network cannot fit $f(x)$ well. (b) Fitting the function with four neurons in the hidden layer works better. (c) Increasing the number of neurons and training examples makes the approximation more accurate. (d) The neural network approximation however only fits $f(x)$ within the training data range of $x \in [-2, 2]$.

Let's assume we have a one-dimensional input variable x and a neural network with one hidden and one output layer. The neurons in the hidden layer are of the form $n(x) = \sigma(wx + b)$ with $\sigma(x) = \frac{1}{(1+\exp(-30x))}$ a (relatively sharp) sigmoid function. w and b are the trainable weight and bias of the neuron. The output layer does not use a non-linearity but simply computes a weighted sum of the hidden layer neurons. We try to model a simple quadratic function $f(x) = x^2$ for $x \in [-2, 2]$.

As shown in Figure 2.1 (a), with a single neuron, we can barely fit the target function using only a constant function with a single step. But when we add more neurons, the steps can be arranged such that the network output coarsely follows the training data from the target function. The more neurons, i.e. steps, we add, the more accurate the approximation becomes (Figure 2.1 (b) and (c)). As long as we provide enough data points in the target interval to place the steps, we can thus approximate every continuous function in this way with arbitrary accuracy.

Note that while the shown examples are the result of a real neural network training, they were specifically designed to illustrate the Universal Approximation Theorem. With a less sharp non-linearity, for example the frequently used ReLU function, the network would be able to fit its target function much better.

What does this example teach us about neural networks? The first important aspect is visualized in Figure 2.1 (d): Given enough neurons and training examples, the neural network can approximate $f(x)$ accurately within the training data range of $x \in [-2, 2]$. However, if we query the network with an input value not seen during training, e.g. $x = 3$, the network's predictions can become arbitrarily bad. This limited ability to *extrapolate* makes it hard to apply neural networks in cases where it is difficult to obtain training data from the full expected range of input values.

A second point that is important to keep in mind is that while the network's approximation can get arbitrarily accurate, it does not learn the actual mathemat-

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Figure 2.2: Finding the right DNN architecture for a task can be difficult and is often a trial-and-error process in practice. Image "Machine Learning" from `xkcd.com/1838` © Randall Munroe (license CC BY-NC 2.5)

ical square operation. In this concrete example, the neural network approximation thus turns out to be much more complex than the true underlying function. If we had instead used a least squares method to fit a polynomial to the training data, we would have been able to recover the true underlying function with much less training data.

To summarize, neural networks are powerful models that can in theory approximate any continuous function inside the domain of their training data. However, they usually do not learn the true underlying function and are thus not able to extrapolate beyond their training domain. If we have prior knowledge about the structure of the target function, we can potentially reduce the amount of required training data and improve the accuracy and extrapolation performance of the fitted function by restricting the approximator to a smaller family of functions that reflects our prior.

2.1.2 Underfitting and Overfitting

While the Universal Approximation Theorem guarantees the existence of a DNN that can model any given function, it does not offer any guidance on how many parameters this DNN needs. In practice, finding the right DNN architecture is often a trial-and-error process that is sometimes made fun of in satirical depictions of deep learning such as Figure 2.2.

To judge how well a given network learns to approximate its target function, we can look at two metrics: The *training risk* shows how well the network is able to predict the data points in its training set. More relevant in practice is the *test risk*, which is the expected loss of the network for data points that were not seen during training. Since, as discussed in the previous section, extrapolation is generally difficult for neural networks, here, we assume that the test data comes from the same distribution as the training data.

Plotting the training and test risk of function approximators against their capacity often results in curves similar to the one shown in Figure 2.3 (Belkin *et al.*, 2019). Models with very small capacity typically struggle to fit the training data and consequently have high training and test risk. These models are said to *underfit* the training data and if we increase their capacity, both training and test risk will decrease.

However, at some point, we can often see that while the training risk keeps decreasing, the test risk starts to rise again. An explanation for this is that the model starts to learn overly complex functions that can fit the training data points well, but behave badly when evaluated at new data points. This behavior is called *overfitting*.

One way to reduce overfitting in models with higher capacity would be to increase the amount of training examples. If this is not possible, we can attempt to *regularize* the model to favor learned functions with certain properties we belief to be desirable. For example, the commonly used *weight decay* regularization penalizes solutions with high network weights and thus favors learning simpler functions. It is also possible to directly change the structure of the network to reflect priors that we may have about the target function. A famous example for this are Convolutional Neural Networks (CNN) for image processing: By applying the same operation (kernel) at every input pixel, they enforce translation invariance on the learned functions.

Nowadays, DNNs often have many more parameters than necessary for fitting the training data optimally. Research shows that for such *over-parameterized* models, the test risk can start to decreases again. Belkin *et al.* (2019) suggest that increasing the capacity of the models in the over-parameterized regime increases the number of different solutions with optimal training risk they could learn. With the right regularization, the training algorithm can thus select solutions that perform well on both, training and test data. If the test risk obtained in the over-parameterized regime will be lower than the optimal trade-off between over- and underfitting depends on the problem and on how well the priors we express through regularization match the underlying function.

2.2 Modeling Dynamic Systems

In this section, we introduce the terminology that we will use throughout this thesis to describe dynamic systems, such as a robot interacting with its environment.

We describe the state of a system at time t by the vector $\mathbf{x}_t$ that contains all quantities that are of interest for the task the robot tries to solve. This state changes over time, possibly in response to robot actions $\mathbf{u}_t$, according to an un-

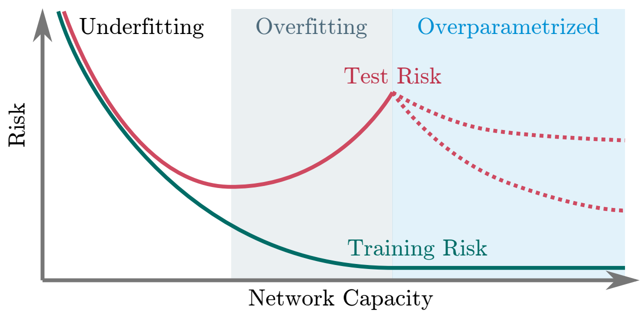


Figure 2.3: Training and test risk as a function of the model capacity: Models with a too small capacity *underfit* the training and test data. However, when the capacity increases over a certain point, the models can start to *overfit* to the training data, which results in an increasing test risk despite the decreasing training risk (grey area). Over-parameterized models like many DNN have more parameters than necessary for optimally fitting the training data (blue area). In this regime, the test risk often decreases again, as the number of possible learned functions that match he training data well increases. Different *regularization* methods can be used to reduce overfitting and guide the learning towards solutions with desirable properties.

known stochastic process. We approximate this process with the *process model* $f(\mathbf{x}, \mathbf{u})$ that describes the dynamics and an associated *process noise* model.

The robot cannot access the true system state directly, but it can use its sensors to get noisy observations $\mathbf{z}_t$ of the system. Just like the system dynamics, the exact process by which observations are generated for a certain state is unknown. We model it with the *observation model* $h(\mathbf{z})$ and the associated *observation noise* model. In many cases, $\mathbf{z}$ will not contain information about all components of $\mathbf{x}_t$.

2.3 Bayesian Filtering for State Estimation

The first requirement for solving any robotic task is knowing the current state of the system. However, as stated before, a robot usually has no access to the true system state. Instead, it has to *estimate* the state based on sensory observations and its internal models of the system.

Filtering algorithms address the state estimation problem by maintaining a probabilistic belief $bel(\mathbf{x})$ about the latent state $\mathbf{x}$ of the system over time given an initial belief $bel(\mathbf{x}_0)$ and a sequence of sensory observations $\mathbf{z}_{0...t}$ and robot actions $\mathbf{u}_{0...t-1}$. Formally, a filtering algorithms seeks the posterior distribution $p(\mathbf{x}_t|\mathbf{x}_{0...t-1}, \mathbf{u}_{0...t-1}, \mathbf{z}_{0...t})$.

Bayesian filters make the Markov assumption, i.e. that the distribution of the future states and observations is conditionally independent from the history of past states and observations given the current state. This assumption makes it possible to compute $p(\mathbf{x}_t|\mathbf{x}_{0...t-1}, \mathbf{u}_{0...t-1}, \mathbf{z}_{0...t})$ recursively from $p(\mathbf{x}_{t-1}|\mathbf{x}_{0...t-2}, \mathbf{u}_{0...t-2}, \mathbf{z}_{0...t-1})$

using the process model f and observation model h.

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{q}_{t-1}) \qquad \mathbf{z}_t = h(\mathbf{x}_t, \mathbf{r}_t)$$

The random variables $\mathbf{q}$ and $\mathbf{r}$ represent the process and observation noise.

In Chapter 4, we will investigate differentiable versions of four different nonlinear Bayesian filtering algorithms: The Extended Kalman filter (EKF), the Unscented Kalman filter (UKF), a sampling-based variant of the UKF that we call Monte Carlo Unscented Kalman filter (MCUKF) and the Particle filter (PF). In the following, we briefly review these algorithms. For more details on EKF, UKF and PF, we refer to Thrun *et al.* (2005).

2.3.1 Kalman Filter

The Kalman filter (Kalman, 1960) is a closed-form solution to the filtering problem for systems with a linear process and observation model and Gaussian additive noise:

$$f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} + \mathbf{q}_t \qquad \mathbf{q}_t \sim N(0, \mathbf{Q}_t) \tag{2.1}$$
$$h(\mathbf{x}_t) = \mathbf{H}\mathbf{x}_t + \mathbf{r}_t \qquad \mathbf{r}_t \sim N(0, \mathbf{R}_t) \tag{2.2}$$

The belief about the state $\mathbf{x}$ is represented by the mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ of a normal distribution. At each timestep, the filter predicts $\hat{\boldsymbol{\mu}}_t$ and $\hat{\boldsymbol{\Sigma}}_t$ using the process model. The innovation $\mathbf{i}_t$ is the difference between the predicted and actual observation and is used to correct the prediction. The Kalman Gain $\mathbf{K}$ trades-off the process noise $\mathbf{Q}$ and the observation noise $\mathbf{R}$ to determine the magnitude of the update.

Prediction Step:

$$\hat{\boldsymbol{\mu}}_t = \mathbf{A}\boldsymbol{\mu}_{t-1} + \mathbf{B}\mathbf{u}_t \tag{2.3}$$
$$\hat{\boldsymbol{\Sigma}}_t = \mathbf{A}\boldsymbol{\Sigma}_{t-1}\mathbf{A}^T + \mathbf{Q}_{t-1} \tag{2.4}$$

Update Step:

$$\mathbf{S}_t = \mathbf{H}\hat{\boldsymbol{\Sigma}}_t\mathbf{H}^T + \mathbf{R}_t \tag{2.5}$$
$$\mathbf{K}_t = \hat{\boldsymbol{\Sigma}}_t\mathbf{H}^T\mathbf{S}_t^{-1} \tag{2.6}$$
$$\mathbf{i}_t = \mathbf{z}_t - \mathbf{H}\hat{\boldsymbol{\mu}}_t \tag{2.7}$$
$$\boldsymbol{\mu}_t = \hat{\boldsymbol{\mu}}_t + \mathbf{K}_t\mathbf{i}_t \tag{2.8}$$
$$\boldsymbol{\Sigma}_t = (\mathbf{I}_n - \mathbf{K}_t\mathbf{H})\hat{\boldsymbol{\Sigma}}_t \tag{2.9}$$

2.3.2 Extended Kalman Filter (EKF)

The EKF (Sorenson, 1985) extends the Kalman filter to systems with non-linear process and observation models. It replaces the linear models for predicting $\hat{\boldsymbol{\mu}}$ in Equation 2.3 and the corresponding observations $\hat{\mathbf{z}}$ in Equation 2.7 with non-linear models $f(\cdot)$ and $h(\cdot)$. For predicting the state covariance $\boldsymbol{\Sigma}$ and computing the Kalman Gain $\mathbf{K}$, these non-linear models are linearized around the current mean of the belief. The Jacobians $\mathbf{F}_{|\mu_t}$ and $\mathbf{H}_{|\mu_t}$ replace $\mathbf{A}$ and $\mathbf{H}$ in Equations 2.4 - 2.6 and 2.9. This first-order approximation can be problematic for systems with strong non-linearity, as it does not take the uncertainty about the mean into account (Van Der Merwe, 2004).

2.3.3 Uncentered Kalman Filter (UKF)

The UKF (Julier and Uhlmann, 1997; Van Der Merwe, 2004) was proposed to address the aforementioned problem of the EKF. Its core idea, the *Unscented Transform* (Julier and Uhlmann, 1997), is to represent a Gaussian random variable that undergoes a non-linear transformation by a set of specifically chosen points in state space, the so called *sigma points* $\boldsymbol{\chi} \in \mathcal{X}$.

$$\boldsymbol{\chi}^0 = \boldsymbol{\mu} \qquad \boldsymbol{\chi}^i = \boldsymbol{\mu} \pm (\sqrt{(n+\kappa)\boldsymbol{\Sigma}})_i \ \forall i \in \{1...n\} \tag{2.10}$$

$$w^0 = \frac{\kappa}{\kappa + n} \qquad w^i = \frac{0.5}{\kappa + n} \qquad \forall i \in \{1...2n\} \tag{2.11}$$

Here, n is the number of dimensions of the state $\mathbf{x}$. Each sigma point $\boldsymbol{\chi}^i$ has a weight w^i. The parameter κ controls the spread of the sigma points and how strongly the original mean $\boldsymbol{\chi}^0$ is weighted in comparison to the other sigma points.

The statistics of the transformed random variable can then be calculated from the transformed sigma points. For example, in the prediction step of the UKF, the non-linear transform is the process model (Equation 2.12) and the new mean and covariance of the belief are computed in Equations 2.13 and 2.14.

$$\hat{\mathcal{X}}_t = f(\mathcal{X}_{t-1}, \mathbf{u}_t) \tag{2.12}$$

$$\hat{\boldsymbol{\mu}}_t = \sum_i w^i \hat{\boldsymbol{\chi}}^i_t \tag{2.13}$$

$$\hat{\boldsymbol{\Sigma}}_t = \sum_i w^i (\hat{\boldsymbol{\chi}}^i_t - \hat{\boldsymbol{\mu}}_t)(\hat{\boldsymbol{\chi}}^i_t - \hat{\boldsymbol{\mu}}_t)^T + \mathbf{Q}_t \tag{2.14}$$

In theory, the UKF conveys the nonlinear transformation of the covariance more faithfully than the EKF and is thus better suited for strongly non-linear problems

(Thrun *et al.*, 2005). In contrast to the EKF, it also does not require computing the Jacobian of the process and observation models, which can be advantageous when those models are learned.

In practice, tuning the parameter κ of the UKF can sometimes be difficult: If $\kappa + n$ is high, the sigma points will be placed far from the mean, which increases prediction uncertainty and can even destabilize the filter. Julier and Uhlmann (1997) suggested to chose κ such that $\kappa + n = 3$. This however results in negative values of κ if $n > 3$, for which the estimated covariance matrix is not guaranteed to be positive semidefinite any more. This problem can be solved by changing the way in which $\mathbf{\Sigma}$ is computed (see Appendix III in Julier *et al.* (2000)). However, for $-n < \kappa < 0$, the sigma point $\boldsymbol{\chi}^0$, which represents the original mean, is also weighted negatively. This not only seems counter-intuitive but strongly negative w^0 can also cause divergence of the estimated mean.

To address the problem of placing the sigma points too far away from the mean, Julier (2002) proposed the Scaled Unscented Transform:

$$\boldsymbol{\chi}^0 = \boldsymbol{\mu} \qquad \boldsymbol{\chi}^i = \boldsymbol{\mu} \pm (\sqrt{(n+\lambda)\mathbf{\Sigma}})_i \qquad \forall i \in \{1...n\} \tag{2.15}$$

$$w_m^0 = \frac{\lambda}{\lambda + n} \qquad w_m^i = \frac{0.5}{\lambda + n} \qquad \forall i \in \{1...2n\} \tag{2.16}$$

$$w_c^0 = \frac{\lambda}{\lambda + n} + (1 - \alpha^2 + \beta) \qquad w_c^i = \frac{0.5}{\lambda + n} \qquad \forall i \in \{1...2n\} \tag{2.17}$$

$$\lambda = \alpha^2(\kappa + n) - n \tag{2.18}$$

The authors suggest using $\kappa = 0$ and small positive values for α, e.g. 10^{-3}. In addition, this formulation uses different weights for computing the mean (w_m) and the covariance (w_c) of the transformed sigma points in Equations 2.13 and 2.14. Here, $\beta = 2$ is recommended if the true distribution of the system is Gaussian. With $\alpha = 1$ and $\beta = 0$, the original parametrization from Julier and Uhlmann (1997) can be recovered. The suggested parameters however again result in large negative weights for $\boldsymbol{\chi}^0$.

2.3.4 Monte Carlo Unscented Kalman Filter (MCUKF)

The UKF represents the belief over the state with as few sigma points as possible. However, finding the correct scaling parameters α, κ and β can be difficult, especially if the state is high dimensional. Instead of relying on the Unscented Transform to calculate the mean and covariance of the next belief, we can also resort to Monte Carlo methods, as proposed by Wüthrich *et al.* (2016). In practice, this means replacing the carefully constructed sigma points and their weights in Equations 2.10, 2.11 or Equations 2.15 - 2.17 with uniformly weighted samples from the current belief. The rest of the UKF algorithm stays the same, but more sampled pseudo sigma points are necessary to represent the distribution of the belief accurately.

2.3.5 Particle Filter (PF)

In contrast to the different variants of the Kalman filter explained before, the Particle filter (Gordon *et al.*, 1993) does not assume a parametric representation of the belief distribution. Instead, it represents the belief with a set of weighted *particles*. This allows the filter to track multiple hypotheses about the state at the same time and makes it a popular choice for tasks like localization or visual object tracking (Thrun *et al.*, 2005).

An initial set of particles $\boldsymbol{\chi}_0^i \in \mathcal{X}_0$ is drawn from some prior belief and initialized with uniform weights π. At each timestep, new particles are generated by applying the process model to the previous particles and sampling additive process noise:

$$\mathcal{X}_t = f(\mathcal{X}_{t-1}, \mathbf{u}_t, \mathbf{q}_t) \tag{2.19}$$

Given an observation $\mathbf{z}_t$, the weight π_t^i of each particle $\boldsymbol{\chi}_t^i$ is updated based on the likelihood $p(\mathbf{z}_t|\boldsymbol{\chi}_t^i)$ by $\boldsymbol{\chi}^i$: $\pi_t^i = \pi_{t-1}^i p(\mathbf{z}_t|\boldsymbol{\chi}_t^i)$.

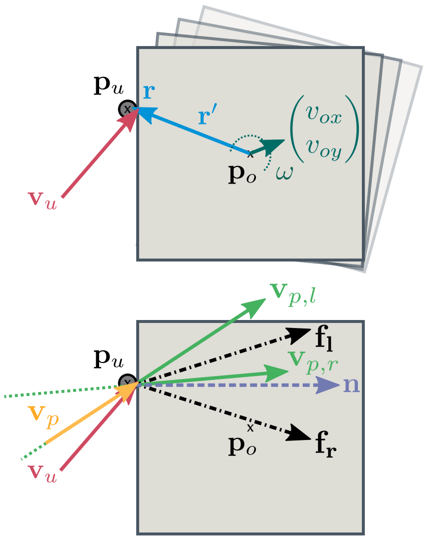
A potential problem of the PF is particle deprivation: Over time, many particles will receive a very low likelihood $p(\mathbf{z}_t|\boldsymbol{\chi}_t^i)$, and eventually the state would be represented by too few particles with high weights. To prevent this, a new set of particles with uniform weights can be drawn (with replacement) from the old set according to the weights. This resampling step focuses the particle set on regions of high likelihood and is usually applied after each timestep.

2.4 Planar Pushing

As explained in Section 1.2, we will use the task of planar pushing as the main test bed for the research presented in this thesis. In the following, we introduce an analytical model for describing the dynamics of planar pushing with a point contact and discuss its underlying assumptions. We also describe a hardware platform that was used to collect data and perform experimental evaluations.

2.4.1 An Analytical Model of Planar Pushing

Throughout this thesis, we use an analytical model of quasi-static planar pushing that was devised by Lynch *et al.* (1992). It predicts the object movement $\mathbf{v}_o$ given the pusher velocity $\mathbf{v}_u$, the contact point $\mathbf{r}$ and associated surface normal $\mathbf{n}$ as well as two friction-related parameters l and m. The model is illustrated in Figure 2.4, which also contains a list of symbols. Note that this model is still approximate



$\mathbf{p}_o$	position of the object
θ	orientation of the object
$\mathbf{v}_o$	linear and angular object velocity
$\mathbf{v}_p$	linear velocity at the contact point - *effective push velocity*
$\mathbf{p}_u$	position of the pusher
$\mathbf{v}_u$	linear pusher velocity - *action*
$\mathbf{r}$	contact point (global)
$\mathbf{r}'$	contact point relative to $\mathbf{p}_o$
$\mathbf{n}$	surface normal at $\mathbf{r}$
l	ratio between maximal torsional and linear friction force
m	friction coefficient pusher-object
$\mathbf{f}_b$	left or right boundary force of the friction cone
τ_b	torques corresponding to the boundary forces
$\mathbf{v}_{o,b}$	object velocities resulting from boundary forces
$\mathbf{v}_{p,b}$	effective push velocities corresponding to the boundary forces
$b = l, r$	placeholder for left or right boundary
s	contact indicator, $s \in [0, 1]$

Figure 2.4: Overview and illustration of the terminology for pushing.

and far from perfectly modeling the stochastic process of planar pushing (Yu *et al.*, 2016).

Predicting the effect of a push with this model has two stages: First, it determines whether the push is stable ("sticking contact") or whether the pusher will slide along the object ("sliding contact"). In the first case, the velocity of the object at the contact point will be the same as the velocity of the pusher. In the sliding case, however, the pusher movement can be almost orthogonal to the resulting motion at the contact point. We call the motion at the contact point "effective push velocity" $\mathbf{v}_p$. It is the output of the first stage. Given $\mathbf{v}_p$ and the contact point, the second stage then predicts the resulting translation and rotation of the object's center of mass which we here assume to correspond to the object position $\mathbf{p}_o$.

Stage 1: Determining the Contact Type and Computing $\mathbf{v}_p$:

To determine the contact type (slipping or sticking), we have to find the left and right boundary forces $\mathbf{f}_l$, $\mathbf{f}_r$ of the friction cone (i.e. the forces for which the pusher will just not start sliding along the object) and the corresponding torques τ_l, τ_r. The opening angle α_m of the friction cone is defined by the friction coefficient m between pusher and object. The forces and torques are then computed by

$$\alpha_m = \arctan(m) \tag{2.20}$$

$$\mathbf{f}_l = \mathbf{R}(-\alpha_m)\mathbf{n} \qquad \mathbf{f}_r = \mathbf{R}(\alpha_m)\mathbf{n} \tag{2.21}$$

$$\tau_l = r'_x f_{ly} - r'_y f_{lx} \qquad \tau_r = r'_x f_{ry} - r'_y f_{rx} \tag{2.22}$$

where $\mathbf{R}(\alpha_m)$ denotes a rotation matrix given α_m and $\mathbf{r}' = \mathbf{r} - \mathbf{p}_o$ is the contact point relative to the object's center of mass.

To relate the forces to object velocities, Lynch *et al.* (1992) use an ellipsoidal approximation to the limit surface. To simplify notation, we use subscript b to refer to quantities associated with either the left l or right r boundary forces. $\mathbf{v}_{o,b}$ and $\omega_{o,b}$ denote linear and angular object velocity, respectively. $\mathbf{v_{p,b}}$ are the push velocities that would create the boundary forces. They span the so called "motion cone".

$$\mathbf{v}_{o,b} = \frac{\omega_{o,b} l^2}{\tau_b}\mathbf{f}_b \tag{2.23}$$

$$\mathbf{v}_{p,b} = \omega_{o,b}(\frac{l^2}{\tau_b}\mathbf{f}_b + \mathbf{k} \times \mathbf{c}') \tag{2.24}$$

Here, $\mathbf{k}$ is the rotation axis of the object and $\omega_{o,b}$ acts as a scaling factor. Since we are only interested in the direction of $\mathbf{v}_{p,b}$ and not in its magnitude, we set $\omega_{o,b} = \tau_b$:

$$\mathbf{v}_{p,b} = l^2\mathbf{f}_b + \tau_b\mathbf{k} \times \mathbf{r}' \tag{2.25}$$

To compute the effective push velocity $\mathbf{v}_p$, we need to determine the contact case: If the push velocity lies outside of the motion cone, the contact will slip. The resulting effective push velocity then acts in the direction of the boundary velocity $\mathbf{v}_{p,b}$ which is closer to the push direction:

$$\mathbf{v}_p = \frac{\mathbf{v}_u \cdot \mathbf{n}}{\mathbf{v}_{p,b} \cdot \mathbf{n}} \mathbf{v}_{p,b} \tag{2.26}$$

Otherwise, the contact is sticking and we can use the pusher velocity as effective push velocity $\mathbf{v}_p = \mathbf{v}_u$. When the norm of $\mathbf{n}$ is zero (due to e.g. a wrong prediction of the perception neural network), we set the output $\mathbf{v}_{p,b}$ to zero.

The object will of course only move if the pusher is in contact with the object. To use the model also in cases where no force acts on the object, we introduce the contact indicator variable s. It takes values between zero and one and is multiplied with $\mathbf{v}_p$ to switch off responses when there is no contact.

$$\mathbf{v}_p = s\mathbf{v}_p$$

We allow s to be continuous instead of binary to give the model a chance to react to the pusher making or breaking contact during the interaction.

Stage 2: Using $\mathbf{v_p}$ to Predict the Object Motion:

Given the effective push velocity $\mathbf{v}_p$ and the contact point $\mathbf{r}'$ relative to the object center of mass, we can compute the linear and angular velocity $\mathbf{v}_o = [v_{ox}, v_{oy}, \omega]$ of the object.

$$v_{ox} = \frac{(l^2 + r_x'^2)v_{px} + r_x' r_y' v_{py}}{l^2 + r_x'^2 + r_y'^2} \tag{2.27}$$

$$v_{oy} = \frac{(l^2 + r_y'^2)v_{py} + r_x' r_y' v_{px}}{l^2 + r_x'^2 + r_y'^2} \tag{2.28}$$

$$\omega = \frac{r_x' v_{oy} - r_y' v_{ox}}{l^2} \tag{2.29}$$

Discussion of Underlying Assumptions

The analytical model is built on three simplifying assumptions:

(i) Quasi-static pushing: the force applied to the object is big enough to move the object, but not to accelerate it.

(ii) The pressure distribution of the object on the surface is uniform and the limit-surface of frictional forces can be approximated by an ellipsoid.

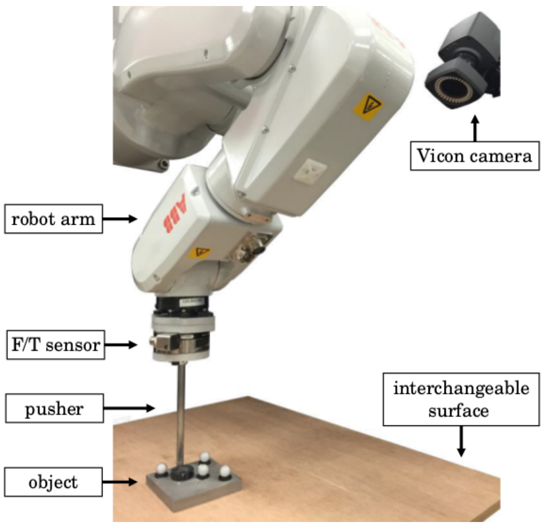


Figure 2.5: Robotic platform for collecting pushing data. Figure reproduced with permission from Yu *et al.* (2016) ©2016 IEEE

(iii) The friction coefficient between surface and object is constant.

The analysis performed by Yu *et al.* (2016) shows that assumption (ii) and (iii) are frequently violated by real world data. Assumption (i) holds for push velocities below 50 $\frac{mm}{s}$. In addition, the contact situation may change during pushing (as the pusher may slide along the object and even lose contact), such that the model predictions become increasingly inaccurate the longer ahead it needs to predict in one step.

2.4.2 Robotic Pushing Platform

To train and evaluate our methods, we frequently use data from a robotic platform shown in Figure 2.5. It is located at MIT's Manipulation and Mechanisms Laboratory (`https://mcube.mit.edu`) and consists of an ABB IRB 120 robot arm with an attached cylindrical pusher that can push steel objects with different shapes on a table with interchangeable surface materials. A Vicon motion capture system is used to track the pose of the manipulated objects with high accuracy. Sensory input can be obtained using an Intel RealSense D415 RGBD camera mounted in front of the robot and the force/torque sensor at the pusher.

Besides from performing our own experiments on the robot in Chapter 5, we will use two datasets of robotic pushes collected with this platform by Yu *et al.* (2016); Bauza *et al.* (2019).

Chapter 3

Models for Perception and Prediction

This chapter is based on a previously published article

Alina Kloss, Stefan Schaal, and Jeannette Bohg. (2020) "Combining learned and analytical models for predicting action effects from sensory data." In *International Journal of Robotics Research*. DOI: 10.1177/0278364920954896. ©2018 SAGE Publications

For author contributions, Alina Kloss developed the theory and methods, and conceived, planned and carried out the experiments. Alina Kloss wrote the manuscript with inputs from all authors. Stefan Schaal helped supervise the project and provided feedback. Jeannette Bohg supervised the project and shaped its direction.

3.1 Introduction

In this chapter, we want to study the advantages and limitations of model-based and data-driven approaches for perception and prediction and how we can combine both. As an exemplary task, we approach the problem of predicting the consequences of push interactions with objects based on raw sensory data.

Traditionally, interaction dynamics are described by physics-based analytical models (Yu *et al.*, 2016; Lynch *et al.*, 1992; Zhang and Trinkle, 2012) which rely on a fixed representation of the environment state. As discussed before, this approach has the advantage that both state representation and dynamics model have physical meaning and are therefore interpretable for humans and easily transferable to similar problems. They also make the underlying assumptions in the model transparent. However, defining such dynamics models for complex scenarios and extracting the required state representation from raw sensory data may be difficult, especially if no assumptions, for example about the shape of objects, are made.

More recently, we have seen approaches for predicting the effects of pushing actions that successfully replace the physics-based models with learned ones (Zhou *et al.*, 2016; Belter *et al.*, 2014; Meriçli *et al.*, 2015; Kopicki *et al.*, 2017; Bauza and Rodriguez, 2017). While often more accurate than analytical models, these methods still assume a predefined state representation as input and do not address the problem of how it may be extracted from raw sensory data.

Other neural network based methods instead simultaneously learn a representation of the sensory input and the associated dynamics from large amounts of training data, for example (Byravan and Fox, 2017; Agrawal *et al.*, 2016; Watters *et al.*, 2017; Finn *et al.*, 2016). They have shown impressive results in predicting the effect of physical interactions directly from sensory observations.

Agrawal *et al.* (2016) argue that a neural network may benefit from choosing its own intermediate representation of the input data instead of being forced to use a predefined state representation. They reason that a problem can often be parameterized in different ways and that some of these parameterizations might be easier to obtain from the given sensory input than others. The disadvantage of a learned representation is, however, that it usually cannot be be mapped to physical quantities. This makes it hard to intuitively understand both the learned functions and the representations. In addition, it remains unclear models with a learned state representation could be transferred to different but similar problems.

As discussed in Chapter 2.1 and shown by Zhang *et al.* (2017), neural networks also often have the capacity to memorize their training data perfectly and learn a mapping from inputs to outputs instead of fitting the true underlying function. This can make perfect sense if enough training data is available that covers the whole problem domain. However, when data is sparse (for example because a robot learns by experimenting), it can easily lead to overfitting. The question of how to generalize beyond the training data then becomes very important.

Our hypothesis is that using prior knowledge from existing physics-based models can reduce the amount of required training data for learning the perception models and at the same time ensure good generalization of the whole system beyond the training domain. In this chapter, we thus investigate using neural networks for extracting a suitable state representation from raw sensory data that can then be consumed by an analytical model for prediction. Optionally, the output of the analytical model can be further refined by adding a learned error term. We compare these *hybrid* approaches to using a neural network for both, perception and prediction, as well as to the analytical model applied on ground truth input values.

As example physical interaction task, we use planar pushing as introduced in Chapter 2.2. For this task, a well-known physical model (Lynch *et al.*, 1992) is available as well as a large, real-world dataset (Yu *et al.*, 2016) which we augmented with simulated images. Given a depth image of a tabletop scene with one object and the position and movement of the pusher, our models need to predict the object position in the given image and its movement due to the push.

Our experiments show that despite of relying only on depth images to extract position and contact information, all our models perform similar to the analytical model applied on the ground truth state. Given enough training data and evaluated inside of its training domain, the pure neural network implementation performs best and even outperforms the analytical model baseline significantly. However, when it comes to generalization to new actions, the hybrid approach is much more accurate. Additionally, we find that the hybrid approach needs significantly less training data than the neural network model to arrive at a high prediction accuracy.

To summarize, in this chapter, we make the following contributions:

- We show how analytical dynamics models and neural networks can be combined and trained end-to-end to predict the effects of robot actions based on visual input like depth images.

- We compare this hybrid approach to using a pure neural network for learning both, perception and prediction. Evaluations on a real world physical interaction task demonstrate improved data efficiency and generalization when including the analytical model into the network over learning everything from scratch.

- We show how the hybrid approach can be further extended by combining the analytical model with a learned error-correction term to better compensate for possible inaccuracies of the analytical model

- For training and evaluation, we augmented an existing dataset of planar pushing with depth and RGB images and additional contact information. The code for this is available online.

3.2 Related Work

3.2.1 Models for Pushing

Analytical models of quasi-static planar pushing have been studied extensively in the past, starting with Mason (1986). Goyal *et al.* (1991) introduced the limit surface to relate frictional forces with object motion, and much work has been done on different approximate representations of it Hong Lee and Cutkosky (1991); Howe and Cutkosky (1996). The model by Lynch *et al.* (1992), which we use here, relies on an ellipsoidal approximation of the limit surface.

More recently, there has also been a lot of work on data-driven approaches to modeling the dynamics of pushing (Zhou *et al.*, 2016; Belter *et al.*, 2014; Meriçli *et al.*, 2015; Kopicki *et al.*, 2017; Bauza and Rodriguez, 2017). Kopicki *et al.* (2017) describe a modular learner that outperforms a physics engine for predicting the results of 3D quasi-static pushing even for generalizing to unseen actions and object shapes. This is achieved by providing the learner not only with the trajectory of the global object frame, but also with multiple local frames that describe contacts. The approach however requires knowledge of the object pose from an external tracking system and the learner does not place the contact-frames itself. Bauza and Rodriguez (2017) train a heteroscedastic Gaussian Process that predicts not only the object movement under a certain push, but also the expected variability of the outcome. The trained model outperforms the analytical model by Lynch *et al.* (1992) given very few training examples. It is however specifically trained for one object and generalization to different objects is not attempted. Moreover, this work, too, assumes access to the ground truth state, including the contact point and the angle between the push and the object surface.

3.2.2 Learning Dynamics Based on Raw Sensory Data

Many recent approaches in reinforcement learning aim to solve the so-called *pixel-to-torque* problem, where the network processes images to extract a representation of the state and then directly returns the required action to achieve a certain task (Lillicrap *et al.*, 2015; Levine *et al.*, 2016). Jonschkowski and Brock (2015) argue that the state-representation learned by such methods can be improved by enforcing *robotic priors* on the extracted state, that may include for example temporal coherence. This is an alternative way of including basic principles of physics in a learning approach, compared to what we propose here. While policy learning requires understanding the effect of actions, the above methods do not acquire an explicit dynamics model. We are interested in learning such an explicit model, as it enables optimal action selection (potentially over a larger time horizon). The following papers share this aim.

Agrawal *et al.* (2016) consider a learning approach for pushing objects. Their network takes as input the pushing action and a pair of images: one before and one after a push. After encoding the images, two different network streams attempt to predict (i) the encoding of the second image given the first and the action and (ii) the action necessary to transition from the first to the second encoding. Simultaneously training for both tasks improves the results on action prediction. The authors do not enforce any physical models or robotic priors. As the learned models directly operate on image encodings instead of physical quantities, we cannot compare the accuracy of the forward prediction part (i) to our results.

SE3-Nets (Byravan and Fox, 2017) process organized (i.e. image shaped) 3D point clouds and an action to predict the next point cloud. For each object in the scene, the network predicts a segmentation mask and the parameters of an SE3 transform (linear velocity, rotation angle and axis). In newer work, Byravan *et al.* (2018) add an intermediate step, that computes the 6D pose of each object before predicting the transforms based on this more structured state representation. The output point cloud is obtained by transforming all input pixels according to the transform for the object they correspond to. The resulting predictions are very sharp and the network is shown to correctly segment the objects and determine which are affected by the action. An evaluation of the generalization to new objects or forces was however not performed.

Our own architecture is inspired by this work. The pure neural network we use to compare to our hybrid approach can be seen as a simplified variant of SE3-Nets, that predicts SE2 transforms (see Section 3.5). Since we define the loss directly on the predicted movement of the object, we omit predicting the next observation and the segmentation masks required for this. We also use a modified perception network, which relies mostly on a small image patch around the robot end-effector.

The work presented by Finn *et al.* (2016); Ebert *et al.* (2017, 2018) is similar to the method by Byravan and Fox (2017) and explores different possibilities of predicting the next frame of a sequence of actions and RGB images using recurrent neural networks.

Visual Interaction Networks (Watters *et al.*, 2017) also take temporal information into account. A convolutional neural network encodes consecutive images into a sequence of object states. Dynamics are predicted by a recurrent network that considers pairs of objects to predict the next state of each object.

3.2.3 Combining Analytical Models and Learning

The idea of using analytical models in combination with learning has also been explored in previous work. Degrave *et al.* (2019) implemented a differentiable physics engine for rigid body dynamics in Theano and demonstrate how it can be used to train a neural network controller. Nguyen-Tuong and Peters (2010) significantly improve Gaussian Process learning of inverse dynamics by using an analytical model of robot dynamics with fixed parameters as the mean function or as feature transform inside the covariance function of the GP. Both works, however, do not cover visual perception.

More recently, Wu *et al.* (2017) used a graphics and physics engine to learn to extract object-based state representations in an unsupervised way: Given a sequence of images, a network learns to produce a state representation that is predicted forward in time using the physics engine. The graphics engine is used to render the predicted state and its output is compared to the next image as training signal. In contrast to the aforementioned work, we not only combine learning and analytical models, but also evaluate the advantages and limitations of this approach.

Finally, Sahoo *et al.* (2018) present an interesting approach to learning functions by training a neural network to combine a number of mathematical base operations (like multiplication, division, sine and cosine). This enables their "Equation Learner" to learn functions which generalize beyond the domain of the training data, just like traditional analytical models. Training these networks is however challenging and involves training many different models and choosing the best in an additional model selection step.

3.2.4 Newer Work

A first preprint of the work presented in this chapter was published in 2017 [1]. Since then, multiple authors have worked on related problems.

Jiang and Liu (2018), for example, mix analytical and learned components for modeling contact dynamics, where the analytical part ensures that physical constraints like non-penetration are observed while the learned part models frictional interactions that are difficult to compute analytically.

Similarly to our hybrid approach with learned error-correction term, Ajay *et al.* (2018); Ajay *et al.* (2019) train recurrent neural networks to correct the predictions of an analytical model over several timesteps and provide estimates of the system uncertainty.

Instead of correcting the prediction of an analytical model with a learned error-term, Zeng *et al.* (2020) propose to directly correct the resulting control parameters for grasping and throwing objects. To compute the residual term, their method takes the analytical control parameters and visual input into account.

3.3 Problem Statement

Our aim is to analyze the benefits of combining neural networks with analytical models. We therefore compare this hybrid approach to models that exclusively rely

[1]Preprint at `https://arxiv.org/abs/1710.04102`

on either approach. As a test bed, we use planar pushing, for which a well-known analytical model and a real-world dataset are available.

We consider the following problem: The input consists of a depth image $\mathbf{D}_t$ of a tabletop scene with one object and the pusher at time t, the starting position $\mathbf{p}_{u,t}$ of the pusher and its movement between this and the next timestep $\mathbf{v}_{u,t} = \mathbf{p}_{u,t+1} - \mathbf{p}_{u,t}$. With this information, the models need to predict the object position $\mathbf{p}_{o,t}$ before the push is applied and its movement $\mathbf{v}_{o,t} = \mathbf{p}_{o,t+1} - \mathbf{p}_{o,t}$ due to the push.

This can be divided into two subproblems:

Perception Extract a suitable state representation of the scene at time t (before the push) $\mathbf{x}_t$ from the input image. The form of $\mathbf{x}_t$ depends on the following prediction model, we only require that $\mathbf{x}_t$ contains the object position $\mathbf{p}_{o,t}$.

$$f_{perception}(\mathbf{D}_t) = \mathbf{x}_t$$

Prediction Given the state representation $\mathbf{x}_t$, the start position $\mathbf{p}_{u,t}$ of the pusher and its movement $\mathbf{v}_{u,t}$, predict how the object will move:

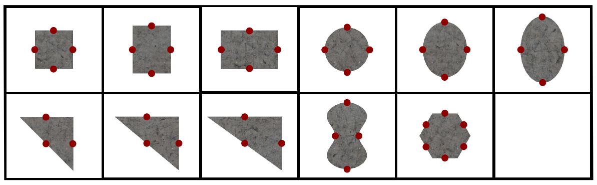
$$f_{prediction}(\mathbf{x}_t, \mathbf{p}_{u,t}, \mathbf{v}_{u,t}) = \mathbf{v}_{o,t}$$

3.4 Data

We use the MIT Push Dataset by Yu *et al.* (2016) for our experiments. It contains object pose and force recordings (not used here) from real robot experiments, where eleven different planar objects are pushed on four different surfaces. For each object-surface combination, the dataset contains about 6000 pushes that vary in the manipulator ("pusher") velocity and acceleration, the point on the object where the pusher makes contact and the angle between the object surface and the push direction. Pushes are 5 cm long and data was recorded at 250 Hz.

As this dataset does not contain RGB or depth images, we render them using OpenGL and the mesh-data supplied with the dataset. In this chapter, we only use the depth images, RGB will be considered Chapter 4. A rendered scene consists of a flat surface with one of four textures (representing the four surface materials), on which one of the objects is placed. The pusher is represented by a vertical cylinder with no arm attached. Figures 3.1a and 3.1b show the different objects and example images. We also annotated the dataset with all information necessary to apply the analytical model to use it as a baseline. The code for annotation and rendering is available at `https://github.com/mcubelab/pdproc`.

For each experiment, we construct datasets for training and testing from a subset of the Push Dataset. As the analytical model does not take acceleration of the pusher into account, we only use push variants with zero pusher acceleration. We,


(a) Rendered objects of the Push Dataset Yu *et al.* (2016): *rect1-3, ellip1-3, tri1-3, butter, hex.* Red dots indicate the subset of contact points we use to collect a test set with held-out pushes for Experiment 3.6.4.

(b) Rendered RGB images showing two of the four surfaces in the MIT dataset, *plywood* and *abs.*

however, do evaluate on data with high pusher velocities, that break the quasi-static assumption made in the analytical model (in Section 3.6.5). One data point in our datasets consists of a depth image showing the scene before the push is applied, the object position before and after the push and the initial position and movement of the pusher. The prediction horizon is 0.5 seconds in all datasets [2]. More information about the specific data for each experiment can be found in the corresponding sections.

We use data from multiple randomly chosen timesteps of each sequence in the Push Dataset. Some of the examples thus contain shorter push-motions than others, as the pusher starts moving with some delay or ends its movement during the 0.5 seconds time-window. To achieve more visual variance and to balance the number of examples per object type, we sample a number of transforms of the scene relative to the camera for each push. Finally, about a third of our dataset consists of examples where we moved the pusher away from the object, such that it is not affected by the push movement.

3.5 Combining Neural Networks and Analytical Models

We now introduce the neural network variants that we will analyze in the following experiments. All architectures share the same first network stage that processes raw depth images and outputs a lower-dimensional encoding and the object position. Given this output, the pushing action (movement $\mathbf{v}_u$ and position $\mathbf{p}_u$) of the pusher, and the friction related parameters m and l, the second part of these networks predicts the linear and angular velocity $\mathbf{v}_o$ of the object. Here, we provide the friction related parameters to the models since such information cannot be obtained from single images. In Chapter 4.7, we demonstrate how m and l can be estimated from sequences of observations and pushes using Bayesian Filtering.

The predictive part differs between the network variants. While three of them

[2]We also evaluated two different prediction horizons but found no significant effect on the performance.

(*simple, full, error*) use variants of the analytical dynamics model established in Chapter 2.2, variant *neural* has to learn the dynamics with a DNN. The prediction part has about 1.8 million trainable parameters for all variants except for *error*, which has 2.7 million parameters.

We implement all our networks as well as the analytical model in tensorflow (Abadi *et al.*, 2015), which allows us to propagate gradients through the analytical models just like any other layer.

3.5.1 Perception

The architecture of the network part that processes the image is depicted in Figure 3.2. We assume that the robot knows the position of its end-effector, which allows us to extract a small (80 $\times$ 80 pixel) image patch ("glimpse") around the tip of the pusher. If the pusher is close enough to the object to make contact, the relevant information for predicting the effect of the push - like the contact point and the normal to the object surface - can be estimated from this smaller image. It thus serves as an attention-mechanism to focus the computations on the most relevant part of the image. Only the position of the object needs to be estimated from the full image. The state representation that our perception model extracts thus contains the estimated object position and an encoding of the information represented in the glimpse.

To obtain the glimpse encoding, we process the glimpse with three convolutional layers with ReLU non-linearity, each followed by max-pooling and batch normalization Ioffe and Szegedy (2015). For estimating the object position, the full image is processed with a sequence of four convolutional and three deconvolution layers. The output of the last deconvolution has the same size as the image input and only has one channel that resembles an object segmentation map. We use spatial softmax (Levine *et al.*, 2016) to calculate the pixel location of the segmented object center.

Initial experiments showed that not using the glimpse strongly decreased performance for all networks. We also found that using both, the glimpse and an encoding of the full image, for estimating the state representation was disadvantageous: Using the full image increases the number of trainable parameters in the prediction network but adds no information that is not already contained in the glimpse.

3.5.2 Prediction

Neural Network Only (*neural*): Figure 3.3 a) shows the prediction part of the variant *neural*, which uses a neural network to learn the dynamics of pushing. The input to this part is a concatenation of the output from perception with the action and parameter l[3]. The network processes this input with three fully connected layers before predicting the object velocity $\mathbf{v}_o$. All intermediate fully connected layers use ReLU non-linearities. The output layers do not apply a non-linearity.

Full Analytical Model (*hybrid*): This variant uses the complete analytical model as described in Chapter 2.2. Several fully connected layers extract the necessary input values from the glimpse encoding and the action, as shown in Figure 3.3 b). These are the contact point $\mathbf{r}$, the surface normal $\mathbf{n}$ and the contact indicator s. For predicting s, we use a sigmoidal non-linearity to limit the predicted values to $[0, 1]$.

Simplified Analytical Model (*simple*): *Simple* (Figure 3.3 c) only uses the second stage of the analytical model. As for *hybrid*, a neural network extracts the model inputs (effective push velocity $\mathbf{v}_p$, contact point $\mathbf{r}$) from the encoded glimpse and the action.

We use this variant as a middle ground between the two other options: It still contains the main mechanics of how an effective push at the contact point moves the object, but leaves it to the neural network to deduce the effective push velocity from the scene and the action. This gives the model more freedom to correct for possible shortcomings of the analytical model. We expect these to manifest mostly in the first stage of the model, as small errors can have a big effect there when they influence whether a contact is estimated as sticking or slipping. Since the second stage of the analytical model does not specify how the input action relates to the object movement, *simple* also allows us to evaluate the importance of this particular aspect of the analytical model.

Full Analytical Model + Error-Correction (*error*): One concern when using a predefined analytical model is that the trained network cannot improve over the performance of the analytical model. If the analytical model is inaccurate, the *hybrid* architecture can only compensate to some degree by manipulating the input values of the model, i.e. by predicting “incorrect” values for the components of the state representation. This limits its ability to compensate for model errors as it might not be possible to account for all types of errors in this way.

As an alternative, we propose to learn an error-correction term which is added to the output prediction of the analytical model. The error-term is thus not constrained by the model and should be able to compensate for a broader class of model errors.

Figure 3.3 d) shows the architecture. As input for predicting the error-term, we use the same values that *neural* receives for predicting the object velocity, i.e.

[3]Since the friction coefficient between pusher and objects m is constant over all examples, we saw no reason for including it in the inputs of the network

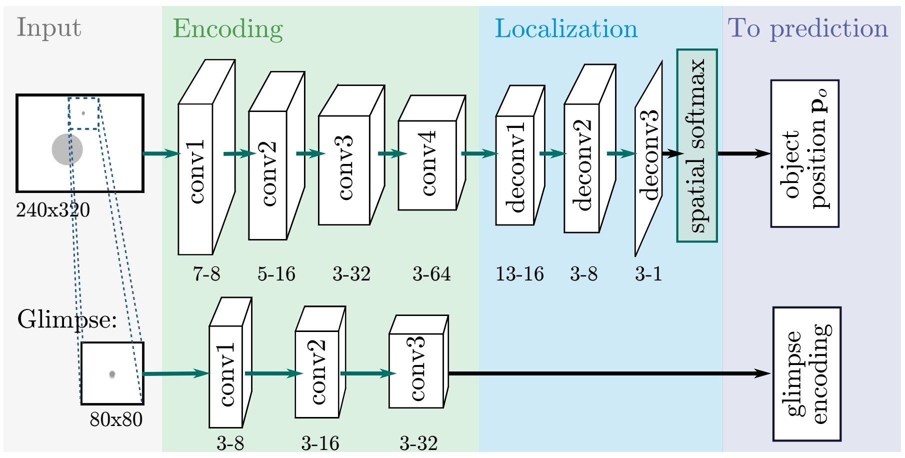


Figure 3.2: Perception part for all network variants. White boxes represent tensors, green arrows and boxes indicate network layers, whereas black arrows represent dataflow without processing. For green arrows, the type of layer (convolution or deconvolution) is denoted in the name of their output tensors. The numbers below the output tensors denote the kernel size and the number of output channels for each layer.
The output of this module, glimpse encoding and the estimated object position $\mathbf{p}_o$, serves as input for the prediction network depicted in Figure 3.3. For training, gradient information is backpropagated through the prediction to the perception network.

the glimpse encoding, the action, the predicted object position and the friction parameter. Note that we do not propagate gradients to the inputs of the error-prediction module. The intuition behind this is that we do not want the error-prediction to interfere with the prediction of the inputs for the analytical model. We evaluate the effect of this architectural decision in Section 3.8. A second variant that we compare to in this section aims to improve the generalizability of the error-prediction to faster push movements. This is achieved by normalizing the input action to unit length before feeding it into the error-prediction module.

3.5.3 Training

All our architectures are trained end-to-end, i.e. the loss is propagated through the prediction to the perception part of the networks. The loss L penalizes the Euclidean distance between the predicted and the real object position in the input image (*pos*), the Euclidean error of the predicted object translation (*trans*), the error in the magnitude of translation (*mag*) and in angular movement (*rot*) in degree (instead of radian, to ensure that all components of the loss have the same order of magnitude). We use weight decay with $\lambda = 0.001$.

Let $\hat{\mathbf{v}}_o$ and $\hat{\mathbf{p}}_o$ denote the predicted and $\mathbf{v}_o$, $\mathbf{p}_o$ the real object movement and position. $\mathbf{w}$ are the network weights and $\boldsymbol{\nu}_o = [v_{ox}, v_{oy}]$ denotes linear object

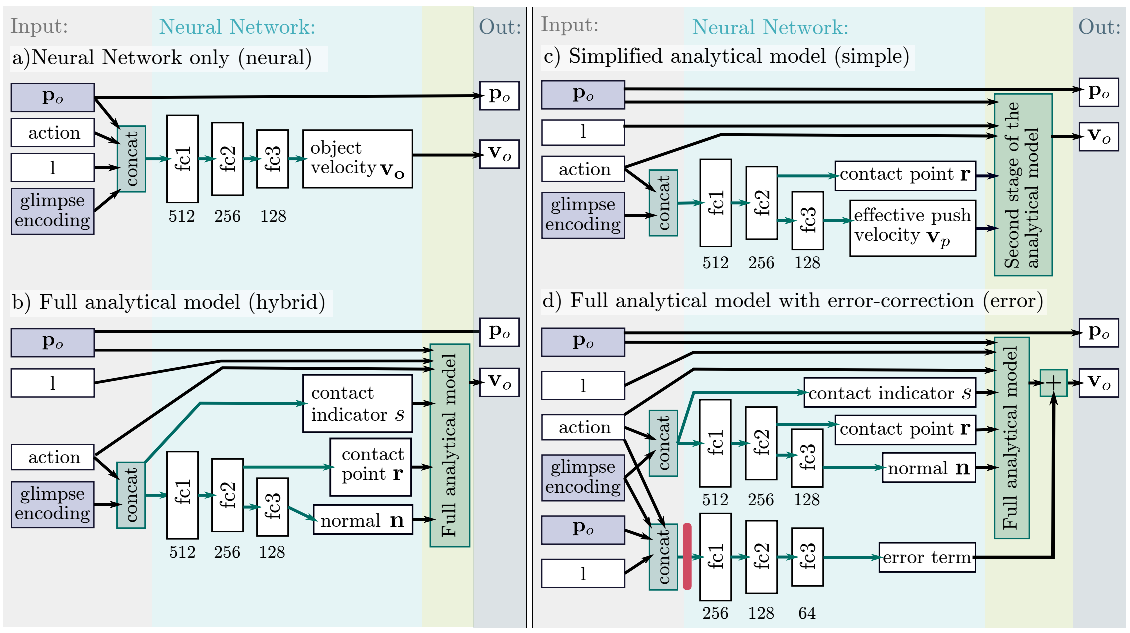

Input:
Neural Network:
Out:
a)Neural Network only (neural)
$\mathbf{p}_o$
action
1
glimpse encoding
concat
fc1
fc2
fc3
512
256
128
object velocity $\mathbf{v_o}$
$\mathbf{v}_o$
b) Full analytical model (hybrid)
contact indicator s
contact point $\mathbf{r}$
normal $\mathbf{n}$
Full analytical model
c) Simplified analytical model (simple)
effective push velocity $\mathbf{v}_p$
Second stage of the analytical model
d) Full analytical model with error-correction (error)
error term
+
64

Figure 3.3: Prediction parts of the four network variants *neural*, *hybrid*, *simple* and *error*. White and purple boxes represent tensors, where the purple color indicates tensors that are computed by the perception part shown in Figure 3.2. During training, the gradient information is back-propagated through these tensors to the perception part.
Green arrows and boxes indicate network layers, whereas black arrows represent dataflow without processing. In this network, all green arrows represent fully connected layers and the numbers beneath their output tensors (fc) denote the number of output channels. The red bar in architecture (d) indicates that no gradients are propagated to the inputs of this layer.

velocity.

$$L(\hat{\mathbf{v}}_o, \hat{\mathbf{o}}, \mathbf{v}_o, \mathbf{p}_o) = trans + mag + rot + pos + \lambda \sum\nolimits_{\mathbf{w}} \parallel \mathbf{w} \parallel$$
$$trans = \|\hat{\boldsymbol{\nu}}_o - \boldsymbol{\nu}_o\| \quad mag = |\|\hat{\boldsymbol{\nu}}_o\| - \|\boldsymbol{\nu}_o\||$$
$$rot = \tfrac{180}{\pi}|\omega - \hat{\omega}| \quad pos = \parallel \mathbf{p}_o - \hat{\mathbf{p}}_o \parallel$$

When using the variant *hybrid*, a major challenge is the contact indicator s: In the beginning of training, the direction of the predicted object movement is mostly wrong. s therefore receives a strong negative gradient, causing it to decrease quickly. Since the predicted motion is effectively multiplied by s, a low s results in the other parts of the network receiving small gradients and thus greatly slows down training. We therefore add the error in the magnitude of the predicted velocity to the loss to prevent s from decreasing too far in the early training phase.

We use Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.0001 and a batch-size of 32 for 75,000 steps.

3.6 Evaluating Generalization

In this section, we test our hypothesis that using an analytical model for prediction together with a neural network for perception improves data efficiency and leads to better generalization than using neural networks for both, perception and prediction. We evaluate how the performance of the networks depends on the amount of training data (Experiment 3.6.3) and how well they generalize to (i) pushes with new pushing angles and contact points (Experiment 3.6.4), (ii) new push velocities (Experiment 3.6.5) and (iii) unseen objects (Experiment 3.6.6).

For the experiments here, we use a top-down view of the scene, such that the object can only move in the image plane and the z-coordinate of all scene components remains constant. This simplifies the application of the analytical model by removing the need for an additional transform between the camera and the table. It also simplifies the perception task and allows us to focus this evaluation on the comparison of the hybrid and the purely neural approach. In Chapter 4.7 we will show how to extend the proposed model to work on more difficult camera settings.

3.6.1 Baselines

We use three baselines in our experiments. All of them use the ground truth input values of the analytical model (action, object position, contact point, surface normal, contact indicator and friction coefficients) instead of depth images. They thus do not solve the full problem of predicting object movement from raw sensory input. Instead, they address the easier problem of prediction given perfect state information. Accordingly, the baselines only output the object velocity, but not its initial position in the scene.

If the pusher makes contact with the object during the push, but is not in contact initially, we use the contact point and normal from when contact is first made and shorten the action accordingly. Note that this gives the baseline models an additional advantage over architectures that have to infer such input values from raw sensory data.

The first baseline is just the average translation and rotation over the dataset. This is equal to the error when always predicting zero movement, and we therefore name it *zero*. The second, *physics*, is the full analytical model evaluated on the ground truth input values. The third baseline, called *neural dyn* is a neural network that has the same three-layer architecture as the prediction module of *neural* (see Figure 3.3 a) for details). The difference between *neural* and *neural dyn* is their input: While *neural* receives the glimpse encoding and object position from the perception network as input, *neural dyn* gets the ground truth physical state representation that is also used in the analytical model. This allows us to evaluate whether *neural* benefits from being able to learn its own state representation (the glimpse encoding) end-to-end through the prediction part.

3.6.2 Metrics

For evaluation, we compute the average Euclidean distance between the predicted and the ground truth object translation (*trans*) and position (*pos*) in millimeters as well as the average error on object rotation (*rot*) in degree. As our datasets differ in the overall object movement, we report errors on translation and rotation normalized by the average motion in the corresponding dataset given by the error of the baseline *zero*.

3.6.3 Data Efficiency

The first hypothesis we test is that combining the analytical model with a neural network for perception reduces the required training data as compared to a pure neural network.

Data: We use a dataset that contains all objects from the MIT Push dataset and all pushes with velocity $20\frac{mm}{s}$ and split it randomly into training and test set. This results in about 190k training examples and about 38k examples for testing. To evaluate how the networks' performance develops with the amount of training data, we train the models on different subsets of the training split with sizes from 2500 to the full 190k. We always evaluate on the full test split. To reduce the influence of dataset composition especially on the small datasets, we average results over multiple different datasets with the same size.

Results: Figure 3.4 shows how the errors in predicted translation, rotation and object position develop with more training data and Table 3.1 contains numeric values for training on the biggest and smallest training split. As expected, the combined approach of neural network and analytical model (*hybrid* and *error*) already performs very well on the smallest dataset (2500 examples) and beats the other models including the *neural dyn* baseline, which uses the ground truth state representation, by a large margin. It takes more than 20k training examples for the other models to reach the performance of *hybrid*, where predicting rotation seems to be harder to learn than translation.

Despite of having to rely on raw depth images instead of the ground truth state representation, all models perform at least close to the *physics* baseline when using the full training set. However, only the pure neural network and the hybrid model with error-correction are able to improve on the baseline. This shows that the analytical model limits *hybrid* in fitting the training data perfectly, since the model itself is not perfect and does not allow for overfitting to noise in the training data. *Neural* and *error* have more freedom for fitting the training distribution, which however also increases the risk of overfitting.

Combining the learned error-correction with the fixed analytical model is especially helpful for predicting the translation of the object. To also improve the prediction of rotations, the model needs more than 20k training examples, which is similar to *neural*. While *neural* makes a larger improvement on the full dataset, *error* combines the comparably good performance of *hybrid* on few training examples with the ability to improve on the model given enough data.

The variant *simple*, which uses only the second part of the analytical model, also combines learning and a fixed model for predicting the dynamics. But in contrast to *error*, this variant seems to combine the disadvantages of both approaches: It needs much more training data than *hybrid* but is still limited by the performance of the analytical model and gets quickly outperformed by the pure neural network when more data is available.

The comparison of *neural* and the baseline *neural dyn* shows that despite of having access to the ground truth data, *neural dyn* actually performs worse than *neural* on the full dataset. This seems to agree with the theory of Agrawal *et al.* (2016), that training perception and prediction end-to-end and letting the network chose its own state representation instead of forcing it to use a predefined state may be beneficial for neural learning.

Finally, we evaluate how accurate the predicted input values to the analytical model are for *simple*, *hybrid* and *error*. If the analytical model was perfect, we would expect the predicted values to be very close to the real physical state. Higher errors could thus indicate that the models learn to compensate for inaccuracies of the analytical model.

As can be seen in Table 3.2, both *hybrid* and *error* make fairly accurate predictions for the object state, with contact point errors around 5 mm and less than

Table 3.1: Error in predicted translation (*trans*) and rotation (*rot*) as percentage of the average movement given by *zero* (standard errors in brackets). *pos* denotes the error in predicted object position. Values shown are for training on the full training set (190k examples) and on a 2500 examples subset.

		trans	rot	pos [mm]
2.5k	*neural*	33.6 (0.18) %	62.5 (0.42) %	0.46 (0.002)
	simple	32.3 (0.19) %	53.6 (0.37) %	**0.44** (**0.002**)
	hybrid	25.4 (0.17) %	**45.5** (**0.36**) %	0.46 (0.002)
	error	**24.7** (**0.16**) %	46.8 (0.36) %	0.45 (0.002)
	neural dyn	32.6 (0.19) %	63.5 (0.46) %	-
190k	*neural*	**17.4** (**0.12**) %	**33.4** (**0.28**) %	**0.31** (**0.002**)
	simple	19.3 (0.13) %	35.7 (0.3) %	0.33 (0.002)
	hybrid	19.3 (0.13) %	36.1 (0.3) %	0.32 (0.002)
	error	18.4 (0.12) %	34.6 (0.29) %	**0.31** (**0.002**)
	neural dyn	19.2 (0.12) %	36.3 (0.29) %	-
	physics	18.95 (0.13) %	35.4 (0.3) %	-
	zero	2.95 (0.02) mm	1.9 (0.01) °	-

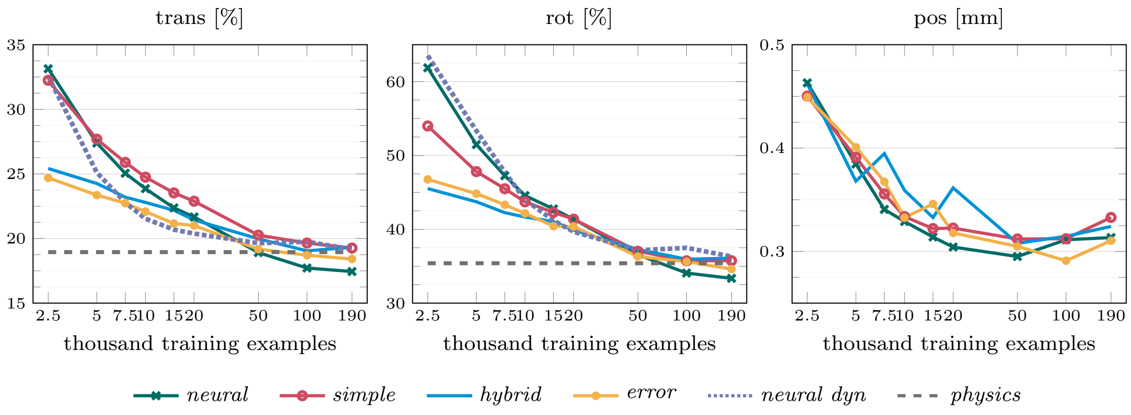


Figure 3.4: Prediction errors versus training set size (x-axis in logarithmic scale). Errors on translation and rotation are given as percentage of the average movement in the test set. The model-based architectures *hybrid* and *error* perform much better than the other networks when training data is sparse.

Table 3.2: Errors of the predicted input values for the analytical model: *Hybrid* and *error* predict the contact point $\mathbf{r}$, then normal $\mathbf{n}$ and the contact indicator s accurately. *Simple* only predicts the contact point and the effective push velocity $\mathbf{v}_p$, which both deviate notably from their ground truth values. Values shown are for training on the full training set (190k examples).

	$\mathbf{r}$ [mm]	$\mathbf{n}$ [°]	s	$\mathbf{v}_p$ [°]
simple	22.4 (0.03)	-	-	18.1 (0.1)
hybrid	4.4 (0.01)	3.6 (0.02)	0.08 (0.001)	-
error	4.8 (0.01)	2.5 (0.02)	0.08 (0.001)	-

5° angle between the predicted and correct normal. The contact point indicator s is also estimated with high accuracy. Only variant *simple* shows a larger error between the predicted and true contact points. The predicted effective push velocity $\mathbf{v}_p$ also does not match the values we got from applying the first stage of the analytical model on ground truth input very closely. Since these errors do not seem to harm the overall prediction accuracy, we conclude that they cancel each other out. This shows that *simple* is not as strongly constrained by its analytical component as *hybrid* and *error* and that it thus has more freedom in choosing its state representation.

Summary: All our models reach the performance of the (perception-free) *physics* baseline given enough training data. Combining neural networks and analytical models strongly improves performance in comparison to to purely learned models when little training data is available. However, *neural* can achieve the highest prediction accuracy and beat the *physics* baseline when trained on a very large dataset.

To further improve the prediction accuracy of *hybrid* while preserving its data-efficiency, an additive error-correction term can be learned. Replacing a part of the analytical model with a learned component in *simple* in contrast harmed the data efficiency.

3.6.4 Generalization to New Pushing Angles and Contact Points

The previous experiment showed the performance of the different models when testing on a dataset with a very similar distribution to the training set. Here, we evaluate the performance of the networks on held-out push configurations that were not part of the training data. Note that while the test set contains *combinations* of object pose and push action that the networks have not encountered during training, the pushing actions or object poses themselves do not lie outside of the value range of the training data. This experiment thus test the models' *interpolation* abilities.

Data: We again train the networks on a dataset that contains all objects and pushes with velocity $20\frac{mm}{s}$. For constructing the test set, we collect all pushes with (i) pushing angles $\pm 20°$ and $0°$ to the surface normal (independent from the contact points) and (ii) at a set of contact points illustrated in Figure 3.1a (independent from the pushing angle).

The remaining pushes are split randomly into a training and a validation set, which we use to monitor the training process. There are about 114k data points in the training split, 23k in the validation split and 91k in the test set.

Table 3.3: Prediction errors for testing on pushes with pushing angles and contact points not seen during training.

	trans	rot	pos [mm]
neural	16.5 (0.06) %	36.1 (0.17) %	0.31 (0.001)
simple	16.4 (0.06) %	37.1 (0.18) %	**0.31 (0.001)**
hybrid	**15.6 (0.07)** %	35.3 (0.19) %	0.31 (0.001)
error	**15.6 (0.07)** %	**34.5 (0.18)** %	0.32 (0.001)
neural dyn	18.1 (0.07) %	44.1 (0.2) %	-
physics	14.6 (0.06) %	32.8 (0.18) %	-
zero	4.36 (0.013) mm	2.27 (0.009) °	-

Results: As Table 3.3 shows, *hybrid* and *error* perform best for predicting the object velocity for pushes that were not part of the training set. Although still being close, none of the networks can outperform the *physics* baseline on this test set.

Note that the difficulty of the test set in this experiment differs from the one in the previous experiment, as can be seen from the different performance of the *physics* baseline: Due to the central contact point locations and small pushing angles, the test set contains a high proportion of pushes with sticking contact (see Section 2.2), for which the resulting object movement is similar to the pusher movement. Prediction in sticking contact cases is therefore generally simpler than in cases in which the pusher slides along the object. This difference in difficulty makes it hard to compare the results between Table 3.1 and Table 3.3 in terms of absolute values.

With more than 100k training examples, we supply enough data for the pure neural model to clearly outperform the combined approach and the baseline in the previous experiment (i.e. when the test set is similar to the training set, see Figure 3.4). The fact that *neural* now performs worse than *hybrid* and *physics* indicates that its advantage over the *physics* baseline may not come from it learning a more accurate dynamics model. Instead, it probably memorizes specific input-output combinations that the analytical model cannot predict as well, e.g. due to noisy object pose data.

This might also be the reason why *error* cannot improve on *hybrid* as much as in the previous experiment, especially when it comes to predicting the translation of the object. It is however encouraging to see that the learned error correction term for the predicted rotation is still beneficial for pushes not seen during training.

In contrast to *hybrid* and *error*, *simple* again does not seem to profit from using the simplified analytical model and performs similar to *neural*.

As in the previous experiment (see Table 3.1), we also tested the generalization ability of the networks when trained on a smaller training set. If we supply only 2500 training examples, the difference between *hybrid* and the purely learned model is again much more pronounced: *Hybrid* achieves 20.3 % translation and 43.8 % rotation error whereas *neural* lies at 38.7 % and 63.4 % respectively.

Summary: The purely learned model performs worse than the hybrid approaches when interpolating to unseen push configurations. For all models, the difference to the *physics* baseline is larger when the training distribution does not match the test distribution.

3.6.5 Generalization to Different Push Velocities

In this experiment, we test how well the networks generalize to unseen push velocities. In contrast to the previous experiment, the test actions in this experiment have a different value range than the actions in the training data, and we are thus looking at *extrapolation*. As neural networks are usually not good at extrapolating beyond their training domain, we expect the model-based network variants to generalize better to push-velocities not seen during training.

Data: We use the networks that were trained in the first experiment (3.6.3) on the full (190k) training set. The push velocity in the training set is thus 20 $\frac{mm}{s}$. We evaluate on datasets with different push velocities ranging from 10 $\frac{mm}{s}$ to 300 $\frac{mm}{s}$. Since seeing only one push velocity during training might be a disadvantage for the learned models, we also compose two new training datasets, one with velocities conform to the quasi-static assumption (10 and 20 $\frac{mm}{s}$) and one with a higher second velocity (20 and 50 $\frac{mm}{s}$) that violates the quasi-static assumption. Both datasets have slightly more than 125k training examples.

Results: Results are shown in Figure 3.5. Since the input action does not influence perception of the object position, we only report the errors on the predicted object motion.

When training on push velocities below 50 $\frac{mm}{s}$, we see a very large difference between the performance of our combined approach and the pure neural network for higher velocities. *Neural's* and *neural dyn's* predictions quickly become very inaccurate, with the error on predicted translation rising to more than 60 % and the error on predicted rotation to more than 80 % of the error when predicting zero movement always. The performance of *hybrid* on the other hand is most constant over the different push velocities and declines only slightly more than the *physics* baseline. *Error*, too, extrapolates well, but only when trained on more than one push velocity.

Like *neural* and *neural dyn*, *simple*, too, gets worse on higher velocities. Its performance when predicting rotations however degrades much less than for predicting translations. The reason for this is that all three architectures struggle mostly with predicting the correct *magnitude* of the object translation and not so much with predicting the translation's *direction*. By using the second stage of the analytical model, *simple* has information about how the direction of the object translation

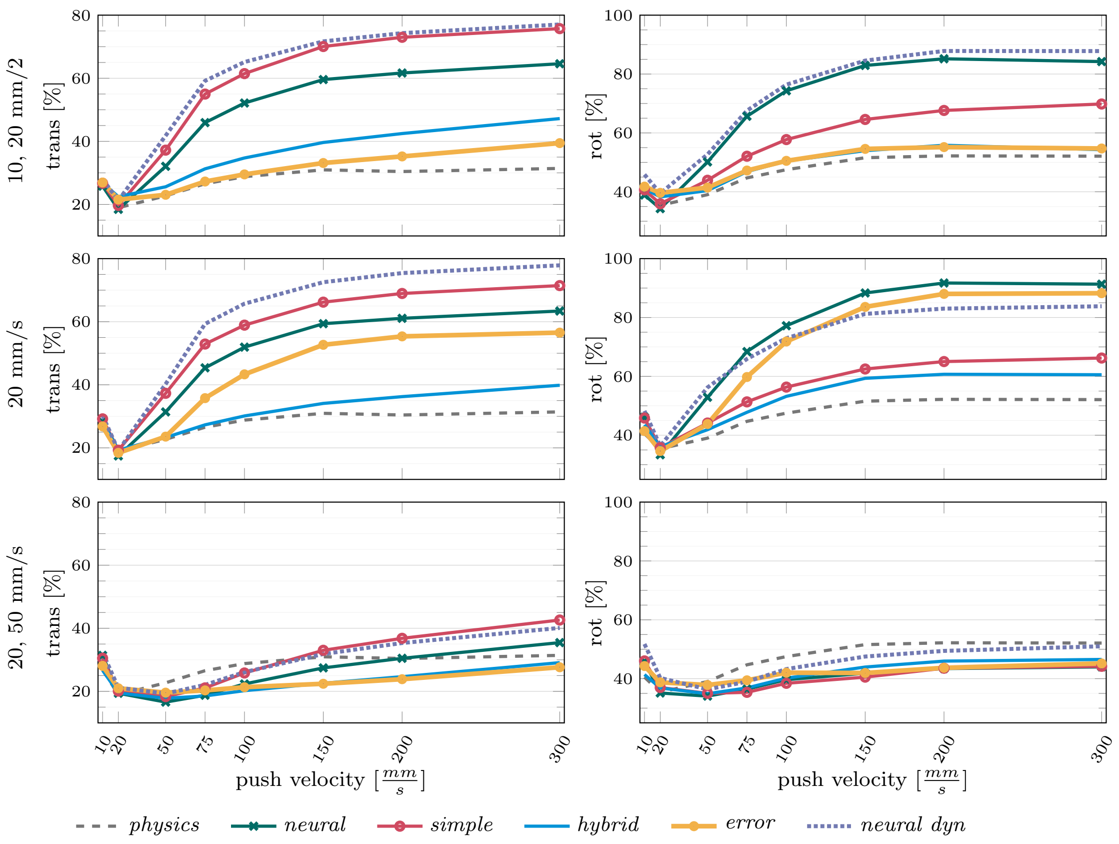


Figure 3.5: Errors on predicted translation and rotation for testing on different push velocities. In the first row, all models were trained on push velocities 10 and 20 $\frac{mm}{s}$, in the second row on velocity 20 $\frac{mm}{s}$ and in the last row on velocities 20 and 50 $\frac{mm}{s}$. When training on velocities that are small enough to ensure quasi-static pushing, all models have trouble extrapolating to higher velocities, but *hybrid* and *error* stay much closer to the *physics* baseline than *simple*, *neural* and *neural dyn*. Seeing additional training data from a higher push velocity (50 $\frac{mm}{s}$) that violates the quasi-static assumption strongly improves the generalization to higher velocities for all models and enables them to beat the *physics* baseline in many cases. Especially for the predicted object translation, we however still see a much stronger decrease in performance for *simple*, *neural* and *neural dyn* than for *hybrid*, *error* and *physics*.

and the contact point relate to its rotation, which results in much more accurate predictions.

The advantage of *hybrid* for extrapolation lies in the first stage of the analytical model, which allows it to scale its predictions according to the magnitude of the action and the contact indicator s. Both are in essence multiplication operations. A general multiplication of inputs can however not be expressed using only fully connected layers (as used by *simple*, *neural*, *neural dyn* and the error-prediction part of *error*) because fully connected layers essentially perform weighted additions of their inputs. So instead of learning the underlying function, the networks are forced to resort to memorizing input-output relations for the magnitude of the object motion, which explains why extrapolation does not work well, especially when training on low push velocities.

When combining the prediction of the analytical model with a learned error-term and training only on one push velocity, the resulting model suffers from the same issues as the other network-based variants. The decline is however less pronounced than for *neural*, and only starts after $50 \frac{mm}{s}$. A possible reason for this is that the error-correction term is rather small compared to the output of the analytical model. This means that the weights with which the action enters the computation of the error term are smaller than for *neural*, *simple* or *neural dyn*.

Interestingly, adding a second training velocity completely changes the picture and makes *error* perform on par with or even better than *hybrid*. Our hypothesis is that seeing different velocities during training prevents the error term from overfitting to the input action and minimizes the effect of the action magnitude on the predicted error-term. In Section 3.8, we show that for training on only one push velocity, *error* can also be made more robust to higher velocities by normalizing the push action before using it as input to the error-prediction.

While the *physics* baseline performs better than the models trained on low push velocities, it predictions also get worse on higher push velocities. The main reason for this is that the quasi-static assumption of the model is violated: For pushes faster than $20 \, \frac{mm}{s}$, the object gets accelerated and can continue sliding even after contact to the pusher was lost.

How different the dynamics of pushing are between the quasi-static and this dynamic behaviour also becomes apparent when we include the push velocity $50 \, \frac{mm}{s}$ in the training data for our learned models: They all extrapolate much better to higher velocities and are often able to outperform the *physics* baseline. This increase of performance for fast pushes however only extends to the slowest push velocity ($10 \, \frac{mm}{s}$) for *hybrid*, whereas all other models perform slightly worse than their counterparts that were only trained on one push velocity.

We also still see that with increasing push velocities, the variants *simple*, *neural* and *neural dyn* make significantly larger errors for predicting the translation of the object than *hybrid* and *error*. Interestingly, for predicting the object rotation, all models except for *neural dyn* perform extremely well, with *hybrid* even doing slightly worse than the others. A possible reason for this difference between translation and rotation could be that the magnitude of rotations does not increase as strongly with the push velocity as the magnitude of translations: The average rotation increases from 1.4° on 10 $\frac{mm}{s}$ pushes to 14.4° on 300 $\frac{mm}{s}$ pushes, whereas translation increases from 2.1 to 24.8 mm. The models therefore need to change their predicted rotations less in response to higher push velocities than they have to for translation.

Summary: Extrapolating to different push velocities is difficult for purely learned models, especially when the training data only contains low pushing velocities. Using the analytical model in *hybrid* and *error* facilitates extrapolation by providing multiplication operations and explaining the influence of the action on the resulting movement. Since the quasi-static assumption of the analytical model is violated by fast pushes, our models can however learn to outperform the *physics* baseline in this regime when they have training data from faster pushes.

3.6.6 Generalization to Different Objects

This experiment tests how well the networks generalize to unseen object shapes and how many different objects the networks have to see during training to generalize well.

Data: We train the networks on three different datasets: With one object (butter), two objects (butter and hex) and three objects (butter, hex and one of the ellipses or triangles). The datasets with fewer objects contain more augmented data, such that the total number of data points is about 35k training examples in each. As test sets, we use one dataset containing the three ellipses and one containing all triangles. While this is fewer training data than in the previous experiments, it should be sufficient for the pure neural network to perform as well as *hybrid*, since the test sets contain only few objects.

Results: The results in Figure 3.6 show that *neural* is consistently worse than the other networks, especially when predicting rotations. It also improves most notably when one example of the test objects is in the training set. The differences between the models are less pronounce when predicting translation, except for *simple* which performs particularly bad on triangles. In contrast to neural, the architecture with added error-term also does not perform very different from *hybrid*. These results suggest that the fixed state representation and the analytical model can act as a regularizer that prevents the perception part from overfitting to the object shape.

In general, all models perform surprisingly well on ellipses, even if the models only had access to data from the butter object. Reaching the baseline performance on

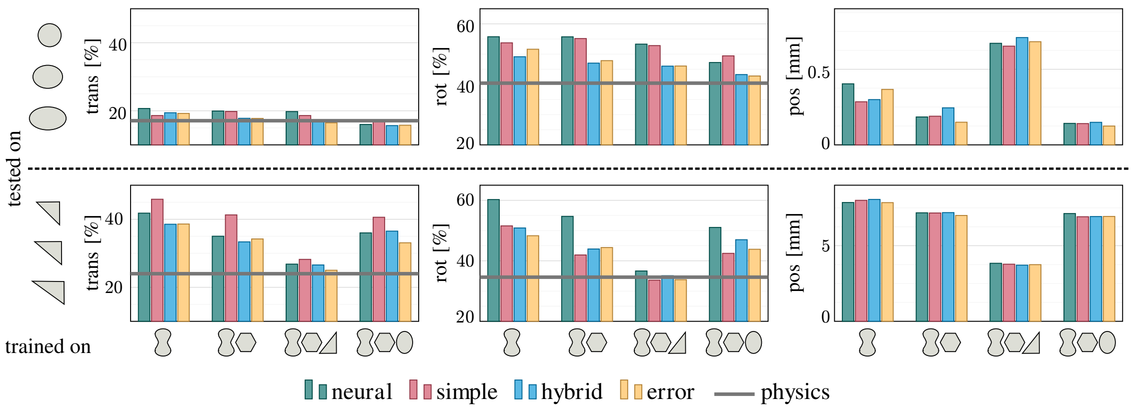


Figure 3.6: Prediction errors in translation, rotation and position on objects not seen during training. Training objects are shown on the x-axis. The top row shows results for evaluating on ellipses, the bottom row on triangles. All networks generalize well to ellipses, but are worse for triangles, where the error in predicted position is by factor ten higher than for the other objects. *Neural* particularly struggles with predicting rotations of previously unseen objects.

triangles is however only possible with a triangle in the training set. Predicting the object's position is most sensitive to the shapes seen during training: It generalizes well to ellipses which have similar shape and size as the butter or hex object. The triangles on the other hand are very different from the other objects in the dataset and the error for localizing triangles is by factor ten higher than for ellipses. The results for predicting the object position do not differ much between the different models. This is not surprising, since they share the same perception architecture.

Summary: Using the analytical model with its fixed state representation in *hybrid* and *error* also facilitates generalization to novel object shapes, which is more difficult for the purely learned model. All models struggle slightly with localizing objects of unknown shapes.

3.7 Visualizations

As a qualitative evaluation, we plot the predictions of our networks, the *physics* and *neural dyn* baselines and the ground truth object motion for 200 repetitions of the same push configuration. The data for these repeated pushes is available with the MIT Push dataset. All repetitions have the same nominal pushing angle ($0°$), velocity ($20 \frac{mm}{s}$) and contact point, but the exact values vary slightly between individual pushes. To keep the visual input diverse, we also sample a different transformation of the whole scene for each repetition, such that the object's pose in the image varies. The networks were trained on the full dataset from Experiment 3.6.3.

The results shown in Figure 3.7 illustrate that the resulting ground truth object motion for the same push configuration varies greatly between trials. Especially in terms of object rotation, the distribution of outcomes shows two distinct modes (one close to the overall mean and one with notably stronger object rotation). By comparing the ground truth with the prediction of the analytical model, we can estimate how much of this variance is due to slight changes in the push configuration between trials (these also reflect in the analytical model) and how much is caused by other, non-deterministic effects.

The predictions of *hybrid* and the analytical model are very similar. This again shows that the state-representation that the neural network part of *hybrid* predicts is mostly accurate. The plotted contact point and normal estimates in Figure 3.8 further confirm this. Adding an error-correction term to the *hybrid* architecture improves the average estimation quality a little, but also increases the variance of the predictions.

The visualizations for the other models (Figure 3.7 (d)-(f)) show that they, too, make good predictions in this example, but *simple* and *neural dyn* have much more variance in the direction of the predicted translation than *hybrid* or *neural*. It is also interesting to see that *neural*, *neural dyn* and *simple* all slightly overestimate the object rotation in comparison to the mean ground truth movement, whereas *physics* slightly underestimates it. Figure 3.8 also shows that *simple* is not very accurate in predicting the contact points, confirming the quantitative results found in Table3.2. As stated before, we believe that this inaccuracy is compensated for by the predicted $\mathbf{v}_p$.

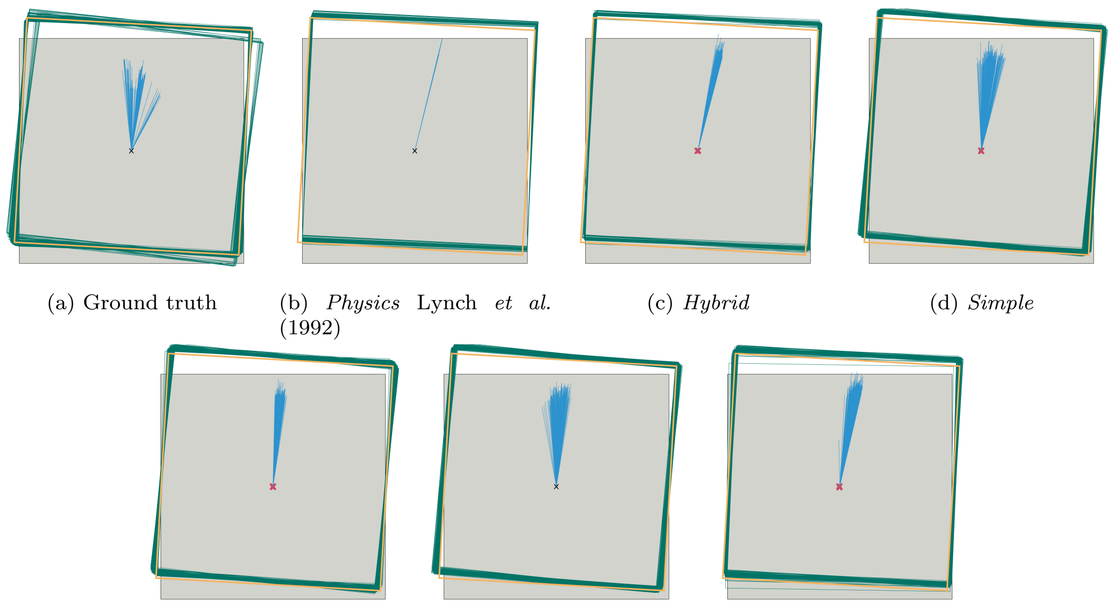
3.8 Evaluation of Models with Error-Correction

The previous results have shown that adding a learned error-correction term to the output of the analytical model in the hybrid architecture enables the network to improve over the performance of the analytical model. The *error* model we analyzed is able to outperform *hybrid* and the *physics* baseline if the training set and the test set are similar (see Experiment 3.6.3).

In the following experiments, we evaluate different choices we made for the architecture of *error*. We also compare the ability of *hybrid* and *error* to compensate for larger errors in the analytical model.

3.8.1 Evaluation of Different Architectures

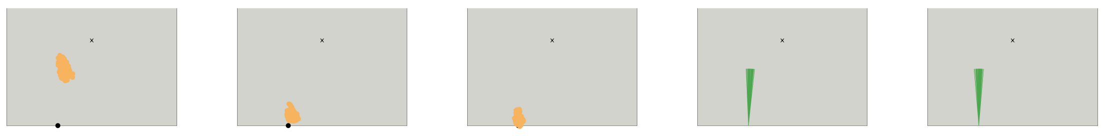
As explained in Section 3.5, we chose to block the propagation of gradients from the error-correction module to the glimpse-encoding, because we did not want the error-computation to interfere with the prediction of the state representation. Here, we also evaluate an architecture *err-grad* that does not block the gradient propagation.



(a) Ground truth (b) *Physics* Lynch *et al.* (1992) (c) *Hybrid* (d) *Simple*

(e) *Neural* (f) *Neural dyn* (g) *Error*

Figure 3.7: Qualitative evaluation on 200 repeated pushes with the same push configuration (angle, velocity, contact point). The green rectangles show the (predicted) pose of the object after the push and the blue lines illustrate the object's translation (for better visibility, we upscaled the lines by factor 5). The thicker orange rectangle is the average ground truth pose of the object after the push. Red crosses indicate the predicted initial object positions. All models predict the movement of the object and its initial position well, but cannot capture the multimodal distribution of the ground truth data.



(a) Contact points predicted by *simple* (b) Contact points predicted by *hybrid* (c) Contact points predicted by *error* (d) Contact normal predicted by *hybrid* (e) Contact normal predicted by *error*

Figure 3.8: Predicted contact points and normals from 200 repeated pushes with the same push configuration (angle, velocity, contact point). The black point marks the (average) ground truth contact point. While *hybrid* and *error* make fairly accurate predictions, *simple* predicts the contact points not on the edge of the object but close to its center.

Table 3.4: Evaluation of different architectures for predicting an error-correction term. In contrast to *error*, *error-grad* allows the propagation of gradients from the error-prediction module to the glimpse encoding. *Error-norm* instead normalizes the push action to unit length before using it as input to the error-prediction. Values shown are for training on the full training set (190k examples). Results for *hybrid* and *neural* are repeated for reference.

	trans	rot
neural	**17.4 (0.12)** %	**33.4 (0.28)** %
hybrid	19.3 (0.13) %	36.1 (0.3) %
error	18.4 (0.12) %	34.6 (0.29) %
error-grad	17.9 (0.12) %	34.4 (0.29) %
error-norm	18.3 (0.12) %	35.3 (0.29) %
physics	18.95 (0.13) %	35.4 (0.3) %
zero	2.95 (0.02) mm	1.9 (0.01) °

This architecture manages to beat *hybrid* by an even bigger margin, as shown in Table 3.4.

The downside of propagating the gradients becomes apparent if we look at generalization to new pushing velocities: While the predictions of *error* become worse with increasing velocity, they still remain more accurate than the predictions of *neural*, as illustrated in Figure 3.9. *Error-grad* on the other hand performs even worse than the pure neural network. A reason for this difference could be that *error-grad* relies more strongly on the error-correction term than *error*. This allows it to fit the training data more closely but at the same time impedes generalization to novel actions.

As explained before, the reason for the decline in performance when extrapolating is that the neural networks cannot scale their predictions correctly according to the input velocity. One possibility to make the error-prediction more robust to higher input velocities is the architecture we call *error-norm*. In this model, we scale the push action to unit length before using it as input to the error-prediction. This makes the error-prediction independent of the magnitude of the action, while still giving it information about the push direction. The resulting model performs only slightly worse than *error* inside the training domain, but much better for extrapolation. It is still worse than *hybrid* though, as it cannot properly adapt the error-term to match higher velocities.

3.8.2 Compensation of Model Errors

Using the error-correction term of course becomes much more interesting if the analytical model is bad. To test how well the *hybrid* and *error* architectures can compensate for wrong models, we manipulate the friction parameter l by setting it to 1.5 or 3 times its real value. The results are shown in Table 3.5.

Wrong values of l are especially harmful for predicting the rotation of the object, and both *hybrid* and *error* perform better than the *physics* baseline under this condition. This shows that the *hybrid* architecture has the ability to compensate for some errors of the analytical model by manipulating the predicted state representation. However, while *hybrid* performs similar to *error* if l is only 1.5 times bigger

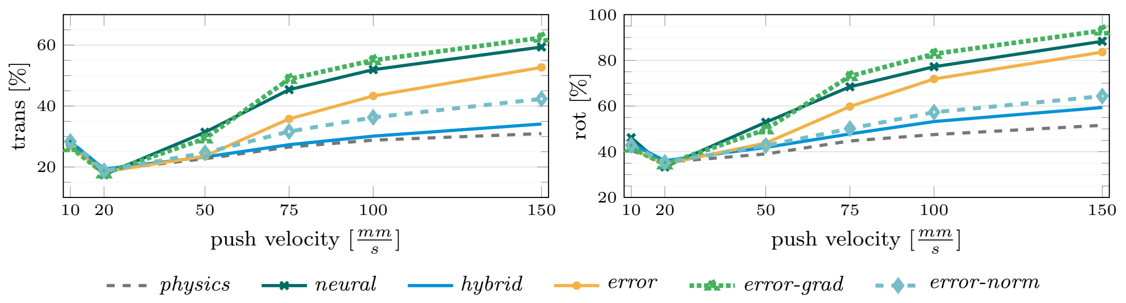


Figure 3.9: Evaluation of the different architectures for predicting an error-correction term on unseen push velocities. All models were trained on push velocity 20 $\frac{mm}{s}$. None of the error-prediction models is as robust as *hybrid* to higher input velocities. *Error-norm* performs best because its predicted error terms are independent from the push velocity. *Error-grad* presumably relies more on the error-prediction term than the other architectures and therefore performs worst outside of the training domain.

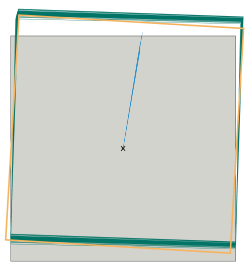
		trans	rot
$1.5 \cdot l$	*hybrid*	20.7 (0.13) %	40.5 (0.32) %
	error	**19.2** (**0.14**) %	**35.9** (**0.3**) %
	physics	23.9 (0.15) %	46.1 (0.37) %
$3 \cdot l$	*hybrid*	25.1 (0.15) %	66.9 (0.45) %
	error	**19.6** (**0.13**) %	**37.2** (**0.3**) %
	physics	35.6 (0.23) %	80.1 (0.53) %

Table 3.5: Prediction errors of *physics*, *hybrid* and *error* when using a manipulated friction parameter l. In contrast to *physics*, both neural networks can compensate for the resulting error of the analytical model. *Hybrid* can however only modify the input values to the analytical model, while *error* can correct the model's output directly and thus compensates the error of the analytical model much better.

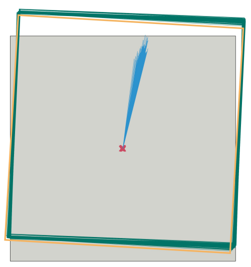
than the correct value, it cannot compensate as well for larger deviations in l. In this case, the ability of *error* to directly alter the output of the analytical model instead of only manipulating its input values proves to be necessary for achieving good performance.

The visualization in Figure 3.10 shows that both models predicted incorrect contact points to counter the effect of the higher friction value. This makes sense, since the location of the contact point influences the tradeoff between how much the object rotates and how much it translates. The predictions from *error* deviate farther from the ground truth values, which shows that the additional error-term does not prevent the model from manipulating the input values to the analytical model. Instead, it achieves its good results by combining both forms of correction.

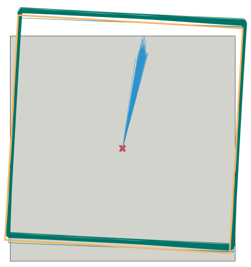
Summary: Adding an learned error-correction term to the hybrid approach improves its ability to compensate for errors in the analytical model. It however does not prevent prediction of "wrong" state representations in such cases. For generalization, we found it helpful to limit the error term's dependency on the magnitude of the pushing action and to stop gradient flow from the error to the perception



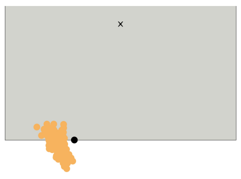
(a) *Physics*



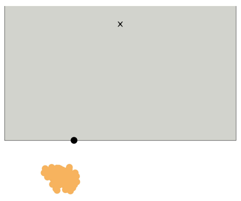
(b) *Hybrid*



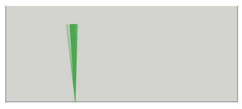
(c) *Error*



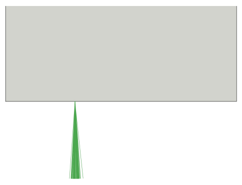
(d) Contact points predicted by *hybrid*



(e) Contact points predicted by *error*



(f) Contact normal predicted by *hybrid*



(g) Contact normal predicted by *error*

Figure 3.10: Predicted movement, contact points and normals from 200 repeated pushes when using a wrong friction parameter ($1.5 \cdot l$). The black point marks the (average) ground truth contact point. Both networks compensate for the wrong friction parameter by predicting the contact point in a slightly wrong position, but the deviation from the ground truth is stronger for *error*, which also flips the direction of the predicted normal (this is however not relevant in our implementation of the analytical model).

module.

3.9 Conclusion

In this chapter, we studied the advantages and limitations of model-based and data-driven approaches for perception and prediction and how we can combine both. As an exemplary task, we approach the problem of predicting the consequences of push interactions with objects based on raw sensory data. We compared a purely learned approach to a hybrid approach that uses a neural network for perception and an analytical model for prediction.

We observed two main advantages of the hybrid architecture. Compared to the pure neural network, it significantly (i) reduces the amount of required training data and (ii) improves generalization to novel physical interaction and object shapes. The analytical model aides generalization by limiting the ability of the hybrid architecture to overfit to the training data and by providing multiplication operations for scaling the output according to the input action and contact indicator. This kind of mathematical operation is hard to learn for fully connected architectures and requires many parameters and diverse training examples for covering a large value range. The drawback of the *hybrid* approach is that it cannot as easily improve on the performance of the underlying analytical model.

The pure neural network on the other hand can beat both, the hybrid approach and the analytical model (with ground truth input values) if trained on enough data. This, however, only holds when we evaluate on actions encountered during training and does not transfer to new push configurations, velocities or object shapes. The challenge in these cases is that the distribution of the training and test data differ significantly.

To enable the hybrid approach to improve more on the prediction accuracy of its analytical model, we experimented with learning an error-correction term that is added to the prediction of the analytical model. These *error* models are almost as data-efficient as *hybrid* and can to some extend retain the ability to generalize to different test data provided by the analytical model. They, however, require more diversity in the training data than *hybrid* to avoid overfitting. Our experiments with a wrong analytical model also showed that the *error* models can compensate for errors of the model much better than *hybrid*, which can only influence the prediction by manipulating the input values of the analytical model.

The last architecture, *simple*, showed that combining learning and analytical models is not automatically guaranteed to lead to good performance. By replacing the first stage of the analytical model with a neural network, we instead combined the disadvantages of both approaches: The architecture needs lots of training data and does not generalize well to new pushes, because it misses the part of the analytical model that explains the influence of the pushing action on the resulting object velocity. In contrast to the pure neural network, it, however, also cannot improve much on the performance of the analytical model.

A limitation of the presented hybrid approach is that it may be hard to find an accurate analytical model for some physical processes and that not all existing models are suitable for our approach, as we require them to be differentiable everywhere. If no analytical model is available, learning the predictive model with a neural network is still a very good option. Especially the switching dynamics encountered when the contact situation changes proved to be challenging and more work needs to be done in this direction.

In perception on the other hand, the strengths of neural networks can be well exploited to extract the input state representation of the analytical model from raw sensory data. By training end-to-end through a given model, we can avoid the effort of labeling data with the ground truth state. Our experiments also showed that training end-to-end allows the hybrid models to compensate for smaller errors in the analytical model by adjusting the predicted input values.

Using the state representation of the analytical model for the *hybrid* architecture has the advantage that the predictions of the network can be visualized and interpreted. This is not easily possible for the intermediate representations learned in the pure neural network. Our results, however, suggest that the pure neural network benefits from being free to chose its own state representation, as learning the dynamics model from the ground truth state representation (*neural dyn*) lead to worse prediction results.

The work presented here mainly serves as a case study for combining analytical and learned models and we thus kept the visual scenes relatively simple. An interesting direction for future work is to extend the concept to more challenging visual problems, like scenes with multiple objects or objects with more complex geometry. The visual processing of point-clouds and understanding of object geometry could potentially be facilitated by using methods like pointnet++ (Qi *et al.*, 2017) that are specifically designed for this type of input data.

A logical next step is also to consider sequences of actions and observations instead of single-step prediction. Working on sequences makes it possible to exploit temporal cues like optical flow or to guide learning by enforcing constraints like temporal consistency. One approach to do so that we will present in the next chapter is to embed the model-learning into the structure of Bayesian filtering algorithms to provide probabilistic estimates of the state of the system over time.

Chapter 4

State Estimation and Uncertainty

This chapter has not been previously published, but an article based on its contents is currently under review. The preprint can be found on arXiv.org

Alina Kloss, Georg Martius, and Jeannette Bohg. (2020) "How to Train Your Differentiable Filter" *arxiv.org/abs/2012.14313.*

For author contributions, Alina Kloss developed the theory and methods, and conceived, planned and carried out the experiments. Alina Kloss wrote the manuscript with feedback from all authors. Jeannette Bohg and Georg Martius supervised the project and shaped its direction.

4.1 Introduction

In the previous chapter, we investigated combining structure and learning at the level of individual models for perception and prediction. Here, we take the focus up to the level algorithms. Algorithms are formal descriptions of how to solve a certain task using the corresponding models. Discovering this type of higher-level logic on top of learning the required models can be challenging for unstructured DNNs. Embedding the model-learning into the given structure of algorithms thus has the potential to facilitate learning and allows us to optimize the models specifically for their respective algorithm.

For evaluating this hypothesis, we look at the problem of state estimation: In many robotic applications, it is crucial to maintain a belief about the state of the system over time, like tracking the location of a mobile robot or the pose of a manipulated object. These state estimates serve as input for planning and decision making and provide feedback during task execution. In addition to tracking the system state, it can also be desirable to estimate the uncertainty associated with the state predictions. This information can be used to detect failures and enables risk-aware planning, where the robot takes more cautions actions when its confidence in the estimated state is low.

Recursive Bayesian filters are a class of algorithms that combine perception and prediction for probabilistic state estimation in a principled way. To do so, they require an observation model that relates the estimated state to the sensory observations and a process model that predicts how the state develops over time. Both have associated noise models that reflect the stochasticity of the underlying system and determine how much trust the filter places in perception and prediction.

Formulating good observation and process models for the filters can, however, be difficult for many problems, especially when the sensory observations are high-dimensional and complex, like camera images. Over the last years, deep learning has become the method of choice for processing such data. While (recurrent) neural networks can be trained to address the full state estimation problem directly, recent work (Jonschkowski and Brock, 2016; Haarnoja *et al.*, 2016; Jonschkowski *et al.*, 2018; Karkus *et al.*, 2018a) showed that it is also possible to include data-driven models into Bayesian filters and train them end-to-end through the filtering algorithm. For Histogram filters (Jonschkowski and Brock, 2016), Kalman filters (Haarnoja *et al.*, 2016) and Particle filters (Jonschkowski *et al.*, 2018; Karkus *et al.*, 2018a), the respective authors showed that such *differentiable filters* (DF) systematically outperform unstructured neural networks like LSTMs. In addition, the end-to-end training of the models also improved the filtering performance compared to using observation and process models that had been trained separately.

A further interesting aspect of differentiable filters is that they allow for learning sophisticated models of the observation and process noise. This is useful because finding appropriate values for the noise models is often difficult and despite much research on identification methods (e.g. (Bavdekar *et al.*, 2011; Valappil and Georgakis, 2000)) they are often tuned manually in practice. To reduce the tedious tuning effort, the noise is then typically assumed to be uncorrelated Gaussian noise with zero mean and *constant* covariance. Many real systems are, however, better described by *heteroscedastic* noise models, where the level of uncertainty depends on the state of the system and/or possible control inputs. Taking heterostochasticity of the dynamics into account has been demonstrated to improve filtering performance in many robotic tasks (Bauza and Rodriguez, 2017; Kersting *et al.*, 2007). Haarnoja *et al.* (2016) also showed that learning heteroscedastic observation noise helped a Kalman filter dealing with occlusions in the observations.

In this chapter, we perform a through evaluation of differentiable filters. Our main goals are to highlight the advantages of DFs over both unstructured learning approaches and manually-tuned filtering algorithms, and to provide practical guidance to researchers interested in applying differentiable filtering to their problems.

To this end, we review and implement existing work on differentiable Kalman and Particle filters and introduce two novel variants of differentiable Unscented Kalman filters. The underlying algorithms are introduced in Chapter 2.3. Our implementation for TensorFlow (Abadi *et al.*, 2015) is publicly available[1]

In extensive experiments on three different tasks, we compare the DFs and evaluate different design choices for implementation and training, including loss functions and training sequence length. We also investigate how well the different filters can learn complex heteroscedastic and correlated noise models and compare the DFs to unstructured LSTM (Hochreiter and Schmidhuber, 1997) models.

4.2 Related Work

4.2.1 Combining Learning and Algorithms

Integrating algorithmic structure into learning methods has been studied for many robotic problems, including state estimation, planning (Tamar *et al.*, 2016; Karkus *et al.*, 2017; Oh *et al.*, 2017; Farquhar *et al.*, 2018; Guez *et al.*, 2018) and control (Donti *et al.*, 2017; Okada *et al.*, 2017; Amos *et al.*, 2018; Pereira *et al.*, 2018; Holl *et al.*, 2020). Most notably, Karkus *et al.* (2019) combine multiple differentiable algorithms into an end-to-end trainable "Differentiable Algorithm Network" to address the complete task of navigating to a goal in a previously unseen environment using visual observations. Here, we focus on addressing the state estimation problem with differentiable implementations of Bayesian filters.

[1]`https://github.com/akloss/differentiable_filters`

4.2.2 Differentiable Bayesian Filters

There have been few works on differentiable filters so far. Haarnoja *et al.* (2016) propose the BackpropKF, a differentiable implementation of the (extended) Kalman filter. Jonschkowski and Brock (2016) present a differentiable Histogram filter for discrete localization tasks in one or two dimensions and Jonschkowski *et al.* (2018) and Karkus *et al.* (2018a) both implement differentiable Particle filters for localization and tracking of a mobile robot. In the following, we focus our discussion on differentiable Kalman and Particle filters, since Histogram filters as used in Jonschkowski and Brock (2016) are usually not feasible in practice, due to the need of discretizing the complete state space.

Observation Model and Noise All three works have in common that the raw observations are processed by a learned neural network that can be trained end-to-end through the filter. In Haarnoja *et al.* (2016), the network outputs a low-dimensional representation of the observations together with input-dependent observation noise $\mathbf{R}$ (see Section 4.3.2 for a detailed explanation), while in Jonschkowski *et al.* (2018); Karkus *et al.* (2018a), a neural network learns to predict the likelihood $p(\mathbf{z}_t|\mathbf{x}^i_t)$ of each particle given an image and a map of the environment.

As a result, all three works use heteroscedastic observation noise, but only Haarnoja *et al.* (2016) evaluate this choice: They show that conditioning $\mathbf{R}$ on the raw image observations drastically improves filter performance when the tracked object can be occluded.

Process Model and Noise For predicting the next state, all three works use a given analytical process model. While Haarnoja *et al.* (2016) and Karkus *et al.* (2018a) also assume known process noise, Jonschkowski *et al.* (2018) train a network to predict $\mathbf{Q}$ that can be conditioned on the actions $\mathbf{u}$. The effect of learning action dependent process noise was however not evaluated.

Effect of End-to-End Learning Jonschkowski *et al.* (2018) compare the results of an end-to-end trained filter with one where the observation model and process noise were trained separately. The end-to-end trained variant performs better, presumably because it learns to overestimate the process noise. Possible differences between the learned observation models are not discussed. The best performance for the filter could be reached by first pretraining the models individually and the finetuning end-to-end through the filter.

Comparison to Unstructured Models All works compare their differentiable filters to LSTM models trained for the same task and find that including the structural priors of the filtering algorithm and the known process models improves performance. Jonschkowski *et al.* (2018) also evaluate a Particle filter with learned process model in one experiment, which performs worse than the filter with analytical process model but still beats the LSTM.

In contrast to the existing work on differentiable filtering, the main purpose of this chapter is not to present a new method for solving a robotic task. Instead, we attempt a thorough evaluation of differentiable filtering and of implementation choices made by the aforementioned seminal works. We also compare differentiable filters with different underlying Bayesian filtering algorithms in a controlled way.

4.2.3 Variational Inference

A second line of research closely related to differentiable filters is variational inference in temporal state space models (Krishnan *et al.*, 2016; Karl *et al.*, 2017; Watter *et al.*, 2015; Fraccaro *et al.*, 2017; Archer *et al.*, 2015). For a recent review of this work, see (Girin *et al.*, 2020). In contrast to DFs, the focus of this research lies more on finding generative models that explain the observed data sequences and are able to generate new sequences. The representation of the underlying state of the system is often not assumed to be known. But even though the goals are different, recent results in this field show that structuring the variational models similarly to Bayesian filters improves their performance (Karl *et al.*, 2017; Fraccaro *et al.*, 2017).

4.3 Implementation

In this section, we describe how we embed model-learning into the nonlinear filtering methods presented in Chapter 2.3. Specifically, we will investigate differentiable versions of the Extended Kalman filter (EKF), the Unscented Kalman filter (UKF), a sampling based variant of the UKF that we call Monte-Carlo Unscented Kalman filter (MCUKF) and the Particle Filter (PF). The differentiable versions of the filters will be denoted by dEKF, dUKF etc. in the following.

4.3.1 Differentiable Filters

We implement the aforementioned filtering algorithms as recurrent neural network layers in TensorFlow. For UKF and MCUKF, this is straight-forward, since all necessary operations are differentiable and available in TensorFlow.

dEKF

In contrast, the dEKF requires the Jacobian of the process model **F**. TensorFlow implements a method for computing Jacobians, with or without vectorization. The former is fast but has a high memory demand, while the latter can become very slow for large batch sizes. Therefore, we recommend to derive the Jacobians manually where applicable.

dPF

The Particle filter is the only filter we investigate that is not fully differentiable: In the resampling step, a new set of particles with uniform weights is drawn (with replacement) from the old set according to the old particle weights. While the drawn particles can propagate gradients to their ancestors, gradient propagation to other old particles or to the weights of the old particle set is disrupted (Jonschkowski *et al.*, 2018; Karkus *et al.*, 2018a; Zhu *et al.*, 2020). If we place the resampling step at the beginning of the per-timestep computations, this only affects the gradient propagation through time, i.e. from one timestep $t + 1$ to its predecessor t. At time t, both particles and weights still receive gradient information about the corresponding loss at this timestep. We therefore hypothesize that the missing gradients through time are not problematic as long as we provide a loss at every timestep.

As an alternative to simply ignoring the disrupted gradients, we can also apply the resampling step less frequently or use soft resampling as proposed by Karkus *et al.* (2018a). We evaluate these options in Experiment 4.5.4.

In addition, we investigate two alternative implementation choices for the dPF: The likelihood used for updating the particle weights in the observation update step can be implemented either with an analytical Gaussian likelihood function or with a trained neural network as in Jonschkowski *et al.* (2018) and Karkus *et al.* (2018a). The learned observation likelihood is potentially more expressive than the analytical solution and can be advantageous for problems where formulating the observation and sensor model is not as straight-forward as in our experiments. A potential drawback is that in contrast to the analytical solution, no explicit noise model or sensor network is learned. We compare these two options in Section 4.5.4.

4.3.2 Observation Model

In Bayesian filtering, the observation model $h(\cdot)$ is a *generative* model that predicts observations from the state $\mathbf{z}_t = h(\mathbf{x}_t)$. In practice, it is, however, often hard to find such models that directly predict the potentially high-dimensional raw sensory signals without making strong assumptions.

We therefore use the method first proposed by Haarnoja *et al.* (2016) and train a *discriminative* neural network n_s with parameters $\mathbf{w}_s$ to preprocess the raw sensory data $\mathbf{D}$ and create a more compact representation of the observations $\mathbf{z} = n_s(\mathbf{D}, \mathbf{w}_s)$. This network can be seen as a virtual sensor, and we thus call it *sensor network*. In addition to $\mathbf{z}_t$, the sensor network can also predict the heteroscedastic observation noise covariance matrix $\mathbf{R}_t$ (see Section 4.3.4) for the current input $\mathbf{D}_t$.

In our experiment, $\mathbf{z}$ contains a subset of the state vector $\mathbf{x}$. The actual observation model $h(\mathbf{x})$ thus reduces to a simple linear selection matrix of the observable components, which we provide to the DFs.

4.3.3 Process Model

Depending on the user's knowledge about the system, the process model $f(\cdot)$ can be implemented using a known analytical model or a neural network $n_p(\cdot)$ with weights $\mathbf{w}_p$. When using neural networks, $n_p(\cdot)$ outputs the change from the last state $n_p(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_p) = \Delta\mathbf{x}_t$ such that $\mathbf{x}_{t+1} = \mathbf{x}_t + \Delta\mathbf{x}_t$. This form ensures stable gradients between timesteps (since $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} = 1 + \frac{\partial p}{\partial \mathbf{x}_t}$) and provides a reasonable initialization of the process model close to identity.

4.3.4 Noise Models

For learning the observation and process noise, we consider two different conditions: constant and heteroscedastic. In both cases, we assume that the process and observation noise at time t can be described by zero-mean Gaussian distributions with covariance matrices $\mathbf{Q}_t$ and $\mathbf{R}_t$.

A common assumption in state-space modeling is that $\mathbf{Q}_t$ and $\mathbf{R}_t$ are diagonal matrices, but we can also use full covariance matrices to model correlated noise. In this case, the output of the noise models are upper-triangular matrices $\mathbf{L}_t$, such that e.g. $\mathbf{Q}_t = \mathbf{L}_t\mathbf{L}_t^T$.

For constant noise, the filters directly learn the diagonal or triangular elements of $\mathbf{Q}$ and $\mathbf{R}$. In the heteroscedastic case, $\mathbf{Q}_t$ is predicted from the current state $\mathbf{x}_t$ and (if available) the control input $\mathbf{u}_t$ by a neural network $n_q(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_q)$ with weights $\mathbf{w}_q$. In dUKF, dMCUKF and dPF, $n_q(\cdot)$ outputs separate $\mathbf{Q}^i$ for each sigma point/particle and $\mathbf{Q}_t$ is computed as their weighted mean. The heteroscedastic observation noise covariance matrix $\mathbf{R}_t$ is an additional output of the sensor model $n_s(\mathbf{D}_t, \mathbf{w}_s)$.

We initialize the diagonals $\mathbf{Q}_t$ and $\mathbf{R}_t$ close to given target values by adding a trainable bias variable to the output of the noise models. To prevent numerical instabilities, we also add a small fixed diagonal matrix to both covariance matrices as a lower bound for the predicted noise. The value of the lower bound depends on the overall value range of the state and observations, but we found that values below 10^{-4} increase the risk of numerical errors.

4.3.5 Training Loss

For training the filters we always assume that we have access to the ground truth trajectory of the state $\mathbf{x}^l_{t=0\dots T}$. In our experiments, we test the two different loss functions used in related work: The first, used by Karkus *et al.* (2018a) is simply the mean squared error (MSE) between the mean of the belief and true state at each timestep:

$$L_{\text{MSE}} = \frac{1}{T}\sum_{t=0}^{T}(\mathbf{x}^l_t - \boldsymbol{\mu}_t)^T(\mathbf{x}^l_t - \boldsymbol{\mu}_t) \tag{4.1}$$

For the dPF, we compute μ as the weighted mean of the particles.

The second loss function, used by Haarnoja *et al.* (2016) and Jonschkowski *et al.* (2018), is the negative log likelihood (NLL) of the true state under the predicted distribution of the belief. In dEKF, dUKF and dMCUKF, the belief is represented by a Gaussian distribution with mean $\boldsymbol{\mu}_t$ and covariance $\boldsymbol{\Sigma}_t$ and the negative log likelihood is computed as

$$L_{\text{NLL}} = \frac{1}{2T}\sum_{t=0}^{T}\log(|\boldsymbol{\Sigma}_t|) + (\mathbf{x}^l_t - \boldsymbol{\mu}_t)^T\boldsymbol{\Sigma}_t^{-1}(\mathbf{x}^l_t - \boldsymbol{\mu}_t) \tag{4.2}$$

The dPF represents its belief using the particles $\boldsymbol{\chi}_i \in \mathcal{X}$ and their weights π_i. We consider two alternative ways of calculating the NLL for training the dPF: The first is to represent the belief by fitting a single Gaussian to the particles, with $\boldsymbol{\mu} = \sum_{i=0}^{N}\pi_i\boldsymbol{\chi}_i$ and $\boldsymbol{\Sigma} = \sum_{i=0}^{N}\pi_i(\boldsymbol{\chi}_i - \boldsymbol{\mu})(\boldsymbol{\chi}_i - \boldsymbol{\mu})^T$ and then apply Equation 4.2. We refer to this variant as dPF-G.

This is, however, only a good representation of the belief if the distribution of the particles is unimodal. To better reflect the potential multimodality of the particle distribution, the belief can also be represented with a Gaussian Mixture Model (GMM) as proposed by Jonschkowski *et al.* (2018). Every particle contributes a separate Gaussian $N_i(\boldsymbol{\chi}^i, \boldsymbol{\Sigma})$ in the GMM and the mixture weights are the particle weights. The drawback of this approach is that the fixed covariance $\boldsymbol{\Sigma}$ of the individual distributions is an additional tuning parameter for the filter. We call this version dPF-M and calculate the negative log likelihood with

$$L_{\text{NLL}} = \frac{1}{T}\sum_{t=0}^{T}\log\sum_{i=0}^{|\mathcal{X}|}\frac{\pi^i}{\sqrt{|\boldsymbol{\Sigma}|}}\exp(\mathbf{x}^l_t - \boldsymbol{\chi}^i_t)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}^l_t - \boldsymbol{\chi}^i_t) \tag{4.3}$$

4.4 Experiments

In the following, we will evaluate the DFs on three different datasets. We start with a simple simulation setting that gives us full control over parameters of the system such as the process noise (Section 4.5). In Sections 4.6 and 4.7, we then study the performance of the DFs on two real-robot tasks: The first is the Kitti Visual Odometry problem, where the filters are used to track the position and heading of a moving car given only RGB images. With the last dataset, we return to the task of robotic pushing, where the filters track the pose of an object while the robot performs a series of pushes.

4.4.1 Training and Initialization

Unless stated otherwise, we will train the DFs end-to-end for 15 epochs using the Adam optimizer (Kingma and Ba, 2015). During training, the initial state is perturbed with noise sampled from a Normal distribution $N_{\text{init}}(0, \mathbf{\Sigma}_{\text{init}})$. For testing, we evaluate all DFs with the correct initial state as well as with few fixed perturbations (sampled from N_{init}) and average the results.

The initial covariance for dEKF, dUKF and dMCUKF are set accordingly to $\mathbf{\Sigma}_0 = \mathbf{\Sigma}_{\text{init}}$. For the dPF, we sample the initial particles around the *perturbed* state from N_{init}.

4.5 Simulated Disc Tracking

We first evaluate the DFs in a simulated environment similar to the one in Haarnoja *et al.* (2016): The task is to track a red disc moving amongst varying numbers of distractor discs, as shown in Figure 4.1. The state consists of the position $\mathbf{p}$ and linear velocity $\mathbf{v}$ of the red disc.

The dynamics model that we used for generating the training data is

$$\begin{aligned}\mathbf{p}_{t+1} &= \mathbf{p}_t + \mathbf{v}_t + \mathbf{q}_{p,t}\\ \mathbf{v}_{t+1} &= \mathbf{v}_t - f_p\mathbf{p}_t - f_d\mathbf{v}_t^2 sign(\mathbf{v}_t) + \mathbf{q}_{v,t}\end{aligned}$$

The velocity update contains a force that pulls the discs towards the origin ($f_p = 0.05$) and a drag force that prevents too high velocities ($f_d = 0.0075$). $\mathbf{q}$ represents the Gaussian process noise.

The sensor network receives the current image at each step, from which it can estimate the position but not the velocity of the target. As we do not model collisions, the red disc can be occluded by the distractors or leave the image temporarily.

4.5.1 Data

We create multiple datasets with varying numbers of distractors, different levels of constant process noise for the disc position and constant or heteroscedastic process noise for the disc velocity. All datasets contain 2400 sequences for training, 300

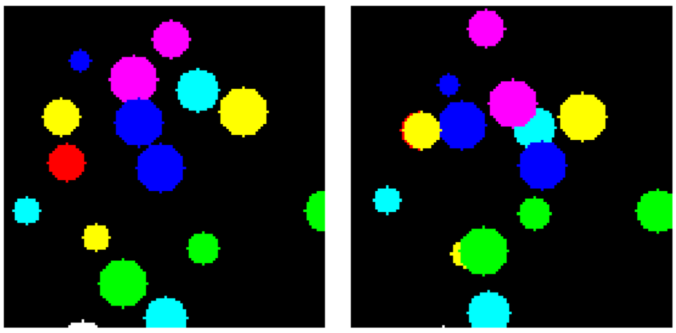


Figure 4.1: Two sequential observations from our simulated task. The filters need to track the red disc, which can be occluded by the other discs or leave the image temporarily.

Table 4.1: Sensor model and heteroscedastic observation noise architecture. Both output layers (for $\mathbf{z}$ and $diag(\mathbf{R})$) get fc 2's output as input.

Layer	Output Size	Kernel	Stride	Activation
Input $\mathbf{D}$	$100 \times 100 \times 3$	-	-	-
conv 1	$50 \times 50 \times 4$	9×9	2	ReLU
conv 2	$25 \times 25 \times 8$	9×9	2	ReLU
fc 1	16	-	-	ReLU
fc 2	32	-	-	ReLU
$\mathbf{z}$ (fc)	2	-	-	-
diag($\mathbf{R}$) (fc)	2	-	-	-

validation sequences and 303 sequences for testing. The sequences have 50 steps and the colors and sizes of the distractors are drawn randomly for each sequence.

4.5.2 Network Architectures and Initialization

The network architectures for the sensor model and heteroscedastic observation noise model are shown in Table 4.1. Tables 4.2a and 4.2b show the architecture for the learned process model and the heteroscedastic process noise. We denote fully connected layers by *fc* and convolutional layers by *conv*.

For the noisy initial state, we use $\mathbf{\Sigma}_{\text{init}} = 25 * \mathbf{I}_4$. When training from scratch, we initialize $\mathbf{Q}$ and $\mathbf{R}$ with $\mathbf{Q} = 100 * \mathbf{I}_4$ and $\mathbf{R} = 900 * \mathbf{I}_2$, reflecting the high

(a) Learned process model architecture

Layer	Output Size	Activation
Input $\mathbf{x}$	4	-
fc 1	32	ReLU
fc 2	64	ReLU
fc 3	64	ReLU
$\Delta\mathbf{x}$ (fc)	4	-

(b) Heteroscedastic process noise model architecture

Layer	Output Size	Activation
Input $\mathbf{x}$	4	-
fc 1	32	ReLU
fc 2	32	ReLU
diag($\mathbf{Q}$) (fc)	4	-

uncertainty of the untrained models.

4.5.3 Implementation and Parameters: dEKF, dUKF, dMCUKF

We first evaluate different design choices and filter-specific parameters for the DFs that are based on different versions of the Kalman filter. We seek settings that perform well and increase the stability of the filters during training. The dPF has much more implementation choices and will thus be treated in a separate section.

All experiments are performed on a dataset with 15 distractors and constant process noise ($\sigma_p = 0.1, \sigma_v = 2$). The filters are trained end-to-end on L_{NLL} and learn the sensor and process model as well as heteroscedastic observation and constant process noise models. We repeat each experiment two times to account for different initializations of the weights and report mean and standard errors.

dEKF

Of all DFs discussed here, the dEKF is the only filter without parameters or relevant implementation choices.

dUKF

The dUKF has three filter-specific scaling parameters, α, κ and β. As explained in Section 2.3.3, α and κ determine how far from the mean of the belief the sigma points are placed and how the mean is weighted in comparison to the other sigma points. β only affects the weight of the central sigma point when computing the covariance of the transformed distribution.

Experiment The original version of the UKF by Julier and Uhlmann (1997) uses a simple parameterization where $\alpha = 1$ and $\beta = 0$ are fixed and only κ varies. The authors recommend setting $\kappa = 3 - n$. α and β are used in the later proposed *scaled unscented transform* (Julier, 2002), for which Van Der Merwe (2004) suggest setting $\kappa = 0$, $\beta = 2$ and α to a small positive value.

We evaluate the original, simple parameterization as well as the one for the scaled transform. For the first, we test training the dUKF with κ values in $[-10, 10]$. In the second case, we evaluate $\alpha \in \{0.001, 0.1, 0.5\}$ but do not vary β, for which the value 2 is optimal when working with Gaussians.

Results The results show no significant differences between the different parameter settings or between using the original parameterization from Julier and Uhlmann (1997) and the scaled transform. Only for $\kappa \in \{-5, -10\}$ (i.e. $\kappa < -n$), the training failed due to a non-invertible matrix in the calculation of the Kalman Gain.

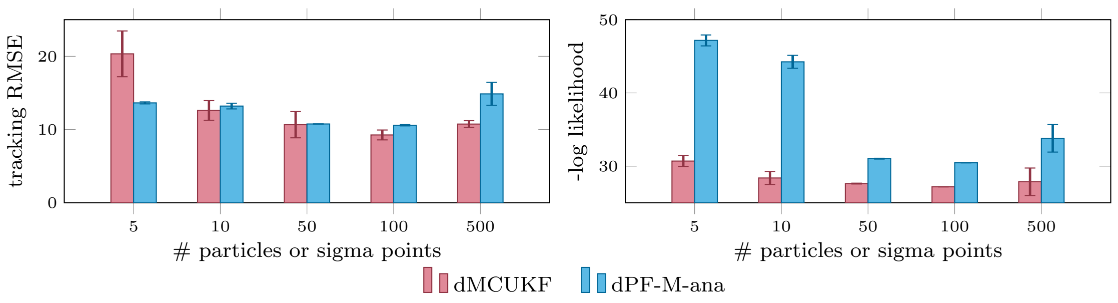


Figure 4.2: Results on disc tracking: Tracking error and negative log likelihood of the dMCUKF and dPF-M-ana (see Experiment 4.5.4) for different numbers of sampled sigma points or particles during training. At test time, we use 500 sigma points or particles.

The choice of the UKF parameters presumably becomes more important for problems with strongly non-linear dynamics or high-dimensional state spaces and should be re-evaluated for each task. We generally recommend using values for which $\lambda = \alpha^2(\kappa + n) - n$ is a small positive number. In the following, we will use $\alpha = 1$, $\kappa = 0.5$ and $\beta = 0$.

dMCUKF

In contrast to the dUKF, the dMCUKF simply samples pseudo sigma points from the current belief. Its only parameter thus is the number of sampled points during training N_{train} and testing N_{test}.

Experiment We train the dMCUKF with $N_{\text{train}} \in \{5, 10, 50, 100, 500\}$ and evaluate with $N_{\text{test}} \in \{10, 100, 500, 1000\}$.

Results Figure 4.2 shows the results for testing with 500 sigma points. We see that as few as ten sampled sigma points are enough for training the dMCUKF relatively successfully. The best results are obtained with 100 sigma points. Using more sigma points than this even slightly decreases the performance again. A possible explanation for this could be that the number of sigma points has a similar effect as the batch size, such that more sigma points result in smaller overall gradients.

The number of sigma points during testing does not have a large overall effect on the filter performance, as long as it is not smaller than the one used during training. In the following, we will use 100 points for training and 500 for testing. More complex problems with higher-dimensional states could, however, require more sigma points.

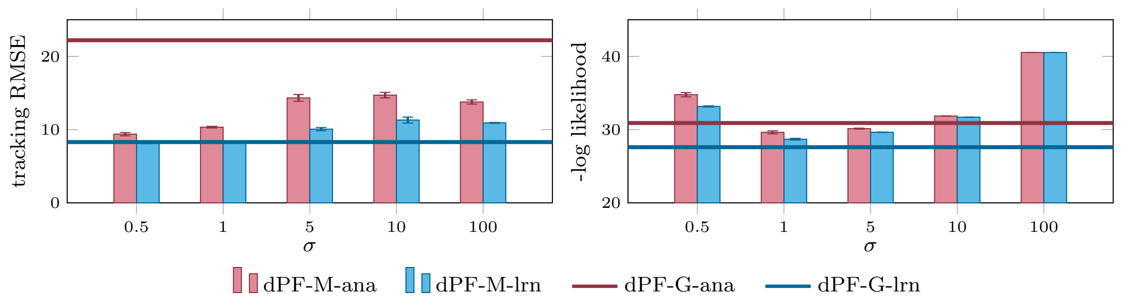


Figure 4.3: Results on disc tracking: Tracking error and negative log likelihood of the dPF-M and dPF-G, each with using the analytical (-ana) or learned (-lrn) observation update (see Experiment 4.5.4). The dPF-M is evaluated for different values of the fixed per-particle covariance matrix $\mathbf{\Sigma} = \sigma^2\mathbf{I}$ in the GMM.

4.5.4 Implementation and Parameters: dPF

The differentiable Particle filter has the highest number of different implementation choices that we will evaluate in the following. The experiments are performed in the same way as in the previous Section.

Belief Representation

When training on L_{NLL}, we have to chose how to represent the belief of the filter for computing the likelihood (see Section 4.3.5). We investigate using a single Gaussian (dPF-G) or a Gaussian Mixture Model (dPF-M). For the dPF-M, the covariance $\mathbf{\Sigma}$ of the single Gaussians in the Mixture Model is an additional parameter that has to be tuned. As our test scenario does not require tracking multiple hypotheses, the representation by a single Gaussian in dPF-G should be accurate for this task.

Experiment We evaluate the dPF-M with $\mathbf{\Sigma} = \sigma^2\mathbf{I}$ for $\sigma \in \{0.1, 0.5, 1, 5, 10, 100\}$ and compare it to the dPF-G. For each variant, we evaluate the dPF with a learned (-lrn) or analytical (-ana) observation update function. This will be further discussed in Experiment 4.5.4.

Results As shown in Figure 4.3, when training on L_{NLL} and using the analytical observation update, representing the belief of a dPF with a single Gaussian leads to much worse results than using a GMM to represent the belief. This could either mean that Equation 4.3 facilitates training or that approximating the belief with a single Gaussian removes useful information even when the task does not obviously require tracking multiple hypothesis. Interestingly, when using a learned observation update, this effect is not noticeable, which suggests the first theory.

When evaluating different values of $\mathbf{\Sigma} = \sigma^2\mathbf{I}$ for the dPF-M, we observe that smaller values generally lead to lower tracking errors for both versions of the observation update. This, however, only holds up to a certain point: using $\sigma = 0.1$ increases the tracking error again significantly and brings the NLL up to over 200 (which is why results are omitted in Figure 4.3).

While $\sigma = 0.5$ results in the best tracking errors, the best NLL values are achieved with $\sigma = 1$. Both smaller and larger values of σ lead to worse uncertainty estimates as the GMM under- or overestimates the uncertainty around the belief. We will thus use $\mathbf{\Sigma} = \mathbf{I}$ for the dPF-M in all following experiments on this task. It is, however, possible that different task could require different settings and we will thus re-evaluate this parameter for other problems.

Observation Update

As mentioned before, the likelihood for the observation update step of the dPF can be implemented with an analytical Gaussian likelihood function (dPF-(G/M)-ana) or with a neural network (dPF-(G/M)-lrn) as in Jonschkowski *et al.* (2018) and Karkus *et al.* (2018a).

Jonschkowski *et al.* (2018) predict the likelihood based on an encoding of the sensory data and the observable components of the (normalized) particle states. Our implementation, too, takes the 64-dimensional encoding of the raw observations (fc 3 in Table 4.1) and the observable particle state components as input. However, we decide not to normalize the particles, since having prior knowledge about the mean and standard deviation of each state component in the dataset might give an unfair advantage to the method over other variants.

Results Results for comparing the learned and analytical observation update can be found in Figure 4.3. Using a learned instead of an analytical likelihood function for updating the particle weights improves the tracking error of the dPF-M from 10.3±0.1 to 8.3±0.1 and the NLL from 29.6±0.2 to 28.7±0.1. For the dPF-G, the difference is even more dramatic, with an RMSE of 23.3±1.1 vs. 8.0±0.3 and an NLL of 31.0±0.05 vs. 27.5±0.1.

One possible explanation for this is that the learned observation likelihood function enables a better gradient flow through the observation update and thus facilitates learning, especially in the dPF-G. It could also help to mitigate numerical stability problems encountered when the analytical observation likelihoods and the resulting particle weights are extremely small.

However, the main difference between the learned an the analytical observation update is that the implicit observation noise model of the learned version does not have a predefined form. With the analytical version, we restrict the filter to use additive Gaussian noise that is ether constant or depends only on the raw sensory observations. The learned update, in contrast, enforces no functional form of the noise model. In addition, the noise can depend not only on the raw sensory data, but also on the observable components of the particle states. This means that the learned observation update is potentially much more expressive than the analytical one, which pays off when the Gaussian assumption made in the filtering algorithms does not hold.

While learning the observation update thus improves the performance of the dPF, we will still use the analytical variant in many of the following evaluations. The main reason for this is that the analytical observation update makes the observation noise model explicit and allows us to look at the sensor model and the observation noise in separation. This facilitates comparing between the dPF and the other DF variants and gives us control over the learned observation noise. For example, in Experiment 4.5.7, we compare learning constant or heteroscedastic observation noise. As the learned observation update always implicitly learns a heteroscedastic noise model, it does not support this kind of evaluation.

Resampling

The resampling step of the Particle filter discards particles with low weights and prevents particle depletion. It may, however, be disadvantageous during training since it is not fully differentiable (Jonschkowski *et al.*, 2018; Karkus *et al.*, 2018a; Zhu *et al.*, 2020). Karkus *et al.* (2018a) proposed soft resampling, where the resampling distribution is traded off with a uniform distribution to enable gradient flow between the weights of the old and new particles. This trade-off is controlled by a parameter $\alpha_{\text{re}} \in [0, 1]$. The higher α_{re}, the more weight is put on the uniform distribution. An alternative to soft resampling is to not resample at every timestep.

Experiment We test dPF-M-lrn and dPF-M-ana with different values of α_{re} and when resampling every 1, 2, 5 or 10 steps. With a training sequence length of 10, the last option results in resampling only once, before the last step of the sequence.

Results Our results in Figure 4.4 show that independent of how the observation update is implemented, resampling frequently improves the filter performance. This is in contrast to the results in Karkus *et al.* (2018a), where resampling only every second step improved performance in comparison to resampling at every step.

Soft-resampling also did not have much of a positive effect in our experiments, presumably because higher values of α_{re} decrease the effectiveness of the resampling step. We do, however, see a slight improvement of performance for the small value of $\alpha_{\text{re}} = 0.05$, especially when resampling is applied less frequently. In the following experiments, we will use $\alpha_{\text{re}} = 0.05$ and resample at every timestep.

Number of Particles

Finally, the user also has to decide how many particles to use during training and testing. As for the dMCUKF, we train the dPF-M-ana with $N_{\text{train}} \in \{5, 10, 50, 100, 500\}$

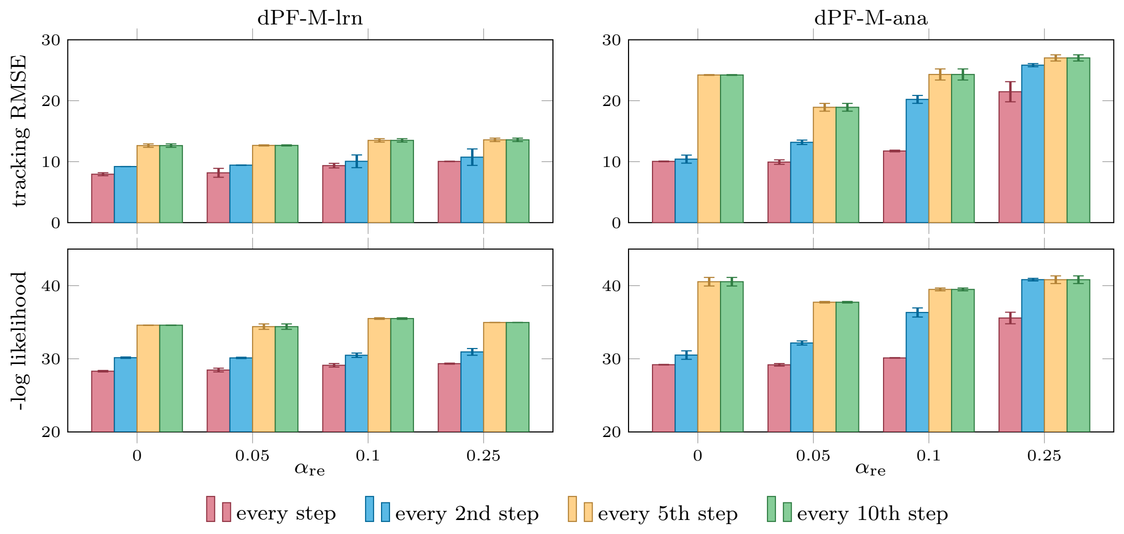


Figure 4.4: Results on disc tracking: Tracking error and negative log likelihood of the two dPF-M variants for different resampling rates and values of the soft resampling parameter α_{re}.

and evaluate with $N_{\text{test}} \in \{10, 100, 500, 1000\}$.

Results The results shown in Figure 4.2 are very similar to the results we obtained for the dMCUKF. In the following, we thus also use 100 particles during training and 1000 particles for testing.

4.5.5 Loss Function

In this experiment we compare the different loss functions introduced in Section 4.3.5, as well as a combination of the two $L_{\text{mix}} = 0.5(L_{\text{MSE}} + L_{\text{NLL}})$. Our hypothesis is that L_{NLL} is better suited for learning noise models, since it requires predicting the uncertainty about the state, while L_{MSE} only optimizes the tracking performance.

Experiment We use a dataset with 15 distractors and constant process noise ($\sigma_{q_p} = 0.1$, $\sigma_{q_v} = 2$). The filters learn the sensor and process model as well as heteroscedastic observation noise and constant process noise models.

Results As expected, training on L_{NLL} leads to much better likelihoods scores than training on L_{MSE} for all DFs, see Figure 4.5. The best tracking errors on the other hand are reached with L_{MSE}, as well as more precise sensor models.

For judging the quality of a DF, both likelihood and tracking error should be taken into account: While a low RMSE is important for all tasks that use the state

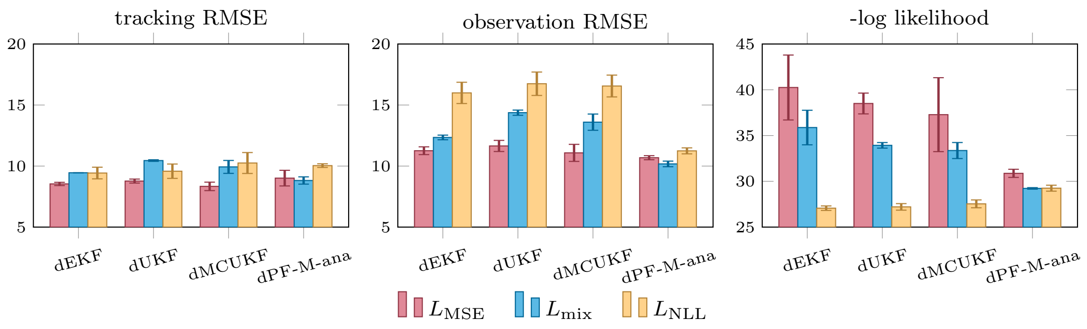


Figure 4.5: Results on disc tracking: Tracking error, observation error and negative log likelihood of dEKF, dUKF, dMCUKF and and dPF-M-ana trained with loss functions L_{MSE}, L_{NLL} or L_{mix}.

estimate, a good likelihood means that the uncertainty about the state is communicated correctly, which enables e.g. risk-aware planning and failure detection.

The combined loss L_{mix} trades off these two objectives during training. It does, however, not outperform the single losses in their respective objective. A possible explanation is that they can result in opposing gradients: All DFs tend to overestimate the process noise when trained only on L_{MSE}. This lowers the tracking error by giving more weight to the observations in dEKF, dUKF and dMCUKF and allowing more exploration in the dPF. But it also results in a higher uncertainty about the state, which is undesirable when optimizing the likelihood.

We generally recommend using L_{NLL} during training to ensure learning accurate noise models. If learning the process and sensor model does not work well, L_{NLL} can either be combined with L_{MSE} or the models can be pretrained.

4.5.6 Training Sequence Length

Karkus *et al.* (2018a) evaluated training their dPF on sequences of length $k \in \{1, 2, 4\}$ and found that using more steps improved results. We want to test if increasing the sequence length even further is beneficial. However, longer training sequences also mean longer training times (or more memory consumption). We thus aim to find a value for k with a good trade off between training speed and model performance.

Experiment We evaluate dEKF, dUKF, dMCUKF and dPF-M-ana on a dataset with 15 distractors and constant process noise ($\sigma_{q_p} = 0.1$, $\sigma_{q_v} = 2$). The filters learn the sensor and process model as well as heteroscedastic observation noise and constant process noise models. We train using L_{NLL} on sequence lengths $k \in \{1, 2, 5, 10, 25, 50\}$ while keeping the total number of examples per batch (steps $\times$ batch size) constant.

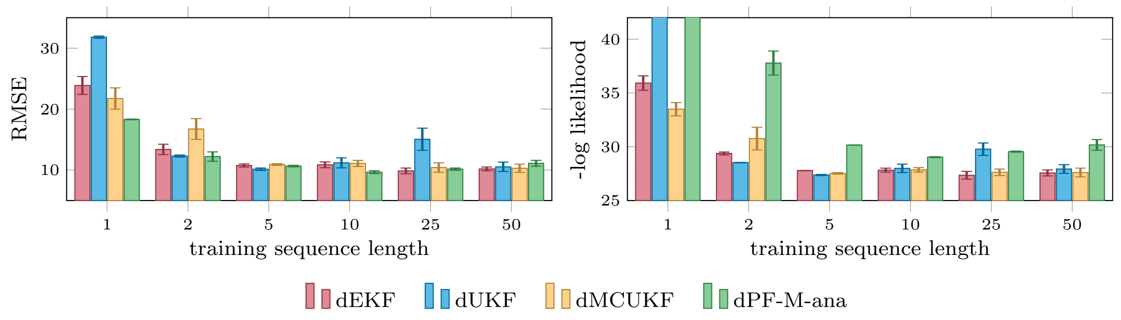


Figure 4.6: Results on disc tracking: Tracking error and negative log likelihood of the DFs trained with different sequence lengths. Each experiment was repeated two times and we report mean and standard error of the statistics. The cut-off NLL values for sequence length 1 are 65.8±3.8 for the dUKF and 85.7±1.6 for the dPF-M.

Results Our results in Figure 4.6 show that all filters benefit from longer training sequences much more than the results in Karkus *et al.* (2018a) indicated. However, while only one time step is clearly too little, returns diminish after around ten steps.

Why are longer training sequences helpful? One issue with short sequences is that we use noisy initial states during training. This reflects real-world conditions, but the noisy inputs hinder learning the process model. On longer sequences, the observation updates can improve the state estimate and thus provide more accurate input values.

We repeated the experiment without corrupting the initial state, but the results with $k \in \{1, 2\}$ got even worse: Since the DFs could now learn accurate process models, they did not need the observations to achieve a low training loss and thus did not learn a proper sensor model. On the longer test sequences, however, even small errors from the noisy dynamics accumulate over time if they are not corrected by the observations.

To summarize, longer sequences are beneficial for training DFs, because they demonstrate error accumulation during filtering and allow for convergence of the state estimate when the initial state is noisy. However, performance eventually saturates and increasing k also increased our training times. We therefore chose $k = 10$ for all experiments, which provides a good trade-off between training speed and performance.

4.5.7 Learning Noise Models

The following experiments analyze if and how well complex models of the process and observation noise can be learned through the filters.

To isolate the effect of the noise models, we use a fixed, pretrained sensor model and the true analytical process model, such that only the noise models are trained. We initialize $\mathbf{Q}$ and $\mathbf{R}$ with $\mathbf{Q} = \mathbf{I}_4$ and $\mathbf{R} = 100\mathbf{I}_2$. All DFs are trained with L_{NLL}

on different datasets with 30 distractors and increasing positional process noise. For the dPF, we only evaluate variants that use the analytical observation update and thus have an explicit observation noise model.

Heteroscedastic Observation Noise

We first test if learning more complex, heteroscedastic observation noise models improves the performance of the filters as compared to learning constant noise models. For this, we compare DFs that learn constant or heteroscedastic observation noise (the learned process noise model is constant) on two datasets with constant process noise ($\sigma_{q_p} \in 0.1, 3$, $\sigma_{q_v} = 2$) and 30 distractors.

To measure how well the predicted observation noise reflects the visibility of the target disc, we compute the correlation coefficient between the predicted $\mathbf{R}$ and the number of visible target pixels. We also evaluate the similarity between the learned and the true process noise model using the Bhattacharyya distance.

Results Results are shown in Table 4.3. When learning constant observation noise, the dPF-M is the only filter that performs well in terms of tracking error on at least one of the datasets: All other filters, (including the dPF-G) learn a very high $\mathbf{R}$ and thus mostly rely on the process model for their prediction. This is expected, since trusting the observations would result in wrong updates to the mean state estimate when the target disc is occluded. The PF-M does not use the mean of the particles in the likelihood computation, which makes it less sensitive to wrong observations and allows it to learn a lower $\mathbf{R}$.

Like Haarnoja *et al.* (2016), we find that heteroscedastic observation noise significantly improves the tracking performance of all DFs (except for the dPF-M). The strong negative correlation between $\mathbf{R}$ and the visible disc pixels shows that the DFs correctly predict higher uncertainty when the target is occluded. Only the dPF-M sometimes fails to learn this correlation well: Since it can already perform well with constant observation noise, it has less incentive to use state-dependent observation noise.

Finally, all DFs learn values of $\mathbf{q}_v$ that are close to the ground truth. For the position noise $\mathbf{q}_p$, however, we see a difference between learning constant or heteroscedastic observation noise: On the datasets with lower ground truth process noise, $\mathbf{q}_p$ is overestimated by all DFs. Results are especially bad when the learned observation noise model is constant. This could be because the bad tracking performance with constant observation noise prevents learning an accurate process model. The results for learning $\mathbf{q}_p$ indeed also improved when we enable learning a better process model by using the true initial state instead of a noisy one.

Table 4.3: Results for disc tracking: End-to-end learning of the noise models through the DFs on datasets with 30 distractors and different levels of process noise. While **Q** is always constant, we evaluate learning constant (const.) or heteroscedastic (hetero) observation noise **R**. We show the tracking error (RMSE), negative log likelihood (NLL), the correlation coefficient between predicted **R** and the number of visible pixels of the target disc (corr.) and the Bhattacharyya distance between true and the learned process noise model ($D_{\mathbf{Q}}$).

		$\sigma_{q_p} = 0.1$				$\sigma_{q_p} = 3.0$			
	R	RMSE	NLL	corr.	$D_{\mathbf{Q}}$	RMSE	NLL	corr.	$D_{\mathbf{Q}}$
dEKF	const.	21.0	32.2	-	2.89	25.9	32.4	-	0.068
	hetero.	**13.4**	**29.5**	-0.67	**0.96**	**13.8**	**29.1**	-0.78	**0.001**
dUKF	const.	21.6	32.4	0	2.812	27.0	32.5	-	0.101
	hetero.	**12.1**	**28.7**	-0.57	**0.756**	**13.9**	**29.2**	-0.78	**0.008**
dMCUKF	const.	22.0	32.2	-	3.253	25.4	32.5	-	0.366
	hetero.	**11.3**	**28.5**	-0.6	**1.423**	**13.8**	**29.1**	-0.78	**0.011**
dPF-G-ana	const.	22.7	32.2	-	3.151	26.7	32.5	-	**0.232**
	hetero.	**16.2**	**30.6**	-0.54	**3.106**	**18.4**	**30.8**	-0.60	0.270
dPF-M-ana	const.	**14.1**	**30.6**	-	2.684	24.4	49.2	-	0.339
	hetero.	15.2	30.8	-0.53	**2.666**	**14.9**	**37.0**	-0.78	**0.309**

Heteroscedastic Process Noise

The effect of learning heteroscedastic process noise has not yet been evaluated in related work. We create datasets with heteroscedastic ground truth noise, where the magnitude of $\mathbf{q}_v$ increases in three steps the closer to the origin the disc is. The positional process noise $\mathbf{q}_p$ remains constant.

We compare the performance of DFs that learn constant and heteroscedastic process noise. The observation noise is heteroscedastic in all cases.

Results As shown in Table 4.4, learning heteroscedastic models of the process noise is a bit more difficult than for the observation noise. This is not surprising, as the input values for predicting the process noise are the noisy state estimates.

Plotting the predicted values for **Q** (see Figure 4.7 for an example from the dEKF) reveals that all DFs learn to follow the real values for the heteroscedastic velocity noise relatively well, but also predict state dependent values for $\mathbf{q}_p$, which is actually constant. This could mean that the models have difficulties distinguishing between $\mathbf{q}_p$ and $\mathbf{q}_v$ as sources of uncertainty about the disc position. However, we can see the same behavior also on a dataset with constant ground truth process noise. We thus assume that the models rather pick up an unintentional pattern in our data: The probability of the disc being occluded turned out to be higher in the middle of the image. The filters react to this by overestimating $\mathbf{q}_p$ in the center, which results in an overall higher uncertainty about the state in regions

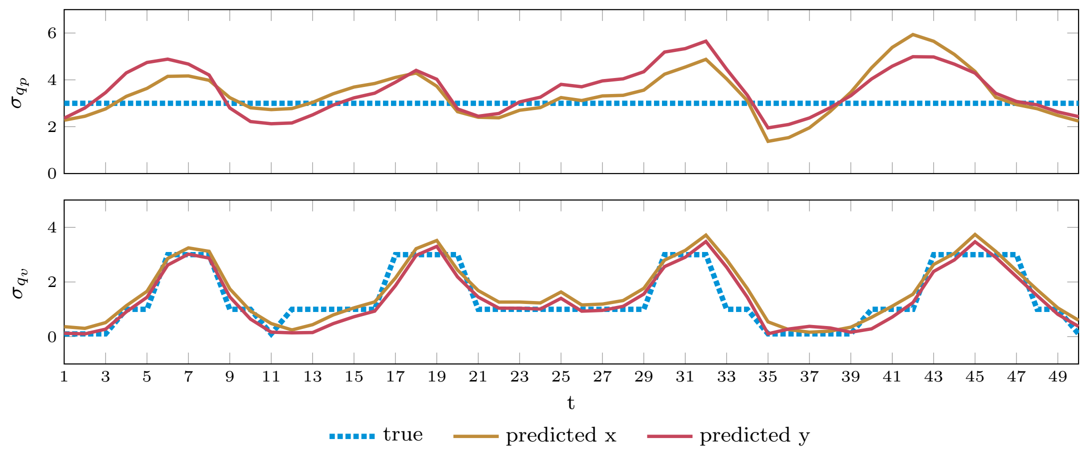


Figure 4.7: Predicted and true process noise from the dEKF over one test sequence of the disc tracking task. Our model predicts separate values for the x and y-coordinates of position and velocity, but the ground truth process noise has the same σ for both coordinates.

where occlusions are more likely.

Despite not being completely accurate, learning heteroscedastic noise models still increases performance of all DFs by a small but reliable value. Even when the ground-truth process noise model is constant, the DFs were able to improve their likelihood scores slightly by learning "wrongly" heteroscedastic noise models.

Correlated Noise

So far, we have only considered noise models with diagonal covariance matrices. In this experiment, we want to study if DFs can learn to identify correlations in the noise. For this, we create a new dataset with 30 distractors and constant, correlated process noise. The ground truth process noise covariance matrix is

$$\mathbf{Q}_{gt} = \begin{pmatrix} 9. & -3.6 & 1.2 & 5.4 \\ -3.6 & 9. & -0.6 & 0. \\ 1.2 & -0.6 & 4. & 0. \\ 5.4 & 0. & 0. & 4. \end{pmatrix}$$

We compare the performance of DFs that learn correlated or diagonal noise models on datasets with and without correlated process noise. Both the process and the observation noise model are also heteroscedastic.

Results Results are shown in Table 4.5. Overall, we note that learning correlated noise models has a small but consistent positive effect on the tracking performance of all DFs, even when the ground truth noise is not correlated. On the dataset with correlated ground truth process noise, we also observe an improvement of the

Table 4.4: Results on disc tracking: End-to-end learning of constant or heteroscedastic process noise $\mathbf{Q}$ on datasets with 30 distractors and different heteroscedastic or constant ($\sigma_{q_p} = 3.0$, $\sigma_{q_v} = 2.0$) process noise. $D_{\mathbf{Q}}$ is the Bhattacharyya distance between true and learned process noise.

		hetero. $\sigma_{q_v}, \sigma_{q_p} = 0.1$			hetero. $\sigma_{q_v}, \sigma_{q_p} = 3.0$			$\sigma_{q_p} = 3.0, \sigma_{q_v} = 2.0$		
	Q	RMSE	NLL	$D_{\mathbf{Q}}$	RMSE	NLL	$D_{\mathbf{Q}}$	RMSE	NLL	$D_{\mathbf{Q}}$
dEKF	const.	9.5	28.4	2.268	12.2	29.7	0.864	13.8	**29.1**	**0.001**
	hetero.	**8.4**	**27.4**	**1.705**	**11.6**	**29.0**	**0.351**	**13.6**	**29.1**	0.024
dUKF	const.	9.4	28.2	2.819	12.3	29.6	0.867	13.9	29.2	**0.008**
	hetero.	**8.6**	**27.4**	**1.679**	**11.9**	**29.1**	**0.4**	**13.7**	**29.1**	0.032
dMCUKF	const.	9.5	28.2	2.972	12.3	29.7	0.882	13.8	**29.1**	**0.011**
	hetero.	**8.6**	**27.5**	**1.915**	**11.8**	**29.1**	**0.42**	**13.7**	**29.1**	0.045
dPF-G-ana	const.	12.4	29.7	3.984	14.1	30.3	1.126	18.4	**30.8**	**0.270**
	hetero.	**12.1**	**29.4**	**3.695**	**13.8**	**29.8**	**0.755**	**18.2**	30.8	0.371
dPF-M-ana	const.	11.3	29.6	3.744	12.6	30.4	0.936	**14.9**	37.0	**0.309**
	hetero.	**9.6**	**28.7**	**3.327**	**11.9**	**29.9**	**0.589**	15.2	**36.2**	0.495

likelihood scores.

In terms of the Bhattacharyya distance between true and learned process noise covariance matrix, learning correlated models leads to a slight improvement for correlated ground truth noise and to slightly worse scores otherwise. This indicates that the models are able to uncover some, but not all correlations in the underlying data.

In summary, while learning correlated noise models does not influence the results negatively, it also does not lead to a very pronounced improvement over models with diagonal covariance matrices. Uncovering correlations in the process noise thus seems to be even more difficult than learning accurate heteroscedastic noise models.

4.5.8 Benchmarking

In the final experiment on this task, we compare the performance of the DFs among each other and to two LSTM models. We use an LSTM architecture similar to Jonschkowski *et al.* (2018), with one or two layers of LSTM cells (512 units each). The LSTM state is decoded into mean and covariance of a Gaussian state estimate.

Experiment All models are trained for 30 epochs. The DFs learn the sensor and process models with heteroscedastic, diagonal noise models. We compare their performance on datasets with 30 distractors and different levels of constant or heteroscedastic process noise. Each experiment is repeated two times to account for different initializations of the weights.

Table 4.5: Results on disc tracking: End-to-end learning of independent (*diagonal* covariance matrix) or correlated (*full* covariance matrix) process and observation noise models. We evaluate on one dataset with independent, constant process noise ($\sigma_{q_p} = 3.0$, $\sigma_{q_v} = 2.0$), one with independent heteroscedastic process noise ($\sigma_{q_p} = 3.0$), and one with correlated constant process noise. $D_\mathbf{Q}$ is the Bhattacharyya distance between true and learned $\mathbf{Q}$.

		diag. const. noise			diag. hetero. noise			correlated const. noise		
	Q	RMSE	NLL	$D_\mathbf{Q}$	RMSE	NLL	$D_\mathbf{Q}$	RMSE	NLL	$D_\mathbf{Q}$
dEKF	diag.	13.6	29.1	**0.024**	11.6	**29.0**	**0.351**	13.8	29.0	1.238
	full	**13.4**	**29.0**	0.096	**11.1**	**29.0**	0.549	**13.1**	**28.6**	**0.954**
dUKF	diag.	13.7	**29.1**	**0.032**	11.9	29.1	**0.4**	13.7	29.0	1.255
	full	**13.6**	**29.1**	0.135	**11.7**	**29.0**	0.512	**13.5**	**28.7**	**1.073**
dMCUKF	diag.	13.7	**29.1**	**0.045**	11.8	**29.1**	**0.42**	13.8	29.0	1.326
	full	**13.6**	**29.1**	0.155	**11.4**	**29.1**	0.602	**13.5**	**28.7**	**1.02**
dPF-G-ana	diag.	18.2	30.8	0.371	13.8	**29.8**	**0.755**	18.5	30.8	1.719
	full	**17.8**	**30.7**	**0.366**	**13.3**	30.0	1.156	**17.8**	**30.5**	**1.564**
dPF-M-ana	diag.	15.2	36.2	**0.495**	**11.9**	**29.9**	**0.589**	14.3	36.5	**1.6**
	full	**14.0**	**34.7**	1.1	12.2	34.0	1.263	**12.9**	**35.1**	1.638

Results The results in Table 4.6 show that all models except for the dPF-G-ana learn to track the target disc well and make reasonable uncertainty predictions. In terms of tracking error, the dPF with learned observation update performs best on all evaluated datasets. This, however, often does not extend to the likelihood scores. For the NLL, the dMCUKF instead mostly achieves the best results, however, not with a significant advantage over the other DFs.

If we exclude the dPF variants with learned observation model (which are more expressive than the other DFs) and the dPF-G-ana, we can see that the choice of the underlying filtering algorithm does not make a big difference for the performance on this task. The unstructured LSTM model, in contrast, requires two layers of LSTM cells (each with 512 units per layer) to reach the performance of the DFs. Unstructured models like LSTM can thus learn to perform similar to differentiable filters, but require a much higher number of trainable parameters than the DFs which increases computational demands and the risk of overfitting.

4.5.9 Summary

Our simulation experiments have shown that all DFs we evaluated are well suited for learning both, sensor and process model, as well as the associated noise models. While the LSTMs models could reach the same performance, they need significantly more trainable weights.

The results also showed that long enough training sequences are important for optimizing the performance of the DFs and confirmed that learning heteroscedastic noise models can be extremely beneficial to allow the filters to deal with events like occlusions.

Table 4.6: Results on disc tracking: Comparison between the DFs and LSTM models with one or two LSTM layers on three different datasets with 30 distractors and constant process noise with increasing magnitude. Each experiment is repeated two times and we report mean and standard error.

	$\sigma_{q_p} = 0.1$		$\sigma_{q_p} = 3.0$		$\sigma_{q_p} = 9.0$	
	RMSE	NLL	RMSE	NLL	RMSE	NLL
dEKF	9.0±0.88	27.1±0.36	11.9±0.48	28.3±0.09	17.6±0.04	29.5±0.04
dUKF	9.1±0.04	27.1±0.04	12.3±0.38	28.7±0.19	18.7±0.74	29.6±0.18
dMCUKF	8.7±0.80	**27.0±0.44**	11.4±0.55	28.2±0.25	18.3±0.99	**29.3±0.24**
dPF-M-ana	9.3±0.18	28.8±0.14	10.3±0.26	29.1±0.07	17.5±0.18	35.6±0.83
dPF-G-ana	19.9±0.08	30.6±0.08	21.5±0.56	30.9±0.12	28.3±0.15	31.8±0.03
dPF-M-lrn	8.2±0.26	28.5±0.16	**8.9±0.24**	29.0±0.12	**15.1±0.33**	34.2±0.47
dPF-G-lrn	**7.3±0.05**	27.3±0.03	8.9±0.44	**28.1±0.1**	15.9±0.34	30.1±0.02
LSTM-1	11.0±2.90	27.7±0.98	14.2±2.05	29.0±0.51	22.9±0.55	30.4±0.08
LSTM-2	9.0±0.2	27.1±0.07	11.9±0.47	28.4±0.06	19.5±1.07	29.9±0.23

In a direct comparison between the different DFs, the dPF with learned observation update had the best tracking performance. This can be explained by it being more expressive than the remaining DFs, which all perform similarly. The dPF is also the DF for which the user faces the most implementation choices. These choices are highly relevant for the filter performance, as can be seen in the large differences between using the learned or the analytical observation update or between using a single Gaussian or a GMM to represent the belief.

The simulated system we studied here, however, has relatively simple dynamics without strong nonlinearities and also does not pose a big challenge for the sensor model. In the following experiments, we will thus test the DFs on more challenging, real-world problems.

4.6 Kitti Visual Odometry

As a first real-world application we study the Kitti Visual Odometry problem (Geiger *et al.*, 2012) that was also evaluated by Haarnoja *et al.* (2016) and Jonschkowski *et al.* (2018). The task is to estimate the position and orientation of a driving car given a sequence of RGB images from a front facing camera and the true initial state.

The state is 5-dimensional and includes the position $\mathbf{p}$ and orientation θ of the car as well as the current linear and angular velocity v and $\dot{\theta}$. The real control input $\mathbf{u} = \begin{pmatrix}\dot{v} & \ddot{\theta}\end{pmatrix}^T$ is unknown and we thus treat changes in v and $\dot{\theta}$ as results of the process noise. The position and heading estimate can be updated analytically by Euler integration.

While the dynamics model is simple, the challenge in this task comes from the unknown actions and the fact that the absolute position and orientation of the car cannot be observed from the RGB images. At each timestep, the filters receive the current images as well as a difference image between the current and previous timestep. From this, the filters can estimate the angular and linear velocity to update the state, but the uncertainty about the position and heading will inevitably grow due to missing feedback.

4.6.1 Data

The Kitti Visual Odometry dataset consists of eleven trajectories of varying length (from 270 to over 4500 steps) with ground truth annotations for position and heading and image sequences from two different cameras collected at 10 Hz.

Following Haarnoja *et al.* (2016) and Jonschkowski *et al.* (2018), we build eleven different datasets. Each of the original trajectories is used as the test split of one dataset, while the remaining 10 sequences are used to construct the training and validation split.

To augment the data, we use the images from both cameras for each trajectory and also mirror the sequences. For training and validation, we extract 200 sequences of length 50 with different random starting points from each augmented trajectory. This results in 1013 training and 287 validation sequences. For testing, we extract sequences of length 100 from the augmented test-trajectory. The number of test sequences depends on the overall length of the test- trajectory.

When looking at the statistics of the eleven trajectories in the original Kitti dataset, Trajectory 1 can be identified as an outlier: It shows driving on a highway, where the velocity of the car is much higher than in all the other trajectories. As a result, the sensor models trained on the other sequences will yield bad results when evaluated on Trajectory 1. We will therefore mostly report results for only a ten-fold cross-validation that excludes the dataset for testing on Trajectory 1. We will refer to this as *kitti-10* while the full, eleven-fold cross validation will be denoted as *kitti-11*. In Section 4.6.6, results for both settings are reported, such that the influence of trajectory 1 becomes visible.

4.6.2 Network Architectures and Initialization

Sensor Network The network architectures for the sensor model and the heteroscedastic observation noise model are shown in Table 4.7. At each timestep, the input consists of the current RGB image and the difference image between the current and previous image. The network architecture for the sensor model is the same as was used in Haarnoja *et al.* (2016) and Jonschkowski *et al.* (2018).

Process Model Tables 4.8a and 4.8b show the architecture for the learned process model and the heteroscedastic process noise. For both models, we found it

Table 4.7: Sensor model and heteroscedastic observation noise architecture. Both output layers (for $\mathbf{z}$ and diag($\mathbf{R}$)) get fc 2's output as input.

Layer	Output Size	Kernel	Stride	Activation	Normalization
Input $\mathbf{D}$	$50 \times 150 \times 6$	-	-	-	-
conv 1	$50 \times 150 \times 16$	7×7	1×1	ReLU	Layer
conv 2	$50 \times 75 \times 16$	5×5	1×2	ReLU	Layer
conv 3	$50 \times 37 \times 16$	5×5	1×2	ReLU	Layer
conv 4	$25 \times 18 \times 16$	5×5	2×2	ReLU	Layer
dropout (0.3)	$25 \times 18 \times 16$	-	-	-	-
fc 1	128	-	-	ReLU	-
fc 2	128	-	-	ReLU	-
$\mathbf{z}$ (fc)	2	-	-	-	-
diag($\mathbf{R}$) (fc)	2	-	-	-	-

(a) Learned process model architecture. We use a modified version of the previous state $\mathbf{x}$ as input: $\bar{\mathbf{x}} = \left(v, \dot{\theta}, \cos\theta, \sin\theta\right)$

Layer	Output Size	Activation
Input $\bar{\mathbf{x}}$	4	-
fc 1	32	ReLU
fc 2	64	ReLU
fc 3	64	ReLU
$\Delta\mathbf{x}$ (fc)	5	-

(b) Heteroscedastic process noise model architecture. We use a modified version of the previous state $\mathbf{x}$ as input: $\bar{\mathbf{x}} = \left(v, \dot{\theta}\right)_{t-1}$

Layer	Output Size	Activation
Input $\left(v, \dot{\theta}\right)$	2	-
fc 1	32	ReLU
fc 2	32	ReLU
diag($\mathbf{Q}$) (fc)	5	-

to be important not to include the absolute position of the vehicle in the input values: The value range for the positions is not bounded, and especially for the dUKF variants, novel values encountered at test time often lead to a divergence of the filter.

Excluding these values from the network inputs for predicting the state update also makes sense intuitively, since they are not required for computing the update analytically, either. For the state-dependent process noise, we not only exclude the position, but also the orientation of the car, as any relationships between vehicle pose and noise that could be learned would be specific to the training trajectories.

In addition, we provide the process model with the sine and cosine of θ as input instead of using the raw orientation, to facilitate the learning. In general, dealing with angles in the state vector requires special attention: First, we correct angles to the range between $[-\pi, \pi]$ after every operation on the state vector. Second, it is important to correctly calculate the difference between angles (e.g. in the loss function) to avoid differences over 180deg. And third, computing the mean of several angles, e.g. for the particle mean in the dPF, requires converting the angles to a vector representation.

Initialization When creating the noisy initial states, we do not add noise to the absolute position and orientation of the vehicle, since the DFs have no way of correcting them. We use $\text{diag}(\mathbf{\Sigma}_{\text{init}}) = \begin{pmatrix} 0.01 & 0.01 & 0.01 & 25 & 25 \end{pmatrix}$ for the initial covariance matrix. When training the DFs from scratch, we initialize the covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ with $\text{diag}(\mathbf{Q}) = \begin{pmatrix} 0.01 & 0.01 & 0.01 & 100 & 100 \end{pmatrix}$ and $\mathbf{R} = 100\mathbf{I}_2$. This reflects the high uncertainty of the untrained models, but also the fact that the process noise should be higher for the velocities (to account for the unknown driver actions) than for the absolute pose.

4.6.3 Training Sequence Length and Filter Parameters

One special feature of the Visual Odometry task is that the the error on the estimated absolute vehicle pose will inevitably grow during filtering. As this could have an effect on the ideal training sequence length, we repeat the experiment from Section 4.5.6.

For the dPF-M, we also evaluate different values of the fixed per-particle covariance $\mathbf{\Sigma}$ for calculating the GMM-likelihood. We anticipate that this parameter, too, could be sensitive to the accumulating uncertainty in the problem.

In addition, we also reevaluate different values for parameterizing the sigma point selection and weighting in the dUKF.

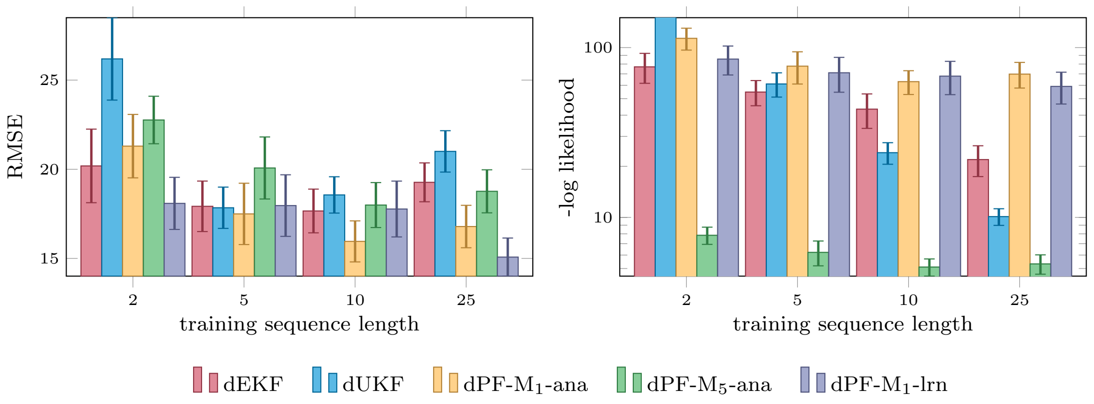


Figure 4.8: Results on *kitti-10*: Tracking error and negative log likelihood (NLL with logarithmic y axis) of dEKF, dUKF, dPF-M-ana and dPF-M-lrn trained with different sequence lengths. For the dPF-M-ana, we show two different values for the covariance $\mathbf{\Sigma}$ of the single Gaussians in the mixture model, $\mathbf{\Sigma} = \mathbf{I}$ and $\mathbf{\Sigma} = 5^2\mathbf{I}$. The cut-off NLL value for the dUKF on sequences of length 2 is 2015.8±518.1.

Training Sequence Length and dPF-M

Experiment We only test with the dEKF, dUKF, dPF-M-ana and dPF-M-lrn on *kitti-10*. The filters learn the sensor and process model as well as constant noise models. We train them using L_{NLL} on sequence lengths $k \in \{2, 5, 10, 25\}$ while keeping the total number of examples per batch (steps $\times$ batch size) constant.

For the dPF-M-ana, we also evaluate two different values of the per-particle covariance, $\mathbf{\Sigma} = \mathbf{I}$ and $\mathbf{\Sigma} = 5^2\mathbf{I}$.

Results The results shown in Figure 4.8 largely confirm the results obtained for the simulation dataset in Section 4.5.6. We again see that longer training sequences increase the tracking performance of all DFs up to a sequence length of around $k = 10$.

The dUKF seems to be most sensitive to the sequence length, with the highest tracking error and an extremely bad NLL score for sequences of length 2. Different from the simulation experiment, for both dEKF and dUKF, the NLL keeps decreasing strongly over the full evaluated sequence length range, despite the best RMSE already being reached at $k = 5$. We attribute this to the accumulating uncertainty about the vehicle pose. For the dPFs, in contrast, the likelihood behaves similarly to the RMSE.

In light of the longer training times with higher sequence lengths, we again decide to keep a training-sequence length of 10 when training the DFs from scratch. However, when only the noise models are trained, longer sequences can be used to improved results on the NLL.

For the dPF-M, the experiment also shows that the covariance of the single distributions in the GMM is an important tuning parameter. With $\mathbf{\Sigma} = \mathbf{I}$, we achieve the best tracking error, however, the likelihood does not reach the performance of dEKF and dUKF. The NLL values can be drastically improved by using larger $\mathbf{\Sigma}$, at the cost of a decreased tracking performance. Visual inspection of the position estimates shows that the particles remain relatively tightly clustered over the complete sequence, such that the likelihood of the GMM is not so different from the likelihood of the individual Gaussian components.

This clustered particle distribution can be explained by the characteristics of the task: The uncertainty in the system mainly stems from the velocity components that are affected by the unknown actions. However, by applying the observation update and resampling the particles at every step, we keep the variance in the velocity components small and thus prevent a stronger diffusion of the unobserved position components. This also explains why the dPF cannot profit as much as the dUKF and dEKF from seeing longer sequences during training.

The large influence of the tuning parameter $\mathbf{\Sigma}$ on the value of the likelihood, independent of the tracking performance, also shows that comparing likelihood scores between different probabilistic models can be difficult. In light of this, we decide to keep using $\mathbf{\Sigma} = \mathbf{I}$ for the better tracking error.

dUKF

We also repeat the evaluation of different values of the parameters α, κ and β for the dUKF described in Experiment 4.5.3. The experiment confirms our finding from the simulation experiment that the exact choice of the values does not have a significant effect on the filter performance. We thus keep the values at $\alpha = 1$, $\kappa = 0.5$ and $\beta = 0$.

4.6.4 Learning Noise Models

In this experiment, we want to test how much the DFs profit from learning the process and observation noise models end-to-end through the filters, as compared to using hand-tuned or individually learned noise models.

We also again compare learning constant or heteroscedastic noise models. In contrast to the previous experiment, we do not expect as large a difference between constant or heteroscedastic observation noise for this task, as the visual input does not contain occlusions or other events that would drastically change the quality of the predicted observations $\mathbf{z}$.

Experiment As in the experiments on simulated data (Section 4.5.7), we use a fixed, pretrained sensor model and the analytical process model, and only train the noise models. We initialize $\mathbf{Q}$ and $\mathbf{R}$ with $\mathbf{Q} = \mathbf{I}_5$ and $\mathbf{R} = \mathbf{I}_2$. All DFs are trained

Table 4.9: Results on *kitti-10*:RMSE and negative log likelihood for the DFs with different noise models (mean and standard error). Hand-tuned and Pretrained use fixed noise models whereas for the other variants, the noise models were trained end-to-end through the DFs. $\mathbf{R}_c$ indicates a constant observation noise model and $\mathbf{R}_h$ a heteroscedastic one (same for $\mathbf{Q}$). The best result per DF is highlighted in bold.

		Hand-tuned $\mathbf{R}_c\mathbf{Q}_c$	Pretrained $\mathbf{R}_c\mathbf{Q}_c$	Pretrained $\mathbf{R}_h\mathbf{Q}_h$	$\mathbf{R}_c\mathbf{Q}_c$	$\mathbf{R}_c\mathbf{Q}_h$	$\mathbf{R}_h\mathbf{Q}_c$	$\mathbf{R}_h\mathbf{Q}_h$
RMSE	dEKF	**15.8±1.1**	15.8±1.1	17.1±1.3	15.9±1.1	15.8±1.1	15.9±1.1	15.8±1.1
	dUKF	16.0±1.1	**15.9±1.1**	17.4±1.3	15.9±1.1	15.9±1.1	16.0±1.1	15.9±1.1
	dMCUKF	16.0±1.1	16.0±1.1	17.4±1.3	**15.9±1.1**	**15.9±1.1**	16.1±1.1	15.9±1.1
	dPF-M-ana	18.8±0.7	16.5±1.0	17.2±1.2	**15.9±1.1**	**15.9±1.1**	**15.9±1.1**	16.0±1.1
NLL	dEKF	292.9±43.8	128.1±16.7	96.2±15.6	29.3±4.1	27.1±5.0	29.2±3.7	**26.1±4.4**
	dUKF	294.4±43.7	128.5±16.6	96.6±15.5	29.0±4.0	**27.7±5.0**	29.8±4.0	28.3±5.4
	dMCUKF	343.7±50.5	149.8±19.3	112.4±17.9	23.6±3.4	**21.5±4.0**	24.9±3.4	21.8±4.2
	dPF-M-ana	86.7±6.6	61.8±8.5	**56.7±6.5**	62.4±9.7	58.4±10.0	61.6±9.9	59.1±9.7

with L_{NLL} and a sequence length of 25, which we found to be beneficial for learning the noise models in the previous experiment.

We compare the DFs when learning constant observation and process noise ($\mathbf{R}_c\mathbf{Q}_c$), constant observation and heteroscedastic process noise ($\mathbf{R}_c\mathbf{Q}_h$), heteroscedastic observation and constant process noise ($\mathbf{R}_h\mathbf{Q}_c$) and heteroscedastic observation and process noise ($\mathbf{R}_h\mathbf{Q}_h$). As one baseline, we use DFs with fixed constant noise models that reflect the average validation error of the pretrained sensor model and the analytical process model for one-step predictions: $\text{diag}(\mathbf{Q}) = \begin{pmatrix}10^{-4} & 10^{-4} & 10^{-6} & 0.01 & 0.16\end{pmatrix}^T$ and $\text{diag}(\mathbf{R}) = \begin{pmatrix}0.36 & 0.36\end{pmatrix}^T$. A second baseline fixes the noise models to those obtained by individual pretraining, where we evaluate both constant and heteroscedastic models. All DFs are evaluated on *kitti-10*.

Results The results in Table 4.9 show that learning the noise models end-to-end through the filters greatly improves the NLL but has no big effect on the tracking errors for this task. The DFs with the hand-tuned, constant noise model have the by far worst NLL because they greatly underestimate the uncertainty about the vehicle pose. The DFs that use individually trained noise models perform better, but are still overly confident.

For most of the DFs, we achieve the best results when learning constant observation and heteroscedastic process noise. The worst results are achieved when instead the observation noise is heteroscedastic and the process noise constant. This could indicate that the true process noise can be better modeled by a state-dependent noise model while learning heteroscedastic observation noise leads to overfitting to the training data. However, the differences are overall not very pronounced.

Finally, we also evaluated the DFs with full covariance matrices for the noise models. For the setting with constant observation and heteroscedastic process noise, using full instead of diagonal covariance matrices barely had any effect on the

Table 4.10: Results on *kitti-10*: RMSE and negative log likelihood for the DFs with different training schemes (mean and standard error). We compare individually trained process, sensor and noise models against finetuning only the sensor and process models (*Finetune Models*), finetuning only the noise models (*Finetune Noise*) and finteuning all models (*Finetune All*) through the DFs. We also report results for DFs trained *from scratch* without individual pretraining. The best results per DF are marked in bold.

		Individual	Finetune Models	Finetune Noise	Finetune All	From Scratch
RMSE	dEKF	15.8±1.1	17.2±1.5	**15.7±1.1**	16.1±1.4	16.7±1.2
	dUKF	15.9±1.1	15.9±1.1	15.8±1.1	15.3±1.1	**15.2±1.0**
	dMCUKF	15.9±1.1	15.6±1.0	15.8±1.1	**15.2±1.0**	15.9±0.8
	dPF-M-ana	16.7±0.9	17.5±1.1	**15.8±1.1**	16.1±1.2	16.5±1.1
NLL	dEKF	118.5±16.3	148.9±28.8	**39.8±5.1**	46.2±5.2	48.1±7.3
	dUKF	115.5±15.6	107.1±14.5	**45.9±5.5**	75.8±9.8	46.5±7.2
	dMCUKF	133.9±18.3	121.9±16.9	**44.1±5.3**	71.7±9.0	44.3±8.7
	dPF-M-ana	62.5±7.6	72.7±10.8	**60.0±9.2**	67.3±12.3	70.1±12.0

tracking error and only slightly improved the NLL (e.g. from 27.1±5.0 to 26.5±4.6 for the dEKF).

4.6.5 End-to-End versus Individual Training

Previous work Jonschkowski *et al.* (2018) has shown that end-to-end training through differentiable filters leads to better results than running the DFs with models that were trained individually. Specifically, pretraining the models individually and finetuning end-to-end resulted in the best tracking performance. As a possible explanation, the authors found that the individually trained process noise model predicted noise close to the ground truth whereas the end-to-end trained model overestimated to noise, which is believed to be beneficial for filter performance.

Does this mean that end-to-end training through DFs mostly affects the noise models? To test this, we pretrain all models individually and compare the performance of the DFs without finetuning, when finetuning only the noise models or only the sensor and process model and when finetuning everything. We also report results for training the DFs from scratch.

Experiment We pretrain sensor and process model and their associated (constant) noise models individually for 30 epochs. For finetuning, we load the pretrained models and finetune the desired parts for 10 epochs, while the end-to-end trained versions are trained for 30 epochs. All variants are evaluated using *kitti-10* and trained using L_{NLL}.

Results The results shown in Table 4.10 seem to confirm our hypothesis that end-to-end training through the DFs is most important for learning the noise models: Finetuning only the noise models improved both RMSE and NLL of all DFs in comparison to the variants without finetuning or with finetuning only the sensor and process model (except for the dMCUKF). For dEKF and dPF, finetuning the sensor and process model even decreased the performance on both measures.

In terms of tracking error, individual pretraining plus finetuning the noise models lead to the best results on dEKF and dPF, while dUKF and dMCUKF performed slightly better when finetuning both sensor and process model and their noise models (dMCUKF) or even learning both from scratch (dUKF). For the NLL, finetuning only the noise models lead to the best results for all DFs, followed in most cases by training from scratch.

To summarize, the results indicate that individual pretraining is helpful for learning the sensor and process models, but not for the noise models. End-to-end training through the DFs, on the other hand, again proved to be important for optimizing the noise models for the respective filtering algorithm but did not offer advantages for learning the sensor and process model.

4.6.6 Benchmarking

In the final experiment on this task, we compare the performance of the DFs to a LSTM model. We again use an LSTM architecture similar to Jonschkowski *et al.* (2018), but with only one layer of LSTM cells with 256 units. The LSTM state is decoded into an update for the mean and the covariance of a Gaussian state estimate. Like the process model of the DFs, the LSTM does not get the full initial state as input, but only those components that are necessary for computing a state update (velocities and sine and cosine of the heading). We chose this architecture in an attempt to make the learning task easier for the LSTM.

Experiment All models are trained for 30 epochs using L_{NLL}, except for the LSTM, for which L_{mix} lead to better results. The DFs learn the sensor and process models with constant noise models. We report their performance on *kitti-10* and *kitti-11*, for comparison with prior work.

Results The results in Table 4.11 show that by training all the models in the DFs from scratch, we can reach a performance that is competitive with prior work by Haarnoja *et al.* (2016), despite not relying on an analytical process model. We were, however, not able to reach the very good performance of the dPF reported by Jonschkowski *et al.* (2018). A possible cause for this could be that the normalization of the particles in the learned observation update used by Jonschkowski *et al.* (2018) helps the method to better deal with the higher overall velocity in Trajectory 1 of the kitti dataset.

In contrast to the DF, we were not able to train LSTM models that reached a good evaluation performance on this task, despite trying multiple different architectures and loss functions. Different from the experiments on the simulation task, increasing the number of units per LSTM-layer or using multiple LSTM layers even decreased the performance here. To complement our results, we also report an LSTM result from Haarnoja *et al.* (2016) that does better on the position error but worse on the orientation error. While these findings do not mean that a better performance could not be reached with unstructured models given better architectures or training routines, it still shows that the added structure of the filtering algorithms greatly facilitates learning in more complex problems.

For this task, the dPF-M-lrn achieves the overall best tracking result on *kitti-11*, closely followed by the dUKF and the dMCUKF. The dUKF has the lowest tracking error if we exclude the outlier Trajectory 1. One reason for their better performance in comparison to the dEKF could be that the dynamics of the Visual Odometry task are more strongly non-linear than in the previous experiments. Both UKF and PF can convey the uncertainty more faithfully in this case, which could lead to better overall results when training on L_{NLL}. Given the relatively large standard errors, the differences between the DFs are, however, not significant.

Interestingly, we find that the difference in performance between the dPF variants with learned or analytical observation update is not as pronounced as in the results we obtained for the simulation experiment (Section 4.5.4). In particular, the dPF-G-lrn now performs similarly bad as the dPF-G-ana.

4.6.7 Summary

Our experiments on the Kitti Visual Odometry problem showed that even on this more complex, real-world task, the DFs can still learn both, observation and process models, as well as the associated noise models from scratch. We were not able to reach a similar performance with an unstructured LSTM model, confirming that the algorithmic structure of the DFs greatly facilitates the model-learning.

In contrast to the simulation experiments, learning heteroscedastic noise models did not improve the performance of the DFs much - most likely because the Kitti task can be described sufficiently well by constant noise models. In particular, the visual observations do not feature occlusions that would require ignoring some of the observed images altogether. The bad performance of all DFs on Trajectory 1 with its much higher velocity, however, also shows that the ability to detect input values outside of the training distribution would be a valuable addition to current DFs.

While the choice of constant or heteroscedastic noise models did not affect the filtering performance much, our experiments still confirmed that learning the noise models end-to-end through the filters is important for the DFs to calibrate their uncertainty estimates. Individual pretraining of the required models works well for the process and sensor model, but results in overly confident noise models. Training or finetuning the noise models end-to-end through the filters improved

Table 4.11: Results on Kitti: Comparison between the DFs and LSTM (mean and standard error). Numbers for prior work BKF*, LSTM* taken from Haarnoja *et al.* (2016) and DPF* taken from (Jonschkowski *et al.*, 2018). Both, BKF* and DPF* use a fixed analytical process model while our DFs learn both, sensor and process model. $\frac{\mathrm{m}}{\mathrm{m}}$ and $\frac{\mathrm{deg}}{\mathrm{m}}$ denote the translation and rotation error at the final step of the sequence divided by the overall distance traveled.

		RMSE	NLL	$\frac{m}{m}$	$\frac{\text{deg}}{m}$
kitti-11	dEKF	26.5±9.9	254.8±206.7	0.25±0.05	0.08±0.009
	dUKF	24.2±9.0	313.9±267.5	**0.21±0.04**	**0.08±0.007**
	dMCUKF	24.7±8.8	292.8±248.6	0.23±0.04	0.08±0.012
	dPF-M-ana	26.3±9.9	102.9±34.6	0.24±0.04	0.08±0.009
	dPF-G-ana	34.3±9.2	90.8±68.8	0.33±0.04	0.17±0.035
	dPF-M-lrn	**23.4±8.2**	**81.8±33.3**	0.22±0.04	0.09±0.014
	dPF-G-lrn	30.5±8.3	122.8±85.1	0.31±0.06	0.17±0.049
	LSTM	40.4±8.6	2836.5±1293.5	0.52±0.05	0.08±0.008
	LSTM*	-	-	0.26	0.29
	BKF*	-	-	0.21	0.08
	DPF*	-	-	0.15±0.015	0.06±0.009
kitti-10	dEKF	16.7±1.2	48.1±7.3	0.21±0.03	0.08±0.01
	dUKF	**15.2±1.0**	46.5±7.2	**0.18±0.02**	**0.08±0.008**
	dMCUKF	15.9±0.8	44.3±8.7	0.2 ±0.03	0.08±0.013
	dPF-M-ana	16.5±1.1	70.1±12.0	0.21±0.02	0.08±0.007
	dPF-G-ana	25.2±1.9	**22.4±4.7**	0.3 ±0.04	0.18±0.038
	dPF-M-lrn	15.2±1.1	48.9±6.2	0.19±0.03	0.09±0.015
	dPF-G-lrn	22.9±3.7	38.4±11.1	0.27±0.06	0.18±0.053
	LSTM	32.4±3.4	1583.0±352.9	0.51±0.06	0.08±0.008

both, tracking performance and uncertainty estimates.

4.7 Planar Pushing

In the Kitti Visual Odometry problem, the main challenges were the unknown actions and dealing with the inevitably increasing uncertainty about the vehicle pose. With planar pushing, our second real-robot experiment in contrast addresses a task with much more complex dynamics. Apart from having non-linear and discontinuous dynamics (when the pusher makes or breaks contact with the object), Bauza and Rodriguez (2017) also showed that the noise in the system can be best captured by a heteroscedastic noise model.

With 10 dimensions, the state representation we use is also much larger than in our previous experiments. $\mathbf{x}$ contains the 2D position $\mathbf{p}_o$ and orientation θ of the object, as well as the two friction-related parameters l and α_m. In addition, we include the 2D contact point between pusher and object $\mathbf{r}$, the normal to the object's surface at the contact point $\mathbf{n}$ and a contact indicator s. The control input $\mathbf{u}$ contains the start position $\mathbf{p}_u$ and movement $\mathbf{v}_u$ of the pusher.

An additional challenge of this task is that $\mathbf{r}$ and $\mathbf{n}$ are only properly defined and observable when the pusher is in contact with the object. We set the labels for $\mathbf{n}$ to zeros and $\mathbf{r} = \mathbf{p}_u$ for non-contact cases.

Dynamics We use the analytical model introduced in Chapter 2.2 to predict the linear and angular velocity of the object ($\mathbf{v}_o$, ω) given the previous state and the pusher motion $\mathbf{v}_u$. However, predicting the next $\mathbf{r}$, $\mathbf{n}$ and s is not possible with this model since this would require access to a representation of the object shape.

For $\mathbf{r}$, we thus use a simple heuristic that predicts the next contact point as $\mathbf{r}_{t+1} = \mathbf{r}_t + \mathbf{v}_{u,t}$. $\mathbf{n}$ and s are only updated when the angle between pusher movement and (inwards facing) normal is greater than 90°. In this case, we assume that the pusher moves away from the object and set s_{t+1} and $\mathbf{n}_{t+1}$ to zeros.

Observations Our sensor network receives simulated RGBXYZ[2] images as input and outputs the pose of the object, the contact point and normal as well as whether the push will be in contact with the object during the push or not.

Apart from from the latent parameters l and α_m, the orientation of the object, θ, is the only state component that cannot be observed directly. Estimating the orientation of an object from a single image would require a predefined "zero-orientation" for each object, which is impractical. Instead, we train the sensor network to predict the orientation relative to the object pose in the initial image of each pushing sequence.

[2] Color images with extra channels for the 3D coordinates of each pixel (in camera frame)

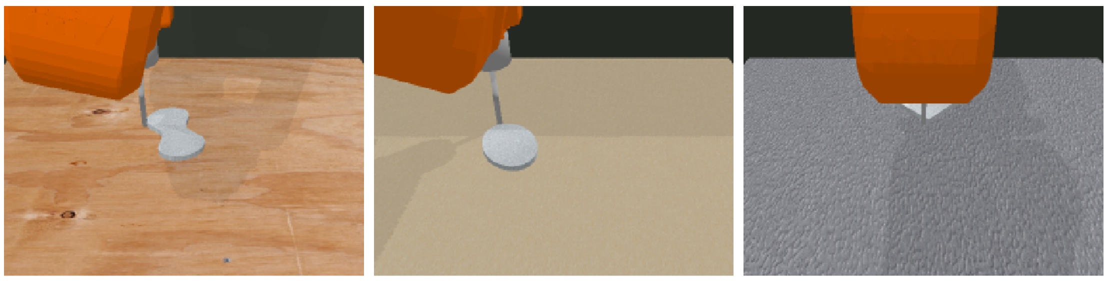


Figure 4.9: Examples of the rendered RGB images that we use as observations in this section. In contrast to the images used in Chapter 3, the images here are taken from a realistic viewpoint and include the robot arm, that can partially occlude the object as in the last example.

4.7.1 Data

We again use the data from the MIT Push dataset (Yu *et al.*, 2016) as a basis for constructing our datasets. However, in contrast to the experiments presented in Chapter 3, here, we chose a more realistic view-point for rendering images that places the camera in front of the robot. The images are thus taken from an angle and also show the robot arm. The arm and pusher frequently occlude parts of the object but complete occlusions are rare. Figure 4.9 shows example views.

We use pushes with a velocity of 50 $\frac{mm}{s}$ and render images with a frequency of 5 Hz. This results in short sequences of about five images for each push in the original dataset. We extend them to 20 steps for training and validation and 50 steps for testing by chaining multiple pushes and adding in-between pusher movement when necessary. The resulting dataset contains 5515 sequences for training, 624 validation sequences and 751 sequences for testing.

4.7.2 Network Architectures and Initialization

Sensor Network Our architectures for the sensor network is very similar to the one used in Chapter 3, where only the object position $\mathbf{p}_o$ is estimated from the full image while the contact-related state components ($\mathbf{r}$, $\mathbf{n}$, s) are computed from a smaller glimpse around the pusher location.

For predicting the orientation of the object, we extract a second glimpse from the full image, this time centered on the estimated object position. A small CNN then predicts the change in orientation between the glimpse extracted from the initial image in the sequence and the glimpse at the current time step.

The sensor network predicts object position, contact point and normal in pixel space because predictions in this space can be most directly related to the input image and the predicted feature maps. To this end, we also transform the action into pixel space before using it (together with the glimpse encoding) as input for predicting the contact point and normal. The pixel predictions are then transformed back to to world-coordinates using the depth measurements and camera

(a) Learned process model architecture.

Layer	Output Size	Activation
Input $(\mathbf{x}, \mathbf{v}_u)$	12	-
fc 1	256	ReLU
fc 2	128	ReLU
fc 3	128	ReLU
$\Delta\mathbf{x}$ (fc)	10	-

(b) Heteroscedastic process noise model architecture. We use a modified version of the previous state $\mathbf{x}$ as input: $\bar{\mathbf{x}}$ does not include the latent parameter l.

Layer	Output Size	Activation
Input $(\bar{\mathbf{x}}, \mathbf{v}_u)$	11	-
fc 1	128	ReLU
fc 2	64	ReLU
diag$(\mathbf{Q})$ (fc)	10	-

information. The resulting sensor network including the layers for computing the heteroscedastic observation noise is illustrated in Figure 4.10.

Process Model Tables 4.12a and 4.12b show the architecture for the learned process model and the heteroscedastic process noise. One problem we noticed is that the estimates for l sometimes diverge during filtering if the DFs estimate that the pusher is in contact with the object while it is not. Just as for the absolute position of the vehicle in the Kitti task, we thus found it important for the stability of the dUKF and dMCUKF to not make the heteroscedastic process noise model dependent on l.

Note that in the filter state, we measure $\mathbf{p}_o$ and $\mathbf{r}$ in millimeter and θ and α_m in degree. To avoid having too large differences between the magnitudes of the state components, we downscale l by a factor of 100. $\mathbf{n}$ is a dimensionless unit vector and s should take values between 0 and 1.

To keep the filters stable during training, we found it necessary to enforce maximum and minimum values for α_m and l. Both α_m and l cannot become negative. The opening angle of the friction cone, α_m, should also not be larger than 90°, while we limit l to be in the range of $[0.1, 5000]$ to ensure that the computations in the analytical model remain numerically stable.

Initialization For the initial covariance matrix, we use $\sqrt{\text{diag}(\mathbf{\Sigma}_{\text{init}})} = \begin{pmatrix}50 & 50 & 10^{-3} & 5 & 5 & 50 & 50 & 0.5 & 0.5 & 0.5\end{pmatrix}^T$. When training the noise models, we initialize $\mathbf{Q}$ and $\mathbf{R}$ with diag$(\mathbf{Q}) = \mathbf{I}_{10}$ and $\mathbf{R} = \mathbf{I}_8$.

4.7.3 Learning Noise Models

In this experiment, we again evaluate how much the DFs profit from learning the process and observation noise models end-to-end through the filters. In contrast to the Kitti task, for pushing, we expect both heteroscedastic observation and process noise to be advantageous, since the visual observations feature at least partial

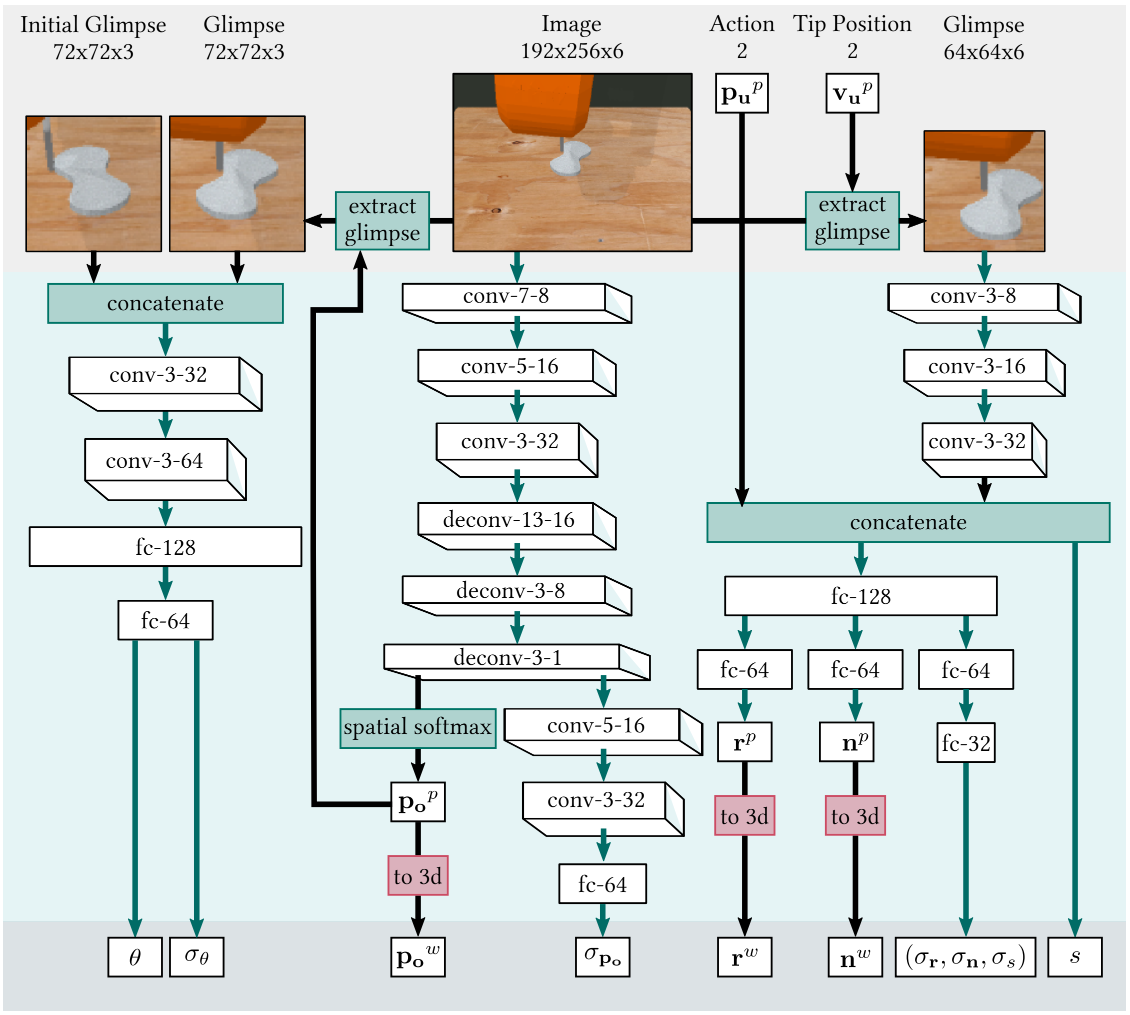


Figure 4.10: Architecture of the sensor network and heteroscedastic observation noise model for planar pushing. We use 6-channel RGBXYZ images as input for computing the object position and contact related state components. The object orientation is estimated relative to the initial orientation by comparing the RGB glimpse centered on the current estimated object position to the initial one.
White boxes represent tensors, green arrows and boxes indicate network layers, whereas black arrows represent dataflow without processing. For convolution (conv) and deconvolution (deconv) layers, the numbers in each tensor are the kernel size and number of output channels of the layer that produced it. For fully connected layers (fc), the number corresponds to the number of output channels.
With the exception of the output layers, all convolution, deconvolution and fully connected layers are followed by ReLU non-linearities. The (de)convolution layers also use layer normalization.

Table 4.13: Results for planar pushing: Translation (tr) and rotation (rot) error and negative log likelihood for the DFs with different noise models evaluated on the pushing task (mean and standard error). The hand-tuned DFs use fixed noise models whereas for the other variants, the noise models were trained end-to-end through the DFs. $\mathbf{R}_c$ indicates a constant observation noise model and $\mathbf{R}_h$ a heteroscedastic one (same for $\mathbf{Q}$). The best result per DF and metric is highlighted in bold.

		Hand-tuned $\mathbf{R}_c\mathbf{Q}_c$	$\mathbf{R}_c\mathbf{Q}_c$	$\mathbf{R}_h\mathbf{Q}_c$	$\mathbf{R}_c\mathbf{Q}_h$	$\mathbf{R}_h\mathbf{Q}_h$
tr [mm]	dEKF	6.22	4.45	4.61	4.44	**4.38**
	dUKF	4.87	4.44	5.25	**4.43**	4.45
	dMCUKF	4.73	4.42	4.8	4.39	**4.35**
	dPF-M-ana	18.13	5.07	4.92	5.32	**4.64**
	dPF-G-ana	17.95	**5.48**	35.57	210.45	10.92
rot [°]	dEKF	10.49	10.00	**9.71**	10.15	9.97
	dUKF	9.87	9.91	**9.73**	10.05	10.00
	dMCUKF	**9.78**	9.95	9.93	10.04	9.85
	dPF-M-ana	16.18	10.18	**9.92**	10.39	10.06
	dPF-G-ana	16.56	10.27	11.27	43.41	**10.25**
NLL	dEKF	265.17	126.69	33.09	79.24	**26.48**
	dUKF	378.08	84.12	33.06	81.55	**27.61**
	dMCUKF	130.22	78.53	30.43	64.12	**30.1**
	dPF-M-ana	353.25	128.15	104.40	103.21	**82.46**
	dPF-G-ana	> 16m	12,089.71	34.18	5,789.83	**31.60**

occlusions and the dynamics of pushing have been previously shown to exhibit heterostochasticity (Bauza and Rodriguez, 2017).

To test this hypothesis, we compare DFs that learn constant or heteroscedastic noise models to DFs with hand-tuned, constant noise models that reflect the average test error of the pretrained sensor model and the analytical process model.

Experiment As in the corresponding experiments on the previous tasks (Section 4.5.7 and Section 4.6.4), we use the fixed, pretrained sensor model and the analytical process model, and only train the noise models. All DFs are trained for 15 epochs on L_{NLL}.

The diagonals of the hand-tuned models are
$\sqrt{\text{diag}(\mathbf{Q})} = \begin{pmatrix}0.23 & 0.23 & 0.37 & 0.01 & 0.01 & 0.7 & 0.7 & 0.1 & 0.1 & 0.13\end{pmatrix}^T$ and
$\sqrt{\text{diag}(\mathbf{R})} = \begin{pmatrix}3.0 & 2.5 & 8.8 & 3.3 & 1.0 & 0.1 & 0.1 & 0.3\end{pmatrix}^T$.

Results The results shown in Table 4.13 again demonstrate that learning the noise models end-to-end through the structure of the filtering algorithms is beneficial. With learned models, all DFs reach much better likelihood scores than with the hand-tuned variants. For the dEKF and especially the dPF, the tracking performance also improves significantly.

Comparing the results between constant and heteroscedastic noise models also confirms our hypothesis that for the pushing task, heteroscedastic noise models are beneficial for both observation and process noise. While all DFs reach the best NLL when both noise models are state-dependent, the effect on the tracking error is, however, less clear.

For For all DFs but the dPF-M-ana, learning a heteroscedastic observation noise model leads to a much bigger improvement of the NLL than learning heteroscedastic process noise. Similar to the simulated disc tracking task, the input dependent noise model allows the DFs to better deal with occlusions in the observations, which again reflects in a negative correlation between the number of visible object pixels and the predicted positional observation noise.

4.7.4 Benchmarking

In the final experiment, we compare the performance of the DFs to an LSTM model on the pushing task. As before, we use a model with one LSTM layer with 256 units. The LSTM state is decoded into an update for the mean and the covariance of a Gaussian state estimate.

Experiment All models are trained for 30 epochs using L_{mix}. As initial experiments showed that learning sensor and process model jointly from scratch is very difficult for this task due to the more complex architectures, we pretrain both models. The sensor and process models are finetuned through the DFs and they learn heteroscedastic noise models. The LSTM, too, uses the pretrained sensor model, but not the process model.

Results As shown in Table 4.14, even with a learned process model, all DFs (except for the dPF-M-lrn) perform at least similar to their pendants in the previous experiment where we used the analytical process model. dEKF, dUKF and dMCUKF even reach a higher tracking performance than before. As noted in Chapter 3.6.5, this can be explained by the quasi-static assumption of the analytical model being violated for push velocities above 20 $\frac{mm}{s}$.

The LSTM model, again, does not reach the performance of the DFs. One disadvantage of the LSTM here is that in contrast to the DFs, we cannot isolate and pretrain the process model. In contrast to the previous tasks, the dPF variant with the learned likelihood function, however, performs even worse than the LSTM for planar pushing. This is likely due to the complex sensor model and the high-dimensional state that make learning the observation likelihood much more challenging.

Table 4.14: Results on pushing: Comparison between the DFs and LSTM. Process and sensor model are pretrained and get finetuned end-to-end. The DFs learn heteroscedastic noise models. Each experiment is repeated three times and we report mean and standard errors.

	RMSE	NLL	tr [mm]	rot [°]
dEKF	26.0±0.72	33.9±3.86	**3.5 ± 0.02**	8.8±0.22
dUKF	**24.4 ± 0.30**	**31.1 ± 1.90**	3.7±0.06	8.8±0.14
dMCUKF	24.7±0.07	34.1±3.57	3.7±0.05	**8.8 ± 0.06**
dPF-M-ana	35.2±0.83	117.6±5.61	5.6±0.23	10.4±0.38
dPF-G-ana	43.5±5.85	35.4±1.48	6.7±1.21	11.8±0.67
dPF-M-lrn	56.0±2.75	483.6±1.49	11.7±0.82	18.9±0.04
dPF-G-lrn	55.0±0.99	40.7±0.83	10.7±0.20	19.9±0.52
LSTM	47.4±0.35	35.4±0.24	8.8±0.17	19.0±0.001

4.8 Conclusions

Our experiments have shown that all DFs we evaluated are well suited for learning both sensor and process model, and the associated noise models. For simpler tasks like the simulated tracking task and the Kitti Visual Odometry task, all of these models can be learned end-to-end without pretraining. Only the pushing problem with its large state and complex dynamics and sensor model required pretraining to achieve good results.

In comparison to unstructured LSTM models, the DFs generally use fewer weights and achieve better results, especially on complex tasks. While training better LSTM models might be possible for a more experienced user, using the algorithmic structure of the filtering algorithms definitely facilitated the learning problem. In addition, the structure of DFs allows us to pretrain components such as the process model that are not explicitly accessible in LSTMs.

The direct comparison between the DFs with different underlying filtering algorithms showed no clear winner. Only the dPF with learned observation update performed notably better than the other variants on the simulated example task and was least affected by the outlier-trajectory of the Kitti-task. This variant relaxes some of the assumptions that the filtering algorithms encode by not relying on an explicit sensor or observation noise model. Its good performance thus shows that the priors enforced by the algorithm choice can also be harmful if they do not hold in practice, such as the Gaussian noise assumption.

Our experiments suggest that for learning the sensor and process model, end-to-end training through the filters is convenient, but provides no advantages over training the models individually. End-to-end training, however, proved to be essential for optimizing the noise models for their respective filtering algorithm. In contrast to end-to-end trained models, both hand-tuned and individually trained noise models did not result in optimal performance of the DFs.

Training noise models through DFs also enables learning more complex noise models than the ones used in learning-free, hand-tuned filters. We demonstrated that noise models with full instead of diagonal covariance matrices, but especially heteroscedastic noise model, can significantly improve the tracking accuracy and uncertainty estimates of DFs.

The main challenge in working with differentiable filters is keeping the training stable and finding good choices for the numerous hyper-parameters and implementation options of the filters. While we hope that this work provides some orientation about which parameters matter and how to set them, we still recommend using the dEKF for getting started with differentiable filters. It is not only the most simple of the DFs we evaluated, but it also proved to be relatively insensitive to sub-optimal initializations of the noise models and was the most numerically stable during training. Especially for tasks with strongly non-linear dynamics, the dUKF, dMCUKF or dPF can, however, ultimately achieve a better tracking performance.

One interesting direction for future research that we have not attempted here is to optimize parameters of the filtering algorithms, such as the scaling parameters of the dUKF or the fixed covariance of the mixture model components in the dPF-M, by end-to-end training. It could also be interesting to implement DFs with other underlying filtering algorithms. For example, the pushing task we evaluated here could potentially be better handled by a Switching Kalman filter (Murphy, 1998) that explicitly treats the contact state as a binary decision variable.

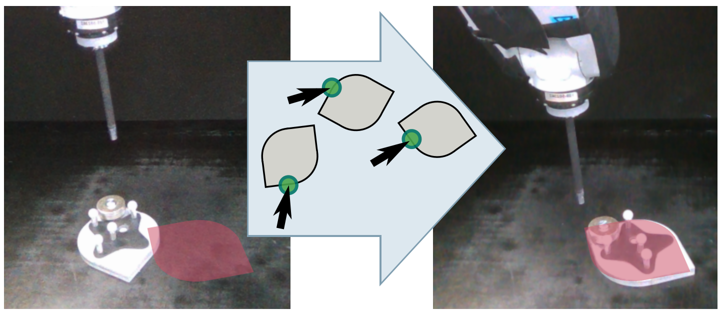
In addition, all of our DFs perform badly on the outlier trajectory of the Kitti dataset which features a much higher driving velocity than the other trajectories we used for training the model. This shows that the ability to detect input values outside of the training distribution would be a valuable addition to current DFs.

Chapter 5
Planning Contact Interactions

This chapter is based on a previously published article

Alina Kloss, Maria Bauza, Jiajun Wu, Joshua B. Tenenbaum, Alberto Rodriguez and Jeannette Bohg. (2020). "Accurate vision-based manipulation through contact reasoning". In *IEEE International Conference on Robotics and Automation (ICRA).* DOI: 10.1109/ICRA40945.2020.9197409 ©2020 IEEE

For author contributions, Alina Kloss developed the theory and methods, and conceived and planned the experiments. Alina Kloss and Maria Bauza carried out the experiments, where Maria Bauza provided the necessary tools and assistance for running experiments on the real robot (Section 5.6). Alina Kloss wrote the manuscript with feedback from all authors. Joshua B. Tenenbaum and Alberto Rodriguez helped supervise the project and provided feedback. Jeannette Bohg and Jiajun Wu supervised the project and shaped its direction.



(a) Planar Pushing: To push an object to a desired pose (red), a robot has to reason over *where* (green contact points) and *how* (black arrows) to push.

(b) The test objects triangle, butter and hexagon.

5.1 Introduction

In this chapter, we finally address the full sensory-motor skill of pushing an object into a desired position and orientation based on visual observations. In addition to modeling the dynamics of pushing and estimating the state of the system from raw sensory data, which we addressed in the previous chapters, this requires planning pushing actions. As illustrated in Figure 5.1a, pushing is a *contact interaction*, for which we not only have to optimize *how* to move the pusher, but also *where* the pusher should make contact with the object.

In many robotics applications that involve manipulation or legged locomotion, planning such contact interactions is one of the core challenges. One problem is the potentially unlimited number of possible contact points and the resulting high computational cost of optimizing contact locations and actions jointly. In addition, for sensory-motor skills, the planned actions also have to be robust against the uncertainty induced by imperfect perception.

Current approaches for planning contact interactions roughly fall into two categories that align with these challenges. The first focuses on reducing the computational cost of action optimization especially for long sequences and complex dynamics. Such approaches typically make the strong assumption of a fully observable state that requires no perception and of prior knowledge of the robot and environment. Both are rarely fulfilled in practice. The second category focuses on including perception and being robust to the resulting uncertainty. Approaches in this category are typically learning-based and provide a larger level of generalization to variations of the environment, such as unknown objects. However, this often comes at the cost of a lower manipulation accuracy.

Prior work that specifically addresses push-planning can be split according to the same principles. Approaches based on analytical models and a physically meaningful state representation often achieve high accuracy, but assume access to the full state information and known object shapes (Zito *et al.*, 2012; Agboh *et al.*, 2019; Hogan *et al.*, 2018; Dafle *et al.*, 2018; Bauza *et al.*, 2018). Learning-based approaches address the perception problem and make fewer assumptions about the environment, but are less accurate (Li *et al.*, 2018; Finn and Levine, 2017; Ebert *et al.*, 2017, 2018; Agrawal *et al.*, 2016; Hermans *et al.*, 2013; Stüber *et al.*, 2018). Moreover, many of these works do not explicitly reason over where to push, but instead sample and evaluate large numbers of random actions (Zito *et al.*, 2012; Agboh *et al.*, 2019; Li *et al.*, 2018; Finn and Levine, 2017; Ebert *et al.*, 2017, 2018).

Our own approach for planning pushing motions based on RGBD images, in contrast, addresses both of the main challenges in planning contact interactions. First, we improve the efficiency of planning by disentangling contact and motion optimization. This allows us to explicitly reason over contact locations and focus the computations for motion optimization on few promising regions. Second, we combine learning-based perception with an explicit state representation and Bayesian filtering to achieve a high manipulation accuracy and enable generalization to novel objects.

We use a learned model to capture the shape of the object from the visual input and predict a physically meaningful representation of the object state. Bayesian filtering makes the state estimate more robust to imperfect perception and allows us to estimate latent object properties like the center of mass of the object online to further increase the accuracy of the dynamics model.

For optimizing the contact locations, we annotate each point on the object outline with approximate predictions of the object motion it affords. This allows sampling promising candidates of *where* to push the object given a desired target object pose. At these fewer contact candidates, we then optimize *how* to push. For predicting the possible object motion, we compare an approach based on a physical model to a learned one. The learned model makes fewer assumptions and shows advantages in cases that are not well-captured by the physical model. However, the physics-based model is generally more accurate and even generalizes to scenarios that violate some of its assumptions.

In summary, we propose a system for planar pushing that:

- allows for efficient planning by explicitly reasoning about contact-locations,
- improves over model-based approaches by including perception and online estimation of latent object properties and
- achieves a higher accuracy than previous vision-based works by combining learned and analytical elements.

We quantitatively evaluate our method in simulation through ablation studies and comparison to state-of-the-art. We also demonstrate that it transfers to a real robotic platform.

5.2 Related Work

As we have shown in previous chapters, there is a wide range of research on robotic pushing, from modeling the dynamics (Zhou *et al.*, 2018; Lynch, 1999; Bauza and Rodriguez, 2017; Kloss *et al.*, 2020b), to state estimation for pushed objects (Yu and Rodriguez, 2018; Lambert *et al.*, 2019). Here, we focus on works that include planning. For a broader review, we refer to Stüber *et al.* (2020).

5.2.1 Efficient Contact Planning under Full Observability

Hogan *et al.* (2018) present a real-time controller for tracking a desired trajectory with a pushed object under full observability. While the push is locally optimized by a neural network that decides between sticking or sliding contact modes, the global contact location is not. Zito *et al.* (2012) present an approach to push an object into a desired pose that combines a global RRT[1] planner with a local, sampling based planner. Dafle *et al.* (2018) reorient a known object in-hand by pushing it against elements in the workspace. Similar to our work, they use motion-cones to efficiently describe the set of possible object movements at each environmental contact. Ajay *et al.* (2019) use a hybrid approach that augments the predictions from a physical model with learned residuals to push two disks that are already in contact. The method evaluates a predefined set of contacts.

The problem of optimizing contacts is also relevant for legged locomotion. Deits and Tedrake (2014) compute a sequence of footsteps given a set of obstacle-free regions. For efficiency, the dynamics of the robot are not taken into account. To address this issue, Lin *et al.* (2019) take a similar approach to ours: they train an approximate dynamics model over a discrete set of actions that can be used for efficient contact planning while taking robot dynamics into account.

All these approaches assume access to the full state information and known models of the dynamics and object or environment geometry.

5.2.2 Push Planning under Partial Observability

Agrawal *et al.* (2016) train a network to predict the pushing action required to transform one RGB image into another. In contrast, Li *et al.* (2018); Finn and Levine (2017); Ebert *et al.* (2017, 2018) do not directly predict actions but instead learn a dynamics model for predicting the effect of sampled pushes. The input is either a segmentation mask or a full RGB image. Push-Net (Li *et al.*, 2018) samples 1000 actions by pairing pixels inside and outside of the object, while Finn and Levine (2017); Ebert *et al.* (2017, 2018) sample pusher motions that are refined iteratively. Neither work reasons explicitly over contact locations, whereas our approach directly samples promising contact points.

[1]Rapidly-exploring random tree

Push-Net can also estimate the center of mass of objects during interaction using an LSTM. Instead of using an LSTM, we rely on an Extended Kalman filter (EKF) to estimate a physically meaningful state representation during interactions.

Similar to our work, Hermans *et al.* (2013) learn a scoring function from histogram features for finding suitable contact points. Stüber *et al.* (2018) learn a contact model and a contact-conditioned predictive model for pushing with a mobile robot.

While making much fewer assumptions, these vision and learning-based methods generally achieve a lower manipulation accuracy than the model-based methods. Our proposed approach significantly improves on this.

5.3 Planning Pushing Actions

Figure 5.2 shows an overview of our system. At each time step, it receives an RGBD image of the current scene, the last robot action and the target object pose as input. In the perception module, we use a Convolutional Neural Network (CNN) to segment the object and estimate its position and orientation. Since we do not assume prior knowledge of the object shape, we extract a representation based on the segmentation map. Together with the last action, the object pose is input to an EKF that estimates the full object state including latent properties like the center of mass (COM).

The next module approximates the object motions that can be produced by applying a discrete set of pushes at each point on the object silhouette. We refer to the output as *push affordances* of the contact points. While this may be considered an abuse of terminology (Osiurak *et al.*, 2017), we use the term for a clear distinction to other parts in our model. The affordances are continuously updated because they depend on object properties that are estimated by the EKF. The object shape, in contrast, has to be computed only once.

Finally, the state estimate and affordances are the input to the planning module which selects suitable contact points and optimizes the pushing actions beyond the discrete set that is considered in the affordance model.

5.3.1 Planar Pushing

As in previous chapters, we consider the task of quasi-static planar pushing of a single object using a point contact. Quasi-static means that the applied force is enough to move but not to further accelerate the object. We parametrize a pushing action $\mathbf{u}$ by the contact point $\mathbf{r}$ and the pushing motion $\mathbf{v}_u$. Pushes are executed at a constant velocity of 20 mm/s.

The dynamics of pushing depend on object shape, friction and pressure distribution of the object on the surface. The relation between push force and resulting

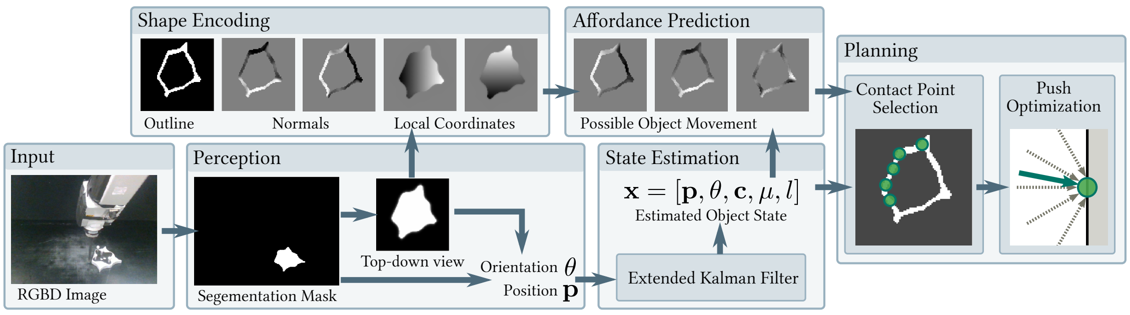


Figure 5.2: Overview: the perception module segments the object and computes its pose. An EKF estimates the full object state including latent properties like the COM $\mathbf{c}$. The object shape is encoded by a silhouette, coordinates and normals in a top-down view. It is input to the affordance prediction module, that approximates the possible object motions at each contact point on the silhouette. The planning module selects contact point candidates using the predicted affordances and optimizes the pushing motion there.

object motion is often modeled using the limit-surface (Howe and Cutkosky, 1996; Goyal *et al.*, 1991). We again use the analytical model by Lynch *et al.* (1992) defined in Chapter 2.2. To recapitulate, it assumes continuous object-surface contact and an uniform pressure distribution for an ellipsoidal approximation of the limit surface parametrized by l. The model predicts object translation and rotation around the COM given l, the push $\mathbf{v}_u$, the normal $\mathbf{n}$ at the contact point and the coefficient of friction between pusher and object m.

Different from the previous chapters, here, we do not assume that the object frame origin $\mathbf{p}_o$ corresponds to the object's COM. Instead, the COM is given by $\mathbf{c}$ relative to $\mathbf{p}_o$. We use $\mathbf{x} = \big(\mathbf{p}_o, \theta, \mathbf{c}, l, m\big)$ as object state, where θ is the orientation of the object.

5.3.2 Perception and State Estimation

We train a CNN to segment the object in each image and compute its world-frame position from segmentation mask and depth values. A bounding box around the segmentation mask is reprojected into a top-down view centered on the object. The orientation of the object is computed relative to the first step by comparing stepwise rotations of the current top-down projection to the initial one. We also evaluated using a neural network for this task (as in Chapter 4.7) but found it to be less reliable.

The output object pose is used as observation for an EKF that estimates the full object state $\mathbf{x}$. The filter uses the analytical model as process model and an identity matrix selecting the object pose from $\mathbf{x}$ as observation model.

5.3.3 Shape Encoding

Our shape encoding needs to be independent of the object shape and position and should contain all the necessary information for predicting the effect of pushes, i.e. the possible contact points together with their surface normals.

We use the object-centric top-down projection of the segmentation mask and depth values to compute the x and y coordinates of each object pixel in this frame. Together with the mask, the coordinates are the input to a CNN that predicts the object outline (i.e. all possible contact points on the object) and the unit 2D surface normals to each point on the outline. Figure 5.2 shows an example of the resulting $100 \times 100 \times 5$ image (outline, coordinates and normals) under Shape Encoding.

5.3.4 Affordance Prediction

For each point on the object outline, the affordance module makes an approximate prediction of the object motions that can be achieved by pushing there. For this, it densely evaluates a predictive model for a fixed set of representative pushing motions. This prediction then informs contact point selection for pushing the object towards the target.

For our experiments, we use a relatively large set of ten representative pushes: we take five directions relative to the respective surface normal ($0°$, $\pm 30°$ and $\pm 60°$) with two push lengths each (1 cm and 5 cm). In general, the expressiveness of the affordance model is a tuning parameter of our method that trades off accuracy against computational speed. Ablation studies showed that including fewer push directions had an overall small effect on our results, as had removing the shorter pushes. Removing the pushes with 5 cm length was most detrimental and increased the average number of steps taken by more than 15 %.

We evaluate two predictive models for obtaining the affordances, the analytical model and a learned model.

Affordances from the Analytical Model

Given the representative pushes, the shape encoding and parameters $\mathbf{c}$, l and m from the state estimation module, we can apply the analytical model at each contact point. We use a one-step prediction, which can be done efficiently on GPU but is potentially not perfectly accurate: During one push, values like the contact point and normal there can change when the the pusher slides along the object or even loses contact completely. Such changes of the model's input values cannot be taken into account by the analytical model as is. This would require dividing the actions into a sequence of much shorter pushes and looking up the new contact point and surface normal at each step for each combination of pushing motion and initial contact point, which quickly becomes computationally challenging.

Affordances from a Learned Model

Alternatively, we train a CNN to predict the object movement given the pushes, **c** and the shape encoding. Different from the analytical model, the learned model does not require the parameters l and μ. In addition, it can take the local shape around the contact point into account to predict effects of pusher sliding like loss of contact. For this, the model uses a 3-layer CNN with max-pooling to process the object outline. The resulting local shape features, the pushes and the shape encoding serve as input for predicting the object motion using a second 3-layer CNN without pooling.

5.3.5 Planning

We use a greedy planner to find the contact point and straight pushing motion that brings the object closest to the desired goal pose at each step. We found this approach to be sufficient in our scenario where no obstacles are present. For planning around obstacles, our model could be combined with a global planner for object poses, as proposed for example by Zito *et al.* (2012); Dafle *et al.* (2018).

Instead of jointly optimizing contact point and pushing motion, we divide the problem into two subtasks. We first propose a set of contact points and then separately optimize the pushing motions at each candidate point before selecting the most promising combination.

Contact Point Proposal

Our method uses the affordances to score each point on the object outline by how close pushing there could bring the object to the target pose:

$$s(\mathbf{r}_i) = \min_{\mathbf{v}_u \in \mathcal{U}_i} \| \mathbf{v}_d - \hat{\mathbf{v}}_o(\mathbf{u}, r_i) \|_2 + \lambda |\omega_d - \hat{\omega}(\mathbf{v}_u, r_i)| \tag{5.1}$$

Here, $\mathbf{v}_d$ and ω_d are the desired object translation and rotation, $\mathbf{v}_u \in \mathcal{U}_i$ are the representative pushing motions at contact point $\mathbf{r}_i$, and $\hat{\mathbf{v}}_o(\mathbf{v}_u, r_i)$ and $\hat{\omega}(\mathbf{v}_u, r_i)$ their predicted object motion from the affordance model. We weight the rotation error (in degree) stronger ($\lambda = 2$) for a good trade-off between translation and rotation. A softmax function turns the scores, $s(\mathbf{r}_i)$, into a probability distribution that is used to sample k candidate points.

We found that sampling the contact points instead of choosing the k best points deterministically improved the robustness of our method against imperfect object outlines and pose estimates. In cases where the planned push fails to make contact with the object, it prevents the model from trying to execute this same action over and over again.

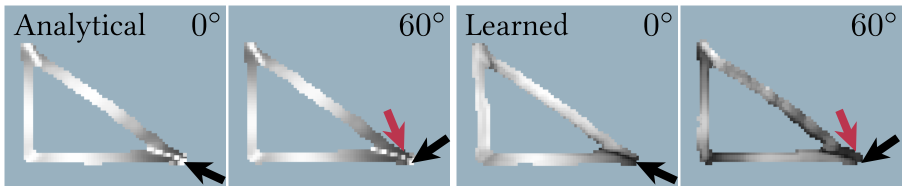


Figure 5.3: Predicted translation magnitude from the analytical and learned affordance model (brighter is higher) for pushes along the normal and at a 60° angle. In contrast to the analytical model, the learned model predicts low magnitudes for pushes that are unlikely to properly hit the object (black arrows) or pushes that are likely to slide off the object (red arrows).

Push Motion Optimization

The discrete set of actions evaluated for the affordance model will in general not contain the optimal pushing motion at each point. We thus optimize push direction and length at each candidate contact point by interpolating between five base pushes $\mathcal{U}_b$ with different directions as follows:

1. For each $\tilde{\mathbf{v}}_u \in \mathcal{U}_b$, roll out the analytical model over the maximum allowed push length of 5 cm in substeps of 0.5 cm.

2. At each step, score the predicted object movement so far using Equation 5.1.

3. Truncate each $\tilde{\mathbf{v}}_u$ at its best-scoring step or to the minimum required push length of 1 cm. This gives $\bar{\mathbf{v}}_u$ with optimal scores $\bar{s}$.

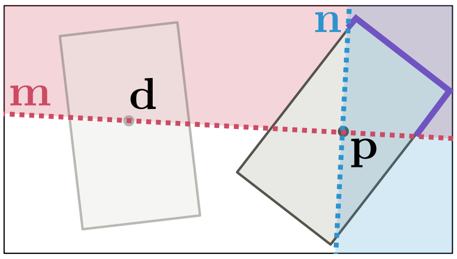
4. Find the optimal push direction $\mathbf{d}$ by interpolating between the $\bar{\mathbf{v}}_u$ with the best $\bar{s}$ and the two $\bar{\mathbf{v}}_u$ with neighboring directions.

5. Optimize the length of $\mathbf{d}$ as in steps (1) - (3) to find the optimal push ${\mathbf{v}^*}_u$ with score s^*

The planner finally returns the contact point and action with the highest score s^*. As explained before, rolling out the analytical model over shorter substeps is more accurate but also more computationally expensive than predicting the outcome of the full 5 cm push in one step.

In our specific case, the push affordances already contain predictions for the same five push directions that we also use for $\mathcal{U}_b$. This allows us to use the affordance predictions in step (1) of the push optimization. Instead of steps (2,3), the push length is then optimized by linearly rescaling the push and predicted object motion to match the desired motion. We compare this approach to our regular method in Experiment 5.5.4.

Figure 5.4: Heuristic for contact point selection (*geo*): line $\mathbf{m}$ connects the current $\mathbf{p}_o$ and desired object position $\mathbf{d}$, $\mathbf{n}$ is its normal. Points on the blue side of $\mathbf{n}$ afford pushing towards $\mathbf{d}$, points to the right of $\mathbf{m}$ (red area) are proposed for counter-clockwise rotation. For rotations below 2°, candidates need to lie within 2 cm of $\mathbf{m}$. The intersection of both areas (purple) defines the set of possible contact points for sampling.



5.4 Training

For training the perception, shape encoding and learned affordance model, we rely mostly on simulated data generated in pybullet (Coumans and Bai, 2016). Each datapoint contains an RGBD image of an object on a surface, its ground truth position, segmentation mask and outline with normals. We annotate 20 random contact points per object with the object movement in response to the ten representative pushing actions defined in Section 5.3.4.

Properties like object mass, center of mass and friction coefficients are sampled randomly. We generate more than 15k examples using 21 objects, of which we hold out three for testing (shown in Figure 5.1b, which also shows the real-world setup after which we modeled the simulation).

While the segmentation and shape encoding network are finetuned on real data from the Omnipush dataset (Bauza *et al.*, 2019), the learned affordance model is only trained on simulated data. We train the models in Tensorflow (Abadi *et al.*, 2015) using Adam (Kingma and Ba, 2015).

5.5 Simulation Experiments

5.5.1 Setup

We evaluate three different tasks: translating the object by 20 cm without changing the orientation (*translation*), rotating the object by 0.5 rad (28.6°) without changing the position (*rotation*) and translating for 10 cm plus rotating by 0.35 rad (20°) (*mixed*). A trial counts as successful if it brings the object within less than 0.75 cm of the desired position and 5° of the desired orientation in at most 30 steps. We evaluate the percentage of successful trials and the average number of steps until the goal pose is reached.

For each task, object and method, we perform 60 trials. At the beginning of each, the object is placed at the center of the workspace. We vary its initial orientation in 20 steps from 0 to 360° and perform three runs with each orientation.

5.5.2 Affordance Prediction

We first qualitatively compare the learned and the analytical affordance model to see if there are any major differences between their predictions. Overall, both models predict similar directions of movement, with the analytical model predicting more pronounced rotation. A potential advantage of using a learned model becomes apparent when we compare the magnitude of the predicted translational movement, which is visualized in Figure 5.3. The analytical model predicts strong translations for pushes at the sharp corners of the triangle, whereas the learned model predicts comparatively low magnitudes there. The same effect is visible for angled pushes that cause the pusher to slide towards corners.

As discussed before, the analytical affordance model cannot predict loss of contact due to pusher sliding or because the planned motion does not properly hit the target contact point. Such events are more likely when pushing at sharp corners or with high angles. The learned model takes the local object shape around the contact point into account and is therefore able to identify such cases and predict a lower movement magnitude.

5.5.3 Contact Point Selection

In this experiment, we test our hypothesis that explicitly reasoning about the contact locations makes planning more efficient as compared to sampling pushes that collide with the object in random locations. For this, we vary the number of sampled contact points and compare our approach (that uses the affordances to propose promising contact points) to two baselines that select the contact points more randomly.

The simplest baseline samples contact points uniformly from all points on the object outline (*rdn*). A more informed approach (*geo*) uses a geometric heuristic explained in Figure 5.4. Based on the desired motion, it defines a quadrant of the object from which the contact points are sampled. In contrast to *rdn*, *geo* better avoids sampling contact points at which the object can only be pushed away from the goal. It however ignores the exact shape of the object and can thus still propose unsuitable contact locations especially for non-convex objects.

To minimize the influence of other components of our system on the results, we do not use filtering for state estimation in this experiment but assume access to perfect state information at every step.

Results

We first compare the success rates in Figure 5.5 (left). By sampling from the affordance model (learned *lrn* or analytical *ana*), our method can already achieve a success rate close to 100% with only one contact point. The geometric heuristic

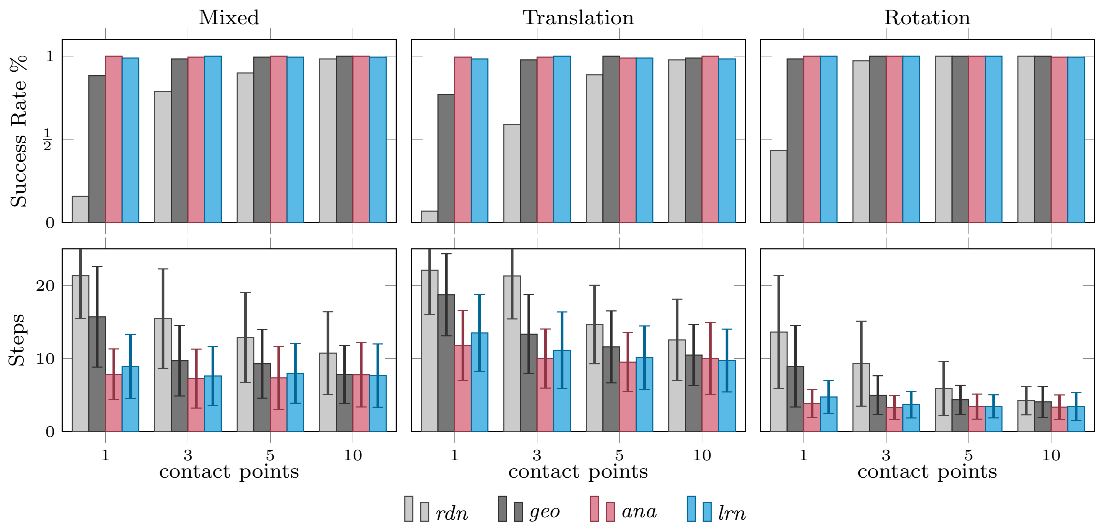


Figure 5.5: Pushing performance over number of sampled contact points. We compare different sampling methods of contact locations: randomly (*rdn*) or according to a geometric heuristic (*geo*), the analytical (*ana*) or the learned (*lrn*) affordances. We analyze performance for three different tasks: Pure object translation, pure object rotation and a mixed motion. Results are averaged over three test objects (See Figure 5.1b). With our proposed affordance models (*ana* and *lrn*), one contact point sample is already sufficient to achieve a high success rate. *Ana* and *lrn* also require the lowest number of steps to get to a target object pose.

also performs well and often reaches 100% with as few as three sampled contact points. We only see a big impact of the number of contact points when sampling randomly. On the tasks that involve translation, *rdn* only reaches the success rate of the other methods with ten contact points. The number of sampled points is generally more important for translating than for rotating the object.

Figure 5.5 (right) shows the number of steps each method took until the goal pose was reached. Even in successful runs, *rdn* needs significantly more steps than the other methods. *Geo* again performs better, although still worse than our proposed method using the affordance predictions. Both *lrn* and *ana* work very well with only one contact point and their performance mostly saturates at three sampled candidates. There is no significant difference between using the analytical or the learned model for obtaining the affordances.

To summarize, using an affordance model to sample contact points makes planning more efficient by reducing the number of contact points that have to be evaluated per step and the number of steps taken until the goal is reached.

5.5.4 Pushing Motion Optimization

In this experiment, we test if the predicted affordances are accurate enough to also be used for optimizing the pushing actions (see Section 5.3.5). This is especially

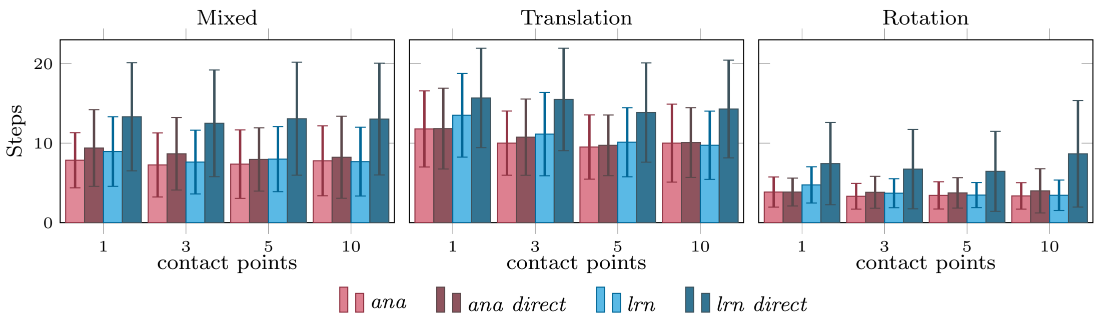


Figure 5.6: Steps taken vs. sampled contact points when rolling out the analytical model for optimizing the push motions (*ana, lrn*) or directly using the affordances (*ana-direct, lrn-direct*). While *ana, lrn* and *ana-direct* perform similar, *lrn-direct* is less accurate and thus needs more steps to succeed.

interesting for evaluating the quality of the learned model. We call the variants that use the predictions from the affordance model directly *ana direct* and *lrn direct* respectively.

Results

As shown in Figure 5.6, using the analytical affordance predictions for action optimization does not significantly increase the number of steps taken as compared to *ana*. This is not surprising since the only difference between both approaches is that *ana* optimizes the actions by rolling out the analytical model over smaller substeps while *ana-direct* relies on one-step predictions.

When using the learned affordances for push optimization, the number of steps taken, however, increases by up to four and the success rate drops by up to 10% compared to *lrn*. This implies that while being sufficient for selecting contact points, the learned model is not as accurate as the analytical model for predicting the outcome of a push. The negative effect of using the learned model can also not be compensated by evaluating more contact points, which emphasizes the value of an accurate predictive model for optimizing the pushes.

5.5.5 Full System

Now we evaluate the accuracy of our full system including the state estimation module, with three contact points sampled per step. In all experiments, $\mathbf{c}$ is initialized to zero and l and m to reasonable estimates.

We first test on objects whose center of mass coincides with the geometric center to evaluate the perception module and how well planning works with imperfect pose information. In the second experiment, we verify the benefit of estimating latent properties of the object online on the example of the COM. For this, we sample

Table 5.1: Comparison of performance and end pose error when using ground truth (gt) pose information vs. using pose estimates from the Extended Kalamn Filter (filter). Results are obtained using the analytical affordance model on the mixed task and are averaged over all test objects.

method	suc	steps	error tr [mm]	error rot [°]
gt	1.0	7.3±3.6	5.0±1.8	1.8±1.3
filter	1.0	7.1±3.1	8.7±4.2	3.0±2.2

Table 5.2: Performance of our approach with (+ COM) and without estimating the center of mass on the mixed task.

	butter		tri		hex	
method	success	steps	success	steps	success	steps
lrn	0.75	13.9±7.6	0.75	12.96±6.6	0.92	13.2±6.7
lrn + COM	0.91	10.6±6.1	0.93	11.86±6.2	0.88	11.4±6.5
ana	0.82	12.1±6.2	0.9	13.3±6.2	0.85	10.5±5.1
ana + COM	0.9	11.4±6.6	0.98	10.9±6.6	0.9	10.7±6.3

the COM uniformly inside the objects. We also compare our approach to Push-Net Li *et al.* (2018) under this condition. Push-Net uses top-down segmentation maps of the current and desired pose as input to evaluate a large number of randomly sampled actions. A local planner generates sub-goals by interpolating between the current and the goal pose with a fixed step size, we use 5 cm and 10°.

Results

COM at Geometric Center In the previous experiments, we used the ground truth object pose information. Here, we compare those results to doing pose estimation by filtering. As shown in Table 5.1, we find that using the filter has no large impact on the success rate of our method or the number of steps taken. However, it increases the (true) average end pose error from 5.0±1.8 mm, 1.8±1.3° to 8.7±4.2 mm and 3.0±2.2°. This is expected as we use the *estimated* object pose to determine if the goal is reached. Therefore, the real pose error can be higher than the (7.5 mm, 5°) margin of the goal region.

Randomly Sampled COM In this experiment, we want to verify that estimating the COM online is possible and beneficial for our approach. For this, we compare the performance of our method with and without estimating the COM on objects with a randomly sampled COM.

On the triangle and butter shape, the average estimation error for $\mathbf{c}$ is 17.7±8.5 mm, on the hexagon it is around 1 cm higher. The average distance between the true COM $\mathbf{c}$ and the object frame position $\mathbf{p}_o$, which corresponds to the error when the

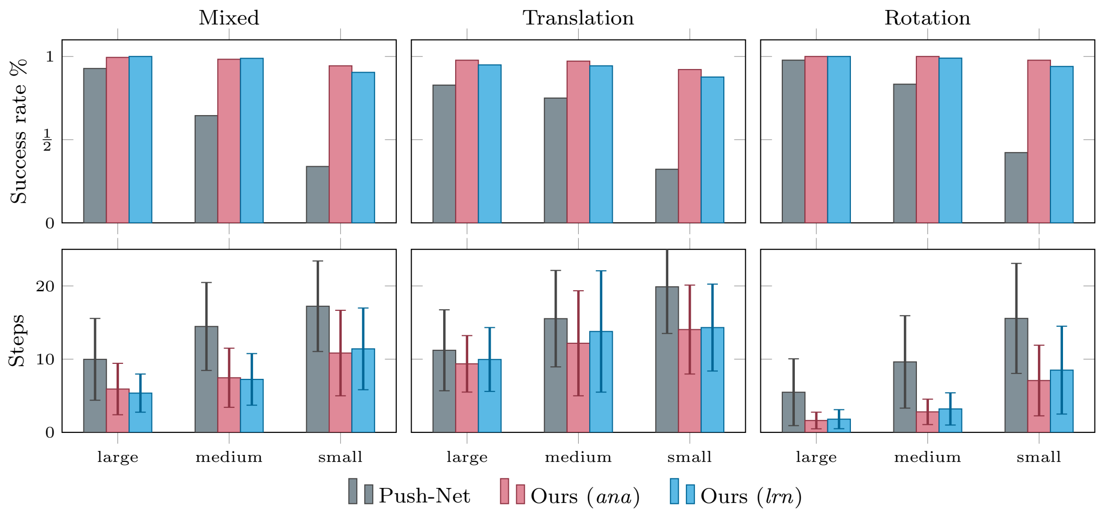


Figure 5.7: Performance of our method and Push-Net on objects with random COM (averaged over objects). We evaluate three goal region sizes, *small* (0.75 cm 5°), *medium* (2.5 cm 7.5°) and *large* (5 cm 10°). Our method has a higher success rate on smaller goal regions and needs fewer steps to reach the goal.

COM is not estimated, is 37.4±12.3 mm. This shows that filtering can significantly improve the estimated COM position. More accurate estimates could potentially be obtained if the pushing actions were specifically optimized to determine the COM location.

Table 5.2 shows that despite not being extremely accurate, estimating **c** increases the success rate of our method by up to 18% on the triangle and 16% on the butter object. The difference is more pronounced when we use the learned instead of the analytical affordance model. Estimating the COM also decreases the number of steps taken on the butter and triangle object. The hexagon shape is the only object for which we do not see much gain from estimating the COM. This is likely linked to the higher estimation error for the COM on this more compact shape.

Comparison to Push-Net We also compare our approach to Push-Net, which uses an LSTM to estimate the COM. We evaluate three sizes of the goal region, from the (0.75 cm, 5°) we used in all previous experiments to the (5 cm, 10°) used in the original Push-Net paper, plus a medium size of (2.5 cm, 7.5°). Results are shown in Figure 5.7. With the largest goal region, Push-Net performs competitive to our approach and it still reaches a good success rate for the medium sized region. On the smallest size however, our approach outperforms Push-Net by a large margin, despite evaluating much fewer actions. Our method also consistently requires fewer steps to reach each level of accuracy. Qualitatively, Push-Net does well for translating the object, but has trouble controlling its orientation precisely.

5.6 Real-Robot Experiments

Finally, we evaluate our approach on the real robotic system introduced in Chapter 2.2 and shown in Figure 5.1a. This is especially interesting with respect to our predictive models: We know that the analytical model makes assumptions that are frequently violated in the real world, while the learned model was trained purely in simulation and might not transfer well to the real world. A video of some trials with visualizations of the predicted affordances can be found at `https://www.youtube.com/watch?v=YLnXLHWTA60`.

Setup

To evaluate our affordance models, we first compare *lrn*, *ana*, *lrn direct* and *ana direct* given ground truth state information on the butter object from the MIT Push Dataset (Yu *et al.*, 2016) that we also used in the simulation experiments (see Figure 5.1b). We also evaluate *rdn* and *geo* again under this conditions.

Then we test the full system with the analytical affordance model on butter, triangle and a new object from the Omnipush Dataset (Bauza *et al.*, 2019) (shown in Figure 5.1a). This object has added weights that alter its pressure distribution and center of mass and thus violates the uniform pressure distribution assumption of the analytical model.

For both experiments, we use a new mixed task with 12 cm translation and 46° rotation. The relatively short translation distance is necessary to ensure that the object does not leave the workspace of the robot during interaction, independent of the direction of pushing. We set the maximum number of steps that the methods can take to 20 and sample three contact points at each step. Every experiment is repeated 15 times.

Results

When comparing *rdn*, *geo*, *lrn* and *ana* as well as *ana direct* and *lrn direct* given full state information on the real butter object, the results (shown in Figure 5.8) are very similar to the results we obtained in simulation. Using the analytic push optimization step, both *lrn* and *ana* succeed in all trials and need 5.5±2.6 and 5.2±2.1 steps respectively. For *lrn direct* and *ana direct*, the average number of steps increases to 8.2±4.3 and 6.3±1.9 respectively, while the success rate stays at 100% in both cases. The two baselines *rdn* and *geo* in contrast do not succeed in every trial and also need a higher number of steps to reach the goal in successful runs. We can thus conclude that both affordance models transfer well between simulation and the real system.

The results with filtering are shown in Table 5.3. We still achieve a high success rate on all three test objects. In comparison to the previous experiment with ground

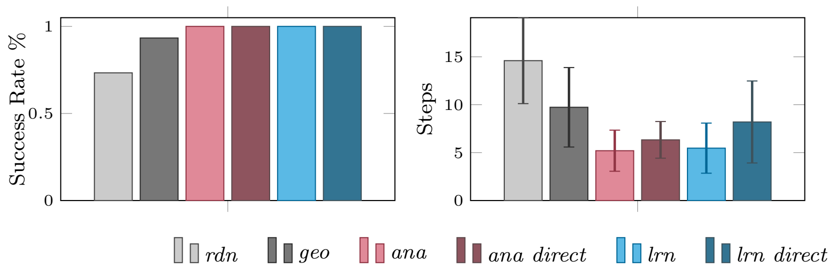


Figure 5.8: Success rate and steps taken for *rdn*, *geo*, *lrn* and *ana* as well as *ana direct* and *lrn direct* on the real robot with ground truth state information. As in simulation, our proposed method reaches a higher success rate and needs fewer steps to reach the goal than the baselines *rdn* and *geo*. Using the analytical model to optimize the pushing motions in *lrn* and *ana* further reduces the number of steps as compared to using the predictions from the affordance models in *ana direct* and *lrn direct*.

Table 5.3: Performance and end pose error of our full approach using the analytical affordance model and filtering on the real robot.

object	success	steps	error tr [mm]	error rot [°]
butter	1	7.4±3.2	8.0±3.2	7.6±3.7
tri	0.93	6.1±3.8	9.2±4.2	5.8±3.7
omni	0.88	8.0±3.6	8.8±4.5	7.2±5.5

truth state information, the number of steps taken for the butter object increased from 5.2±2.1 to 7.4±3.2.

The biggest challenge in this experiment turned out to be estimating the orientation of the object correctly. Especially for the relatively compact Omnipush object, the estimation from the visual information sometimes failed drastically, resulting in the lowest success rate of 88% and the highest number of steps taken.

Averaged over all test objects, we still reach a success rate of 94% and an end pose error of 8.6±4.0 mm and 6.9±4.3°. This confirms that our approach generalizes to real-world conditions, even when they potentially violate the assumptions of the analytical model.

5.7 Conclusion

In this chapter, we presented an approach that addresses the full sensory-motor skill of planar pushing. Besides from the uncertainty about the system state that arises from estimating the state from raw sensory data, this task comes with the additional challenge of optimizing not only the pushing motions but also the contact locations for pushing efficiently.

We proposed to address this problem by decomposing the action optimization into first proposing promising contact point locations using an affordance model and then optimizing the pushing motion at each of the contact point candidates. Our experiments showed that explicitly reasoning over contact locations allows our method to evaluate fewer actions and at the same time plan more optimal pushing motions than when sampling random contact locations.

In comparison to the purely learned Push-Net, our method reaches a much higher manipulation accuracy. To do so, we combined learned components for perception with structure in the form of a physically meaningful state representation, the analytical model of the pushing dynamics, a Bayesian filtering algorithm for state estimation, and the aforementioned decomposition of the planning problem into contact and motion optimization.

A particularly important factor for the accuracy of our approach turned out to be using the analytical model for optimizing the pushing actions. However, for predicting the push affordances to select promising contact locations, the learned model was also sufficient. In contrast to the analytical model, the learned model is able to take the object shape into account, which can be advantageous for identifying unstable contact locations where the pusher is likely to loose contact quickly. We thus find that learning is not only well suited for perception but also for providing "intuitive physics" models that can quickly narrow down large search spaces to few promising candidates that are then optimized using more accurate but also costly analytical models.

Some limitations of our approach are the relatively simple scenes we considered and that our method assumes a mostly unoccluded object outline. Dealing with strong occlusion is an interesting problem for future work, as is working with more complex object shapes. Preliminary results in simulation suggest that our approach is robust to objects with a non-planar surface contact, but real-world experiments are necessary to confirm these results.

Chapter 6

Conclusions

6.1 Summary

In this thesis, we have investigated different ways of combining "traditional" model-based robotics with data-driven learning techniques in the context of sensory-motor manipulation skills and in particular robotic planar pushing.

The three projects we presented in Chapters 3-5 looked at different aspects of sensory-motor skills and different forms of structure that we can use to solve them. We went from the level of individual models for perception and prediction, over algorithms used for state and uncertainty estimation to a full system that solves the planar pushing task in an efficient and accurate manner. In the following, we briefly summarize the contributions and results of each project.

6.1.1 Models for Perception and Prediction

In Chapter 3, we considered the problem of predicting the effect of physical interaction from raw sensory data. We decomposed the task into a perception model for compressing the sensory observations into a description of the current state of the system and a dynamics model that predicts the outcome of the interaction given this state representation and the action.

We proposed a hybrid architecture that addresses this task by training a DNN for perception end-to-end through an analytical model of the dynamics. To further improve the predictions, a learned error-correction or residual term could be added to the analytical predictions. Our experiments compared this hybrid approach to a purely learned model in terms of *accuracy*, *data-efficiency* and the ability to *generalize* to situations not seen during training. While the pure neural network achieved the best accuracy when training and test distribution are identical, we observed two main advantages of the hybrid architectures: Compared to the pure neural network, they required significantly fewer training examples and generalized better to novel interactions and object shapes.

By using a physics-based analytical model with its fixed state representation to predict the dynamics of the system, we limit the ability of the full model to overfit to the training data. In addition, the analytical model provides multiplication operations that relate the velocity of the input actions to the resulting object movement and thus facilitate extrapolation to faster pushes. From the perspective of function approximation and model capacity that we discussed in Chapter 2.1, the analytical model thus both restricts and at the same time extends the family of functions that the full system can learn.

End-to-end training is an important aspect for combining learned and analytical components in a robotic system. Our experiments showed that training the perception part through the analytical model allowed it to compensate for smaller errors of the analytical model by adjusting the predicted input values. In addition, end-to-end training can reduce the labeling effort for supervised training. It allows learning the intermediate state representation instead of specifying a fixed representation. While physically meaningful state representations are generally desirable to ensure an interpretable solution and prevent overfitting, the purely learned model in our experiments could reach a higher accuracy (on the training domain) when it was free to learn its own representation.

6.1.2 State Estimation and Uncertainty

For the second project (Chapter 4), we investigated differentiable Bayesian filtering algorithms (DF) for tracking the state of a system over multiple time steps and providing estimates of the uncertainty about the predicted state. The filtering algorithms formalize prior knowledge about how to solve the state estimation task and structure the learning problem into learning models of the sensor and the dynamics as well as their respective noise models.

Our main goal was not only to explore the advantages of DFs over their purely learned or purely analytical counterparts, but also to provide a comprehensive overview of existing methods and the factors that are important for making them work. While the DFs proved to be great tools for state estimation, especially the differentiable Particle Filter comes with many different and relevant implementation choices. We hope that our work can provide useful guidance to other researchers interested in using DFs.

Our experiments confirmed the findings from prior work that the algorithmic structure of the DFs greatly facilitates training as compared to using less structured LSTM models. In contrast to their non-differentiable counterparts, DFs enable learning not only the dynamics and sensor models, but also their associated noise models. We could show that training these noise models end-to-end through the filters is important for the performance of DFs and that learning more complex, heteroscedastic noise models can give them a big advantage over filters that use hand-tuned noise models.

While we found no large difference between the performance of DFs with different underlying filtering algorithms, the Particle Filter with learned observation update had the lowest tracking errors in many cases. In contrast to the other DFs, this variant relaxes some of the assumptions made in Bayesian filtering algorithms (e.g. that the observation noise is Gaussian). This shows that in cases where the prior structure does not reflect the true process well, deep learning is a promising tool for improving the traditional methods.

6.1.3 Planning Contact Interactions

In our last project (Chapter 5), we finally studied the complete task of using planar pushing to move an object into a desired pose based on visual input. In addition to the sub-problems of state estimation from raw sensory data and predicting the outcome of pushing actions, this task comes with the additional challenge of optimizing not only the pushing motions but also the contact points for pushing.

We proposed to address this problem by decomposing the optimization task into a module that proposes promising contact point locations using what we called an *affordance* model and a module that optimizes the pushing motion at each of the contact point candidates. Our experiments showed that this two-step approach not only reduced the number of contact points and actions that had to be evaluated at each step but also enabled planning more optimal pushing actions than when sampling random contact locations.

In comparison to a recent, purely learned approach, our method reaches a much higher accuracy by relying on a physically meaningful state representation, a Bayesian Filter for state estimation and an analytical model of the pushing dynamics. An important factor for the accuracy of our approach turned out to be using the analytical model for optimizing the pushing actions. However, for selecting promising contact points, a learned affordance model was also sufficient and even advantageous in some cases. Learning is thus not only well suited for perception but also for providing “intuitive physics” models that can quickly narrow down large search spaces to few promising candidates that are then optimized using more accurate but also costly analytical models.

6.2 Lessons Learned

In the following, we briefly summarize some of the main lessons we learned about applying deep learning in robotics.

Don’t reinvent the wheel! While nowadays, applying a DNN seems to be our first impulse, there already exists a wealth of successful models and algorithms for many tasks. Finding and understanding them might require a lot of work, but it is also rewarding: Even when the “traditional” methods prove not be sufficient for solving a problem, studying them still helps to get a better understanding of the problem and its challenges.

Choose the right function approximator! While DNNs can in theory model any function, in practice we can often obtain better results if we tailor the function approximator to the problem we address. In the context of deep learning, a trivial example for this is using CNNs for processing image data or RNNs for modelling time series. The more we know about the problem, the more we can restrict the class of functions that can be learned without impeding the model's ability to fit the training data. This is exactly what we do when we combine structure and learning. Functions learned in this way often generalize better to unseen data and need fewer training examples than functions learned with a more general approximator.

Residual models often work well! An easy way to combine structure and learning is to learn a residual (error-correction) term on top of the output of an analytical model or algorithm. As we could show in Chapter 3, residual models can retain much of the desirable generalization and data efficiency properties of the underlying analytical solution while at the same time improving its accuracy.

Know your data! Our experiments have shown that differences between the distributions of training and test data can be problematic for learning methods. To leverage the full potential of deep learning, it is thus important to make sure that the training data covers the whole problem domain. This does not necessarily mean that the training data needs to contain examples of every possible input - if we think about image processing, it is clear that DNNs can *interpolate* surprisingly well in some cases. However, *extrapolation* to new value ranges is still problematic.

In addition to analysing the training data in the context of the problem domain, it can be useful to amplify the frequency of rare but important events in the training data. For example, for predicting the outcome of a push, it is clearly important to detect when the pusher is not in contact with the object. However, since this happens rarely in the MIT Push dataset, our perception models in Chapter 3 would learn to always predict contact until we augmented the training data with more non-contact examples.

Representations matter! If a learned model does not work well, one problem could be the representation of its input data. In Chapter 3, we have seen large differences in performance between neural networks that used a physically meaningful state representation or a learned encoding of an image. Supplying sines and cosines instead of raw orientations proofed to be helpful for learning dynamics in Chapter 4. Similarly to the input representation, some output representations may also be easier to predict for a neural network than others.

6.3 Directions for Future Work

We are still a long way away from the robotic butlers and boardgame mates that we envisioned in the introduction of this thesis and there are thus many exciting directions for future research. While our work showed that the traditional approach to robotics with its explicit and transparent structure, algorithms and models is nowhere close to being obsolete, we still expect that the next big advances in robotics will result from approaches that also use deep learning. In a recent study, Sünderhauf *et al.* (2018) give a good overview of the special aspects and challenges when applying learning in robotic systems. Here, we want to highlight some problems that our work left open and that we belief will be important for taking todays robots a step closer towards their fictional role models.

Robotic Manipulation With planar pushing, we mainly studied a vision-based object manipulation task. One constant challenge for such problems is dealing with novel object shapes and other physical object properties. In our projects, we side-stepped much of this challenge by using flat objects, often with uniform weight distribution. This not only eliminates self-occlusions by the object, but also allowed us to effectively summarize the object shape with its 2D outline and the center of mass.

Real-world objects, however, have much more interesting shapes and properties. They can be composed of different materials, be articulated or even be deformable. Finding representations of object shapes and related properties that are compact, well suited for prediction and can at the same time be easily extracted from partial views will be an important step for advancing robotic manipulation skills. Wu *et al.* (2018); Rempe *et al.* (2019); Park *et al.* (2019) are examples of recent work that make progress in this direction.

A second aspect that we did not address are interactions between multiple objects or between objects and fixed structures in the environment. Efficiently predicting which objects will interact and how they will affect each other is still an active area of research (e.g. Watters *et al.* (2017); Janner *et al.* (2019)).

Finally, in this thesis, we only used visual and depth information as sensory inputs. However, we know that other sensory modalities, especially tactile information, play a large role when humans solve manipulation tasks. Challenges here range from fusing information from different sensory modalities (as was studied for example in Lee *et al.* (2020); Izatt *et al.* (2017)) to designing better tactile sensors (such as GelSight (Yuan *et al.*, 2017)).

Embracing Imperfection One important rule-of-thumb for roboticists is that no model - be it learned or analytical - is ever perfect. This means that in addition to trying to make our models better and better, a second avenue for improving our robots is to account for modeling errors in our algorithms.

A key ingredient for this is constant monitoring of feedback and the ability to detect when a model fails. While the differentiable filters we discussed in Chapter 4 present a promising way for learning about uncertainty, there is still much room for improvements. In the context of deep learning, this means that we need better methods for estimating how (un)certain a trained model is about its predictions (as studied e.g. by Gal (2016)) and especially for detecting input data that lies outside of the model's training domain (as proposed by e.g. Limoyo *et al.* (2020)).

Apart from preventing possibly catastrophic failures of the robotic system, the ability to detect mismatches between model predictions and the observed data also presents an opportunity for improving the models. Online and self-supervised learning techniques can thus be expected to play a large role for the future of robotics. A particularly interesting challenge here is to determine where errors originated. For example, if a learned dynamics model does not make good predictions in a novel scenario, this could be caused by the model itself, but also, for example, by a failure of a perception module that lead to wrong input values. In the latter case, adapting the dynamics model to work with faulty input values would clearly be undesirable.

Learning from Deep Learning Another question that is still largely open is what successful DNNs can teach us about the problems they address and about previous solutions to those problems. For example, if we can figure out where and how a DNN's prediction deviates from an analytical solution, this might help to identify and resolve weaknesses of existing models.

One important aspect for distilling knowledge from DNNs is to understand their intermediate representations. While there exist methods for visualizing the activations of hidden units or analyzing to which patterns in the input a neuron reacts, they are still too cumbersome to use.

6.4 Personal Reflections on Deep Learning for Robotics

When I started my PhD in 2015, Deep Learning was still a relatively young field and researchers were just starting to apply DNNs to robotics problems. Since then, the number of published papers about learning in robotics has rapidly risen and when ICRA and IROS finally created the keyword "Deep Learning in Robotics and Automation" in 2018, it immediately became the most used keyword of both conferences. In addition, a learning-themed robotics conference (Conference on Robot Learning (CoRL)) has been founded and more and more robotic papers appear in machine learning conferences like NeurIPS, ICLR or ICML.

By now, Deep Learning has thus become a fixed part of robotics research and finding any group that does not leverage learning for any project would be a difficult task. However, in conversations and reviews alike, one can still get the impression that large parts of the research community meet this success story with a lot of skepticism. One expression of this is the growing debate about "structure vs learning" or "model-based vs. data-driven robotics". Over the past few years, there have been numerous workshops and discussion panels on this topic and on methods for combining the two approaches at all major conferences. When we ourselves organized a workshop on "Combining Learning and Reasoning" at R:SS 2018 (Karkus *et al.*, 2018b), the general feedback we received matched my overall impression that while the robotics community has accepted that deep learning is the most promising avenue for advancing the field, many researchers do not feel comfortable with taking the "fully learned" approach.

The missing generalizability of fully learned systems and the resulting safety issues that we also explored in this thesis surely are one reason for this sentiment. Another aspect seems to be a general feeling of resentment towards declaring decades of past research in analytical models and methods obsolete. In some sense, accepting that learning approaches could lead to better results than all the models and algorithms the community has created would mean to admit defeat.

A third, related aspect only came to my attention more recently: During a panel about "the Roles of Physics-Based Models and Data-Driven Learning in Robotics" (Hsu *et al.*, 2020), Aude Billard voiced her belief that as researchers, our task is to generate new knowledge and understanding about our world. But how can we learn new things if we just apply black-box function approximators to solve all robotic problems? From this perspective, the skepticism of the robotics community towards learning could also be read as a reluctance to accept solutions that work well in practice but for which we cannot determine exactly *how* or *why* they work.

Following these considerations, combining learning and structure seems to be a logical solution for us as a research community to profit from the huge potential of deep learning without sacrificing the achievements of decades of prior research or our mission to create not only solutions but also understanding. Given the current rapid progress in deep learning techniques as well as tensor processing hardware, it might eventually turn out that putting structure into learning approaches is not necessary anymore for system performance and robustness. But even then, structure will still be a necessary tool for us to create transparent solutions that can be broken down into smaller parts to be analyzed and understood individually.

As a field, we might never get rid of the black-boxes of learning again, but we can still chose how big we make these boxes and where we place them in our overall architectures. And hopefully, with time, we will also be getting better at peering into the boxes and create new understanding from them.

Abbreviations

AI	artificial intelligence
CNN	convolutional neural network
COM	center of mass
DF	differentiable filter
DNN	deep neural network
EKF	Extended Kalman filter
GP	Gaussian process
GMM	Gaussian mixture model
LSTM	Long Short-Term Memory, a form of recurrent neural network (Hochreiter and Schmidhuber, 1997)
MCUKF	Monte-Carlo Unscented Kalman filter
MPC	model predictive control
NLL	negative log likelihood
PF	Particle filter
RMSE	root mean squared error
RNN	recurrent neural network
UKF	Unscented Kalman filter

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Agboh, W., Ruprecht, D., and Dogar, M. (2019). Combining coarse and fine physics for manipulation using parallel-in-time integration. In *Springer Tracts in Advanced Robotics (STAR)*. Springer.

Agrawal, P., Nair, A. V., Abbeel, P., Malik, J., and Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in neural information processing systems*, pages 5074–5082.

Ajay, A., Wu, J., Fazeli, N., Bauza, M., Kaelbling, L. P., Tenenbaum, J. B., and Rodriguez, A. (2018). Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3066–3073.

Ajay, A., Bauza, M., Wu, J., Fazeli, N., Tenenbaum, J. B., Rodriguez, A., and Kaelbling, L. P. (2019). Combining physical simulators and object-based networks for control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. (2018). Differentiable mpc for end-to-end planning and control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8289–8300. Curran Associates, Inc.

Archer, E., Park, I. M., Buesing, L., Cunningham, J., and Paninski, L. (2015). Black box variational inference for state space models. *arXiv preprint arXiv:1511.07367*.

Bauza, M. and Rodriguez, A. (2017). A probabilistic data-driven model for planar pushing. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3008–3015.

Bauza, M., Hogan, F. R., and Rodriguez, A. (2018). A data-efficient approach to precise and controlled pushing. In *Conference on Robot Learning*, pages 336–345.

Bauza, M., Alet, F., Lin, Y., Lozano-Perez, T., Kaelbling, L., Isola, P., and Rodriguez, A. (2019). Omnipush: accurate, diverse, real-world dataset of pushing dynamics with rgb-d video. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Bavdekar, V. A., Deshpande, A. P., and Patwardhan, S. C. (2011). Identification of process and measurement noise covariance for state and parameter estimation using extended kalman filter. *Journal of Process Control*, **21**(4), 585 – 601.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, **116**(32), 15849–15854.

Belter, D., Kopicki, M., Zurek, S., and Wyatt, J. (2014). Kinematically optimised predictions of object motion. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4422–4427. IEEE.

Byravan, A. and Fox, D. (2017). Se3-nets: Learning rigid body motion using deep neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 173–180. IEEE.

Byravan, A., Leeb, F., Meier, F., and Fox, D. (2018). Se3-pose-nets: Structured deep dynamics models for visuomotor planning and control. volume abs/1710.00489.

Corso, A., Moss, R. J., Koren, M., Lee, R., and Kochenderfer, M. J. (2020). A survey of algorithms for black-box safety validation. *arXiv preprint arXiv:2005.02979*.

Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, **2**(4), 303–314.

Dafle, N. C., Holladay, R., and Rodriguez, A. (2018). In-hand manipulation via motion cones. In *Robotics: Science and Systems*.

Degrave, J., Hermans, M., Dambre, J., and Wyffels, F. (2019). A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, **13**, 6.

Deits, R. and Tedrake, R. (2014). Footstep planning on uneven terrain with mixed-integer convex optimization. In *IEEE-RAS International Conference on Humanoid Robots*.

Donti, P., Amos, B., and Kolter, J. Z. (2017). Task-based end-to-end model learning in stochastic optimization. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5484–5494. Curran Associates, Inc.

Du, G., Wang, K., Lian, S., and Zhao, K. (2020). Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review. *Artificial Intelligence Review*.

Ebert, F., Finn, C., Lee, A. X., and Levine, S. (2017). Self-supervised visual planning with temporal skip connections. In *Conference on Robot Learning*.

Ebert, F., Dasari, S., Lee, A. X., Levine, S., and Finn, C. (2018). Robustness via retrying: Closed-loop robotic manipulation with self-supervised learning. In *Conference on Robot Learning*.

Farquhar, G., Rocktaeschel, T., Igl, M., and Whiteson, S. (2018). TreeQN and ATreeC: Differentiable tree planning for deep reinforcement learning. In *International Conference on Learning Representations*.

Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. In *IEEE International Conference on Robotics and Automation*.

Finn, C., Goodfellow, I., and Levine, S. (2016). Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems 29*, pages 64–72.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, **24**, 381–395.

Fraccaro, M., Kamronn, S., Paquet, U., and Winther, O. (2017). A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems*, pages 3601–3610.

Gal, Y. (2016). Uncertainty in deep learning.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Girin, L., Leglaive, S., Bie, X., Diard, J., Hueber, T., and Alameda-Pineda, X. (2020). Dynamical variational autoencoders: A comprehensive review. *arXiv preprint arXiv:2008.12595*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Gordon, N. J., Salmond, D. J., and Smith, A. F. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET.

Goyal, S., Ruina, A., and Papadopoulos, J. (1991). Planar sliding with dry friction part 1. limit surface and moment function. *Wear*, **143**(2), 307 – 330.

Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., and Silver, D. (2018). Learning to search with mctsnets. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1817–1826. PMLR.

Haarnoja, T., Ajay, A., Levine, S., and Abbeel, P. (2016). Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems*, pages 4376–4384.

Hermans, T., Li, F., Rehg, J. M., and Bobick, A. F. (2013). Learning contact locations for pushing and orienting unknown objects. In *IEEE-RAS International Conference on Humanoid Robots*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Hogan, F. R., Grau, E. R., and Rodriguez, A. (2018). Reactive planar manipulation with convex hybrid mpc. In *IEEE International Conference on Robotics and Automation*.

Holl, P., Thuerey, N., and Koltun, V. (2020). Learning to control pdes with differentiable physics. In *International Conference on Learning Representations*.

Hong Lee, S. and Cutkosky, M. (1991). Fixture planning with friction. *Journal of Engineering for Industry*, **113**.

Hornik, K., Stinchcombe, M., White, H., *et al.* (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, **2**(5), 359–366.

Howe, R. D. and Cutkosky, M. R. (1996). Practical force-motion models for sliding manipulation. *The International Journal of Robotics Research*, **15**(6), 557–572.

Hsu, D., Billard, A., Levine, S., Tedrake, R., and Wang, M. (2020). Ifrr colloquium: A conversation on the roles of physics-based models and data-driven learning in robotics.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. 32nd Int. Conf. on Machine Learning*, volume 37, pages 448–456.

Izatt, G., Mirano, G., Adelson, E., and Tedrake, R. (2017). Tracking objects with point clouds from vision and touch. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4000–4007.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2019). Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Janner, M., Levine, S., Freeman, W. T., Tenenbaum, J. B., Finn, C., and Wu, J. (2019). Reasoning about physical interactions with object-centric models. In *International Conference on Learning Representations*.

Jiang, Y. and Liu, C. K. (2018). Data-augmented contact model for rigid body simulation. *CoRR*, **abs/1803.04019**.

Jonschkowski, R. and Brock, O. (2015). Learning state representations with robotic priors. *Autonomous Robots*, **39**(3), 407–428.

Jonschkowski, R. and Brock, O. (2016). End-to-end learnable histogram filters.

Jonschkowski, R., Rastogi, D., and Brock, O. (2018). Differentiable particle filters: End-to-end learning with algorithmic priors. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, USA.

Julier, S., Uhlmann, J., and Durrant-Whyte, H. F. (2000). A new method for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Transactions on Automatic Control*, **45**(3), 477–482.

Julier, S. J. (2002). The scaled unscented transformation. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, volume 6, pages 4555–4559 vol.6.

Julier, S. J. and Uhlmann, J. K. (1997). New extension of the kalman filter to nonlinear systems. *Proc.SPIE*, **3068**, 3068 – 3068 – 12.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., and Levine, S. (2018). Scalable deep reinforcement learning for vision-based robotic manipulation. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, **82**(1), 35–45.

Karkus, P., Hsu, D., and Lee, W. S. (2017). Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, pages 4694–4704.

Karkus, P., Hsu, D., and Lee, W. S. (2018a). Particle filter networks with application to visual localization. In *Conference on Robot Learning*, pages 169–178.

Karkus, P., Kloss, A., Jonschkowski, R., and Kaelbling, L. P. (2018b). R:ss workshop on combining learning and reasoning – towards human-level robot intelligence.

Karkus, P., Ma, X., Hsu, D., Kaelbling, L. P., Lee, W. S., and Lozano-Pérez, T. (2019). Differentiable algorithm networks for composable robot learning. In *Robotics: Science and Systems*.

Karl, M., Soelch, M., Bayer, J., and van der Smagt, P. (2017). Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations (ICLR)*.

Kersting, K., Plagemann, C., Pfaff, P., and Burgard, W. (2007). Most likely heteroscedastic gaussian process regression. In *Proceedings of the 24th international conference on Machine learning*, pages 393–400. ACM.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kloss, A., Bauza, M., Wu, J., Tenenbaum, J. B., Rodriguez, A., and Bohg, J. (2020a). Accurate vision-based manipulation through contact reasoning. In *IEEE International Conference on Robotics and Automation*.

Kloss, A., Schaal, S., and Bohg, J. (2020b). Combining learned and analytical models for predicting action effects from sensory data. *The International Journal of Robotics Research*.

Kopicki, M., Zurek, S., Stolkin, R., Moerwald, T., and Wyatt, J. L. (2017). Learning modular and transferable forward models of the motions of push manipulated objects. *Autonomous Robots*, **41**(5), 1061–1082.

Krishnan, R. G., Shalit, U., and Sontag, D. (2016). Structured inference networks for nonlinear state space models. *arXiv preprint arXiv:1609.09869*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

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.

Lee, M. A., Zhu, Y., Zachares, P., Tan, M., Srinivasan, K., Savarese, S., Fei-Fei, L., Garg, A., and Bohg, J. (2020). Making sense of vision and touch: Learning multimodal representations for contact-rich tasks. *IEEE Transactions on Robotics*.

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, **6**(6), 861 – 867.

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

Li, J., Lee, W. S., and Hsu, D. (2018). Push-net: Deep planar pushing for objects with unknown physical properties. In *Robotics: Science and Systems*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *CoRR*, **abs/1509.02971**.

Limoyo, O., Chan, B., Marić, F., Wagstaff, B., Mahmood, A. R., and Kelly, J. (2020). Heteroscedastic uncertainty for robust generative latent dynamics. *IEEE Robotics and Automation Letters*, **5**(4), 6654–6661.

Lin, Y.-C., Ponton, B., Righetti, L., and Berenson, D. (2019). Efficient humanoid contact planning using learned centroidal dynamics prediction. In *IEEE International Conference on Robotics and Automation*.

Lynch, K. M. (1999). Locally controllable manipulation by stable pushing. *IEEE Transactions on Robotics and Automation*, **15**(2), 318–327.

Lynch, K. M., Maekawa, H., and Tanie, K. (1992). Manipulation and active sensing by pushing using tactile feedback. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, volume 1, pages 416–421.

Mason, M. T. (1986). Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, **5**(3), 53–71.

Meriçli, T., Veloso, M., and Akın, H. L. (2015). Push-manipulation of complex passive mobile objects using experimentally acquired motion models. *Autonomous Robots*, **38**(3), 317–329.

Murphy, K. P. (1998). Switching kalman filters.

Nguyen-Tuong, D. and Peters, J. (2010). Using model knowledge for learning inverse dynamics. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2677–2682. IEEE.

Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6118–6128. Curran Associates, Inc.

Okada, M., Rigazio, L., and Aoshima, T. (2017). Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*.

Osiurak, F., Rossetti, Y., and Badets, A. (2017). What is an affordance? 40 years later. *Neuroscience & Biobehavioral Reviews*, **77**, 403 – 417.

Otter, D. W., Medina, J. R., and Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Pereira, M., Fan, D. D., An, G. N., and Theodorou, E. (2018). Mpc-inspired neural network policies for sequential decision making. *arXiv preprint arXiv:1802.05803*.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5105–5114.

Rahimi, A. and Recht, B. (2017). Reflections on random kitchen sinks. `http://www.argmin.net/2017/12/05/kitchen-sinks/`. Accessed: 2020-09-02.

Rempe, D., Sridhar, S., Wang, H., and Guibas, L. J. (2019). Learning generalizable physical dynamics of 3d rigid objects. *CoRR*, **abs/1901.00466**.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., *et al.* (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, **115**(3), 211–252.

Sahoo, S., Lampert, C., and Martius, G. (2018). Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pages 4442–4450.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, **529**(7587), 484–489.

Sorenson, H. (1985). *Kalman Filtering: Theory and Application*. IEEE Press selected reprint series. IEEE Press.

Stanton, A., Reardon, J., and andJim Morris, P. D. (2008). Wall•e.

Stüber, J., Kopicki, M., and Zito, C. (2018). Feature-based transfer learning for robotic push manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5643–5650.

Stüber, J., Zito, C., and Stolkin, R. (2020). Let's push things forward: A survey on robot pushing. *Frontiers in Robotics and AI*, **7**, 8.

Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., and Corke, P. (2018). The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, **37**(4-5), 405–420.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *International Conference on Learning Representations*.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. (2016). Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. MIT press.

Valappil, J. and Georgakis, C. (2000). Systematic estimation of state noise statistics for extended kalman filters. *AIChE Journal*, **46**(2), 292–308.

Van Der Merwe, R. (2004). Sigma-point kalman filters for probabilistic inference in dynamic state-space models.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., and Silver, D. (2019). AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`.

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.

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. (2017). Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547.

Wu, J., Lu, E., Kohli, P., Freeman, B., and Tenenbaum, J. (2017). Learning to see physics via visual deanimation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 152–163. Curran Associates, Inc.

Wu, J., Zhang, C., Zhang, X., Zhang, Z., Freeman, W. T., and Tenenbaum, J. B. (2018). Learning shape priors for single-view 3d completion and reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Wüthrich, M., Garcia Cifuentes, C., Trimpe, S., Meier, F., Bohg, J., Issac, J., and Schaal, S. (2016). Robust gaussian filtering using a pseudo measurement. In *Proceedings of the American Control Conference*, Boston, MA, USA.

Yu, K.-T. and Rodriguez, A. (2018). Realtime state estimation with tactile and visual sensing. application to planar manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7778–7785. IEEE.

Yu, K. T., Bauza, M., Fazeli, N., and Rodriguez, A. (2016). More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *2016 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 30–37. Data available from `http://web.mit.edu/mcube//push-dataset`.

Yuan, W., Dong, S., and Adelson, E. H. (2017). Gelsight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors*, **17**(12), 2762.

Zendel, O., Alhaija, H. A., Benenson, R., Cordts, M., Dai, A., Fernandez, X. P., Geiger, A., Hanselmann, N., Jourdan, N., Koltun, V., Kontschieder, P., Kuznetsova, A., Kuang, Y., Lin, T.-Y., Michaelis, C., Neuhold, G., Nießner, M., Pollefeys, M., Ranftl, R., Richter, S., Rother, C., Sattler, T., Scharstein, D., Schilling, H., Schneider, N., Uhrig, J., Wulff, J., and Zhou, B. (2018). Robust vision challenge 2018. `http://www.robustvision.net/rvc2018.php`. Accessed: 2020-08-27.

Zeng, A., Song, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2020). Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, **36**(4), 1307–1319.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Zhang, L. and Trinkle, J. C. (2012). The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3805–3812.

Zhou, J., Paolini, R., Bagnell, J. A., and Mason, M. T. (2016). A convex polynomial force-motion model for planar sliding: Identification and application. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 372–377. IEEE.

Zhou, J., Mason, M. T., Paolini, R., and Bagnell, D. (2018). A convex polynomial model for planar sliding mechanics: theory, application, and experimental validation. *The International Journal of Robotics Research*, **37**(2-3), 249–265.

Zhu, M., Murphy, K., and Jonschkowski, R. (2020). Towards differentiable resampling.

Zito, C., Stolkin, R., Kopicki, M., and Wyatt, J. L. (2012). Two-level rrt planning for robotic push manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.