# Using Mean Embeddings for State Estimation and Reinforcement Learning

**Die Anwendung von Mittelwerteinbettungen zur Zustandsabschätzung und für das Bestärkende Lernen**
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von Gregor H. W. Gebhardt aus Heidelberg
Darmstadt 2019

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Gerhard Neumann

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik

Computational Learning for Autonomous Systems

Using Mean Embeddings for State Estimation and Reinforcement Learning
Die Anwendung von Mittelwerteinbettungen zur Zustandsabschätzung und für das Bestärkende Lernen

Genehmigte Dissertation von Gregor H. W. Gebhardt aus Heidelberg

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Gerhard Neumann

Tag der Einreichung: 27.11.2018
Tag der Prüfung: 17.01.2019

Darmstadt 2019

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-84340
URL: http://tuprints.ulb.tu-darmstadt.de/8434

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

Die Veröffentlichung steht unter folgender Creative Commons Lizenz:
Namensnennung – Nicht kommerziell – Keine Bearbeitung 4.0 International
http://creativecommons.org/licenses/by-nc-nd/4.0

# Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Bei der vorliegenden Dissertation stimmen schriftliche und elektronische Version überein.

Darmstadt, den 27.11.2018

(Gregor H. W. Gebhardt)

# Abstract

To act in complex, high-dimensional environments, autonomous systems require versatile state estimation techniques and compact state representations. State estimation is crucial when the system only has access to stochastic measurements or partial observations. Furthermore, in combination with models of the system such techniques allow to predict the future which enables the system to asses the outcome of possible decisions. Compact state representations alleviate the curse of dimensionality by distilling the important information from high-dimensional observations.

Due to noisy sensory information and non-perfect models of the system, estimates of the state never reflect the true state perfectly but are always subject to errors. The natural choice to incorporate the *uncertainty* about the state estimate is to use a probability distribution as representation. This results in the so called *belief state*.

High-dimensional observations, for example images, often contain much less information than conveyed by their dimensionality. But also if all the information is necessary to describe the state of the system—for example, think of the state of a swarm with the positions of all agents—a less complex description might be a sufficient representation. In such situations, finding the *generative distribution* that explains the state would give a much more compact while informative representation.

Traditionally, parametric distributions have been used as state representations such as most prevalently the Gaussian distribution. However, in many cases a unimodal distribution might not be sufficient to represent the belief state. Using multi-modal probability distributions, instead, requires more advanced approaches such as mixture models or particle-based Monte Carlo methods. Learning mixture models is however not straightforward and often results in locally optimal solutions. Similarly, maintaining a good population of particles during inference is a complicated and cumbersome process. A third approach is kernel density estimation which is located at the intersection of mixture models and particle-based approaches. Still, performing inference with any of these approaches requires heuristics that lead to poor performance and a limited scalability to higher dimensional spaces.

A recent technique that alleviates this problem are the embeddings of probability distributions into reproducing kernel Hilbert spaces (RKHS). Conditional distributions can be embedded as operators based on which a framework for inference has been presented that allows to apply the sum rule, the product rule and Bayes' rule entirely in Hilbert space. Using sample based estimators and the kernel-trick of the representer theorem allows to represent the operations as vector-matrix manipulations. The contributions of this thesis are based on or inspired by the embeddings of distributions into reproducing kernel Hilbert spaces.

In the first part of this thesis, I propose additions to the framework for non-parametric inference that allow the inference operators to scale more gracefully with the number of samples in the training set. The first contribution is an alternative approach to the condi-

tional embedding operator formulated as a least-squares problem which allows to use only a subset of the data as representation while using the full data set to learn the conditional operator. I call this operator the *subspace conditional embedding operator*. Inspired by the least-squares derivations of the Kalman filter, I furthermore propose an alternative operator for Bayesian updates in Hilbert space, the *kernel Kalman rule*. This alternative approach is numerically more robust than the kernel Bayes rule presented in the framework for non-parametric inference and scales better with the number of samples. Based on the kernel Kalman rule, I derive the *kernel Kalman filter* and the *kernel forward-backward smoother* to perform state estimation, prediction and smoothing based on Hilbert space embeddings of the belief state. This representation is able to capture multi-modal distributions and inference resolves—due to the kernel trick—into easy matrix manipulations.

In the second part of this thesis, I propose a representation for large sets of homogeneous observations. Specifically, I consider the problem of learning a controller for object assembly and object manipulation with a robotic swarm. I assume a swarm of homogeneous robots that are controlled by a common input signal, e.g., the gradient of a light source or a magnetic field. Learning policies for swarms is a challenging problem since the state space grows with the number of agents and becomes quickly very high dimensional. Furthermore, the exact number of agents and the order of the agents in the observation is not important to solve the task. To approach this issue, I propose the *swarm kernel* which uses a Hilbert space embedding to represent the swarm. Instead of the exact positions of the agents in the swarm, the embedding estimates the generative distribution behind the swarm configuration. The specific agent positions are regarded as samples of this distribution. Since the swarm kernel compares the embeddings of distributions, it can compare swarm configurations with varying numbers of individuals and is invariant to the permutation of the agents. I present a hierarchical approach for solving the object manipulation task where I assume a high-level object assembly policy as given. To learn the low-level object pushing policy, I use the swarm kernel with an actor-critic policy search method. The policies which I learn in simulation can be directly transferred to a real robotic system.

In the last part of this thesis, I investigate how we can employ the idea of kernel mean embeddings to deep reinforcement learning. As in the previous part, I consider a variable number of homogeneous observations—such as robot swarms where the number of agents can change. Another example is the representation of 3D structures as point clouds. The number of points in such clouds can vary strongly and the order of the points in a vectorized representation is arbitrary. The common architectures for neural networks have a fixed structure that requires that the dimensionality of inputs and outputs is known in advance. A variable number of inputs can only be processed by applying tricks. To approach this problem, I propose the *deep M-embeddings* which are inspired by the kernel mean embeddings. The deep M-embeddings provide a network structure to compute a fixed length representation from a variable number of inputs. Additionally, the deep M-embeddings exploit the homogeneous nature of the inputs to reduce the number of parameters in the network and, thus, make the learning easier. Similar to the swarm kernel, the policies learned with the deep M-embeddings can be transferred to different swarm sizes and different number of objects in the environment without further learning.

# Zusammenfassung

Autonome Systeme benötigen vielseitige Techniken zur Zustandsabschätzung und kompakte Zustandsrepräsentationen, um in komplexen, hochdimensionalen Umgebungen zu agieren. Zustandsabschätzungen sind wichtig, wenn das System nur stochastische Messungen oder partielle Beobachtungen zur Verfügung hat. Mit Modellen des Systems sind durch solche Techniken zudem Vorhersagen in die Zukunft möglich, welche die Beurteilung der Folgen von möglichen Entscheidungen des Systems erlauben. Kompakte Repräsentationen mildern den Fluch der Dimensionalität indem sie die wichtigen Informationen aus hochdimensionalen Beobachtungen herausdestillieren.

Aufgrund verrauschter Sensorinformationen und fehlerhafter Systemmodelle sind Zustandsabschätzungen niemals exakt sondern immer fehlerbehaftet. Die naheliegende Wahl, um die Unsicherheit einer Zustandsabschätzung darzustellen, ist die Repräsentation als Wahrscheinlichkeitsverteilung, dem sogenannten *belief state*.

Hochdimensionale Beobachtungen, zum Beispiel Bilder, beinhalten oft viel weniger Informationen als die Dimensionalität vermittelt. Aber auch wenn die gesamte Information notwendig ist, um den Zustand zu beschreiben – zum Beispiel im Falle eines Schwarms, dessen Zustand die Positionen der einzelnen Agenten enthält – könnte eine einfachere Repräsentation ausreichend sein. In solchen Situationen würde eine generative Verteilung, die den Zustand erklärt, eine kompaktere und trotzdem informative Darstellung erlauben.

Herkömmlich werden parametrische Verteilungen, vor allem die Gaußverteilung, als Repräsentation verwendet. In vielen Fällen ist solch eine unimodale Verteilung nicht ausreichend, um den *belief state* darzustellen. Die Verwendung von multimodalen Modellen erfordert fortgeschrittenere Ansätze, wie Mischverteilungen oder partikelbasierte Monte-Carlo-Methoden. Mischverteilungen zu lernen ist jedoch nicht trivial und resultiert oft in lokalen Optima. Genauso ist die Erhaltung einer guten Population während der Inferenz mit Monte-Carlo-Methoden ein komplizierter Prozess. Ein weiterer Ansatz sind Kerndichteschätzer, die als Hybrid von Mischverteilungen und partikelbasierten Ansätzen gesehen werden können. Jeder dieser Ansätze erfordert für die Inferenz Heuristiken, die zu einer schlechten Leistung und begrenzter Skalierbarkeit auf hochdimensionale Räume führen.

Eine neuere Technik, die dieses Problem reduziert, sind die Mittelwerteinbettungen von Wahrscheinlichkeitsverteilungen in die *Reproducing Kernel Hilbert Spaces* (RKHS). Bedingte Verteilungen können als Operatoren in solche Räume eingebettet werden. Basierend darauf wurde ein Regelwerk für nicht-parametrische Inferenz präsentiert, welches erlaubt, die Summenregel, die Produktregel und die Bayes'sche Regel gänzlich im Hilbertraum anzuwenden. Die Verwendung von datenbasierten Schätzfunktionen und der Kernel-Trick des *Representer-Theorems* ermöglichen es, die Operationen als 'Vektor-Matrix-Manipulationen' darzustellen. Die Ergebnisse dieser Arbeit basieren auf bzw. sind inspiriert von diesen Einbettungen von Verteilungen in *Reproducing Kernel Hilbert Spaces*.

Im ersten Teil dieser Arbeit werden Ergänzungen zum *Framework for Non-parametric Inference* präsentiert, die die Skalierbarkeit der Inferenz-Operatoren mit der Anzahl an

Datenpunkten im Trainingsdatensatz verbessern. Basierend auf der Formulierung eines Optimierungsproblems wird hierzu zunächst der *Subspace Conditional Embedding Operator* als alternative Formulierung des Conditional Embedding Operators vorgeschlagen. Dieser erlaubt nur eine Teilmenge der Daten als Repräsentation der Verteilung zu verwenden, während der vollständige Datensatz zum Lernen des Operators verwendet wird. Inspiriert von der Herleitung des Kalman-Filters basierend auf der Methode der kleinsten Quadrate, wird die *Kernel Kalman Rule* außerdem als alternativer Operator für die Bayes'sche Regel im Hilbertraum vorgeschlagen. Dieser alternative Ansatz ist numerisch robuster als die im *Framework for Non-parametric Inference* enthaltene *Kernel Bayes' Rule* und skaliert besser mit der Anzahl der Datenpunkte. Basierend auf der *Kernel Kalman Rule* werden das *Kernel Kalman Filter* und der *Kernel Forward-Backward Smoother* hergeleitet, die es erlauben, den Zustand mittels Einbettungen des *belief state* in den Hilbertraum abzuschätzen, vorherzusagen oder zu glätten. Diese Darstellung des *belief state* ist in der Lage, multimodale Verteilungen darzustellen und Inferenz lässt sich – aufgrund des Kernel-Tricks – durch einfache Matrixmanipulationen durchführen.

Im zweiten Teil dieser Arbeit wird eine Repräsentation für große Mengen homogener Beobachtungen vorgeschlagen. Im Speziellen wird das Problem der Regelung eines Roboterschwarms zur Manipulation und Zusammenführung von Objekten betrachtet. Es wird ein Schwarm homogener Roboter, der durch ein gemeinsames Signal – z.B. der Gradient einer Lichtquelle oder eines Magnetfelds – kontrolliert werden kann, verwendet. Solche Regelungen für Roboterschwärme zu erlernen ist nicht einfach, da der Zustandsraum mit der Anzahl der Agenten anwächst und schnell sehr hochdimensional wird. Außerdem ist die genaue Anzahl der Agenten im Schwarm sowie die Zuordnung der Agenten zu den Positionen nicht ausschlaggebend, um die Aufgabe zu erledigen. Der *Swarm Kernel* löst dieses Problem indem der Schwarm durch eine Einbettung in einen Hilbertraum dargestellt wird. Statt der exakten Positionen der Agenten stellt die Einbettung die generative Verteilung hinter der Schwarmkonfiguration dar. Die einzelnen Agentenpositionen können als Stichproben dieser Verteilung betrachtet werden. Da die generativen Verteilungen verglichen werden, ist der *Swarm Kernel* invariant zur Anzahl der Agenten und bezüglich der Zuordnung der Agenten zu ihren Positionen im Schwarm. Es wird ein hierarchischer Ansatz präsentiert um das Problem der Objektzusammenführung zu lösen. Der übergeordnete Plan zur Zusammenführung wird als gegeben betrachtet, während die lokale Regelung zur Objektmanipulation mittels einer Actor-Critic Policy Search Methode erlernt wird. Die in Simulationen gelernten Regler können direkt auf ein Robotersystem angewendet werden.

Im dritten Teil dieser Arbeit wird untersucht, wie die Idee der Einbettung von Verteilungen in Hilberträume auf das tiefe bestärkende Lernen übertragen werden kann. Wie im vorherigen Teil wird eine variable Anzahl von homogenen Beobachtungen angenommen, beispielsweise von einem Roboterschwarm, dessen Größe sich ändern kann. Ein weiteres Beispiel ist die Darstellung von dreidimensionalen Strukturen als Punktwolken. Die Anzahl der Punkte in solchen Wolken kann stark variieren und die Ordnung der Punkte in einer vektorisierten Darstellung ist beliebig. Die gängigen Architekturen für *Neuronale Netze* verlangen eine starre Struktur, bei der die Dimensionalität der Ein- und Ausgänge von vornherein festgelegt werden muss. Eine variable Anzahl von Beobachtungen lässt sich nur über Kunstgriffe verarbeiten. Um dieses Problem zu lösen, werden die *Deep M-Embeddings* vorgeschlagen, eine Netzwerkstruktur die von den Einbettungen in Hilberträume inspiriert ist. Die *Deep M-Embeddings* erlauben die Repräsentation einer Menge von Beobachtungen durch einen Vektor fester Dimensionalität. Zusätzlich ermöglichen sie die Homogenität der Daten auszunutzen, um die Parameteranzahl des Netzwerks zu reduzieren und dadurch ein schnelleres Lernen zu ermöglichen.

# Acknowledgments

The work that led to this thesis would not have been possible without the support of numerous people. First and foremost I want to thank my ‘Doktorvater’ Geri who accompanied me with endless support during the four years of my Ph.D. studies and had always helpful and valuable advice for all of my problems. The visit at his lab in Lincoln gave me a huge thrust for successfully finishing this thesis. I also want to thank Jan for his support and convincing suggestions and for giving me from time to time a push into the right direction. I am very happy that I could be part of his IAS group at TU Darmstadt also after Geri left to England.

With numerous inspiring and exciting discussions and with wonderful moments after work, my colleagues made my time at CLAS and IAS an unforgettable period of my life for which I am very grateful. Especially, I want to thank my office mates Herke, Chris, Daniel, Hany, Svenja, Doro and Rudi, for the pleasant, relaxed and helpful atmosphere at the lab. My coauthors Andras and Max gave me a lot of great feedback on how to improve my work and how to strengthen my arguments. Further, I want to thank the staff at IAS, Veronika, Marion, Nanette, and Sabine, for their support and help with all the bureaucratic and technical hurdles. The students that have worked with me, Philipp, Kevin, Marius, Daniel, and Alexander, have accomplished excellent work that has greatly supported me with my research. I want to thank the members of my committee, Prof. Roth, Prof. Kersting, and Prof. Fürnkranz for the work and time which they have put into my defense of this thesis.

I am most grateful to my parents who have always supported me during my studies and have contributed a big part in the success of this thesis. My brothers have always been ideals to follow, to learn from, and to improve upon. I want to thank my friends for all the great days and nights at Darmstadt or anywhere else. They provided the essential counterpart to my work. And last but most, I want to thank Paula for all her love and support and for bearing all my bad mood in the last years. Better together—this is the beginning!

# Contents

**1 Introduction** **1**
1.1 Probability Distributions and their Representations . . . . . . . . . . . . . . . . . . . . 2
  1.1.1 Discrete Probability Distributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.1.2 Continuous Probability Distributions . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.2 State Estimation with Models Learned from Data . . . . . . . . . . . . . . . . . . . . 10
1.3 Swarm Representations for Learning Policies . . . . . . . . . . . . . . . . . . . . . . . 11
1.4 Learning Deep Representations for Sets of Homogeneous Inputs . . . . . . . . . 12
1.5 Challenges Addressed by the Contributions of this Thesis . . . . . . . . . . . . . . 12

**2 The Kernel Kalman Rule** **15**
2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  2.1.1 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.2 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
  2.2.1 Nonparametric Inference with Hilbert Space Embeddings of Distributions 19
  2.2.2 The Kalman Filter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
2.3 Efficient Nonparametric Inference in a Subspace . . . . . . . . . . . . . . . . . . . . 23
  2.3.1 Selecting the Sample Set to Span the Subspace . . . . . . . . . . . . . . . . . . 25
  2.3.2 Relation to Other Sparsification Approaches . . . . . . . . . . . . . . . . . . . . 25
  2.3.3 The Subspace Kernel Sum Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
  2.3.4 The Subspace Kernel Chain Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
  2.3.5 The Subspace Kernel Bayes' Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
  2.3.6 Experimental Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.4 The Kernel Kalman Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
  2.4.1 Estimating the Posterior Mean Embedding from a Least Squares Objective 29
  2.4.2 Using Recursive Least Squares to Estimate the Posterior Embedding . . . 30
  2.4.3 Empirical Kernel Kalman Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
  2.4.4 The Subspace Kernel Kalman Rule . . . . . . . . . . . . . . . . . . . . . . . . . . 34
  2.4.5 Experimental Comparison of (sub)KKR and (sub)KBR . . . . . . . . . . . . . 35
2.5 Applications of the Kernel Kalman Rule . . . . . . . . . . . . . . . . . . . . . . . . . 37
  2.5.1 The Kernel Kalman Filter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
  2.5.2 The Subspace Kernel Kalman Filter . . . . . . . . . . . . . . . . . . . . . . . . . . 41
  2.5.3 Experimental Evaluation of the Kernel Kalman Filter . . . . . . . . . . . . . . 42
  2.5.4 The Kernel Forward-Backward Smoother . . . . . . . . . . . . . . . . . . . . . . 47
  2.5.5 The Subspace Kernel Forward-Backward Smoother . . . . . . . . . . . . . . . 51
  2.5.6 Experimental Evaluation of the Kernel Forward Backward Smoother . . . 51
2.6 Conclusion & Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

3 Learning Swarm Policies for Pushing Objects 59
3.1 Introduction . . . 59
3.2 Related Work . . . 61
3.3 Preliminaries . . . 61
3.3.1 Actor-Critic Relative Entropy Policy Search . . . 61
3.3.2 Kernel Embeddings of Distributions . . . 65
3.3.3 Planning Strategies . . . 66
3.4 Learning Control Policies for Object Assembly . . . 67
3.4.1 The Object Movement Policy . . . 67
3.4.2 Assembly Policy and Path Planning Strategy . . . 69
3.5 Experimental Setup & Results . . . 70
3.5.1 Evaluation of the Learning Algorithm . . . 70
3.5.2 The Assembly Task in Simulation . . . 76
3.5.3 The Kilobot Setup . . . 79
3.5.4 The Assembly Task on the Kilobots . . . 81
3.6 Conclusions . . . 81

4 Using M-Embeddings to Learn Control Strategies for Robot Swarms 83
4.1 Introduction . . . 83
4.1.1 Related Work . . . 85
4.2 Preliminaries . . . 86
4.2.1 Trust Region Policy Optimization . . . 86
4.2.2 Neural Networks as Function Approximator for Policy and Value . . . 88
4.2.3 Mean Embeddings of Distributions . . . 88
4.3 Deep M-embeddings . . . 89
4.4 Learning Swarm Policies . . . 90
4.4.1 Swarm Agents . . . 91
4.4.2 Policy and Value Function Network . . . 92
4.5 Experimental Setup and Evaluation . . . 92
4.5.1 The Kilobot Gym . . . 93
4.5.2 Tasks and Reward Functions . . . 94
4.5.3 Experimental Evaluation . . . 95
4.6 Conclusions . . . 99

5 Conclusion 103
5.1 Future Work . . . 104

List of Figures 107

List of Tables 109

List of Algorithms 109

Publications 111

**Bibliography** **113**

**Appendix** **123**
A Derivations for the Subspace Conditional Operator . . . . . . . . . . . . . . . . . . 123
B Derivations for the Kernel Kalman Rule and its Applications . . . . . . . . . . . 123
B.1 The Residual of the Observation Operator is Unbiased . . . . . . . . . . . 124
B.2 Derivation of the Optimal Kalman Gain Operator . . . . . . . . . . . . . . 124
B.3 Simplifying the Update of the Covariance Operator . . . . . . . . . . . . 125
B.4 Derivations for the Sample-based Kernel Kalman Rule . . . . . . . . . . . 126
B.5 Derivation of the Subspace Kernel Kalman Gain Operator/Matrix . . . . 127

**Curriculum Vitae** **129**

# 1 Introduction

Intelligence can be seen as the process of comprehending our surroundings and making decisions that “make sense” [31], where “making sense” is usually related to an optimization of some outcome. This comprehension of the surroundings requires that we have an internal estimate of the state of our environment. Such an estimate is naturally not perfect as we do not have perfect sensory equipment. The entire process of comprehending and making decisions could be modeled in three steps. First, a sensory system delivers observations of the environment. Second, these observations are then processed and used to update an internal state. And third, from this internal state an optimal action can be deducted and executed. In this thesis, I concentrate mostly on the second part of this model which is concerned with the internal state representations of the observations.

Consider as an example the driver of a car. While she is driving, the driver usually points her attention, i.e., her gaze, towards the street. She needs to observe the road and the traffic in order to stay on the street and to avoid accidents. Hence, the driver is not tracking the speed of the car exactly. Merely, she is looking at the speedometer only from time to time and otherwise estimates the speed from the observed environmental cues.[1] The driver internally maintains an estimate of the speed which is updated by various observations. But this estimate of the speed alone does not explain when the driver should make the decision to look at the speedometer. This observation suggests that not only the estimation of the state is important for making intelligent decisions, but also to assess the *uncertainty* of this estimate. In addition, uncertainty not only enables us to asses the quality of a point estimate but also provides a measure for weighting the impact of new measurements on the update of the estimate. Consider, for example, a robotic arm with mediocre sensors for the joint states that only give poor estimates about the current state of the robot. To be able to control the robot arm, we maintain an estimate of the true joint states. If we get new sensor readings, we can integrate the measurements into our estimate. However, it is hard to decide only from the estimate and the measurement to which extend we trust our estimate and how much credit we give to the new measurement. If we have access to the uncertainty of both values, we can use this information in the update of our estimate.

*Probability distributions* allow us to express the uncertainty about a quantity in mathematical terms by modeling it as a random variable. In this understanding, the true value of the quantity we want to estimate is a single, specific value which we do not know. The distribution describes our confidence that the true value is in a certain area of the state space. Yet, probability distributions can also help us to represent the state of a system from another perspective. Instead of describing a single state with uncertainty, we can also use

---

[1] Interestingly, this ability to estimate the speed from visual cues is better during night than during day [112] and, moreover, especially during maneuvers that require a more precise control of speed, the driver tends to not look at the speedometer [15].

a probability distribution as abstraction from a high dimensional state if this state satisfies certain properties.

Consider, for example, the state of a swarm of simple robots where we assume that it is sufficient to describe the state of each agent by its position. But even if we assume such simple agents, the state of the entire swarm grows with the number of agents and becomes quickly very high dimensional. We might, however, not be interested in the exact location of each individual agent. Also, we do not care which of the individual agents is at which specific position in the swarm. Swapping the location of two agents would not have any impact on our perception of the swarm as a whole. Hence, in contrast to describing the state of the swarm by the individual states of the agents we could also try to find a more abstract representation. One possibility is to assume a generative probability distribution of which the current state of the swarm is one particular set of samples. The position of one agent would then be a single sample of that distribution. Instead of a high dimensional vector of agent positions, we would only need a representation of the distribution to describe the state of the swarm. In general, such a representation would allow to represent a set of observations while abstracting from the number and the order of the observations. Besides the state of a swarm, other examples for such sets of observations are point clouds or detected obstacles in the environment of a robot.

This latter use of probability distributions as representation of the state is different from the use case I have described before. In the first case, we assume that there is one true state which we do not know exactly but which we belief to be in a certain area around our estimate. In the second case, the probability distribution is instead used as an abstraction of the state where the current true state is just one possible realization of the abstraction. To give a brief overview, I discuss different forms of probability distributions, their representations, and their limitations in the following sections. This overview motivates my choice of using non-parametric representations of probability distributions in this thesis.

## 1.1 Probability Distributions and their Representations

From a traditional perspective, probability distributions describe the *relative occurrence* of an event with respect to all other possible events. Here, an event can be defined as the specific outcome of an experiment that can be described by a logical sentence which is either true or false. For example, if the experiment is to roll a die, the event could be ⚂ (i.e., ‘3 facing up’). This approach is called the *frequentist approach* since the definition assumes that the experiment can be repeated many times to find frequencies of the events in the outcomes. To roll a die can be easily repeated for many times until we converge to frequency values, as the experiment is simple to setup, cheap, and finishes after a short time. From a frequentist perspective, we could roll a die 600 times and would observe around 100 times the event ⚂. Thus we can deduct that the probability of this event is roughly $\frac{1}{6}$. However, there are many experiments that we do not want to repeat, or that are to hard if not impossible to repeat. The weather, for example, can not be influenced in a way that would render it a repeatable experiment. Still, we want to know how likely it is that it will rain tomorrow or not.

Thus, an alternative approach assumes that probability distributions describe the *uncertainty* that is inherently connected to the potential events that could be an outcome of an experiment. This more modern viewpoint is called the *Bayesian approach*. In the example of rolling a die, the Bayesian approach would consider that each face is equally likely to show up and thus, the probability of ⚂ is $\frac{1}{6}$.

In probability theory, an experiment is represented by a random variable $X$ which takes for a certain event a certain value from the set of all possible events, called the sample space $\Omega$. The probability mass function (pmf) $pmf(X)$ defines a probability value for each event $X = x, x \in \Omega$. The value of the pmf is between 0 and 1, i.e., $0 \leq p(X = x) \leq 1$, where 1 means that the event will definitely occur and, thus, 0 means that the event will definitely not occur. Furthermore, the pmf integrates to 1 over the sample space, i.e., $\int_{x \in \Omega} pmf(X = x) = 1$. Probability distributions can be classified into discrete and continuous distributions, where the first assume a finite or countable sample set $\Omega$ and the second assume a continuous sample space $\Omega$.

Often, we do not only want to express probabilities for the outcomes of a single random variable (marginal probability distributions). If we have two or more random variables, e.g., $X$ with sample space $\Omega_X$ and $Y$ with sample space $\Omega_Y$, we can use joint probability distributions (or multivariate distributions) $p(X = x, Y = y)$ to describe the probabilities for different outcomes of the random variables. Similarly, a conditional probability distribution $p(X|Y)$ describes the probabilities of the random variable $X$ given the value of the random variable $Y$. A joint probability distribution $p(X, Y)$ can be transformed into the marginal distribution $p(X)$ by marginalization w.r.t. the random variable $X$, or into the conditional probability distributions $p(X|Y)$ by conditioning on the variable $Y$ as

$$p(X) = \sum_{\Omega_Y} p(X, Y), \qquad \text{and} \qquad p(X|Y) = \frac{p(X, Y)}{p(Y)}, \tag{1.1}$$

respectively. Two random variables $X$ and $Y$ are called independent, if their joint distribution equals the product of their marginals, i.e., $p(X, Y) = p(X)p(Y)$. In this case, the outcome of one random variable yields no information on the distribution of the other random variable. This can be also seen from the conditional distribution which is equal to the marginal distribution if $X$ and $Y$ are independent.

### 1.1.1 Discrete Probability Distributions

Discrete probability distributions define a probability mass for each element $x$ of the sample space $\Omega$. This means that we need in general $N - 1$ parameters to represent the distribution over a finite sample space $\Omega$, where $N$ is the cardinality of $\Omega$. We need one parameter less than elements in the sample space because the constraint that the probability values have to sum to 1 takes one degree of freedom from the model. For joint and conditional distributions, this form of representation explodes exponentially with the number of random variables. Exceptions are parametric representations, such as the uniform distribution or the binomial distribution which have for example only one and two parameters, respectively. For example, the binomial distribution with pmf

$$p_{\text{bin}}(X = k) = \binom{n}{k} p^k (1 - p)^{n-k} \tag{1.2}$$

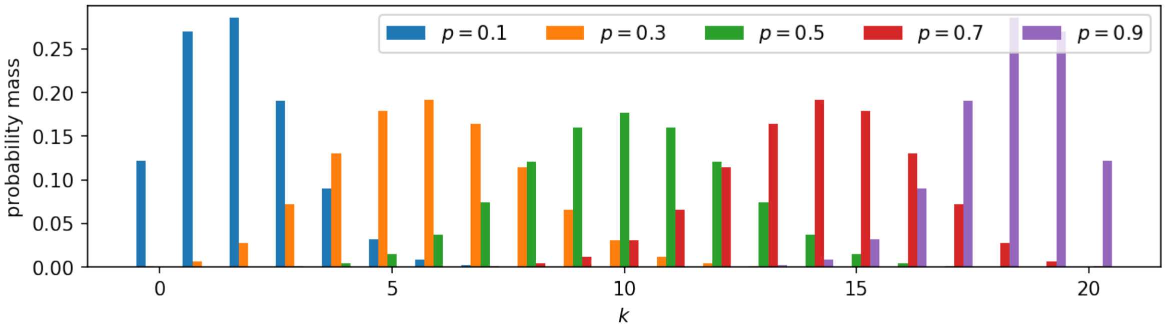


**Figure 1.1:** Binomial distribution with $n = 20$ and different values of $p$.

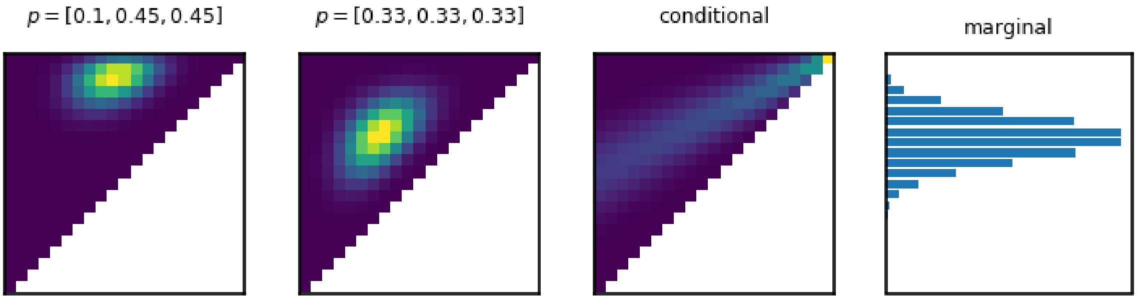


**Figure 1.2:** Multinomial distribution $p(X, Y)$ with $n = 20$ and different categorical distributions $p$, as well as the conditional distribution $p(X|Y)$ and the marginal distribution $p(X)$ of the second multinomial distribution.

describes the probability of having $k$ successes in $n$ experiments, where $p$ is the Bernoulli distribution of success and non-success for a single experiment. This parametric representation of a distribution with $n$ discrete values only has two parameters, i.e., $p$ and $n$. Consequently, the range of distributions that can be represented is limited. Figure 1.1 shows the probability masses with $n = 20$ for different values of $p$. The single degree of freedom in this representation with the parameter $p$ only allows to shift the single mode of the distribution between $k = 0$ for $p \to 0$ and $k = n$ for $p \to 1$. An example of a multivariate, parametric, discrete probability distribution is the multinomial distribution. Now, the underlying experiment has not a Bernoulli distribution with two possible outcomes ('success' and 'no success'), but a categorical distribution (also multinoulli distribution) with $K$ possible outcomes. For an experiment with 3 possible outcomes $x, y, z$, the multinomial distribution has two random variables $X, Y$ which count the number of the respective outcomes $x$ and $y$ in $n$ trials. Figure 1.2 depicts the multinomial distribution for $n = 20$ trials with two different categorical distributions, as well as the conditional distribution $p(X|Y)$ and the marginal distribution $p(X)$ which are obtained by conditioning and marginalizing, respectively, the second multinomial distribution.

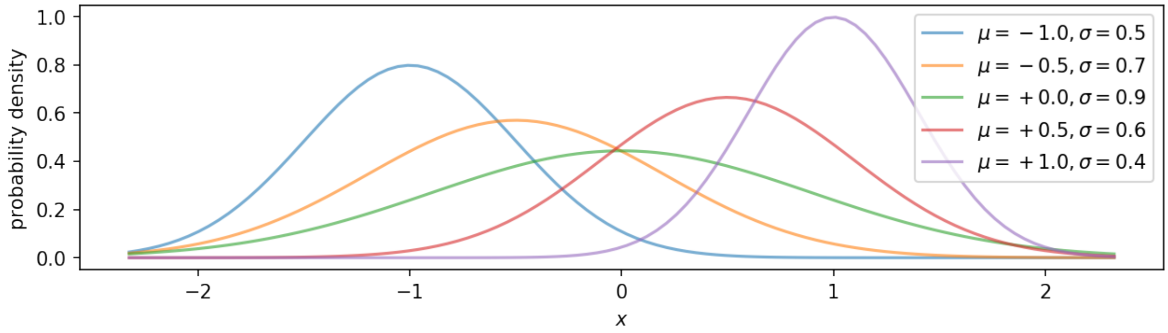


**Figure 1.3:** Probability density of a normal distribution for different values of $\mu$ and $\sigma$.

## 1.1.2 Continuous Probability Distributions

Continuous probability distributions are defined by cumulative distribution functions (cdf) which describe the probability that the random variable $X$ is less than a given value $x$. The derivative of the cdf is the probability density function (pdf) which gives the probability mass for an infinitesimal value of the sample space $\Omega$.

Continuous probability distributions usually do require parametric models as representation. Besides the continuous uniform distribution, the most prominent example is probably the Gaussian distribution, also called the normal distribution, that is defined by the pdf

$$\mathcal{N}(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]. \tag{1.3}$$

The parameters $\mu$ and $\sigma$ allow to move the mode of the distribution and to scale the width, respectively. Figure 1.3 depicts normal distributions for different values of $\mu$ and $\sigma$.

Many other parametric continuous distributions exist, however, they usually share the common property that they have their probability mass distributed around a single mode. Exceptions are, for example, the bimodal beta and arcsine distributions which have two modes, one at each limit of the sample space. Most of these distributions have two or even only one parameter.

In the multivariate case with $k$ random variables, the Gaussian distribution has the probability density function

$$\mathcal{N}(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\intercal\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right], \tag{1.4}$$

where $\boldsymbol{x}$ and $\boldsymbol{\mu}$ are $k$-dimensional vectors and $\boldsymbol{\Sigma} \in \mathbb{R}^{k\times k}$ is the covariance matrix. If we partition the random variables into two subsets $X_1$ and $X_2$, we get the conditional distribution of $X_1$ given $X_2$ with pdf

$$p(X_1 = \boldsymbol{x}_1 | X_2 = \boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{x}_1; \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}). \tag{1.5}$$

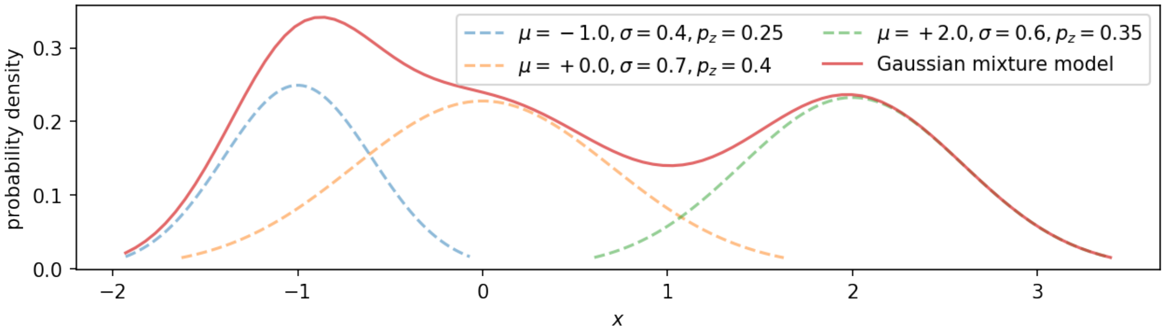


**Figure 1.4:** Gaussian mixture model with three components.

Here, $\mathbf{\Sigma}_{11}$, $\mathbf{\Sigma}_{22}$, and $\mathbf{\Sigma}_{12}$ are the covariance matrices of the two subsets and the cross-covariance matrix, respectively. A marginal distribution of the subset can be obtained by dropping the remaining variables from mean and covariance, i.e., $p(X_1 = \boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_1; \boldsymbol{\mu}_1, \mathbf{\Sigma}_{11})$.

## Mixture Models, Histograms and Kernel Density Estimation

Modeling distributions with more than one mode, i.e., bimodal or especially multi-modal distributions, requires more advanced, often hierarchical approaches such as *mixture models*. A mixture model consists of multiple, but a fixed number of components. These components are usually continuous, parametric distributions, for example Gaussian distributions. Mixture models furthermore assume a latent variable $z_i$ that determines which component is responsible for each sample $x_i$. A discrete distribution $p(z_i)$ models the prior probability over the latent variables. By using normal distributions as components we obtain a Gaussian mixture model with pdf

$$p(x) = \sum_{i=1}^{N} \mathcal{N}(x; \mu_i, \sigma_i) p(z_i). \tag{1.6}$$

Figure 1.4 shows a mixture model with three Gaussian components. Learning such mixture models from data is not trivial and requires techniques such as the iterative expectation-maximization algorithm to find locally optimal parameters. Another learning approach for mixture models is spectral learning. However, this approach assumes that the individual components are well separated [1].

Alternatively, *histograms* estimate probability distributions by discretization of the sample space into equidistant bins and estimate the probability of each discrete state by counting the samples that fall into each of these bins. The most prevalent issue of histograms is however the exponential growth of the number of parameters with the dimensionality of the data. A trade-off between accuracy, i.e., small bins, and efficiency in the number of parameters and the number of samples that are required to get a good estimate of the distribution, i.e., large bins, is necessary. Figure 1.6 shows a representation of the Gaussian mixture model using histograms with different bin sizes estimated from 1000 samples.

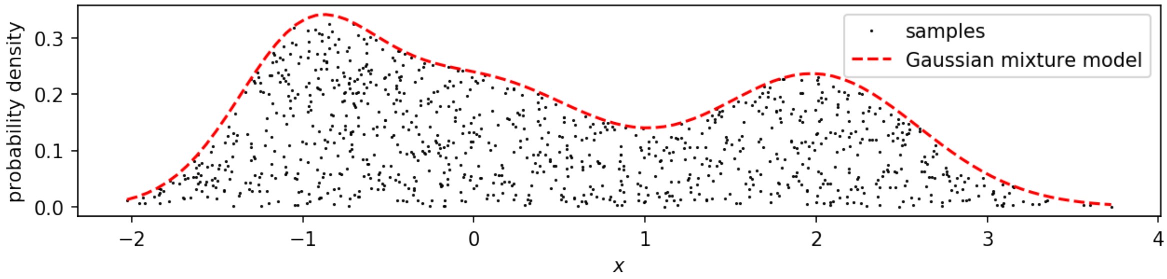


**Figure 1.5:** 1000 samples drawn from the Gaussian mixture model. For visualization the samples are randomly distributed between zero and the probability density value at their location.

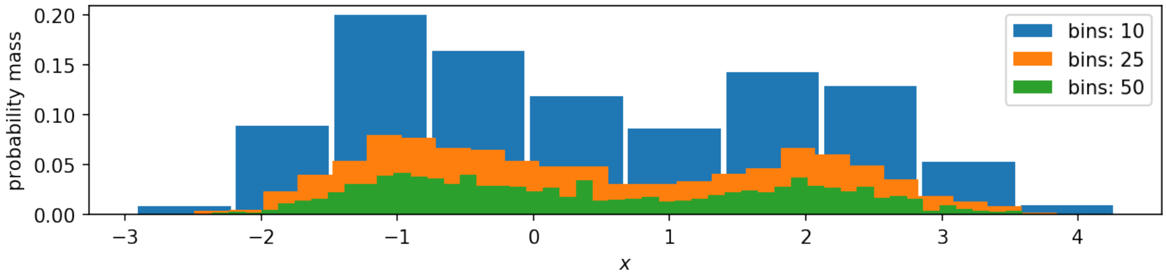


**Figure 1.6:** Histograms generated from 1000 samples taken from the Gaussian mixture model (c.f. Figure 1.4) with different numbers of bins.

A non-parametric extension of histograms is *kernel-density estimation*. Here, the pdf is a linear combination of kernel functions $k$ that are centered at the i.i.d. samples $x_i$ from which we want to estimate the density, i.e.,

$$\tilde{p}(x) = \frac{1}{n}\sum_{i}^{N} k(x, x_i). \tag{1.7}$$

In this context, the term *kernel function* refers to a positive function that integrates to 1. Examples are the uniform kernel, the triangular kernel, or the Gaussian kernel. In the case of a Gaussian kernel, kernel density estimation is related to a Gaussian mixture-model with a mixture component centered at each sample point and a common variance which is also called bandwidth in the case of a kernel function. Since we assume the samples to be i.i.d., the discrete distribution over the mixture components is a uniform distribution $p(x_i) = \frac{1}{n}$. For mixture models the problem is to find an optimal number of components and a good initialization before optimizing the parameters of each component and of the prior distribution. In kernel density estimation, this problem is transferred to finding an optimal bandwidth of the kernel function.

The density of a joint probability distribution can be estimated by using a product kernel or a suitable kernel function for all variables, e.g., a multivariate Gaussian kernel. From

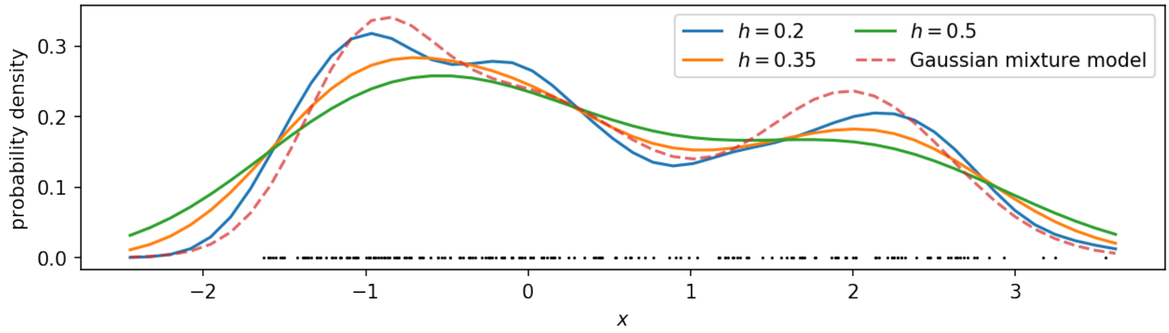


**Figure 1.7:** Kernel density estimation of the Gaussian mixture model using 200 samples (depicted as black dots) with a Gaussian kernel.

the joint kernel density estimate and a marginal density estimate, we can obtain the kernel density estimate of a conditional distribution by applying Equation 1.1 as

$$\tilde{p}(x|y) = \frac{\tilde{p}(x,y)}{\tilde{p}(y)} = \frac{\sum_i^N k(x,x_i)g(y,y_i)}{\sum_i^N g(y,y_i)}. \tag{1.8}$$

## Embedding Probability Distributions into Reproducing Kernel Hilbert Spaces

An alternative representation of probability distributions, which is related to the kernel density estimate, is the non-parametric embedding of a probability distribution into a reproducing kernel Hilbert space [100]. Note that the term kernel has in this case a more restricted meaning than in the case of kernel density estimation. In this case, a kernel, specifically a reproducing kernel, is a symmetric, positive definite function. The embeddings of probability distributions into reproducing kernel Hilbert spaces and a framework for inference allow us to deduct conclusions using sample-based representations of arbitrary distributions.

Intuitively, a Hilbert space is an extension of the well known two- or three-dimensional Euclidean space to high-dimensional or even infinite dimensional vector spaces. An example for such infinite dimensional Hilbert spaces is the space of functions, i.e., infinite dimensional vectors that contain for each element of the domain the corresponding function value in the image. In addition, a Hilbert space has an inner product that allows to measure distances and angles between its elements. For a reproducing kernel Hilbert space $\mathcal{H}_k$, this inner product $\langle \cdot, \cdot \rangle$ is implicitly defined by a reproducing kernel, with $k : \Omega \times \Omega \to \mathbb{R}$ and $k(\boldsymbol{x}, \boldsymbol{y}) =: \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle$, where $\varphi(\boldsymbol{x})$ is a feature mapping into a possibly infinite dimensional space, intrinsic to the kernel function. An example for such a kernel function with intrinsic, infinite dimensional feature vectors is the Gaussian kernel. Due to the reproducing property of the kernel, all elements $f$ of the RKHS can be reproduced by $k$ in the sense that the outcome $f(x)$ of the function for a specific value $x$ can obtained by an evaluation of the kernel function [3], i.e., $f(\boldsymbol{y}) = \langle f, \boldsymbol{\phi}(\boldsymbol{y}) \rangle$ for any $f \in \mathcal{H}_k$. Based on the representer theorem [87] and the reproducing property, the elements $f$ of an RKHS $\mathcal{H}_k$ can then be written as

$$f(\cdot)=\sum_{i=1}^{n}\boldsymbol{\alpha}_i k(\boldsymbol{x}_i,\cdot)=\sum_{i=1}^{n}\boldsymbol{\alpha}_i\langle\varphi(\boldsymbol{x}_i),\varphi(\cdot)\rangle=\boldsymbol{\alpha}^{\mathsf{T}}\boldsymbol{\Upsilon}_{\boldsymbol{x}}^{\mathsf{T}}\varphi(\cdot), \tag{1.9}$$

with the weights $\alpha_i \in \mathbb{R}$ and where $\boldsymbol{\Upsilon}_{\boldsymbol{x}}=[\varphi(\boldsymbol{x}_1),\dots,\varphi(\boldsymbol{x}_n)]$ denotes the feature matrix of samples $\boldsymbol{x}_i$.

The embedding of a marginal density $p(X)$ into an RKHS is defined as the expected feature mapping

$$\mu_X := \mathbb{E}_X\left[\varphi(X)\right]=\int_{\Omega_X}\varphi(X)p(X)\mathrm{d}X, \tag{1.10}$$

also called the *mean map* [100]. Using a finite set of i.i.d. samples from $p(X)$, the mean map can be estimated as

$$\hat{\mu}_X=\frac{1}{n}\sum_{i=1}^{n}\varphi(\boldsymbol{x}_i)=\frac{1}{n}\boldsymbol{\Upsilon}_{\boldsymbol{x}}^{\mathsf{T}}\mathbf{1}_n, \tag{1.11}$$

where $\boldsymbol{\Upsilon}_x$ is the feature matrix of the samples from the distribution and $\mathbf{1}_n \in \mathbb{R}^n$ is an $n$ dimensional vector of ones. Because of the reproducing property of the kernel function, computing the expectation of a function which is an element of the same RKHS resolves to simple matrix operations. On the other hand, obtaining the probability of a single outcome or higher order statistics of the distributions is not straight forward. A joint distribution $p(X,Y)$ of the random variables $X$ and $Y$ can be embedded in a tensor product reproducing kernel Hilbert space $\mathcal{H}_k \times \mathcal{H}_k$ as the expected tensor product of the feature mappings [100] as

$$\begin{align}\mathcal{C}_{XY} &:= \mathbb{E}_{XY}\left[\varphi(X)\otimes\varphi(Y)\right]-\mu_X\otimes\mu_Y \tag{1.12}\\ &=\int_{\Omega_X}\int_{\Omega_Y}(\varphi(X)-\mu_X)\otimes(\varphi(Y)-\mu_Y)\mathrm{d}X\mathrm{d}Y, \tag{1.13}\end{align}$$

where we use $\otimes$ to denote the tensor product (or outer product) of two vectors. This embedding is also called the *centered covariance operator*. The finite sample estimator is given by

$$\hat{\mathcal{C}}_{XY}=\frac{1}{m}\sum_{i=1}^{m}\varphi(\boldsymbol{x}_i)\otimes\varphi(\boldsymbol{y}_i)-\hat{\mu}_X\otimes\hat{\mu}_Y, \tag{1.14}$$

assuming a set of $m$ i.i.d. samples $\mathcal{D}=\left\{(\boldsymbol{x}_1,\boldsymbol{y}_1),\dots,(\boldsymbol{x}_n,\boldsymbol{y}_n)\right\}$ from the joint distribution $p(X,Y)$. The embedding of a conditional distribution $P(Y|X)$ is not like the mean map a single element of the RKHS, but rather a family of embeddings where there is one embedding for each realization of the conditioning variable $X$. To obtain the conditional distribution for a specific value $X=\boldsymbol{x}_*$, Song et al. [104] define the *conditional embedding* *operator* $\mathcal{C}_{Y|X}$ which, if applied to the feature mapping of $\boldsymbol{x}_*$ returns the embedding of $P(Y|X=\boldsymbol{x}_*)$

$$\mu_{Y|\boldsymbol{x}_*} := \mathbb{E}_{Y|\boldsymbol{x}}\left[\phi(Y)\right] = \mathcal{C}_{Y|X}\varphi(\boldsymbol{x}_*). \tag{1.15}$$

Using a set of samples $\mathcal{D} = \left\{(\boldsymbol{x}_1,\boldsymbol{y}_1),\ldots,(\boldsymbol{x}_n,\boldsymbol{y}_n)\right\}$, the conditional embedding operator can be derived from a least-squares objective [33] as

$$\hat{\mathcal{C}}_{Y|X} = \boldsymbol{\Phi}(\boldsymbol{K}_{xx}+\lambda\boldsymbol{I}_n)^{-1}\boldsymbol{\Upsilon}_x^{\intercal}, \tag{1.16}$$

with the feature matrices $\boldsymbol{\Phi} := [\phi(\boldsymbol{y}_1),\ldots,\phi(\boldsymbol{y}_n)]$ and $\boldsymbol{\Upsilon}_x := [\varphi(\boldsymbol{x}_1),\ldots,\varphi(\boldsymbol{x}_n)]$, the Gram matrix $\boldsymbol{K}_{xx} = \boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Upsilon}_x \in \mathbb{R}^{n\times n}$, the regularization parameter $\lambda$, and the identity matrix $\boldsymbol{I}_n \in \mathbb{R}^{n\times n}$. With the feature mapping of the realization $\boldsymbol{x}_*$ this results in

$$\mu_{Y|\boldsymbol{x}_*} = \hat{\mathcal{C}}_{Y|X}\varphi(\boldsymbol{x}_*) = \boldsymbol{\Phi}(\boldsymbol{K}_{xx}+\lambda\boldsymbol{I}_n)^{-1}\boldsymbol{\Upsilon}_x^{\intercal}\varphi(\boldsymbol{x}_*) = \boldsymbol{\Phi}(\boldsymbol{K}_{xx}+\lambda\boldsymbol{I}_n)^{-1}\boldsymbol{k}_{\boldsymbol{x}_*}, \tag{1.17}$$

where $\boldsymbol{k}_{\boldsymbol{x}_*}$ is the kernel vector of the samples $[\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n]$ and the realization $\boldsymbol{x}_*$. As the kernel matrices in the inverse and the kernel vector of the realization are finite, the embedding of the conditional distribution can be represented as a weighted sum of feature mappings

$$\mu_{Y|\boldsymbol{x}_*} = \boldsymbol{\Phi}\boldsymbol{\alpha} = \sum_{i=1}^{n}\alpha_i\phi(\boldsymbol{y}_i), \tag{1.18}$$

with the finite weight vector $\boldsymbol{\alpha} = (\boldsymbol{K}_{xx}+\lambda\boldsymbol{I}_n)^{-1}\boldsymbol{k}_{\boldsymbol{x}_*} \in \mathbb{R}^n$.

## 1.2 State Estimation with Models Learned from Data

The framework for nonparametric inference [103, 100, 20]—which is based on the embeddings of distributions discussed above—allows to perform inference on arbitrary probability distributions. High-dimensional embeddings in reproducing kernel Hilbert spaces (RKHS) are manipulated by kernelized inference rules. The conditional embedding operator is used to derive the kernel sum rule, the kernel product rule, and the kernel Bayes' rule (KBR). However, the computational demands of the conditional embedding operator and of the KBR do not scale well with the number of samples used in their estimators. In addition, the KBR often suffers from numerical instabilities.

In the first part of this thesis, I propose two additions to the framework for nonparametric inference to address these issues. First, I present a sparsification technique for the conditional embedding operator based on the least-squares objective. This sparsification allows to learn a conditional embedding operator only on a subset of the data points while still leveraging from the full data set. I call this new approach the *subspace conditional embedding operator*. This work has been presented at the Large-Scale Kernel Learning Workshop at ICML 2015 [28]. Second, I present the kernel Kalman rule (KKR) as an alternative to the KBR. The derivation of the KKR follows from the clear objective of recursive least squares and is inspired by the well known Kalman filter. Based on the

KKR, I present the kernel Kalman filter (KKF) which uses RKHS embeddings to represent its belief state and learns the system and observation models as conditional embedding operators from data. I further derive the kernel forward backward smoother (KFBS) based on a forward and backward KKF and a smoothing update in Hilbert space. In addition, I derive the KKR, the KKF and the KFBS based on the subspace conditional embedding operator to leverage from the improved scalability with respect to the number of samples used for learning. I demonstrate on nonlinear state estimation tasks that my approaches provide a significantly improved estimation accuracy while the computational demands are considerably decreased. The work on the KKR, the KKF, and the KFBS is under review at the Machine Learning Journal [24] and has been presented at the AAAI Conference on Artificial Intelligence 2017 [29].

## 1.3 Swarm Representations for Learning Policies

In the second part of this thesis, I use the RKHS embeddings of probability distributions to represent the state of a robot swarm by its generative distribution. Swarm robotics investigates how a large population of robots with simple actuation and limited sensors can collectively solve complex tasks. One particular interesting application with robot swarms is autonomous object assembly. Such tasks have been solved successfully with robot swarms that are controlled by a human operator using a global control signal [85]. The application of such problem settings is mainly in the field of nano-robotics. Here, self-propelled, magnetotactic bacteria are used as agents [61] for the manipulation of nano-structures, for example drug containers. These bacteria have flagatella for propulsion and their orientation can be controlled by a magnetic field.

I propose a method to solve such assembly tasks autonomously based on policy search methods. Hereby, the assembly process is split into two subtasks: generating a high-level assembly plan and learning a low-level object movement policy. The high-level assembly policy plans the trajectories for each object and the low-level object movement policy controls the trajectory execution. Learning the object movement policy is challenging as it depends on the complex state of the swarm which consists of an individual state for each agent. To approach this problem, I introduce a representation of the swarm which is based on Hilbert space embeddings of distributions. The proposed representation is invariant to the number of agents in the swarm as well as to the allocation of an agent to its position in the swarm. These invariances make the learned policy robust to changes in the swarm and also reduce the search space for the policy search method significantly. I show that the resulting system is able to solve assembly tasks with varying object shapes in multiple simulation scenarios and evaluate the robustness of our representation to changes in the swarm size. Furthermore, I demonstrate that the policies learned in simulation are robust enough to be transferred to real robots. This work is under review at Advanced Robotics [22] and has been presented at the International Conference on Robotics and Automation (ICRA) 2018 [27]. It has been furthermore presented as extended abstract and as workshop contribution at the International Conference on Autonomous Agents and Multiagent Systems 2017 [26] and at the MALIC workshop at the Annual Conference on Neural Information Processing Systems 2016.

## 1.4 Learning Deep Representations for Sets of Homogeneous Inputs

Non-parametric, kernel-based methods for state estimation and prediction provide a versatile framework for inference on arbitrary shaped probability distributions. The problem of parameter learning is shifted to the problem of hyper-parameter learning with the promise of obtaining a more robust and better generalizing machine learning method. In fact, the ability of kernel methods to generalize well to unseen data trades off against the ability to model non-smooth functions.

In the recent years, neural networks have seen a revival due to the availability of massive amounts of data and the computational resources to process this data in a reasonable amount of time. Besides the applications of neural networks in supervised learning for regression and classification problems, deep learning has also found its way into the field of reinforcement learning. Using the methods of deep reinforcement learning (DRL), machines are able to learn complex control strategies directly from high dimensional observations and in large state spaces [68, 95, 58].

In the third part of my thesis, I investigate how we can employ the idea of mean embeddings in the realm of deep reinforcement learning. Specifically, I want to learn policies for object manipulation with a robot swarm. Neural networks usually have a predefined structure which requires that the number of inputs and outputs is known in advance. In the case of swarms this is a severe limitation, as we might not always have the same number of agents in the swarm. However, also in other situations we might have to deal with variable numbers of homogeneous observations, as for example point clouds. Furthermore, such data has usually no ordering (i.e., if we exchange two swarm agents, we still have semantically the same state of the swarm, if we exchange two points in a point cloud, it still represents the same 3D structure) which cannot be exploited by standard neural network architectures. I present a structure, called the deep M-embeddings which are inspired by the kernel mean embeddings and allow for a compact representation of a variable set of homogeneous inputs as a feature vector of fixed size. In experimental evaluations, I show that this representation allows to learn complex policies in a multi-agent environment outperforming a standard multi-layer perceptron both in the achieved average episode return and in sample efficiency. This work has been submitted to Swarm Intelligence [23].

## 1.5 Challenges Addressed by the Contributions of this Thesis

In this thesis, I present novel approaches for state estimation and state representation using the concept of non-parametric mean embeddings. In this section, I want to discuss the challenges addressed in this thesis and refer to the chapters of this thesis in which I present the specific approaches.

A major challenge for state representation is imposed by *high dimensional* data. Yet, such data is often located on a much lower dimensional manifold of the high dimensional space. Identifying such manifolds and the corresponding projections that map the data to a lower dimensional space is a hard task. Kernel-based methods address this problem through identifying the manifold of the data by learning the representation and models directly from samples. However, to find good kernel machines requires proper settings of the hyper-parameters. In Chapter 2, I present additions to the framework for non-parametric

| Challenge | Addressed by | Chapter |
|---|---|---|
| High dimensionality | • Kernel Kalman Rule<br>• Deep M-Embeddings | 2<br>4 |
| Partial observability | • Kernel Kalman Rule<br>• Swarm Kernel<br>• Deep M-Embeddings | 2<br>3<br>4 |
| Sample complexity | • Subspace Conditional Embedding Operator<br>• Deep M-Embeddings | 2<br>4 |
| Permutation invariance | • Swarm Kernel<br>• Deep M-Embeddings | 3<br>4 |

**Table 1.1:** Challenges addressed in this thesis, proposed approaches and references.

inference [103] that address this problem. The presented kernel Kalman rule, alleviates the process of hyper-parameter identification by being more sample efficient and thus allowing for a faster optimization than existing techniques. The deep M-embeddings which I present in Chapter 4, address the issue of high dimensional state spaces by introducing a compact representation for sets of observations that allows to learn the parameters of neural network policies more sample efficient than with a standard multi-layer perceptron.

A second challenge of state estimation is *partial observability*. Partial observability can arise from two reasons: occlusions in the sensory system or parts of the state are hard or impossible to measure. Both issues are addressed by the kernel Kalman filter presented in Chapter 2. First, the proposed kernel Kalman filter allows to learn observation models that deal with partial observations which are integrated into the belief state. Second, latent variables in the system are identified by learning a probabilistic, multi-modal dynamics model in the high-dimensional feature space of the kernel function. In Chapter 3, I present a kernel function for swarms which also addresses the problem of partial observability. By representing the swarm as a distribution, the kernel is inherently able to deal with incomplete observations where we do not observe all the agents of the swarm. In a similar fashion, the deep M-embeddings presented in Chapter 4 also allow for a variable number of inputs to a neural network and thus can be used to deal with incomplete observations.

When using kernel methods, a major downside is the poor scaling with the number of data points used for training. I propose two approaches in this thesis to address the problem of *sample complexity*. In Chapter 2, I present the subspace conditional embedding operator This addition to the framework for non-parametric inference allows to represent belief states in a much smaller subset from the whole sample set while still learning the operators from the full sample set. Based on the subspace conditional embedding operator, I further re-derive the kernelized inference rules of the framework for non-parametric inference. A second approach is presented in Chapter 4, in which I present the deep M-embeddings. Based on the deep learning techniques, I propose a method that allows to learn a compact feature representation from data which provides an improved sample complexity compared to a standard multi-layer perceptron.

When using probabilistic representations as state estimators to abstract from complex states, we often want *permutation invariance*. Given a set of observations, the order of the observations in the set is of no importance and, thus, the representation should be invariant to changes in this order. The representation for swarms which I present in Chapter 3 addresses this challenge. By representing the swarm as a distribution embedded into an RKHS, the representation is invariant to permutations of the single agents in the swarm. The deep M-embeddings address the same issue by representing sets of observations with variable size and without order as a feature vector of fixed size in a neural network structure. Table 1.1 gives an overview about the challenges addressed in this thesis, the proposed approaches and in which parts of my theses they can be found.

# 2 The Kernel Kalman Rule

In this chapter, I address the problem of state estimation in unstructured and unknown environments. Traditional state estimation methods require known models and make strong assumptions about the dynamics. Novel, versatile techniques should be able to deal with high dimensional observations and non-linear, unknown system dynamics. The recent framework for nonparametric inference [103] allows to perform inference on arbitrary probability distributions. High-dimensional embeddings of distributions into reproducing kernel Hilbert spaces (RKHS) are manipulated by kernelized inference rules, most prominently the kernel Bayes' rule (KBR). However, the computational demands of the KBR do not scale well with the number of samples.

In this chapter, I propose two techniques to increase the computational efficiency of nonparametric inference. First, I derive the kernel Kalman rule (KKR) from a recursive least squares objective and propose the KKR as an alternative to the KBR to perform Bayesian updates on mean embeddings of distributions. Based on the KKR, I present the kernel Kalman filter (KKF) that embeds the belief state into an RKHS and learns the system and observation models from data. I further derive the kernel forward backward smoother (KFBS) based on a forward and backward KKF and a smoothing update in Hilbert space. Second, I present the subspace conditional embedding operator as a sparsification technique that still leverages from the full data set. I apply this sparsification to the KKR and derive the corresponding sparse KKF and KFBS algorithms. I show on nonlinear state estimation tasks that my approaches provide a significantly improved estimation accuracy while the computational demands are considerably decreased.

## 2.1 Introduction

The ability to reason about past, current and future states in continuous, partially observable stochastic processes is a fundamental stepstone towards fully autonomos and intelligent systems. Such models are required in many applications as for example state estimation in case of incomplete sensory data, smoothing noisy data from mediocre sensors, or predicting future states from past and current observations.

Traditional state estimation techniques usually require analytical models of the underlying system, are often limited to a set of models with a special structure, and require knowledge about the moments of the stochstic processes. When assuming linear Gaussian models with known mean and covariance for instance, the Kalman filter [49] yields the optimal solution. However, the required linear Gaussian models with known statistics impose a strong limitation to the applicability of this method. For more complex processes, approximate solutions have to be used instead. Examples are the extended Kalman filter [66, 98] or the unscented Kalman filter [47, 113]. These solution inherit the Gaussian representation of the belief state to which they apply the non-linear system dynamics. However, the Gaussian distribution with its unimodal nature is a strong assumption about the belief state which leads to poor results for systems that require a more complex distribution over possible states. Moreover, both the Kalman filter but also it's extensions to non-linear systems, require that the dynamics of the systems are given as analytical models. Yet, these analytical models are often hard to obtain or make simplifying assumptions about the system.

The recently introduced framework for nonparametric inference [103, 20] alleviates the problems of traditional state estimation methods for nonlinear systems. The basic idea of these methods is to embed the probability distributions into reproducing kernel Hilbert spaces (RKHS). These embeddings allow the representation of arbitrary probability distributions using empirical estimators. Inference on the embedded distribution can then be performed efficiently and entirely in the RKHS using the kernelized versions of the sum rule, the chain rule, and the Bayes' rule. Additionally, Song, Fukumizu, and Gretton [103] use the kernel sum rule and the kernel Bayes' rule to construct the kernel Bayes' filter (KBF). The KBF learns the transition and observation models from observed samples and can be applied to nonlinear systems with high-dimensional observations. However, the computational complexity of the KBR update does not scale well with the number of samples such that hyper-parameter optimization becomes prohibitively expensive. Moreover, the KBR requires mathematical tricks that may cause numerical instabilities and also render the objective that is optimized by the KBR unclear.

In this paper, we present two approaches to overcome the limitations named above. First, we introduce the *subspace conditional embedding operator*. In contrast to the conditional embedding operator [104], this operator allows to estimate its empirical estimator with a much larger data set while maintaining computational efficiency. We further apply the subspace conditional embedding operator to the kernel sum rule, kernel chain rule and kernel Bayes rule to derive their subspace versions. We have presented these results at the large-scale kernel learning workshop at ICML 2015 [28].

Furthermore, we present the *kernel Kalman rule* (KKR) as an approximate alternative to the kernel Bayes' rule. Our derivations closely follow the derivations of the innovation update used in the Kalman filter and are based on a recursive least squares minimization objective in a reproducing kernel Hilbert space (RKHS). The KKR does not perform an exact Bayesian update as it uses a regularization term in the least squares objective and assumes constant noise on the conditioning variable. While the update equations are formulated in a potentially infinite dimensional RKHS, we derive through application of the kernel trick and by virtue of the representer theorem an algorithm that uses only operations of finite kernel matrices and vectors. We employ the kernel Kalman rule together with the kernel sum rule for filtering, which results in the *kernel Kalman filter* (KKF). In contrast to filtering techniques that rely on the KBR, the KKF allows to precompute expensive matrix inversions which significantly reduces the computational complexity and which also allows us to apply hyper-parameter optimization for the KKF. This work has been presented at AAAI 2017 [29].

In addition to the KKF, we introduce the *kernel forward backward smoother* (KFBS) which computes the embedding of the belief state given all available observations from the past and the future. The kernel forward backward smoother combines the belief state embeddings of a forward pass and a backward pass into smoothed embeddings using Hilbert space operations. Both, the forward and the backward pass are realized by a KKF, where the backward KKF operates backwards in time starting at the last observation. To scale gracefully with larger data sets, we rederive the KKR, the KKF and the KFBS with the subspace conditional operator [28].

We compare our approach to different versions of the KBR and demonstrate its improved estimation accuracy and computational efficiency. Furthermore, we evaluate the KKR on a simulated 4-link pendulum task, on a human motion capture data set [117] and on data from a table-tennis setup [30].

### 2.1.1 Related Work

To the best of our knowledge the kernel Bayes' rule exists in three different versions. It was first introduced in its original version by Fukumizu, Song, and Gretton [20]. Here, the KBR is derived, similar to the conditional operator, using prior modified covariance operators. These prior-modified covariance operators are approximated by weighting the feature mappings with the weights of the embedding of the prior distribution. Since these weights are potentially negative, the covariance operator might become indefinite, and thus, rendering its inversion impossible. To overcome this drawback, the authors have to apply a form of the Tikhonov regularization that decreases accuracy and increases the computational costs. A second version of the KBR was introduced by Song, Fukumizu, and Gretton [103] in which they use a different approach to approximate the prior-modified covariance operator. In the experiments conducted for this paper, this second version often leads to more stable algorithms than the first version. Boots, Gretton, and Gordon [8] introduced a third version of the KBR where they apply only the simple form of the Tikhonov regularization. However, this rule requires the inversion of a matrix that is often indefinite, and therefore, high regularization constants are required, which again degrades the performance. In our experiments, we refer to these different versions with KBR(b) for the first, KBR(a) for the second (order adapted from the literature), and KBR(c) for the third version. [103] propose in their framework for nonparametric inference to combine the KBR with the kernel sum rule to obtain the kernel Bayes filter (KBF). The kernel Kalman filter presented in this work is closely related to this, as we simply replace the KBR with the KKR. We compare to the KBF in our experiments. [70] recently proposed the nonparametric kernel Bayes smoother. This approach builds on top of the kernel Bayes filter, which is used to compute the estimates of a normal forward pass. The smoothing update is then obtained by propagating the embeddings backwards in time without performing a second filtering pass.

For filtering tasks with known linear system equations and Gaussian noise, the Kalman filter (KF) yields the solution that minimizes the squared error of the estimate to the true state. Two widely known and applied approaches to extend the Kalman filter to non-linear systems are the *extended Kalman filter* (EKF) [66, 98] and the *unscented Kalman filter* (UKF) [113, 47]. Both, the EKF and the UKF, assume that the non-linear system dynamics are known and use them to update the prediction mean. Yet, updating the prediction covariance is not straightforward. In the EKF the system dynamics are linearized at the current estimate of the state, and in the UKF the covariance is updated by applying the system dynamics to a set of sample-points (sigma points). While these approximations make the computations tractable, they can significantly reduce the quality of the state estimation, in particular for high-dimensional systems.

Hsu, Kakade, and Zhang [42] recently proposed an algorithm for learning Hidden Markov Models (HMMs) by exploiting the spectral properties of observable measures to derive an observable representation of the HMM [46]. An RKHS embedded version thereof was presented by [102]. While this method is applicable for continuous state spaces, it still assumes a finite number of discrete hidden states.

Other closely related algorithms to our approach are the kernelized version of the Kalman filter and Kalman smoother by Ralaivola and d'Alche-Buc [81] and the kernel Kalman filter based on the conditional embedding operator (KKF-CEO) by Zhu, Chen, and Principe [120]. The former approach formulates the Kalman filter in a sub-space of the infinite feature space that is defined by the kernels. Hence, this approach does not fully leverage the kernel idea of using an infinite feature space. In contrast, the KKF-CEO approach embeds the belief state also in an RKHS. However, they require that the observation is a noisy version of the full state of the system, and thus, they cannot handle partial observations. Moreover, they also deviate from the standard derivation of the Kalman filter, which—as our experiments show—decreases the estimation accuracy. The full observability assumption is needed in order to implement a simplified version of the innovation update of the Kalman filter in the RKHS. The KKF does not suffer from this restriction. It also provides update equations that are much closer to the original Kalman filter and outperforms the KKF-CEO algorithm as shown in our experiments. Another approach to state estimation is presented in [50], where the authors propose to estimate low-dimensional state vectors based on kernel canonical correlation analysis and then regress a linear transition model of the estimated state vectors and the nonlinear features of the input.

Learning predictors in the space of predictive state representations to perform filtering has been proposed in [107] and later extended to smoothing in [106]. They introduce predictive state inference machines (PSIM) which are (nonlinear) regressors on predictive states learned from data to perform filtering. With the smoothing machine (SMACH) they extend this concept for smoothing.

## 2.2 Preliminaries

Our work is based on the recent formulations of embedding distributions into reproducing kernel Hilbert spaces [100, 103]. These embeddings allow to represent arbitrary probability distributions non-parametrically by a potentially infinite dimensional feature vector. Through the application of derived operators [104, 19] it is furthermore possible to apply inference rules entirely in the Hilbert space. In the first part of this section, we want to give the reader an introduction into this technology and define the notation we will use throughout this article.

One of the main contributions of our paper is a novel method for performing approximate Bayesian updates on a distribution embedded into an RKHS. The derivations of this update rule are based on a least-squares objective and inspired by the derivations of the Kalman filter update, thus we name this method *kernel Kalman rule*. In the second part of this section, we will recapitulate the classical Kalman filter equations and review the derivations of the innovation update based on the least-squares objective.

### 2.2.1 Nonparametric Inference with Hilbert Space Embeddings of Distributions

Intuitively, a Hilbert space is an extension of the well known two- or three-dimensional Euclidean vector space to arbitrary many dimensions, specifically including infinite dimensional vector spaces. Such infinite dimensional Hilbert spaces include spaces where the single elements are functions, i.e., infinite dimensional vectors that contain for each element of the domain the corresponding function value in the image. In addition, a Hilbert space has an inner product that allows to measure distances and angles between its elements. For a reproducing kernel Hilbert space $\mathcal{H}_k$, this inner product $\langle\cdot,\cdot\rangle$ is implicitly defined by a reproducing kernel $k(\boldsymbol{x},\boldsymbol{x}') = \langle\varphi(\boldsymbol{x}),\varphi(\boldsymbol{x}')\rangle$, where $\varphi(\boldsymbol{x})$ is a feature mapping into a possibly infinite dimensional space, intrinsic to the kernel function. For example the Gaussian kernel computes the inner product of the feature mappings of its inputs where the feature mappings itself cannot be written down explicitly as they are into an infinite dimensional space. Due to the reproducing property of the kernel, all elements $f$ of the RKHS can be reproduced by $k$ in the sense that the outcome $f(x)$ of the function for a specific value $x$ can obtained by an evaluation of the kernel function [3], i.e., $f(\boldsymbol{x}) = \langle f,\varphi(\boldsymbol{x})\rangle$ for any $f \in \mathcal{H}_k$.

In a practical setting, we want to embed probability distributions in an RKHS spanned by samples $\mathcal{D}_X = \{\boldsymbol{x}_1,\dots,\boldsymbol{x}_n\}$. Based on the representer theorem [87] and the reproducing property, the elements $f$ of an RKHS $\mathcal{H}_k$ can then be written as

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i,\cdot) = \sum_{i=1}^{n} \alpha_i \langle\varphi(\boldsymbol{x}_i),\varphi(\cdot)\rangle = \boldsymbol{\alpha}^\intercal \boldsymbol{\Upsilon}_x^\intercal \varphi(\cdot), \tag{2.1}$$

with the weights $\alpha_i \in \mathbb{R}$ and where we denote the feature matrix of samples $\boldsymbol{x}_i$ by $\boldsymbol{\Upsilon}_{\boldsymbol{x}} = [\varphi(\boldsymbol{x}_1),\dots,\varphi(\boldsymbol{x}_n)]$. In the following paragraphs we will show how probability distributions can be represented as an embedding in such a reproducing kernel Hilbert space and how the operators for performing inference in the RKHS can be derived.

#### Embeddings of Marginal and Joint Distributions

The embedding of a marginal density $P(X)$ over the random variable $X$ is defined as the expected feature mapping $\mu_X := \mathbb{E}_X[\varphi(X)]$, also called the *mean map* [100]. Using a finite set of samples $\{\boldsymbol{x}_1,\dots,\boldsymbol{x}_n\}$ from $P(X)$, the mean map can be estimated as

$$\hat{\mu}_X = \frac{1}{n}\sum_{i=1}^{n} \varphi(\boldsymbol{x}_i) = \frac{1}{n}\boldsymbol{\Upsilon}_x^\intercal \mathbf{1}_n, \tag{2.2}$$

where $\mathbf{1}_n \in \mathbb{R}^n$ is an $n$ dimensional vector of ones. Because of the reproducing property of the kernel function, computing the expectation of a function which is an element of the same RKHS resolves to simple matrix operations. On the other hand, the probability of a single outcome or higher order statistics of the distributions are not straight forward to obtain.

Alternatively, a distribution can be embedded in a tensor product RKHS $\mathcal{H}_k \times \mathcal{H}_k$ as the expected tensor product of the feature mappings [100]

$$\mathcal{C}_{XX} := \mathbb{E}_{XX}\left[\varphi(X) \otimes \varphi(X)\right] - \mu_X \otimes \mu_X, \tag{2.3}$$

where we use $\otimes$ to denote the tensor product (or outer product) of two vectors. This embedding is also called the *centered covariance operator*. The finite sample estimator is given by

$$\hat{\mathcal{C}}_{XX} = \frac{1}{m}\sum_{i=1}^{m} \varphi(\boldsymbol{x}_i) \otimes \varphi(\boldsymbol{x}_i) - \hat{\mu}_X \otimes \hat{\mu}_X. \tag{2.4}$$

Similarly, we can define the *uncentered cross-covariance operator* for a joint distribution $p(X,Y)$ of two variables $X$ and $Y$ as $\hat{\mathcal{C}}_{XY} = \frac{1}{m}\sum_{i=1}^{m} \varphi(\boldsymbol{x}_i) \otimes \phi(\boldsymbol{y}_i)$. Here, we have used a data set of tuples $\mathcal{D}_{XY} = \left\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\right\}$ sampled from $p(X,Y)$ and a second RKHS $\mathcal{H}_g$ with kernel function $g(\boldsymbol{y}, \boldsymbol{y}') =: \langle \phi(\boldsymbol{y}), \phi(\boldsymbol{y}')\rangle$.

## The Conditional Embedding Operator

The embedding of a conditional distribution $P(Y|X)$ is not like the mean map a single element of the RKHS, but rather a family of embeddings that yields a mean embedding for each realization of the conditioning variable $X$. To obtain the conditional distribution for a specific value $X = \boldsymbol{x}_*$, Song et al. [104] defined the *conditional embedding operator* $\mathcal{C}_{Y|X}$ which, if applied to the feature mapping of $\boldsymbol{x}$, returns the embedding of $P(Y|X = \boldsymbol{x}_*)$

$$\mu_{Y|\boldsymbol{x}} := \mathbb{E}_{Y|\boldsymbol{x}}\left[\phi(Y)\right] = \mathcal{C}_{Y|X}\varphi(\boldsymbol{x}). \tag{2.5}$$

Using the data set $\mathcal{D}_{XY}$ from the joint distribution, an estimator of the conditional embedding operator can be derived from a least-squares objective [33] as

$$\hat{\mathcal{C}}_{Y|X} = \boldsymbol{\Phi}(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1}\boldsymbol{\Upsilon}_x^{\intercal}, \tag{2.6}$$

with the feature matrices $\boldsymbol{\Phi} := [\phi(\boldsymbol{y}_1), \dots, \phi(\boldsymbol{y}_n)]$ and $\boldsymbol{\Upsilon}_x := [\varphi(\boldsymbol{x}_1), \dots, \varphi(\boldsymbol{x}_n)]$, the Gram matrix $\boldsymbol{K}_{xx} = \boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Upsilon}_x \in \mathbb{R}^{n\times n}$, the regularization parameter $\lambda$, and the identity matrix $\boldsymbol{I}_n \in \mathbb{R}^{n\times n}$. With the feature mapping of the realization $\boldsymbol{x}_*$ this results in

$$\hat{\mu}_{Y|\boldsymbol{x}_*} = \hat{\mathcal{C}}_{Y|X}\varphi(\boldsymbol{x}_*) = \boldsymbol{\Phi}(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1}\boldsymbol{\Upsilon}_x^{\intercal}\varphi(\boldsymbol{x}_*) = \boldsymbol{\Phi}(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1}\boldsymbol{k}_{\boldsymbol{x}_*}, \tag{2.7}$$

where $\boldsymbol{k}_{\boldsymbol{x}_*} = [k(\boldsymbol{x}_1, \boldsymbol{x}_*), \dots, k(\boldsymbol{x}_n, \boldsymbol{x}_*)]^{\intercal}$ is the kernel vector of the samples $\boldsymbol{x}_i$ and the realization $\boldsymbol{x}_*$. As the kernel matrices in the inverse and the kernel vector of the realization are finite, the embedding of the conditional distribution can be represented as a weighted sum of feature mappings

$$\hat{\mu}_{Y|\boldsymbol{x}_*} = \boldsymbol{\Phi}\boldsymbol{\alpha} = \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{y}_i), \tag{2.8}$$

with the finite weight vector $\boldsymbol{\alpha} \in \mathbb{R}^n$. Based on the two definitions for Hilbert space embeddings of probability distributions and the conditional embedding operator discussed above, all the rules of the framework for non-parametric inference [103] can be derived.

### The Kernel Sum Rule

The embedding of $Q(Y) = \sum_X P(Y|X)\pi(X)$ can be obtained from the *kernel sum rule* [103]. To that end, the conditional operator is applied to the embedding $\hat{\mu}^{\pi}_X = \boldsymbol{\Upsilon}_x \boldsymbol{\alpha}_\pi$ of the prior distribution $\pi(X)$,

$$\hat{\mu}^{\pi}_Y = \hat{\mathcal{C}}_{Y|X}\hat{\mu}^{\pi}_X = \boldsymbol{\Phi}(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1}\boldsymbol{K}_{xx}\boldsymbol{\alpha}_\pi. \tag{2.9}$$

Again, the result can be represented as a weighted sum over feature mappings. In order to obtain the distribution $Q(Y)$ as a covariance operator instead of a mean map, Song, Fukumizu, and Gretton [103] also proposed the kernel sum rule for tensor product features which yields the prior modified covariance operator $\mathcal{C}^{\pi}_{YY}$ as

$$\mathcal{C}^{\pi}_{YY} = \mathcal{C}_{(YY)|X}\mu^{\pi}_X \tag{2.10}$$

$$\hat{\mathcal{C}}^{\pi}_{YY} = \boldsymbol{\Phi}\mathrm{diag}((\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1}\boldsymbol{K}_{xx}\boldsymbol{\alpha})\boldsymbol{\Phi}^{\intercal}, \tag{2.11}$$

where $\mathcal{C}_{(YY)|X}$ is the conditional operator for tensor product features.

### The Kernel Chain Rule

The *kernel chain rule* [103] yields an embedding of the joint distribution $Q(X,Y) = P(Y|X)\pi(X)$ as a prior modified covariance operator. There are two versions of the kernel chain rule. Both apply the conditional embedding operator of $P(Y|X)$ to an embedding of the prior distribution $\pi(X)$. In the first version the conditional operator is applied to a covariance embedding of the prior distribution. This covariance operator is not estimated directly from samples but approximated from the weight vector $\boldsymbol{\alpha}_\pi$ of the embedding of the prior distributions as $\hat{\mathcal{C}}^{\pi}_{XX} = \boldsymbol{\Upsilon}_x \mathrm{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{\Upsilon}_x^{\intercal}$. This yields Version (a) of the kernel chain rule as

$$\hat{\mathcal{C}}^{\pi}_{YX} = \hat{\mathcal{C}}_{Y|X}\hat{\mathcal{C}}^{\pi}_{XX} = \boldsymbol{\Phi}(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1}\boldsymbol{K}_{xx}\mathrm{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{\Upsilon}_x^{\intercal}. \tag{2.12}$$

Version (b) of the kernel chain rule first computes the mean map $\mu^{\pi}_Y$ conditioned on the prior distribution $\pi(X)$ by applying the conditional embedding operator to the mean map $\mu^{\pi}_X$. Afterwards, the prior-modified covariance operator of the joint distribution is constructed from the resulting weight vector which results in

$$\hat{\mathcal{C}}^{\pi}_{YX} = \boldsymbol{\Phi}\mathrm{diag}((\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1}\boldsymbol{K}_{xx}\boldsymbol{\alpha}_\pi)\boldsymbol{\Upsilon}_x^{\intercal}. \tag{2.13}$$

Both versions of the kernel chain rule have been used to derive different versions of the kernel Bayes' rule as we will depict below.

Given the embedding of a prior distribution $\pi(X)$ and the feature mapping of an observation $\phi(\boldsymbol{y}_*)$, the *kernel Bayes' rule* (KBR) infers the mean embedding of the posterior distribution $Q_\pi(X|Y = \boldsymbol{y}_*)$. The idea is to construct a prior-modified conditional embedding operator that yields the mean map of the posterior if applied to the feature mapping of the observation [20]

$$\mu^\pi_{X|\boldsymbol{y}} = \mathcal{C}^\pi_{X|Y}\phi(\boldsymbol{y}_*). \tag{2.14}$$

This prior-modified conditional operator is constructed from two prior-modified covariance operators $\mathcal{C}^\pi_{YY}$ and $\mathcal{C}^\pi_{YX}$ obtained from the kernel sum and the kernel chain rule, respectively, using the relation

$$\mathcal{C}^\pi_{X|Y} = \mathcal{C}^\pi_{XY}\left(\mathcal{C}^\pi_{YY}\right)^{-1}. \tag{2.15}$$

In the first version, which we denote by KBR(b) following the notation of Song, Fukumizu, and Gretton [103], Fukumizu, Song, and Gretton [20] derived the kernel Bayes' rule using the tensor product conditional operator in the kernel chain rule (c.f. Equation 2.13) and arrived at

$$\hat{\mu}^\pi_{X|\boldsymbol{y}} = \boldsymbol{\Upsilon}_x \boldsymbol{D}\boldsymbol{G}_{yy}\left((\boldsymbol{D}\boldsymbol{G}_{yy})^2 + \kappa\boldsymbol{I}_n\right)^{-1}\boldsymbol{D}\boldsymbol{g}_{\boldsymbol{y}_*}, \tag{2.16}$$

with the diagonal $\boldsymbol{D} := \operatorname{diag}((\boldsymbol{K}_{xx} + \lambda\boldsymbol{I}_n)^{-1}\boldsymbol{K}_{xx}\boldsymbol{\alpha}_\pi)$, the gram matrix $\boldsymbol{G}_{yy} = \boldsymbol{\Phi}^\intercal\boldsymbol{\Phi}$, the kernel vector $\boldsymbol{g}_{\boldsymbol{y}_*} = [g(\boldsymbol{y}_1, \boldsymbol{y}_*), \ldots, g(\boldsymbol{y}_n, \boldsymbol{y}_*)]^\intercal$ and $\kappa$ as regularization parameter. Song, Fukumizu, and Gretton [103] derived the KBR using the first formulation of the kernel chain rule shown in Equation 2.12 which results in

$$\hat{\mu}^\pi_{X|\boldsymbol{y}} = \boldsymbol{\Upsilon}_x \boldsymbol{\Lambda}^\intercal\left((\boldsymbol{D}\boldsymbol{G}_{yy})^2 + \kappa\boldsymbol{I}_n\right)^{-1}\boldsymbol{G}_{yy}\boldsymbol{D}\boldsymbol{g}_{\boldsymbol{y}_*}, \tag{2.17}$$

with $\boldsymbol{\Lambda} := (\boldsymbol{K}_{xx} + \lambda\boldsymbol{I}_n)^{-1}\boldsymbol{K}_{xx}\operatorname{diag}(\boldsymbol{\alpha}_\pi)$. This second version of the kernel Bayes' rule is denoted by KBR(a). As the matrix $\boldsymbol{D}\boldsymbol{G}_{yy}$ is typically not invertible, both of these versions of the KBR use a form of the Tikhonov regularization in which the matrix in the inverse is squared. Boots, Gretton, and Gordon [8] use a third form of the KBR which is derived analogously to the first version but does not use the squared form of Tikhonov regularization, i.e.,

$$\hat{\mu}^\pi_{X|\boldsymbol{y}} = \boldsymbol{\Upsilon}_x\left(\boldsymbol{D}\boldsymbol{G}_{yy} + \kappa\boldsymbol{I}_n\right)^{-1}\boldsymbol{D}\boldsymbol{g}_{\boldsymbol{y}_*}. \tag{2.18}$$

Since the product $\boldsymbol{D}\boldsymbol{G}_{yy}$ is often not positive definite, a strong regularization parameter is required to make the matrix invertible. We denote this third version of the kernel Bayes' rule consequently by KBR(c).

### 2.2.2 The Kalman Filter

The Kalman filter is a well known technique for state estimation, prediction, and smoothing in environments with linear system dynamics that are subject to zero-mean Gaussian noise with known covariances [49]. The system equations can be formulated as

$$\boldsymbol{x}_{t+1} = \boldsymbol{F}\boldsymbol{x}_t + v_t, \quad \boldsymbol{y}_t = \boldsymbol{H}\boldsymbol{x}_t + \boldsymbol{w}_t, \tag{2.19}$$

where $\boldsymbol{x}_t$ is the latent state of the system at time $t$ and $\boldsymbol{y}_t$ is the corresponding observation. The linear Gaussian model is defined by the system matrix $\boldsymbol{F}$, the observation matrix $\boldsymbol{H}$, and noise vectors $v_t$ and $\boldsymbol{w}_t$ which are sampled from $\mathcal{N}(\mathbf{0}, \boldsymbol{P})$ and $\mathcal{N}(\mathbf{0}, \boldsymbol{R})$, respectively.

From the assumption of Gaussian transition noise and Gaussian observation noise, it follows that the belief state over the latent state $\boldsymbol{x}_t$ is as well a Gaussian distribution with mean $\boldsymbol{\eta}_{\boldsymbol{x},t}$ and covariance $\boldsymbol{\Sigma}_{\boldsymbol{x},t}$. The Kalman filter applies iteratively two update procedures to the belief state to which we will refer to as *prediction* and *innovation* update. During the prediction update the Kalman filter propagates the belief state in time by applying the transition model, i.e.,

$$\boldsymbol{\eta}^-_{\boldsymbol{x},t+1} = \boldsymbol{F}\boldsymbol{\eta}^+_{\boldsymbol{x},t}, \quad \boldsymbol{\Sigma}^-_{\boldsymbol{x},t+1} = \boldsymbol{F}\boldsymbol{\Sigma}^+_{\boldsymbol{x},t}\boldsymbol{F}^\intercal + \boldsymbol{P}. \tag{2.20}$$

On new observations $\boldsymbol{y}_t$, the innovation update applies Bayes' theorem to the *a-priori* belief state $\{\boldsymbol{\eta}^-_{\boldsymbol{x},t}, \boldsymbol{\Sigma}^-_{\boldsymbol{x},t}\}$ to obtain the *a-posteriori* mean and covariance as

$$\boldsymbol{\eta}^+_{\boldsymbol{x},t} = \boldsymbol{\eta}^-_{\boldsymbol{x},t} + \boldsymbol{Q}_t(\boldsymbol{y}_t - \boldsymbol{H}\boldsymbol{\eta}^-_{\boldsymbol{x},t}), \tag{2.21}$$

$$\boldsymbol{\Sigma}^+_{\boldsymbol{x},t} = \boldsymbol{\Sigma}^-_{\boldsymbol{x},t} - \boldsymbol{Q}_t\boldsymbol{H}\boldsymbol{\Sigma}^-_{\boldsymbol{x},t}, \tag{2.22}$$

with the Kalman gain matrix

$$\boldsymbol{Q}_t = \boldsymbol{\Sigma}^-_{\boldsymbol{x},t}\boldsymbol{H}^\intercal(\boldsymbol{H}\boldsymbol{\Sigma}^-_{\boldsymbol{x},t}\boldsymbol{H}^\intercal + \boldsymbol{R})^{-1}.$$

Another approach to derive the Kalman filter equations follows from a least-squares objective between the state estimator a-priori and a-posteriori to the observation [97]. This second approach does not make the explicit assumption that the belief state can be represented as a Gaussian random variable. Rather this representation follows from the objective to minimize the variance of the error between the a-priori and a-posteriori estimators. We will take this second approach as inspiration to derive the kernel Kalman rule in Section 2.4.

## 2.3 Efficient Nonparametric Inference in a Subspace

A general drawback of kernel methods is that the complexities of the algorithms scale poorly with the number of samples in the kernel matrices. As the conditional embedding operator and the kernel inference rules require the inversion of a kernel matrix, the complexity scales cubically with the number of data points. To overcome this drawback, several approaches exist that aim to find a good trade-off between a compact representation and leveraging from a large data set. Examples are the sparse Gaussian processes that use pseudo-inputs [101, 13], or a sparse subset of the data which is selected by maximizing the posterior probability [99]. Other techniques are based on approximating the kernel matrices using the Nyström method [115] or random Fourier features [80]. We approach this problem by proposing the subspace conditional embedding operators [28]. The basic idea is to use only a subset of the available training data as representation for the embeddings but the full data set to learn the conditional operators. In the following sections, we will recapitulate this approach and show how it can be applied to the framework for nonparamteric inference.

Given the feature matrices $\boldsymbol{\Phi} := [\phi(\boldsymbol{y}_1), \dots, \phi(\boldsymbol{y}_n)]$ and $\boldsymbol{\Upsilon}_x := [\varphi(\boldsymbol{x}_1), \dots, \varphi(\boldsymbol{x}_n)]$, we can define the respective subsets $\boldsymbol{\Psi} \subset \boldsymbol{\Phi}$ and $\boldsymbol{\Gamma} \subset \boldsymbol{\Upsilon}_x$, where $|\boldsymbol{\Psi}| = |\boldsymbol{\Gamma}| = m \ll n$. We assume that the subsets are representative for the embedded distributions. Similar to the conditional operator discussed in Section 2.2.1, we define the subspace conditional operator $\mathcal{C}^S_{Y|X}$ as the mapping from an embedding $\varphi(\boldsymbol{x}) \in \mathcal{H}_k$ to the mean embedding $\mu_{Y|\boldsymbol{x}} \in \mathcal{H}_g$ of the conditional distribution $P(Y|\boldsymbol{x})$ conditioned on the variate $\boldsymbol{x}$. To obtain this subspace conditional operator, we first introduce an auxiliary conditional operator $\mathcal{C}^{\text{aux}}_{Y|X}$ which maps from the subspace projection of the embedding $\boldsymbol{\Gamma}^\intercal \varphi(\boldsymbol{x})$ to the mean map of the conditional distribution, i.e.,

$$\hat{\mu}_{Y|\boldsymbol{x}} = \hat{\mathcal{C}}^{\text{aux}}_{Y|X} \boldsymbol{\Gamma}^\intercal \varphi(\boldsymbol{x}). \tag{2.23}$$

We can then derive this auxiliary conditional operator by minimizing the squared error on the full data set

$$\begin{aligned} \hat{\mathcal{C}}^{\text{aux}}_{Y|X} &= \arg\min_{\mathcal{C}} \left\| \boldsymbol{\Phi} - \mathcal{C} \boldsymbol{\Gamma}^\intercal \boldsymbol{\Upsilon}_x \right\|_2 && (2.24) \\ &= \boldsymbol{\Phi} \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Gamma} \left( \boldsymbol{\Gamma}^\intercal \boldsymbol{\Upsilon}_x \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Gamma} + \lambda \boldsymbol{I}_m \right)^{-1}, && (2.25) \end{aligned}$$

with the identity $\boldsymbol{I}_m \in \mathbb{R}^{m \times m}$. Substituting this result for the auxiliary conditional operator in Equation 2.23 gives us the subspace conditional operator as

$$\begin{aligned} \hat{\mathcal{C}}^S_{Y|X} &= \hat{\mathcal{C}}^{\text{aux}}_{Y|X} \boldsymbol{\Gamma}^\intercal \\ &= \boldsymbol{\Phi} \boldsymbol{K}_{x\bar{x}} \left( \boldsymbol{K}^\intercal_{x\bar{x}} \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I}_m) \right)^{-1} \boldsymbol{\Gamma}^\intercal, && (2.26) \end{aligned}$$

where $\boldsymbol{K}_{x\bar{x}} = \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Gamma} \in \mathbb{R}^{n \times m}$ is the kernel matrix of the sample feature set $\boldsymbol{\Upsilon}_x$ and the subset $\boldsymbol{\Gamma}$. Since we assume that $m \ll n$, the inverse in the subspace conditional operator is in $\mathbb{R}^{m \times m}$ and, thus, of a much smaller size than the inverse in the standard conditional operator shown in Equation 2.6. Additionally, we can use the feature matrix $\boldsymbol{\Gamma}^\intercal$ on the right hand side in Equation 2.26 to represent the mean embedding always in the subspace spanned by the features $\boldsymbol{\Gamma}$. This allows to avoid representations and computations in the high-dimensional space spanned by the features of the full sample set. Before we will rederive the non-parametric inference rules analogously to [103] but based on the subspace conditional embedding operator, we will discuss the selection of the samples for spanning the subspace and the relation of the subspace conditional operator to other sparsification approaches in the next sections.

### 2.3.1 Selecting the Sample Set to Span the Subspace

To learn the subspace conditional embedding operator, we need to choose $m$ points for the representation of the embedding from a data set of $n$ data points, where $m \ll n$. The selection of these data points is a crucial step as we want a subset that is descriptive enough to represent the belief state well. We propose two approaches to address this problem which aim at different characteristics of the subset.

The first approach simply samples uniformly without replacement from the full data set. The result is a subset that resembles statistically the full data set, i.e., regions that have a high density in the full data set will have a high density in the subset and vice versa.

The goal of the second approach is to get a subset with an optimal coverage of the sample space. We select the first sample randomly from the full data set into the subset. Afterwards, we iteratively extend the subset by adding samples according to the following criterion: we compute the maximum kernel activation for each sample in the full data set with the samples in the current subset, then we extend the current subset by taking the sample from the full data set with the minimal maximum activation. We call this second strategy the *kernel activation heuristic*.

### 2.3.2 Relation to Other Sparsification Approaches

Many other sparsification techniques for kernel methods exist. The two most important techniques among these are probably the Nyström method [115, 16] and the random Fourier features [80]. Both methods are closely related to our approach.

[115] approximate the Gram matrix $\boldsymbol{K}$ based on the eigendecomposition $\boldsymbol{K} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\intercal$, where $\boldsymbol{U}$ are the eigenvectors and $\boldsymbol{\Lambda}$ is a diagonal of the eigenvalues $\lambda_1, \ldots, \lambda_n$. By taking only the first $m < n$ eigenvectors as $\boldsymbol{U}^{(m)}$ and the first $m$ eigenvalues as diagonal $\boldsymbol{\Lambda}^{(m)}$, $\boldsymbol{K}$ can be approximated as $\boldsymbol{K} \approx \boldsymbol{U}^{(m)}\boldsymbol{\Lambda}^{(m)}(U^{(m)})^\intercal$. However, since the eigendecomposition is computationally costly and more efficient methods only significantly decrease the running time for $m \ll n$, [115] propose to use instead the Nyström approximation of the eigenvectors which can be computed in only $O(m^2 n)$ instead of $O(n^3)$ for the true eigenvectors. The resulting approximation of the Gram matrix $\boldsymbol{K}$ has the form $\tilde{\boldsymbol{K}} = \boldsymbol{K}_{n,m}\boldsymbol{K}_{m,m}^{-1}\boldsymbol{K}_{m,n}$. Using the Nyström method to approximate the conditional embedding operator would result in the following equations

$$\begin{aligned} \hat{\mathcal{C}}_{Y|X}\hat{\mu}_X &= \boldsymbol{\Phi}(\boldsymbol{K}_{n,m}\boldsymbol{K}_{m,m}^{-1}\boldsymbol{K}_{m,n} + \lambda\boldsymbol{I}_n)^{-1}\boldsymbol{K}_{n,m}\boldsymbol{K}_{m,m}^{-1}\boldsymbol{K}_{m,X} && (2.27)\\ &= \boldsymbol{\Phi}\boldsymbol{K}_{n,m}\boldsymbol{K}_{m,m}^{-1}(\boldsymbol{K}_{m,n}\boldsymbol{K}_{n,m}\boldsymbol{K}_{m,m}^{-1} + \lambda\boldsymbol{I}_m)^{-1}\boldsymbol{K}_{m,X}, && (2.28) \end{aligned}$$

where $\boldsymbol{K}_{m,X}$ is the approximator of the kernel mean embedding $\mu_X$ using the $m$ samples of the Nyström approximation. Thus, if we would assume $\boldsymbol{K}_{m,m} = \boldsymbol{I}$, the subspace conditional operator is equivalent to the Nyström approximation. This assumption requires that features of the selected data points are orthogonal, i.e., $\phi(\boldsymbol{x}_i)^\intercal\phi(\boldsymbol{x}_j) = \delta_{ij}$. Note that the kernel activation heuristic presented in the previous sections selects the data points by minimizing this inner product for all points that are already in the subset.

The idea of the random Fourier features [80] is to compute the Fourier transform $p$ of the kernel method $k$. Random samples are drawn from the distribution over frequencies $p$ which are then used to construct a feature function $\boldsymbol{z}(\boldsymbol{x})$. Rather than using a projection of the high dimensional feater as in our approach, the random features directly transform the samples into a finite dimensional feature space whose inner product approximates the kernel function, i.e., $\boldsymbol{z}(\boldsymbol{x})^\intercal\boldsymbol{z}(\boldsymbol{y}) \approx k(\boldsymbol{x},\boldsymbol{y})$. Let $\boldsymbol{Z} \in \mathbb{R}^{n\times m}$ be the feature matrix of the $n$ data points with $m$ random features. We could approximate the conditional embedding operator as

$$\begin{aligned}\hat{\mathcal{C}}_{Y|X}\hat{\mu}_X &= \boldsymbol{\Phi}(\boldsymbol{Z}\boldsymbol{Z}^\intercal + \lambda\boldsymbol{I}_n)^{-1}\boldsymbol{Z}\boldsymbol{Z}^\intercal\boldsymbol{m}_X && (2.29)\\ &= \boldsymbol{\Phi}\boldsymbol{Z}(\boldsymbol{Z}^\intercal\boldsymbol{Z} + \lambda\boldsymbol{I}_m)^{-1}\boldsymbol{Z}^\intercal\boldsymbol{m}_X. && (2.30)\end{aligned}$$

Again, it is easy to observe the similarity to our approach if we replace $\boldsymbol{Z}$ by $\boldsymbol{K}_{x\bar{x}}$. Note that this representation, in contrast to our approach and to the Nyström method, does not allow to derive an operator in the reproducing kernel Hilbert space but directly uses finite vector/matrix representations.

### 2.3.3 The Subspace Kernel Sum Rule

Analogously to [103], the subspace kernel sum rule is the application of the subspace conditional operator to the embedding of a distribution $\pi(X)$, i.e.,

$$\hat{\mu}_Y^\pi = \hat{\mathcal{C}}^S_{Y|X}\hat{\mu}_X^\pi = \boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}\left(\boldsymbol{K}_{x\bar{x}}^\intercal\boldsymbol{K}_{x\bar{x}} + \lambda\boldsymbol{I}_m\right)^{-1}\boldsymbol{\Gamma}^\intercal\boldsymbol{\Upsilon}_x\boldsymbol{\alpha}_\pi, \tag{2.31}$$

where $\mu_X^\pi = \boldsymbol{\Upsilon}_x\boldsymbol{\alpha}_\pi$ is the embedding of the prior distribution $\pi(X)$. We construct the subspace kernel sum rule for tensor product features differently than Song, Fukumizu, and Gretton [103]. Instead of applying the conditional operator to the mean embedding and then approximating the covariance operator with the resulting weights (c.f. Equation 2.11), we first approximate the covariance operator $\mathcal{C}_{XX}^\pi$ from the weights $\boldsymbol{\alpha}_\pi$ and then apply the subspace conditional operator to both sides, i.e.,

$$\begin{aligned}\hat{\mathcal{C}}_{YY}^{S,\pi} = \hat{\mathcal{C}}^S_{Y|X}\hat{\mathcal{C}}^\pi_{XX}\left(\hat{\mathcal{C}}^S_{Y|X}\right)^\intercal &= \hat{\mathcal{C}}^S_{Y|X}\boldsymbol{\Upsilon}_x\operatorname{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{\Upsilon}_x^\intercal\left(\hat{\mathcal{C}}^S_{Y|X}\right)^\intercal\\ &= \boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}\boldsymbol{L}\operatorname{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{L}^\intercal\boldsymbol{K}_{x\bar{x}}^\intercal\boldsymbol{\Phi}^\intercal. && (2.32)\end{aligned}$$

Here, we denote $\boldsymbol{L} := \left(\boldsymbol{K}_{x\bar{x}}^\intercal\boldsymbol{K}_{x\bar{x}} + \lambda\boldsymbol{I}_m\right)^{-1}\boldsymbol{K}_{x\bar{x}}^\intercal \in \mathbb{R}^{m\times n}$ to keep the notation uncluttered. This definition of the subspace kernel sum rule follows from the kernel chain rule for tensor product features, where the conditional operator $\mathcal{C}_{Y|X}$ is applied to the covariance embedding $\mathcal{C}_{XX}^\pi$ to obtain the covariance embedding $\mathcal{C}_{YX}^\pi$ (c.f. Equation 2.12). The subspace kernel sum rule follows from applying the transpose of the condition operator a second time from the right-hand side.

### 2.3.4 The Subspace Kernel Chain Rule

The subspace kernel chain rule is a straight forward modification of the kernel chain rule by [103]. We simply apply the subspace conditional operator $\mathcal{C}^{S}_{Y|X}$ from the left side to a covariance operator $\mathcal{C}^{\pi}_{XX} = \boldsymbol{\Upsilon}_x \operatorname{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{\Upsilon}_x^\mathsf{T}$ approximated from the weights $\boldsymbol{\alpha}_\pi$ of the prior mean map $\mu^{\pi}_{X}$

$$\hat{\mathcal{C}}^{S,\pi}_{YX} = \hat{\mathcal{C}}^{S}_{Y|X}\hat{\mathcal{C}}^{\pi}_{XX} = \boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}\boldsymbol{L}\operatorname{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{\Upsilon}_x^\mathsf{T}. \tag{2.33}$$

With the subspace kernel sum rule and the subspace kernel chain rule we can now construct the subspace kernel Bayes' rule.

### 2.3.5 The Subspace Kernel Bayes' Rule

The Bayes' rule computes a posterior distribution $P(X|Y)$ from a prior distribution $\pi(X)$ and a likelihood function $P(Y|X)$. Fukumizu, Song, and Gretton [20] derive a conditional operator $\mathcal{C}^{\pi}_{X|Y}$ from the prior modified covariance operators $\mathcal{C}^{\pi}_{XY}$ and $\mathcal{C}^{\pi}_{YY}$. We follow this approach and construct the subspace kernel Bayes' rule (subKBR) from the prior modified covariance operators $\hat{\mathcal{C}}^{S,\pi}_{XY}$ and $\hat{\mathcal{C}}^{S,\pi}_{YY}$ which we obtain from the subspace kernel chain rule and the subspace kernel sum rule for tensor product features, respectively. When applied to the embedding of a variate $\boldsymbol{y}_*$, the subspace kernel Bayes' rule returns the mean embedding of the conditional distribution $P(X|\boldsymbol{y}_*)$ as

$$\hat{\mu}^{\pi}_{X|y} = \hat{\mathcal{C}}^{S,\pi}_{X|Y}\phi(\boldsymbol{y}_*) \tag{2.34}$$

$$\hat{\mu}^{\pi}_{X|y} = \hat{\mathcal{C}}^{S,\pi}_{XY}\left(\hat{\mathcal{C}}^{S,\pi}_{YY}\right)^{-1}\phi(\boldsymbol{y}_*). \tag{2.35}$$

From the subspace kernel chain rule, we obtain

$$\begin{aligned}\hat{\mathcal{C}}^{S,\pi}_{XY} &= \left(\hat{\mathcal{C}}^{S}_{Y|X}\hat{\mathcal{C}}^{\pi}_{XX}\right)^\mathsf{T} &(2.36)\\ &= \boldsymbol{\Upsilon}_x \operatorname{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{L}^\mathsf{T}\boldsymbol{K}^\mathsf{T}_{x\bar{x}}\boldsymbol{\Phi}^\mathsf{T} &(2.37)\end{aligned}$$

and from the subspace kernel sum rule

$$\begin{aligned}\mathcal{C}^{S,\pi}_{YY} &= \mathcal{C}^{S}_{Y|X}\mathcal{C}^{\pi}_{XX}\left(\mathcal{C}^{S}_{Y|X}\right)^\mathsf{T} &(2.38)\\ &= \boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}\boldsymbol{L}\operatorname{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{L}^\mathsf{T}\boldsymbol{K}^\mathsf{T}_{x\bar{x}}\boldsymbol{\Phi}^\mathsf{T}. &(2.39)\end{aligned}$$

To keep the notation of the subspace kernel Kalman rule uncluttered, we define the following matrices

$$\begin{aligned}\bar{\boldsymbol{\Lambda}} &:= \operatorname{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{L}^\mathsf{T} &(2.40)\\ \bar{\boldsymbol{D}} &:= \boldsymbol{L}\operatorname{diag}(\boldsymbol{\alpha}_\pi)\boldsymbol{L}^\mathsf{T} \in \mathbb{R}^{m\times m}, &(2.41)\\ \boldsymbol{E} &:= \boldsymbol{K}^\mathsf{T}_{x\bar{x}}\boldsymbol{G}_{yy}\boldsymbol{K}_{x\bar{x}} \in \mathbb{R}^{m\times m}, &(2.42)\end{aligned}$$

where $\boldsymbol{G}_{yy} = \boldsymbol{\Phi}^\intercal\boldsymbol{\Phi}$ is the kernel matrix of the samples $\boldsymbol{y}_i$. Using the same form of the Tikhonov regularization as the kernel Bayes' rule in [19] and substituting the prior modified subspace covariance operators from Equations 2.37 and 2.39 results in

$$\begin{aligned}\hat{\mu}^{\pi}_{X|y} &= \hat{\mathcal{C}}^{S,\pi}_{XY}\left[\left(\hat{\mathcal{C}}^{S,\pi}_{YY}\right)^2 + \gamma\boldsymbol{I}_m\right]^{-1}\hat{\mathcal{C}}^{S,\pi}_{YY}\phi(\boldsymbol{y}_*) &&(2.43)\\ &= \boldsymbol{\Upsilon}_x\bar{\boldsymbol{\Lambda}}\boldsymbol{K}^\intercal_{x\bar{x}}\boldsymbol{\Phi}^\intercal\left[\left(\boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}\bar{\boldsymbol{D}}\boldsymbol{K}^\intercal_{x\bar{x}}\boldsymbol{\Phi}^\intercal\right)\boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}\bar{\boldsymbol{D}}\boldsymbol{K}^\intercal_{x\bar{x}}\boldsymbol{\Phi}^\intercal + \gamma\boldsymbol{I}_m\right]\boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}\bar{\boldsymbol{D}}\boldsymbol{K}^\intercal_{x\bar{x}}\boldsymbol{\Phi}^\intercal\phi(\boldsymbol{y}_*) &&(2.44)\\ &= \boldsymbol{\Upsilon}_x\bar{\boldsymbol{\Lambda}}\boldsymbol{E}\left[(\bar{\boldsymbol{D}}\boldsymbol{E})^2 + \gamma\boldsymbol{I}_m\right]^{-1}\bar{\boldsymbol{D}}\boldsymbol{K}^\intercal_{x\bar{x}}\boldsymbol{g}_{\boldsymbol{y}_*}, &&(2.45)\end{aligned}$$

with kernel vector $\boldsymbol{g}_{\boldsymbol{y}_*} = [g(\boldsymbol{y}_1, \boldsymbol{y}_*), \ldots, g(\boldsymbol{y}_n, \boldsymbol{y}_*)]^\intercal$, and where we apply the matrix identity $\boldsymbol{A}(\boldsymbol{B}\boldsymbol{A} + \lambda\boldsymbol{I})^{-1} = (\boldsymbol{A}\boldsymbol{B} + \lambda\boldsymbol{I})^{-1}\boldsymbol{A}$ with $\boldsymbol{A} = \boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}$ to obtain a finite matrix in the inverse. Since $\boldsymbol{E}$ and $\bar{\boldsymbol{D}}$ are both in $\mathbb{R}^{m\times m}$, the matrix inversion has complexity $O(m^3)$ instead of $O(n^3)$. The entire subspace kernel Bayes' rule has complexity $O(nm^2)$ and, thus, scales linearly with the number of sample points (given a fixed reference set) instead of cubically as for the original kernel Bayes' rule.

### 2.3.6 Experimental Evaluation

We compare the performance, learning time and run time of the subspace kernel Bayes' filter in comparison to the standard kernel Bayes' filter on a simple toy task. We simulate a pendulum which we randomly initialize in the ranges $[0.1\pi, 0.4\pi]$ for the angle $\theta$ and $[-0.5\pi, 0.5\pi]$ for the angular velocity $\dot{\theta}$. The pendulum has a mass of 5kg and a friction coefficient of 1. We apply Gaussian white noise to the system with a variance of 1, and to the observations with a variance of 0.1. Additionally, the observed angles are randomly perturbed by an offset of $\pi/4$. These random perturbations occur with a probability of 0.1 in every time step. Each episode consists of 30 time steps with $\Delta t = 0.1$.

Figure 2.1 shows that the subspace KBF has a slightly better performance when the training set equals the subspace set and maintains the performance of the standard KBF with an increasing number of training samples while the subspace set is fixed to 100 samples. However, at the same time the learning time of the subspace KBF increases at a much lower magnitude and the run time is nearly constant while the learning and run time of the standard KBR grow cubically. The samples for the subspace kernel Bayes rule are drawn uniformly without replacement from the full sample set.

## 2.4 The Kernel Kalman Rule

All three versions of the kernel Bayes' rule discussed in Section 2.2.1 have drawbacks. First, due to the approximation of the prior modified covariance operators, these operators are not guaranteed to be positive definite and, thus, their inversion requires either a harsh form of the Tikhonov regularization or a strong regularization factor and are often still numerically instable. Furthermore, the inverse is dependent on the embedding of the prior distribution and, hence, needs to be recomputed for every Bayesian update of the mean map. This recomputation significantly increases the computational costs, for example, if we want to optimize the hyper-parameter.

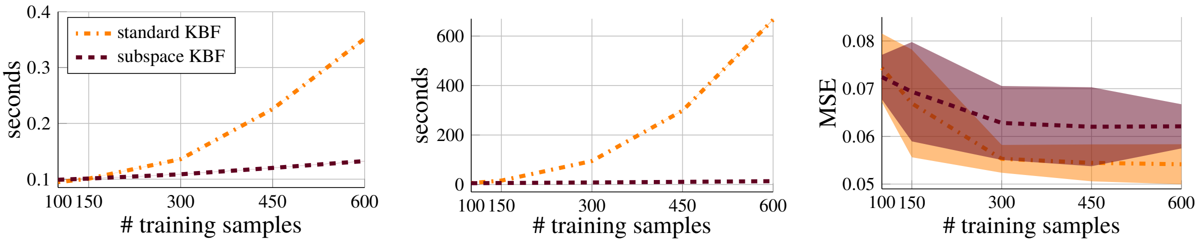


**Figure 2.1:** The subspace variant of the kernel Bayes' filter outperforms the standard kernel Bayes' filter in both, the training time depicted in the left plot and the run time depicted in the middle plot, while maintaining a similar performance to the standard kernel Bayes' rule as depicted in the right plot. The size of the subspace is fixed to 100 samples. The plots show median and the [0.25, 0.75] quantiles over 20 evaluations.

In this section we will present the *kernel Kalman rule* (KKR) as an approximate alternative to the kernel Bayes' rule [19]. We assume a prior belief state over a variable $X$ embedded in a Hilbert space $\mathcal{H}_k$ as

$$\mu^-_{X,t} = \mathbb{E}_{X_t|\boldsymbol{y}_{1:t-1}}[\varphi(X)]$$

and new measurement $\boldsymbol{y}_t$ embedded in a Hilbert space $\mathcal{H}_g$ as $\phi(\boldsymbol{y}_t)$. With the kernel Kalman rule we want to infer the embedding of the posterior belief state

$$\mu^+_{X,t} = \mathbb{E}_{X_t|\boldsymbol{y}_{1:t}}[\varphi(X)] \in \mathcal{H}_k$$

from the prior belief and the new measurement. The derivations for the KKR are inspired by the ansatz from recursive least squares [21, 105, 97], and thus the resulting update equations follow from a clear optimality criterion.

### 2.4.1 Estimating the Posterior Mean Embedding from a Least Squares Objective

Let $\mathcal{C}_{Y|X}$ be a conditional embedding operator of the observation model $P(Y|X)$ that yields for a given belief state embedded in the Hilbert space $\mathcal{H}_k$ the distribution over possible observations embedded into the Hilbert space $\mathcal{H}_g$. We call this conditional embedding operator also *observation operator*. For a single sample $(\boldsymbol{x}_t, \boldsymbol{y}_t)$, the observation operator yields the relation

$$\phi(\boldsymbol{y}_t) = \mathcal{C}_{Y|X}\varphi(\boldsymbol{x}_t) + \zeta_t, \tag{2.46}$$

where $\zeta_t$ is zero mean noise with covariance $\mathcal{R}$. Let us assume that the distribution $p(\boldsymbol{x})$ is unknown and we can only observe the samples $\boldsymbol{y}_t$. The objective of the KKR is then to find the mean embedding $\mu_X$ that minimizes the squared error

$$L = \mathbb{E}_{XY}\left[\left(\phi(\boldsymbol{y}_t) - \mathcal{C}_{Y|X}\mu_X\right)^{\intercal}\mathcal{R}^{-1}\left(\phi(\boldsymbol{y}_t) - \mathcal{C}_{Y|X}\mu_X\right)\right]. \tag{2.47}$$

Note that the use of $\mathcal{R}^{-1}$ as metric for the least squares is somewhat arbitrary (it works with any invertible matrix). This definition becomes more important once we regularize the estimate of $\mu_X$ (see below). We only assume that $\mathcal{R}$ is constant given a single sample $\boldsymbol{x}_t$. We do *not* assume that the noise is constant if we use a mean embedding on the operator $\mathcal{C}_{Y|X}$.

To show that $\mu_X = \mathbb{E}_X[\varphi(\boldsymbol{x})]$, i.e., that $\mu_X$ is indeed the mean embedding of the distribution $p(\boldsymbol{x})$, we can solve for $\mu_X$ by setting the derivative of $L$ to zero, i.e.,

$$\frac{\mathrm{d}L}{\mathrm{d}\mu_X} = \mathbb{E}_{XY}\left[(\phi(\boldsymbol{y}_t) - \mathcal{C}_{Y|X}\mu_X)^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X}\right] \tag{2.48}$$

$$= \mathbb{E}_Y\left[\phi(\boldsymbol{y}_t)^\intercal\right]\mathcal{R}^{-1}\mathcal{C}_{Y|X} - \mu_X^\intercal \mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X} \tag{2.49}$$

$$= 0. \tag{2.50}$$

Assuming that $\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X}$ is invertible, this yields

$$\mu_X = (\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X})^{-1}\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathbb{E}_Y\left[\phi(\boldsymbol{y}_t)\right] \tag{2.51}$$

$$= (\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X})^{-1}\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X}\mathbb{E}_X\left[\varphi(\boldsymbol{x}_t)\right] = \mathbb{E}_X\left[\varphi(\boldsymbol{x}_t)\right], \tag{2.52}$$

where we have used the kernel sum rule $\mathbb{E}_Y[\phi(\boldsymbol{y}_t)] = \mathcal{C}_{Y|X}\mathbb{E}_X[\varphi(\boldsymbol{x}_t)]$. Hence, under the assumption that $\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X}$ is invertible, $\mu_X$ indeed estimates the mean embedding of the unobserved distribution $p(\boldsymbol{x})$. Note that the derivations hold for a constant $\boldsymbol{x}$, i.e, $\boldsymbol{x}_t = \boldsymbol{x}$ as well as for samples $\boldsymbol{x}_t$ drawn from the distribution $p(\boldsymbol{x})$.

In practice, inverting $\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X}$ is, however, not always feasible. Hence, we can introduce an additional regularization objective, i.e.,

$$L_{\text{reg}} = L + (\mu_X - \mu_X^-)^\intercal(\mathcal{C}_{XX}^-)^{-1}(\mu_X - \mu_X^-), \tag{2.53}$$

where $\mu_X^-$ and $\mathcal{C}_{XX}^-$ denote a prior belief embedded as mean embedding and covariance operator, respectively. The solution of this optimization problem is given by

$$\mu_X = \left(\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathcal{C}_{Y|X} + (\mathcal{C}_{XX}^-)^{-1}\right)^{-1}\left(\mathcal{C}_{Y|X}^\intercal \mathcal{R}^{-1}\mathbb{E}_Y\left[\phi(\boldsymbol{y}_t)\right] + (\mathcal{C}_{XX}^-)^{-1}\mu_X^-\right). \tag{2.54}$$

Note that this regularization is the only approximation we make in the derivation of the kernel Kalman rule.

### 2.4.2 Using Recursive Least Squares to Estimate the Posterior Embedding

Since we want to update our estimate $\mu_X$ iteratively with each new observation $\boldsymbol{y}_t$, we assume a prior mean map $\mu_{X,t}^-$. In each iteration, we update the prior mean map with the measurement $\boldsymbol{y}_t$ to obtain the posterior mean map $\mu^+_{X,t}$. From the recursive least squares solution, we know that the update rule for obtaining the posterior mean map $\mu^+_{X,t}$ is

$$\mu^+_{X,t} = \mu^-_{X,t} + \mathcal{Q}_t \left( \phi(\boldsymbol{y}_t) - \mathcal{C}_{Y|X} \mu^-_{X,t} \right), \tag{2.55}$$

where $\mathcal{Q}_t$ is the Hilbert space *Kalman gain operator* that is applied to the correction term $\delta_t = \phi(\boldsymbol{y}_t) - \mathcal{C}_{Y|X} \mu^-_{X,t}$. We call this rule the *kernel Kalman rule* (KKR). It remains to find an optimal value for the Kalman gain operator. In the next section, we will show that this rule is an unbiased estimator of the posterior mean map. Thus, we cannot obtain the optimal $\mathcal{Q}_t$ by minimizing directly the error. Instead, we will show in Section 2.4.2, how we can find an optimal $\mathcal{Q}_t$ by minimizing the covariance of the error instead.

### The Kernel Kalman Update is an Unbiased Estimator of the Posterior Mean Map

The embedding of the observation $\phi(\boldsymbol{y}_t)$ in the correction term $\delta_t$ is a single-sample estimator of the embedding of the true distribution over observations $\mu^-_{Y|x,t} = \mathcal{C}_{Y|X} \varphi(\boldsymbol{x}_t)$. Let us assume for now that we have access to the true embedding $\varphi(\boldsymbol{x}_t)$. Thus, we have for the embedding of the observation

$$\phi(\boldsymbol{y}_t) = \mu^-_{Y|x,t} + \zeta_t = \mathcal{C}_{Y|X} \varphi(\boldsymbol{x}_t) + \zeta_t, \tag{2.56}$$

where $\zeta_t$ denotes the error of the single sample estimator $\phi(\boldsymbol{y}_t)$ to the embedding of the true distribution $\mu^-_{Y|x,t}$. By taking the expectation it is easy to show that the error of the single-sample estimator is zero-mean and thus $\phi(\boldsymbol{y}_t)$ is an unbiased estimator for $\mu^-_{Y|X,t}$. We refer to Appendix B.1 for a more detailed derivation. We further assume that $\zeta_t$ is independent from the state $x_t$ and has constant covariance $\mathcal{R}$. Following from the delta method [2], this assumption is a reasonable choice as we assume i.i.d, zero mean observation noise. Similarly, the error of the a-posteriori mean embedding to the embedding of the true state is given as

$$\begin{aligned} \varepsilon^+_t &= \varphi(\boldsymbol{x}_t) - \mu^+_{X,t} \qquad (2.57) \\ &= \varphi(\boldsymbol{x}_t) - \mu^-_{X,t} - \mathcal{Q}_t(\phi(\boldsymbol{y}_t) - \mathcal{C}_{Y|X} \mu^-_{X,t}), \qquad (2.58) \end{aligned}$$

where we use Equation 2.55 to substitute the embedding of the posterior belief. By substituting $\phi(\boldsymbol{y}_t)$ with Equation 2.56 and defining the error of the a-priori mean embedding analogously as $\varepsilon^-_t = \varphi(\boldsymbol{x}_t) - \mu^-_{X,t}$, we arrive at

$$\begin{aligned} \varepsilon^+_t &= \varphi(\boldsymbol{x}_t) - \mu^-_{X,t} - \mathcal{Q}_t(\mathcal{C}_{Y|X} \varphi(\boldsymbol{x}_t) + \zeta_t - \mathcal{C}_{Y|X} \mu^-_{X,t}) \qquad (2.59) \\ &= \left( \mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X} \right) (\varphi(\boldsymbol{x}_t) - \mu^-_{X,t}) - \mathcal{Q}_t \zeta_t \qquad (2.60) \\ &= \left( \mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X} \right) \varepsilon^-_t - \mathcal{Q}_t \zeta_t \qquad (2.61) \end{aligned}$$

with identity operator $\mathcal{I}$. We can now apply the expectation operator and exploit its linearity to obtain

$$\begin{aligned}\mathbb{E}\left[\varepsilon_t^+\right] &= \mathbb{E}\left[\left(\mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X}\right)\varepsilon_t^- - \mathcal{Q}_t \zeta_t\right] \\ &= \left(\mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X}\right)\mathbb{E}\left[\varepsilon_t^-\right] - \mathcal{Q}_t \mathbb{E}\left[\zeta_t\right]. \end{aligned} \tag{2.62}$$

Since the residual of the observation operator is zero mean ($\mathbb{E}[\zeta_t] = 0$), we see that, given an unbiased a-priori mean embedding ($\mathbb{E}[\varepsilon_t^-] = 0$), the a-posteriori mean embedding obtained from the kernel Kalman update is unbiased ($\mathbb{E}[\varepsilon_t^+] = 0$) independent of the choice of $\mathcal{Q}$. Thus, we cannot use the expected error as an optimality criterion for the Kalman gain operator.

### Finding the Optimal Kernel Kalman Gain Operator

If the expected error—or the first moment of the error distribution—is already zero, taking the covariance of the error—or the second moment—is a consequent choice. Hence, we chose the kernel Kalman gain operator $\mathcal{Q}_t$ which minimizes the expected squared loss $\mathbb{E}\left[\left(\varepsilon_t^+\right)^\intercal \varepsilon_t^+\right]$ or equivalently the variance of the estimator. The objective for minimizing the variance can also be reformulated as minimizing the trace of the a-posteriori covariance operator $\mathcal{C}_{XX,t}^+$ of the state $\boldsymbol{x}_t$ at time $t$, i.e., $\min_{\mathcal{Q}_t} \mathbb{E}\left[\left(\varepsilon_t^+\right)^\intercal \varepsilon_t^+\right] = \min_{\mathcal{Q}_t} \operatorname{Tr} \mathcal{C}_{XX,t}^+$. Using the formulation of the posterior error from Equation 2.61 and the independence assumption of $\zeta_t$ and $\varepsilon_t$ allows us to reformulate the a-posteriori covariance operator as

$$\mathcal{C}_{XX,t}^+ = \left(\mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X}\right)\mathcal{C}_{XX,t}^-\left(\mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X}\right)^\intercal + \mathcal{Q}_t \mathcal{R} \mathcal{Q}_t^\intercal, \tag{2.63}$$

where $\mathcal{R} = \mathbb{E}[\zeta_t \zeta_t^\intercal]$ is the covariance of the residual of the observation operator. Taking the derivative of the trace of the covariance operator and setting it to zero leads to the solution for the kernel Kalman gain operator

$$\mathcal{Q}_t = \mathcal{C}_{XX,t}^- \mathcal{C}_{Y|X}^\intercal \left(\mathcal{C}_{Y|X} \mathcal{C}_{XX,t}^- \mathcal{C}_{Y|X}^\intercal + \mathcal{R}\right)^{-1}. \tag{2.64}$$

We provide a detailed derivation of the optimal Kalman gain operator in Appendix B.2. From Equations 2.63 and 2.64, we can also see that it is possible to recursively estimate the covariance embedding operator independently of the mean map and of the observations. This property will allow us later to precompute the covariance embedding operator as well as the kernel Kalman gain operator to further improve the computational complexity of our algorithm. Following [97], the update of the covariance operator can be further simplified to

$$\mathcal{C}_{XX,t}^+ = \mathcal{C}_{XX,t}^- - \mathcal{Q}_t \mathcal{C}_{Y|X} \mathcal{C}_{XX,t}^-. \tag{2.65}$$

The derivations of this simplification can be found in Appendix B.3. In the following section we will show how to obtain the empirical Kalman update rule from a finite data set.

### 2.4.3 Empirical Kernel Kalman Rule

The equations for the kernel Kalman rule that we derived in the previous section are based on embeddings in infinite dimensional Hilbert spaces and operators that map between these spaces. In practice, these embeddings and operators are estimated from a finite set of samples $\mathcal{D}_{XY} = \left\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\right\}$. In this section we will show how we can reformulate the kernel Kalman rule to manipulations of finite matrices by applying the kernel trick (i.e., matrix identities). Based on the data set $\mathcal{D}_{XY}$ and the corresponding feature matrices $\boldsymbol{\Upsilon}_x$ and $\boldsymbol{\Phi}$, the finite sample estimators of the prior mean embedding and the prior covariance operator are given as

$$\hat{\mu}^-_{X,t} = \boldsymbol{\Upsilon}_x \boldsymbol{m}^-_t \qquad \text{and} \qquad \hat{\mathcal{C}}^-_{XX,t} = \boldsymbol{\Upsilon}_x \boldsymbol{S}^-_t \boldsymbol{\Upsilon}^\intercal_x, \tag{2.66}$$

respectively, with weight vector $\boldsymbol{m}^-_t$ and positive definite weight matrix $\boldsymbol{S}^-_t$. Using this finite sample estimator of the covariance operator, the finite sample estimator of the conditional operator from Equation 2.6, and by approximating the covariance of the residual of the observation operator with a diagonal $\mathcal{R} = \kappa \mathcal{I}$, we can rewrite the kernel Kalman gain operator as

$$\hat{\mathcal{Q}}_t = \boldsymbol{\Upsilon}_x \boldsymbol{S}^-_t \boldsymbol{O}^\intercal \boldsymbol{\Phi}^\intercal \left(\boldsymbol{\Phi} \boldsymbol{O} \boldsymbol{S}^-_t \boldsymbol{O}^\intercal \boldsymbol{\Phi}^\intercal + \kappa \mathcal{I}\right)^{-1}, \tag{2.67}$$

with the observation matrix $\boldsymbol{O} = (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I})^{-1} \boldsymbol{K}_{xx}$. Here, the approximation of $\mathcal{R} = \kappa \mathcal{I}$ also acts as a small regularization in the inverse to ensure its positive definiteness and to improve the numerical stability of the kernel Kalman rule. However, $\hat{\mathcal{Q}}_t$ still requires the inversion of an infinite dimensional matrix. Using matrix identities, we can solve this problem and arrive at

$$\hat{\mathcal{Q}}_t = \boldsymbol{\Upsilon}_x \underbrace{\boldsymbol{S}^-_t \boldsymbol{O}^\intercal (\boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{S}^-_t \boldsymbol{O}^\intercal + \kappa \boldsymbol{I}_n)^{-1}}_{\boldsymbol{Q}_t} \boldsymbol{\Phi}^\intercal, \tag{2.68}$$

where we defined $\boldsymbol{Q}_t = \boldsymbol{S}^-_t \boldsymbol{O}^\intercal (\boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{S}^-_t \boldsymbol{O}^\intercal + \kappa \boldsymbol{I})^{-1} \in \mathbb{R}^{n \times n}$ where $\boldsymbol{G}_{yy} = \boldsymbol{\Phi}^\intercal \boldsymbol{\Phi}$ is the Gram matrix of the observations. Based on this reformulation of the kernel Kalman gain operator, we can obtain finite vector/matrix representations of the update equations for the estimator of the mean embedding (Equation 2.55) and the estimator of the covariance operator (Equation 2.65). For the weight vector $\boldsymbol{m}_t$, we arrive at

$$\boldsymbol{m}^+_t = \boldsymbol{m}^-_t + \boldsymbol{Q}_t \left(\boldsymbol{g}_{\boldsymbol{y}_t} - \boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{m}^-_t\right), \tag{2.69}$$

where $\boldsymbol{g}_{\boldsymbol{y}_t} = [g(\boldsymbol{y}_1, \boldsymbol{y}_t), \dots, g(\boldsymbol{y}_n, \boldsymbol{y}_t)]^\intercal$ is the kernel vector of the measurement at time $t$. Similarly, we can also obtain the update equation for the weight matrix $\boldsymbol{S}_t$ as

$$\boldsymbol{S}_t = \boldsymbol{S}^-_t - \boldsymbol{Q}_t \boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{S}^-_t. \tag{2.70}$$

The algorithm requires the inversion of a $m \times m$ matrix in every iteration for computing the kernel Kalman gain matrix $\boldsymbol{Q}_t$. Hence, similar to the kernel Bayes' rule, the computational complexity of a straightforward implementation would scale cubically with the number of data points $m$. However, in contrast to the KBR, the inverse in $\boldsymbol{Q}_t$ is only dependent on time and not on the estimate of the mean map. Thus, the kernel Kalman gain matrix can be precomputed since it is identical for multiple parallel runs of the algorithm. Furthermore, if the stream of incoming measurements is reliable (no time steps without incoming measurement), $\boldsymbol{S}_t$ will converge to a stationary matrix and by that $\boldsymbol{Q}_t$ will become stationary as well. While many applications do not require to perform state estimations in parallel, it is a huge advantage for hyper-parameter optimization as we can evaluate multiple trajectories from a validation set simultaneously. As for most kernel-based methods, hyper-parameter optimization is crucial for scaling the approach to complex tasks. So far, the hyper-parameters of the kernels for the KBF have typically been set by heuristics as optimization would be too expensive.

Besides the hyper-parameters in the kernel functions (e.g. bandwidths) and the regularization constants of the conditional operators, we also treat the approximation of the covariance operator $\mathcal{R} \approx \kappa\mathcal{I}$ as a hyper-parameter which we optimize. Note that the selection of the minimization objective, e.g., mean squared error (MSE) or negative log-likelihood (NLL), has a substantial effect on the selection of this parameter. For the MSE objective, the parameters are chosen to only optimize the expectation of the filter output. Consequently the parameter $\kappa$ acts more as a regularizer and is chosen as small as possible. In contrast, using the NLL objective also respects the variance of the filter output and thus the role as approximation to the variance $\mathcal{R}$ is more important for choosing the parameter value.

### 2.4.4 The Subspace Kernel Kalman Rule

In Section 2.3 we have already shown how we can apply the subspace conditional embedding operator to leverage from large data sets but at the same time maintain the computational tractability of the learned models. In this section, we will now show how we can apply this technique to the kernel Kalman rule to obtain the *subspace kernel Kalman rule* (subKKR).

A core difference between the KKR and the subKKR is the representation of the embedded distributions. While we represent the embeddings for the kernel Kalman rule as weight vector $\boldsymbol{m}_t^-$ and weight matrix $\boldsymbol{S}_t^-$, we use the projections into a subspace

$$\boldsymbol{n}_t = \boldsymbol{\Gamma}^\intercal \mu_t = \boldsymbol{\Gamma}^\intercal \boldsymbol{\Upsilon}_x \boldsymbol{m}_t = \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{m}_t, \tag{2.71}$$

$$\boldsymbol{P}_t = \boldsymbol{\Gamma}^\intercal \mathcal{C}_{XX,t} \boldsymbol{\Gamma} = \boldsymbol{\Gamma}^\intercal \boldsymbol{\Upsilon}_x \boldsymbol{S}_t \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Gamma} = \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{S}_t \boldsymbol{K}_{x\bar{x}}. \tag{2.72}$$

to represent the distribution for the subspace kernel Kalman rule. These projections will later allow us to express all operations in the lower dimensional subspace instead of the space spanned by the full data set. Matrix manipulations with the full data set are then only necessary during the learning phase of the KKR not while performing inference.

We use a slightly modified version of the kernel Kalman gain from Equation 2.64 where we approximate the covariance operator $\mathcal{R}$ with a diagonal operator $\kappa\mathcal{I}$. With the subspace conditional embedding operator $\hat{\mathcal{C}}^S_{Y|X}$ of the distribution $P(Y|X)$, as derived in Section 2.2.1 we obtain the subspace kernel Kalman gain operator as

$$\hat{\mathcal{Q}}_t^S = \hat{\mathcal{C}}^-_{XX,t}\left(\hat{\mathcal{C}}^S_{Y|X}\right)^\intercal\left(\hat{\mathcal{C}}^S_{Y|X}\hat{\mathcal{C}}^-_{XX,t}\left(\hat{\mathcal{C}}^S_{Y|X}\right)^\intercal + \kappa\mathcal{I}\right)^{-1} \tag{2.73}$$

We can further derive a finite matrix representation of the operator using matrix identities and the projection into the subspace spanned by the features $\boldsymbol{\Gamma}^\intercal$ as

$$\boldsymbol{\Gamma}^\intercal\hat{\mathcal{Q}}_t^S = \underbrace{\boldsymbol{P}_t^-(\boldsymbol{O}^S)^\intercal\left(\boldsymbol{K}_{x\bar{x}}^\intercal\boldsymbol{G}_{yy}\boldsymbol{K}_{x\bar{x}}\boldsymbol{O}^S\boldsymbol{P}_t^-(\boldsymbol{O}^S)^\intercal + \kappa\boldsymbol{I}_m\right)^{-1}\boldsymbol{K}_{x\bar{x}}^\intercal}_{\boldsymbol{Q}_t^S}\boldsymbol{\Phi}^\intercal, \tag{2.74}$$

where we define the subspace kernel Kalman gain matrix $\boldsymbol{Q}_t^S$ using the short hand $\boldsymbol{O}^S := (\boldsymbol{K}_{x\bar{x}}^\intercal\boldsymbol{K}_{x\bar{x}} + \lambda\boldsymbol{I}_m)^{-1}$. A detailed derivation can be found in Appendix B.5. Note that $\boldsymbol{Q}_t^S \in \mathbb{R}^{m\times n}$ and not $\mathbb{R}^{m\times m}$, however when applying the subspace KKR in an inference algorithm, we can use the matrix $\boldsymbol{K}_{x\bar{x}}^\intercal$ on the right side as a projection for the high-dimensional embedding of the distribution over the variable $Y$ to which the gain is applied (see Algorithm 2 for an example). From here, the update equation for the projection of the mean map becomes

$$\boldsymbol{n}_{X,t}^+ = \boldsymbol{n}_{X,t}^- + \boldsymbol{Q}_t^S\left(\boldsymbol{g}_{y_t} - \boldsymbol{G}_{yy}\boldsymbol{K}_{x\bar{x}}\boldsymbol{O}^S\boldsymbol{n}_t^-\right), \tag{2.75}$$

And similarly, we can derive the update equation for the covariance embedding as

$$\boldsymbol{P}_t^+ = \boldsymbol{P}_t^- - \boldsymbol{Q}_t^S\boldsymbol{G}_{yy}\boldsymbol{K}_{x\bar{x}}\boldsymbol{O}^S\boldsymbol{P}_t^- \tag{2.76}$$

In contrast to the kernel Kalman gain presented in the previous section, but also in contrast to the variants of the kernel Bayes rule discussed in Section 2.2.1, the subspace kernel Kalman gain requires only the inversion of an $m \times m$ matrix instead of an $n \times n$ matrix, where $m \ll n$. Still, the full data set of $n$ samples can be used to learn the Kalman gain operator.

### 2.4.5 Experimental Comparison of (sub)KKR and (sub)KBR

We compare the performance of the (subspace) kernel Kalman rule to the performance of the (subspace) kernel Bayes rule on a simple stationary filtering task for estimating the expectation of a Gaussian distribution. The graphical model that we assume for this task is depicted in Figure 2.2. We sample $N = 500$ latent context variables $c_i$ uniformly from the interval $[-5, 5]$ as the mean of the Gaussian distributions. Afterwards we draw one single ($M = 1$) observed sample $s_i$ for each context from $\mathcal{N}(c_i, \frac{1}{3})$ and learn the kernel Kalman rule and the different versions of the kernel Bayes rule with the context variables as states and the samples as observations. For the performance comparison (Figure 2.3), the KKR and the KBR are learned with a kernel size of 200 samples, subKKR and subKBR are both learned with 200 samples to span the subspace and the full set of 500 samples to learn the operators. The comparison of time efficiency is summarized in Table 2.1 where the

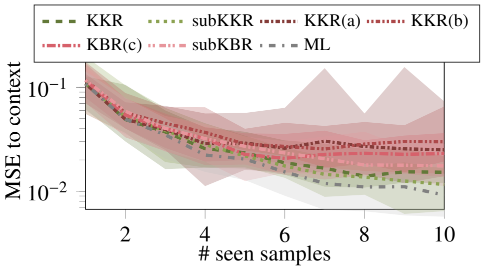


**Figure 2.3:** Performance of the KKR updates vs the KBR updates for estimating the mean of a Gaussian distribution with 1-10 seen samples. The ML estimate serves as a baseline. Depicted are the median and the $(0.15, 0.85)$-quantiles of the MSE to the true mean over 20 runs.

| # | 200 | 300 | 400 | 500 |
|---|---|---|---|---|
| **KKR** | **0.1110** | **0.3360** | **0.7155** | **1.3965** |
| **subKKR** | **0.1820** | **0.4655** | **0.9130** | **1.6075** |
| KBR(a) | 2.9395 | 9.2395 | 21.5035 | 41.6955 |
| KBR(b) | 0.8695 | 2.5840 | 5.7580 | 10.9900 |
| KBR(c) | 0.5455 | 1.5575 | 3.3695 | 6.4465 |
| subKBR | 2.3530 | 4.7625 | 7.6540 | 12.7960 |

**Table 2.1:** Time consumptions of the KKR and KBR update methods for different kernel sizes. The update was performed on 10 samples from 10 different distributions. The subspace KKR and KBR updates are trained with 500 samples in the full data set. Both KKR methods outperform the KBR methods clearly as they are able to process the update on the 10 different distributions in parallel.

respective kernel size and subspace size is denoted as column header. The subKKR and subKBR have always been learned with the full data set of 500 samples. The data points for the subspace have been drawn uniformly without replacement from the full data set.

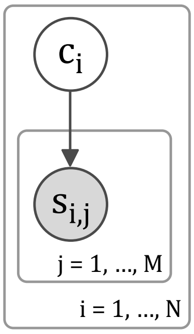


**Figure 2.2:** Graphical model for comparing KKR to KBR.

For the optimization of the hyper-parameters and for the evaluation, we have respectively generated a data set with $N = 10$ latent context variables from the same uniform distribution. These context variables are not be observed by the the filter methods. Next, we draw $M = 10$ samples from the Gaussian distribution around each context and update each method iteratively with these ten samples. For each update we compute the squared error to the true context and take the mean over all ten context variables. We use squared exponentials as kernel functions and optimize their bandwidths as well as the regularization parameters using CMA-ES [35]. Figure 2.3 shows the median and the $(0.15, 0.85)$-quantiles of the MSE to the true context over the number of seen samples. As a baseline we depict the maximum-likelihood (ML) estimate of the expectation. We see that while in the beginning all methods perform similar to the ML estimate, with more seen samples KKR and subKKR outperform all variants of the KBR. Moreover the choice to depict the median and $(0.15, 0.85)$-quantiles over mean and standard deviation is due to the instable optimization behavior of the KBR which produced a lot of outliers. In Table 2.1 we state the time consumed to perform ten KKR/KBR updates on 10 estimation tasks for different kernel sizes. Here, the KKR/subKKR methods benefit from their ability to process the updates for all 10 estimation tasks in parallel. Yet, the ability to precompute $\boldsymbol{Q}_t/\boldsymbol{Q}_t^S$ and $\boldsymbol{S}_t/\boldsymbol{P}_t$ has not even been exploited.

In a second experiment, we have investigated how sensible the KKR is to non-constant noise in comparison to the KBR. We have sampled data similar to the previous experiment with a context variable $c_i$ in the range [-5, 5] and observations $s_{i,j}$ from the distribution $\mathcal{N}(c_i, \sigma(c_i))$. The variance of the Gaussian distribution is dependent on the context variable

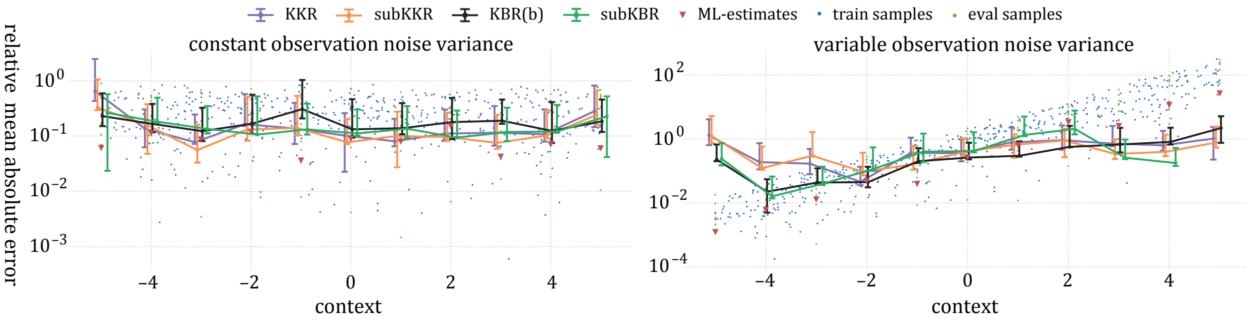


**Figure 2.4:** Comparison of the performances of the KKR, subKKR, KBR(b), and subKBR on the estimation of the mean $c_i$ of a Gaussian random variable, left with constant variance and right with variable variance $\exp(c_i)$. The y-axis depicts the mean absolute error relative to the context using a log-scale. Because of the exponential relation between the observation noise and the context variable, we get the linear slope in the distribution of the samples in the right plot.

by $\sigma(c_i) = \exp(c_i)$. Again, we sample $N = 500$ context and one observation ($M = 1$) for each context for learning the models. For the optimization of the hyper-parameters, we have sampled a data set of $N = 10$ context variables with $M = 10$ observations for each context. And for the evaluation of the methods, we have chosen the context variables at the integers $[-5, -4, \ldots, 5]$ and have sampled again $M = 10$ observations per context. For each sampled context, we perform updates with all ten observations. The plots in Figure 2.4 show mean and min/max of the estimated mean relative to the true mean (context) from which the observations have been sampled for both cases, constant noise $\sigma = \frac{1}{3}$ and variable noise $\sigma = \exp(c_i)$. As expected from the previous experiment, it can be clearly seen that all models perform similarly well for the case of a constant noise variance. In the case of variable noise variance, all models perform worse for smaller variances where the impact is larger in the performances of the KKR and the subKKR. Note, however, that the KBR methods suffered from numerical instabilities for large noise variances. For instance, KBR(b) was the only KBR method that yielded results for the largest variance $\sigma = \exp(5)$.

## 2.5 Applications of the Kernel Kalman Rule

In Section 2.4, we have shown how we can derive the KKR as an operator for approximate Bayesian updates in the framework for nonparametric inference. In this section we will present two applications of the kernel Kalman rule. In Section 2.5.1 we will first present the *kernel Kalman filter* (KKF) and discuss details about the implementation. A subspace variate of the KKF is presented in Section 2.5.2 and experimental results of both are shown in Section 2.5.3. The *kernel forward backward smoother* (KFBS) is presented as another application of the KKR in Section 2.5.4 and a subspace variate is discussed in Section 2.5.5. We finally show experimental evaluations of the KFBS and the subKFBS in Section 2.5.6.

## 2.5.1 The Kernel Kalman Filter

Similar to the kernel Bayes' filter [20, 103], we can combine the kernel Kalman rule with the kernel sum rule to formulate the *kernel Kalman filter* (KKF). To learn the models of the KKF, we assume a data set $\mathcal{D}_{\tilde{X}XY} = \left\{(\tilde{\boldsymbol{x}}_1, \boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\tilde{\boldsymbol{x}}_n, \boldsymbol{x}_n, \boldsymbol{y}_n)\right\}$ consisting of triples with preceding state $\tilde{x}_i$, state $x_i$, and measurement $y_i$ as given. We further assume the states to be Markov, i.e., the state $x_i$ is only dependent on its predecessor $\tilde{x}_i$. Based on this data set we define the feature matrices $\boldsymbol{\Upsilon}_x := [\varphi(\boldsymbol{x}_1), \dots, \varphi(\boldsymbol{x}_n)]$, $\boldsymbol{\Upsilon}_{\tilde{x}} := [\varphi(\tilde{\boldsymbol{x}}_1), \dots, \varphi(\tilde{\boldsymbol{x}}_n)]$, and $\boldsymbol{\Phi} := [\phi(\boldsymbol{y}_1), \dots, \phi(\boldsymbol{y}_n)]$. In contrast to the KBF, we represent the belief state as mean map $\hat{\mu}_{X,t} = \boldsymbol{\Upsilon}_x \boldsymbol{m}_t$ and as covariance operator $\hat{\mathcal{C}}_{XX,t} = \boldsymbol{\Upsilon}_x \boldsymbol{S}_t \boldsymbol{\Upsilon}_x^\intercal$.

The forward model $P(X|\tilde{X})$ that propagates the posterior belief state at time $t$ to the prior belief state at time $t+1$ can then be learned as conditional embedding operator

$$\hat{\mathcal{C}}_{X|\tilde{X}} = \boldsymbol{\Upsilon}_x (\boldsymbol{K}_{\tilde{x}\tilde{x}} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{\Upsilon}_{\tilde{x}}^\intercal, \tag{2.77}$$

which we also call *transition operator*. Here, $\boldsymbol{K}_{\tilde{x}\tilde{x}}$ is the Gram matrix of the features of the preceding states $\boldsymbol{\Upsilon}_{\tilde{x}}$. The posterior belief state at time $t$ is then propagated to the prior belief state at time $t+1$ time by applying the kernel sum rule. That is, we apply the transition operator to the posterior mean map and the posterior covariance embedding at time $t$ and obtain prior mean map and prior covariance embedding at time $t+1$, i.e.,

$$\hat{\mu}_{X,t+1}^{-} = \hat{\mathcal{C}}_{X|\tilde{X}} \hat{\mu}_{X,t}^{+} = \boldsymbol{\Upsilon}_x \boldsymbol{T} \boldsymbol{m}_t^{+}, \qquad \Leftrightarrow \qquad \boldsymbol{m}_{t+1}^{-} = \boldsymbol{T} \boldsymbol{m}_t^{+} \tag{2.78}$$

$$\hat{\mathcal{C}}_{XX,t+1}^{-} = \hat{\mathcal{C}}_{X|\tilde{X}} \hat{\mathcal{C}}_{XX,t}^{+} \hat{\mathcal{C}}_{X|\tilde{X}}^\intercal + \mathcal{V} \qquad \Leftrightarrow \qquad \boldsymbol{S}_{t+1}^{-} = \boldsymbol{T} \boldsymbol{S}_t^{+} \boldsymbol{T}^\intercal + \boldsymbol{V}. \tag{2.79}$$

Note that the propagation of the covariance embedding is slightly different to the kernel sum rule by [103], however this formulation follows directly from the kernel chain rule (c.f. Equations 2.32 and 2.11). Analog to the observation matrix $\boldsymbol{O}$ (c.f. Section 2.4.3), we denote the transition matrix $\boldsymbol{T} = (\boldsymbol{K}_{\tilde{x}\tilde{x}} + \lambda_{\boldsymbol{T}} \boldsymbol{I})^{-1} \boldsymbol{K}_{\tilde{x}x}$, where $\boldsymbol{K}_{\tilde{x}x} = \boldsymbol{\Upsilon}_{\tilde{x}}^\intercal \boldsymbol{\Upsilon}_x$ is the kernel matrix of the preceding states and the current states. The covariance of the transition residual $\mathcal{V}$ and its finite matrix representation $\boldsymbol{V}$ can be obtained as

$$\mathcal{V} = \frac{1}{m} \left(\hat{\mathcal{C}}_{X|\tilde{X}} \boldsymbol{\Upsilon}_{\tilde{x}} - \boldsymbol{\Upsilon}_x\right) \left(\hat{\mathcal{C}}_{X|\tilde{X}} \boldsymbol{\Upsilon}_{\tilde{x}} - \boldsymbol{\Upsilon}_x\right)^\intercal \tag{2.80}$$

$$= \frac{1}{m} \left(\boldsymbol{\Upsilon}_x (\boldsymbol{K}_{\tilde{x}\tilde{x}} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{\Upsilon}_{\tilde{x}}^\intercal \boldsymbol{\Upsilon}_{\tilde{x}} - \boldsymbol{\Upsilon}_x\right) \left(\boldsymbol{\Upsilon}_x (\boldsymbol{K}_{\tilde{x}\tilde{x}} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{\Upsilon}_{\tilde{x}}^\intercal \boldsymbol{\Upsilon}_{\tilde{x}} - \boldsymbol{\Upsilon}_x\right)^\intercal \tag{2.81}$$

$$= \boldsymbol{\Upsilon}_x \underbrace{\frac{1}{m} \left((\boldsymbol{K}_{\tilde{x}\tilde{x}} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{K}_{\tilde{x}\tilde{x}} - \boldsymbol{I}_n\right) \left((\boldsymbol{K}_{\tilde{x}\tilde{x}} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{K}_{\tilde{x}\tilde{x}} - \boldsymbol{I}_n\right)^\intercal}_{\boldsymbol{V}} \boldsymbol{\Upsilon}_x^\intercal. \tag{2.82}$$

On the new prior belief state that we obtain from the transition update, we can afterwards apply the kernel Kalman rule as observation update. Before we give a condensed summary of the kernel Kalman filter in Algorithm 1, we will discuss how we obtain the embedding of the distribution over the initial states in the next section. To extract some meaningful information from the RKHS-embedded distributions, we furthermore need to find a mapping of the embedded distribution back into the state space. In Section 2.5.1, show how we approached the so-called *preimage problem* and shortly discuss other solutions.

## Embedding the Initial State Distribution

Before running the filter on incoming measurements $y_t$, we need to initialize the belief state with an initial mean map $\mu_{X,0}$ and an initial covariance operator $\mathcal{C}_{XX,0}$. We can obtain these initial embeddings from a data set $\mathcal{D}_0 = \{\boldsymbol{x}_1^0, \dots, \boldsymbol{x}_N^0\}$ which consists in general of samples from the initial distribution of the system. Practically, we can obtain this data set by taking the initial states from multiple training episodes or—if we assume a stationary distribution—we can also take all training samples for the initialization. We can obtain the initial mean map by first embedding a uniform distribution into the RKHS spanned by the features of the initial states $\boldsymbol{\Upsilon}_{x,0}$. Afterwards, we apply a conditional operator to map this distribution into the Hilbert space spanned by the features $\boldsymbol{\Upsilon}_x$ as

$$\hat{\mu}_{X,0} = \boldsymbol{\Upsilon}_x \boldsymbol{m}_0 = \boldsymbol{\Upsilon}_x (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Upsilon}_{x,0} \boldsymbol{1}_N \frac{1}{N} \tag{2.83}$$

$$\Leftrightarrow \quad \boldsymbol{m}_0 = (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{K}_{x0} \boldsymbol{1}_N \frac{1}{N}. \tag{2.84}$$

where $\boldsymbol{K}_{x0} = \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Upsilon}_{x,0}$ is the kernel matrix of the training samples and the samples in $\mathcal{D}_0$, and $\boldsymbol{1}_N$ denotes the $N$-dimensional all-ones vector. Similarly, we can obtain the initial covariance embedding operator as

$$\hat{\mathcal{C}}_{XX,0} = \boldsymbol{\Upsilon}_x \boldsymbol{S}_0 \boldsymbol{\Upsilon}_x^\intercal = \frac{1}{N} \boldsymbol{\Upsilon}_x (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{K}_{x0} \boldsymbol{K}_{x0}^\intercal (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{\Upsilon}_x^\intercal - \boldsymbol{\Upsilon}_x \boldsymbol{m}_0 \boldsymbol{m}_0^\intercal \boldsymbol{\Upsilon}_x^\intercal. \tag{2.85}$$

Hence, we can obtain the initial weight vector $\boldsymbol{m}_0$ and the initial weight matrix $\boldsymbol{S}_0$ by computing the mean and the covariance over the columns of the matrix $\boldsymbol{C}_0 = (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{K}_{x0}$.

## The Pre-image Problem / Recovering the State-Space Distribution

Recovering a distribution in the state space that is a pre-image of a given mean map is still a topic of ongoing research. There are several approaches to this problem, such as fitting a Gaussian mixture model [65], or sampling from the embedded distribution by optimization [12]. In the experiments conducted for this paper, we approach the preimage problem by matching a Gaussian distribution, which is a reasonable choice if the recovered distribution is unimodal. Since we embed the belief state for the kernel Kalman rule as a mean map and as a covariance operator, we can obtain the mean and covariance of a Gaussian approximations by simple matrix manipulations. The space of the samples $\mathbb{R}^d$ together with the linear kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle = \boldsymbol{x}_1^\intercal \boldsymbol{x}_2$ forms an RKHS as well. Therfore, we can simply define a conditional embedding operator that maps from the Hilbert space of the feature vectors to the Hilbert space of the samples as

$$\hat{\mathcal{C}}_{\text{pre}} = \boldsymbol{X} (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_n)^{-1} \boldsymbol{\Upsilon}_x^\intercal. \tag{2.86}$$

```
Algorithm 1: The Kernel Kalman Filter
input: triples $\{(\tilde{\boldsymbol{x}}_1, \boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\tilde{\boldsymbol{x}}_m, \boldsymbol{x}_m, \boldsymbol{y}_m)\}$,
       kernel functions $k$ and $g$, regularization parameters $\lambda$ and $\kappa$,
       let $\boldsymbol{X}$ be the matrix of all $x_i$ as columns, $\tilde{\boldsymbol{X}}$ and $\boldsymbol{Y}$ analogously,
       let $\boldsymbol{X}_0$ be the matrix of all data points used to compute the initial embeddings
compute kernel matrices
$\boldsymbol{K}_{xx} = k(\boldsymbol{X}, \boldsymbol{X})$, $\boldsymbol{K}_{\tilde{x}\tilde{x}} = k(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}})$, $\boldsymbol{K}_{\tilde{x}x} = k(\tilde{\boldsymbol{X}}, \boldsymbol{X})$, and $\boldsymbol{G}_{yy} = g(\boldsymbol{Y}, \boldsymbol{Y})$
compute model matrices
$\boldsymbol{T} = (\boldsymbol{K}_{\tilde{x}\tilde{x}} + \lambda \boldsymbol{I}_m)^{-1} \boldsymbol{K}_{\tilde{x}x}$ and $\boldsymbol{O} = (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_m)^{-1} \boldsymbol{K}_{xx}$
compute initial embeddings
kernel matrix with samples of the initial distribution: $\boldsymbol{K}_0 = k(\boldsymbol{X}, \boldsymbol{X}_0)$,
 $\boldsymbol{C}_0 = (\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_m)^{-1} \boldsymbol{K}_0$,
compute mean and variance over the columns: $\boldsymbol{m}_0 = \mathrm{mean}(\boldsymbol{C}_0)$, $\boldsymbol{S}_0 = \mathrm{var}(\boldsymbol{C}_0)$
loop
    if new observation $\boldsymbol{y}_t$ available then
        compute kernel Kalman gain
        $\boldsymbol{Q}_t = \boldsymbol{S}_t^- \boldsymbol{O}^\intercal (\boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{S}_t^- \boldsymbol{O}^\intercal + \kappa \boldsymbol{I}_m)^{-1}$
        innovation update
        $\boldsymbol{m}_t^+ = \boldsymbol{m}_t^- + \boldsymbol{Q}_t (\boldsymbol{g}_{y_t} - \boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{m}_t^-)$
        $\boldsymbol{S}_t^+ = \boldsymbol{S}_t^- - \boldsymbol{Q}_t \boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{S}_t^-$
    transition update
    $\boldsymbol{m}_{t+1}^- = \boldsymbol{T} \boldsymbol{m}_t^+, \quad \boldsymbol{S}_{t+1}^- = \boldsymbol{T} \boldsymbol{S}_t^+ \boldsymbol{T}^\intercal + \boldsymbol{V}$
    project into state space
    $\boldsymbol{\eta}_t = \boldsymbol{X} \boldsymbol{O} \boldsymbol{m}_t, \quad \boldsymbol{\Sigma}_t = \boldsymbol{X} \boldsymbol{O} \boldsymbol{S}_t \boldsymbol{O}^\intercal \boldsymbol{X}^\intercal$
```

By applying this conditional operator now to the belief state, we obtain the mean of the embedded distribution in the sample space

$$\boldsymbol{\eta}_t = \hat{\mathcal{C}}_{\mathrm{pre}} \mu_{X,t} = \hat{\mathcal{C}}_{\mathrm{pre}} \mathbb{E}_{b_t}[\varphi(X)] = \mathbb{E}_{b_t}[\hat{\mathcal{C}}_{\mathrm{pre}} \varphi(X)] = \mathbb{E}_{b_t}[X]. \tag{2.87}$$

Similarly, we can also apply this operator to the covariance embedding to obtain the covariance of the belief state in the sample space

$$\begin{aligned} \boldsymbol{\Sigma}_t &= \hat{\mathcal{C}}_{\mathrm{pre}} \hat{\mathcal{C}}_{XX,t} \hat{\mathcal{C}}_{\mathrm{pre}}^\intercal \\ &= \hat{\mathcal{C}}_{\mathrm{pre}} \left( \mathbb{E}_{b_t} [\varphi(X) \otimes \varphi(X)] - \mu_{X,t} \otimes \mu_{X,t} \right) \hat{\mathcal{C}}_{\mathrm{pre}}^\intercal \\ &= \mathbb{E}_{b_t} \left[ \hat{\mathcal{C}}_{\mathrm{pre}} \varphi(X) \otimes \varphi(X) \hat{\mathcal{C}}_{\mathrm{pre}}^\intercal \right] - \hat{\mathcal{C}}_{\mathrm{pre}} \mu_{X,t} \otimes \mu_{X,t} \mathcal{C}_{\mathrm{pre}}^\intercal \\ &= \mathbb{E}_{b_t} [X \otimes X] - \boldsymbol{\eta}_t \otimes \boldsymbol{\eta}_t^\intercal \end{aligned} \tag{2.88}$$

However, also any other approach from the literature can be used in the kernel Kalman filter algorithm.

### Embedding Observation Windows

So far, we assumed that we have access to the latent states $\boldsymbol{x}_i$ in our training set. However, in many setups we only have access to the partial observations $\boldsymbol{y}_i$ which do not have the Markov property. Yet, we can still learn a KKR model from the provided data by embedding time windows $\boldsymbol{y}_{t-k+1:t}$ of size $k$ as internal state representation. Similar approaches have been used by auto-regressive HMMs [93]. With longer data windows, the transitions become more and more Markov. How many observation each data window has to contain depends on two factors: on the dimensionality of the underlying system and on the signal-to-noise ratio of the measurements $\boldsymbol{y}_i$.

## 2.5.2 The Subspace Kernel Kalman Filter

The subspace kernel Kalman filter (subKKF) is an extension of the KKF that applies the subspace conditional embedding operator presented in Equation 2.26 as well as the subspace formulation of the kernel Kalman rule derived in Section 2.4.4. In contrast to the KKF, we assume for the subKKF a data set of triples $\{(\boldsymbol{x}_1, \boldsymbol{x}'_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{x}'_n, \boldsymbol{y}_n)\}$, where $\boldsymbol{x}'_i$ is the successor state to $\boldsymbol{x}_i$. The representation of the belief state changes from weight vector $\boldsymbol{m}_t$ and weight matrix $\boldsymbol{S}_t$ to the subspace projections of the embeddings $\boldsymbol{n}_t = \boldsymbol{\Gamma}^\intercal \boldsymbol{\Upsilon}_x \boldsymbol{m}_t = \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{m}_t$ and $\boldsymbol{P}_t = \boldsymbol{\Gamma}^\intercal \boldsymbol{\Upsilon}_x \boldsymbol{S}_t \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Gamma} = \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{S}_t \boldsymbol{K}_{x\bar{x}}$, respectively. Additionally, both update procedures of the kernel Kalman filter, the transition update and the innovation update, have to be substituted by their subspace counterparts. The transition update is realized by the subspace kernel sum rule and the innovation update by the subspace kernel Kalman rule. The equations are depicted in Algorithm 2.

Since we represent the belief state as a projection into the subspace defined by $\boldsymbol{\Gamma}$, we can directly obtain the initial belief state by projecting the uniform embedding in the RKHS spanned by the samples from the initial distribution as

$$\boldsymbol{n}_0 = \boldsymbol{\Gamma} \boldsymbol{\Upsilon}_{x,0} \mathbf{1}_N \frac{1}{N} = \boldsymbol{K}_{\bar{x}0} \mathbf{1}_N \frac{1}{N}, \tag{2.89}$$

$$\boldsymbol{P}_0 = \frac{1}{N} \boldsymbol{\Gamma} \boldsymbol{\Upsilon}_{x,0} \boldsymbol{\Upsilon}_{x,0}^\intercal \boldsymbol{\Gamma}^\intercal = \frac{1}{N} \boldsymbol{K}_{\bar{x}0} (\boldsymbol{K}_{\bar{x}0})^\intercal . \tag{2.90}$$

Here, $\boldsymbol{K}_{\bar{x}0}$ is the feature matrix of the subset and the samples from the initial state distribution. For the mapping back into the state space, we can similarly to the KKF define a subspace conditional operator as

$$\hat{\mathcal{C}}^S_{\text{pre}} := \boldsymbol{X} \boldsymbol{K}_{x\bar{x}} \left( \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I}_m \right)^{-1} \boldsymbol{\Gamma}^\intercal. \tag{2.91}$$

By applying this operator to mean map and covariance embedding, we obtain the mean and variance in state space from the subspace projections as

$$\boldsymbol{\eta}_t = \boldsymbol{X} \boldsymbol{K}_{x\bar{x}} \left( \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I}_m \right)^{-1} \boldsymbol{n}_t, \tag{2.92}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{X} \boldsymbol{K}_{x\bar{x}} \left( \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I}_m \right)^{-1} \boldsymbol{P}_t \left( \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I}_m \right)^{-1} \boldsymbol{K}_{x\bar{x}}^\intercal \boldsymbol{X}^\intercal. \tag{2.93}$$

A concise description of the subspace kernel Kalman filter can be found in Algorithm 2.

### 2.5.3 Experimental Evaluation of the Kernel Kalman Filter

We evaluate the performance of the KKF and the subKKF on two experiments on simulated environments, a pendulum and a quad-link, and one experiment on real-world data from a human motion tracking data set [117]. For all kernel based methods, we use the squared exponential kernel, where we choose the kernel bandwidths according to the *median trick* [45] and scale the median distances with a single optimized parameter.

#### Pendulum

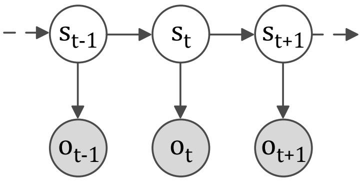


**Figure 2.5:** Graphical model that we assume for the pendulum experiment.

In this experiment, we use a simulated pendulum as system dynamics. The state $s_0 = (q_0, \dot{q}_0)$ of the pendulum is initialized uniformly in the range $[0.1\pi, 0.4\pi]$ for the angle $q_0$ and in the range $[-0.5\frac{\pi}{s}, 0.5\frac{\pi}{s}]$ for the angular velocity $\dot{q}_0$. We simulate the pendulum with a frequency of 10,000 Hz and add normally distributed process noise with $\sigma = 0.1$. The filter methods observe the joint positions with additive Gaussian noise, i.e., $o_t \sim \mathcal{N}(q_t, 0.01)$ at a rate of 10 Hz. A graphical model of the pendulum is depicted in Figure 2.5.

We compare the KKF, the subspace KKF (subKKF) and the KKF learned with the full data set (fullKKF) to version (a) of the kernel Bayes filter (KBF(a)) [103] (the other versions, KBF(b) and KBF(c), have yielded worse results in this experiment) and the kernel Kalman filter with covariance embedding operator (KKF-CEO) [120], as well as to standard filtering approaches such as the EKF [47] and the UKF [113] (which require a model of the system dynamics). To learn the models, we simulate 10 episodes with a length of 30 steps (3 sec), i.e., 300 samples in total. Instead of the true state $s_t$ of the pendulum, we use a window of 4 samples to represent the latent state. For the KKF and all KBF models, we use a kernel size of 100 samples, for the fullKKF, we use all available training samples and for the subKKF we use a set of 100 samples to span the subspace and the full data set to learn the operators. The samples for the subspace are selected from the full data set using the kernel activation heuristic. The results are shown in Figure 2.6. The KKF and subKKF show clearly better results than all other non-parametric filtering methods and reach a performance level close to the EKF and UKF.

#### Quad-Link

In this experiment, we use a simulated 4-link pendulum where we observe the 2-D end-effector positions. The state $\boldsymbol{s}_t$ of the pendulum consists of the four joint angles $\boldsymbol{q}_t$ and joint velocities $\dot{\boldsymbol{q}}_t$. The first and the last joints $q_{0,t=0}, q_{3,t=0}$ are initialized uniformly in the range $[-0.55\pi, -0.45\pi]$ for $q_{0,t=0}$, and $[-0.5\pi, 0.5\pi]$ for $q_{3,t=0}$. The remaining joints and the joint velocities have all been initialized at 0.0. We simulate with Gaussian process noise with $\sigma = 0.01$. The filter methods observe the end-effector positions $x_t$ with Gaussian observation noise as $o_t \sim \mathcal{N}(x_t, 0.001)$ at a rate of 10 Hz. As we assume, that we have no access to the true states, we use data windows of size 4 as representation of the latent state to learn the models.

```
Algorithm 2: The Subspace Kernel Kalman Filter
input: triples $\{(\boldsymbol{x}_1, \boldsymbol{x}'_1, \boldsymbol{y}_1), \dots, (\boldsymbol{x}_n, \boldsymbol{x}'_n, \boldsymbol{y}_n)\}$,
       kernel functions $k$ and $g$, regularization parameters $\lambda$ and $\kappa$,
       let $\boldsymbol{X}$ be the matrix of all $x_i$ as columns, $\boldsymbol{X}'$ and $\boldsymbol{Y}$ analogously,
       let $\boldsymbol{X}_0$ be the matrix of all data points used to compute the initial embeddings

select subset of n samples
$\boldsymbol{X}_S \sim$ random strategy, or $\boldsymbol{X}_S \sim$ kernel activation heuristic

compute kernel matrices
$\boldsymbol{K}_{x\bar{x}} = k(\boldsymbol{X}, \boldsymbol{X}_S)$, $\boldsymbol{K}'_{x\bar{x}} = k(\boldsymbol{X}', \boldsymbol{X}_S)$, and $\boldsymbol{G}_{yy} = g(\boldsymbol{Y}, \boldsymbol{Y})$

compute model matrices
$\boldsymbol{T}^S = \boldsymbol{K}'^{\intercal}_{x\bar{x}} \boldsymbol{K}_{x\bar{x}} \left(\boldsymbol{K}^{\intercal}_{x\bar{x}} \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I}_n\right)^{-1}$ and $\boldsymbol{O}^S := \left(\boldsymbol{K}^{\intercal}_{x\bar{x}} \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I}_n\right)^{-1}$

compute initial embeddings
kernel matrix with samples of the initial distribution: $\boldsymbol{K}_0^S = k(\boldsymbol{X}_S, \boldsymbol{X}_0)$
compute mean and variance over the columns: $\boldsymbol{m}_0 = \mathrm{mean}\left(\boldsymbol{K}_0^S\right)$, $\boldsymbol{S}_0 = \mathrm{var}\left(\boldsymbol{K}_0^S\right)$

loop
    if new observation $\boldsymbol{y}_t$ available then
        compute subspace kernel Kalman gain
        $\boldsymbol{Q}_t^S = \boldsymbol{P}_t^- \left(\boldsymbol{O}^S\right)^{\intercal} \left(\boldsymbol{K}^{\intercal}_{x\bar{x}} \boldsymbol{G}_{yy} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{P}_t^- \left(\boldsymbol{O}^S\right)^{\intercal} + \kappa \boldsymbol{I}_n\right)^{-1} \boldsymbol{K}^{\intercal}_{x\bar{x}}$
        note that you can apply the matrix $\boldsymbol{K}^{\intercal}_{x\bar{x}}$ to the kernel matrix $\boldsymbol{G}$ in the innovation
          update already at learning time to increase computational efficiency.

        innovation update
        $\boldsymbol{n}_t^+ = \boldsymbol{n}_t^- + \boldsymbol{Q}_t^S \left(\boldsymbol{g}_{y_t} - \boldsymbol{G}_{yy} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{n}_t^-\right)$
        $\boldsymbol{P}_t^+ = \boldsymbol{P}_t^- - \boldsymbol{Q}_t^S \boldsymbol{G}_{yy} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{P}_t^-$

    transition update
    $\boldsymbol{n}_{t+1}^- = \boldsymbol{T}^S \boldsymbol{n}_t^+, \quad \boldsymbol{P}_{t+1}^- = \boldsymbol{T}^S \boldsymbol{P}_t^+ \left(\boldsymbol{T}^S\right)^{\intercal} + \boldsymbol{V}_S$
    project into state space
    $\boldsymbol{\eta}_t = \boldsymbol{X} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{n}_t, \quad \Sigma_t = \boldsymbol{X} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{P}_t \boldsymbol{O}^S \boldsymbol{K}^{\intercal}_{x\bar{x}} \boldsymbol{X}^{\intercal}$
```

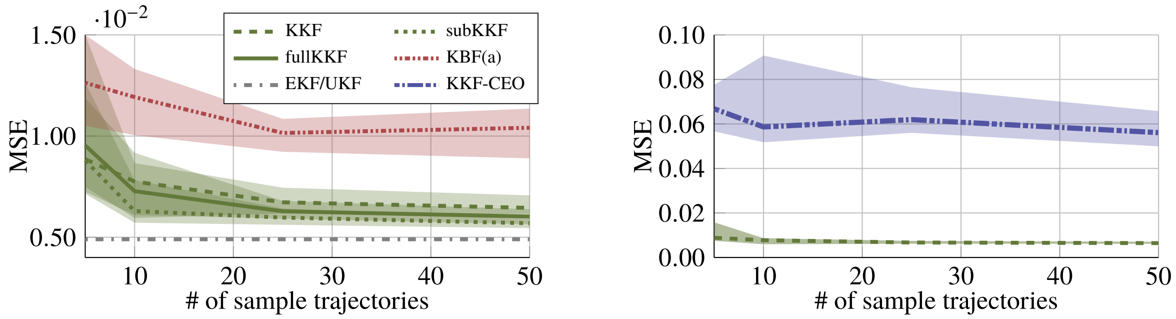
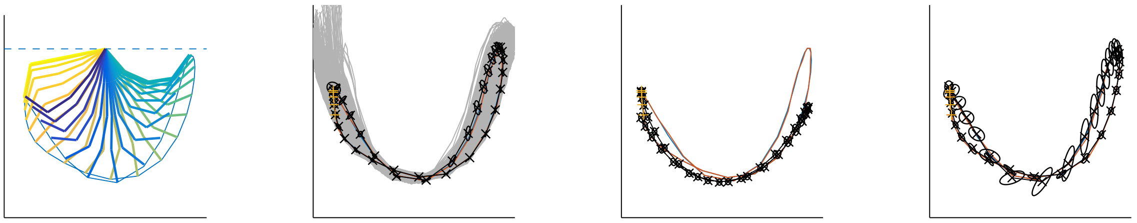


**Figure 2.6:** Comparison of KKF to KBF(b), KKF-CEO, EKF and UKF. All kernel methods (except fullKKF) use kernel matrices of 100 samples. The subKKF method uses a subset of 100 samples and the whole data set to learn the conditional operators. Depicted is the median MSE to the ground-truth of 20 trials with the [0.25 0.75] quantiles.



**(a)** anim. QL **(b)** MCF **(c)** UKF **(d)** subKKF

**Figure 2.8:** Example trajectory of the quad-link end-effector. The filter outputs in black, where the ellipses enclose 90% of the probability mass. All filters were updated with the first five measurements (yellow marks) and predicted the following 30 steps. Figure (a) is an animation of the trajectory.

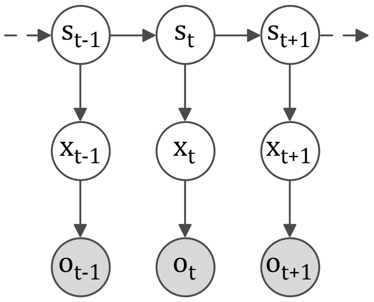


**Figure 2.7:** Graphical model that we assume for the quad-link experiment. The states $s_t$ contain the joint angles and velocities, $x_t$ is the position of the endeffector, and $o_t$ the noise observation thereof.

We evaluate the prediction performance of the subKKF in comparison to the KKF-CEO, the EKF and the UKF. All other non-parametric filtering methods could not achieve a good performance or are not feasible due to the very high computation times. As the subKKF outperformed the KKF in the previous experiments and is also computationally much cheaper, we skip the comparison to the standard KKF in this and also the following experiments. We use a subspace of 500 samples, which is selected according to the kernel activation heuristic, and learn the subKKF with the full data set of 3000 samples.

In a first qualitative evaluation, we compare the long-term prediction performance of the subKKF in comparison to the UKF, the EKF and the Monte-Carlo filter (MCF) as a baseline. This evaluation is shown in Figure 2.8. The first five steps of of the end-effector trajectories are observed by the filters, the following 30 steps are predicted. The UKF is not able to predict the movements of the quad-link end-effector due to the high non-linearity, while the subKKF is able to predict the whole trajectory.

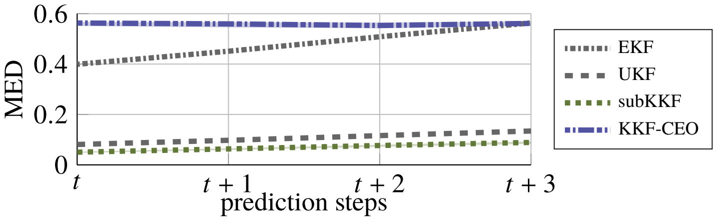


**Figure 2.9:** 1, 2 and 3 step prediction performances in mean euclidean distances (MED) to the true end-effector positions of the quad-link.

We also compared the 1, 2 and 3-step prediction performance of the subKKF to the KKF-CEO, EKF and UKF (Fig. 2.9). The KKF-CEO provides poor results already for the filtering task. The EKF performs equally bad, since the observation model is highly non-linear. The UKF already yields a much better performance as it does not suffer from the linearizion of the system dynamics. The subKKF performs slightly better than the UKF.

### Human Motion Data

The human motion dynamics (HuMoD) database by Wojtusch and Stryk [117] consists of the data sets of several motions executed by two subjects. All data sets contain the recordings from a motion capture system with 36 markers as well as the recordings of the electrical activity of 14 muscles in the legs. Additionally, data from the treadmill such as ground reaction forces and velocities are available. The x-, y-, and z-locations of the markers were recorded at 500Hz, the muscle activities at 2000Hz, and the data from the treadmill at 1000Hz. Furthermore, the database contains joint positions and joint trajectories derived from the marker positions via kinematic models of the human body. In our experiments, we use the marker locations, the derived locations of the joints and the muscle activities. We subsample all data to a common frame rate of 50Hz and transpose the x- and z-position of all markers such that the T12-marker (marker at the 12th thoracic vertebra) has $(x = 0, z = 0)$ in all frames. Note that in the HuMoD database, the x-axis points in the motion direction (i.e., along the treadmill), the y-axis points upwards and the z-axis forms a right-hand coordinate system towards the right side of the treadmill. We used walking motions at 1.0m/s, 1.5m/s, 2.0m/s, and running motions at 2.0m/s, 3.0m/s, and 4m/s, captured from one subject. For evaluating the trained model, we used a test data-set in which the subject transitions linearly from 0m/s up to 4m/s and back to 0m/s.

In the experiment, we compare the performance of subKKF, subKBF, and sparse Gaussian process (SGP) in restoring the marker and joint positions from the muscle activities. We learn all three models using the marker and joint positions as state variables (or outputs) $\boldsymbol{x}_i$ and the muscle activities as observations (or inputs) $\boldsymbol{y}_i$. We use a set of 2000 samples to learn the kernel matrices and a subset of 500 samples to define the subspace (or as inducing inputs). For subKKF and subKBF, we use a window size of 2. While we could easily carry out the optimization of the parameters for the subKKF and for the SGP, the optimization of the parameters for the subKBF was not feasible in a considerable amount of time.

Figure 2.10 depicts marker and joint positions of four exemplary postures together with the muscle activities during that period of time. The locations of the exemplary postures

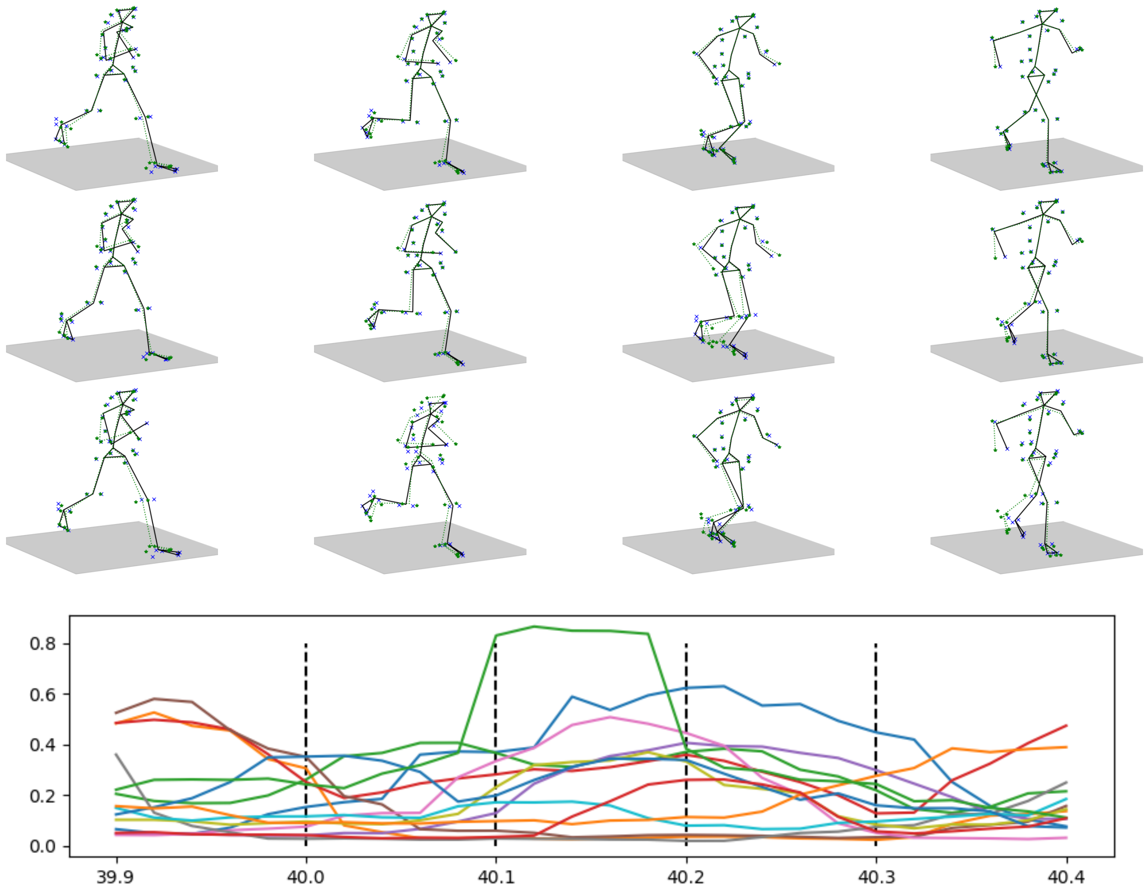


**Figure 2.10:** Example sequence of 4 postures and the measured muscle activities. The marker and skeleton in green depict the ground-truth, the estimates from the models are depicted in black/blue. The learned models estimate the marker and joint positions from the muscle activities. The first row shows the estimated positions from the subKKF, the second row shows the estimated positions from the subKBF and the third row shows the estimated positions from a sparse GP. For all three models, we use a sample set of 2000 samples and a sparse subset of 500 samples.

in the time line are depicted by vertical lines in the plot of the muscle activities. While this is only a qualitative example, it depicts how the subKKF outperforms the subKBF and the SGP in restoring the positions of the markers and the joints.

We compare the performance of subKKF, subKBF and SGP for different sizes of the subspace (inducing inputs). Figure 2.11 depicts the performance of subKKF and subKBF. The results of the SGP can be seen in Table 2.2 which are clearly worse in comparison to the filtering approaches which take the temporal correlation of the data into account. Furthermore, Table 2.2 also depicts the time consumption of subKKF and subKBF for filtering 100 test sequences of length 50. The gain in efficiency of the subKKF over the subKBF which is around the factor 100 can be seen clearly. In Figure 2.12, we depict the performance gain of the subKKF over the number of iterations of the CMA-ES optimizer. We see that in this case, the first 10 iterations yield a bigger jump in performance than the following 40 steps. However, from our experience, this is very specific to the problem and the initial

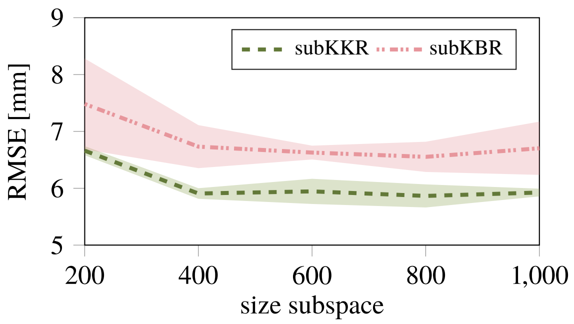


**Figure 2.11:** Performance of the subKKF and the subKBF on the HuMoD transition data for different sizes of the subspace.

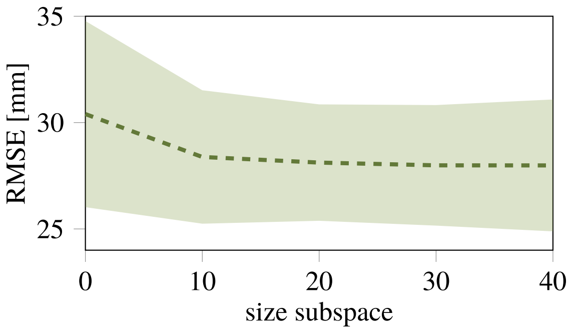


**Figure 2.12:** Performance of the subKKF on HuMoD test sequences after 0, 10, 20, 30 and 40 iterations of the optimizer.

| # | 200 | 400 | 600 | 800 | 1000 |
|---|---|---|---|---|---|
| **subKKF** | **6.66** | **5.91** | **5.94** | **5.87** | **5.92** |
| subKBF | 7.48 | 6.73 | 6.63 | 6.55 | 6.70 |
| SGP | 76.33 | 70.15 | 68.46 | 63.13 | 58.97 |
| **subKKF** | **0.61±0.10 s** | **1.71±0.17 s** | **3.84±0.40 s** | **7.21±0.85 s** | **11.72±0.41 s** |
| subKBF | 59±1.3 s | 170±11.3 s | 352±15 s | 620±15 s | 927±174 s |

**Table 2.2:** Top: performance of subKKF, subKBF, and SGP for the HuMoD transition data for different sizes of the subspace. Bottom: time consumptions of subKKF and subKBF for filtering 100 test sequences of length 50 from the HuMoD data set.

setting of the parameters, which were in this experiment already very close to the optimal parameters.

### 2.5.4 The Kernel Forward-Backward Smoother

Smoothing is in contrast to filtering a post-processing routine. While filtering refers to a routine where the current state is estimated recursively from all past observations, smoothing computes the best state estimates given all available observations from the past and the future. Hence, for a given time series of observations $[\boldsymbol{y}_1, \dots, \boldsymbol{y}_T]$, we want to obtain the belief $p(\boldsymbol{x}_t|\boldsymbol{y}_1, \dots, \boldsymbol{y}_T)$ for all $1 \leq t \leq T$.

One well-known and simple approach to smoothing is the forward-backward smoother. During a forward pass the standard filtering algorithm is applied to the observations. Afterwards, during the backward pass, an inverse filter is applied to the same time series of observations. The filter estimates of forward and backward pass are finally combined into the smoothed estimates. Since the information from the observation should be incorporated only once into the smoothed estimates, we need to combine the posterior estimates of the forward pass with the prior estimates of the backward pass (or vice versa). For the case of ordinary Kalman filters, the backward pass is hard to realize because of two problems. First, it requires an inverse model of the underlying system for the backward pass, and second, an initialization of the belief at the final state is necessary. These issues are however not applicable for the KKF as we can learn both, the inverse models and the embedding of initial distribution of the final states from data.

### Computing the Smoothed Belief State as a Weighted Average

Assuming that we have the a-posteriori belief states from the forward pass and the a-priori belief states from the backward pass as

$$\left\{\left(\mu^{+}_{f,1}, \mathcal{C}^{+}_{f,1}\right), \dots, \left(\mu^{+}_{f,T}, \mathcal{C}^{+}_{f,T}\right)\right\} \qquad \text{and} \qquad \left\{\left(\mu^{-}_{b,1}, \mathcal{C}^{-}_{b,1}\right), \dots, \left(\mu^{-}_{b,T}, \mathcal{C}^{-}_{b,T}\right)\right\}, \tag{2.94}$$

respectively, we can combine the mean maps into a smoothed belief state as the weighted average

$$\mu_{s,t} = \mathcal{Z}_{f,t}\mu^{+}_{f,t} + \mathcal{Z}_{b,t}\mu^{-}_{b,t}. \tag{2.95}$$

Since both, the estimator from the forward pass and the estimator from the backward pass, are unbiased, the weighting operators $\mathcal{Z}_{f,t}$ and $\mathcal{Z}_{b,t}$ need to satisfy $\mathcal{I} = \mathcal{Z}_{f,t} + \mathcal{Z}_{b,t}$ in order to get an unbiased estimator of the smoothed mean map, i.e.,

$$\begin{aligned}
\mathbb{E}\left[\varphi(X_t) - \mu^{s}_{t}\right] &\overset{!}{=} 0 &(2.96)\\
\mathbb{E}\left[\varphi(X_t) - \mathcal{Z}_{f,t}\mu^{f+}_{t} + \mathcal{Z}_{b,t}\mu^{b-}_{t}\right] &\overset{!}{=} 0 &(2.97)\\
\mathbb{E}\left[\varphi(X_t)\right] - \mathcal{Z}_{f,t}\mathbb{E}\left[\mu^{f+}_{t}\right] + \mathcal{Z}_{b,t}\mathbb{E}\left[\mu^{b-}_{t}\right] &\overset{!}{=} 0 &(2.98)\\
\mu_t - \left(\mathcal{Z}_{f,t} + \mathcal{Z}_{b,t}\right)\mu_t &\overset{!}{=} 0 &(2.99)\\
\Rightarrow \quad \mathcal{Z}_{f,t} + \mathcal{Z}_{b,t} &= \mathcal{I} &(2.100)
\end{aligned}$$

Thus, the weighting operators can be expressed by each other as $\mathcal{Z}_{b,t} = \mathcal{I} - \mathcal{Z}_{f,t}$ and vice versa. We substitute this representation back into the smoothing update in Equation 2.95 to obtain

$$\mu_{s,t} = \mathcal{Z}_{f,t}\mu^{+}_{f,t} + \left(\mathcal{I} - \mathcal{Z}_{f,t}\right)\mu^{-}_{b,t}. \tag{2.101}$$

### Finding the Optimal Weighting Operators

We obtain the optimal weighting operators by minimizing the squared error of the smoothing mean map which is equivalent to minimizing the trace of the smoothed covariance embedding operator $\mathcal{C}_{s,t}$, i.e.,

$$\begin{aligned}
\min \mathbb{E}\left[\left(\varphi(X_t) - \mu^{s}_{t}\right)^{\intercal}\left(\varphi(X_t) - \mu^{s}_{t}\right)\right] &= \min \mathbb{E}\left[\operatorname{Tr}\left(\varphi(X_t) - \mu^{s}_{t}\right)\left(\varphi(X_t) - \mu^{s}_{t}\right)^{\intercal}\right] &(2.102)\\
&= \min \operatorname{Tr} \mathcal{C}_{s,t}. &(2.103)
\end{aligned}$$

We can then use Equation 2.101 to rewrite the covariance operator as

$$\mathcal{C}_{s,t} = \mathbb{E}\left[\left(\varphi(X_t) - \mathcal{Z}_{f,t}\mu^+_{f,t} + \left(\mathcal{I} - \mathcal{Z}_{f,t}\right)\mu^-_{b,t}\right)\left(\dots\right)^\intercal\right] \tag{2.104}$$

$$= \mathbb{E}\left[\left(\mathcal{Z}_{f,t}(\epsilon_f - \epsilon_b) + \epsilon_b\right)\left(\dots\right)^\intercal\right], \tag{2.105}$$

where $\epsilon_f = \varphi(x_t) - \mu^+_{f,t}$ is the error of the a-posteriori estimate of the forward pass and $\epsilon_b = \varphi(x_t) - \mu^-_{b,t}$ is the error of the a-priori estimate of the backward pass and where we used the relation

$$\varphi(X_t) = \mathcal{I}\varphi(X_t) = (\mathcal{Z}_{f,t} + \mathcal{Z}_{b,t})\varphi(X_t) = (\mathcal{Z}_{f,t} + (\mathcal{I} - \mathcal{Z}_{f,t}))\varphi(X_t).$$

By expanding the square and since the cross-covariances of the errors from forward and the backward pass are zero (i.e., $\mathbb{E}[\epsilon_f\epsilon_b^\intercal] = 0$), we arrive at

$$\mathcal{C}_{s,t} = \mathbb{E}\left[\mathcal{Z}_{f,t}\left(\epsilon_f\epsilon_f^\intercal + \epsilon_b\epsilon_b^\intercal\right)\mathcal{Z}_{f,t}^\intercal - \mathcal{Z}_{f,t}\epsilon_b\epsilon_b^\intercal - \epsilon_b\epsilon_b^\intercal\mathcal{Z}_{f,t}^\intercal + \epsilon_b\epsilon_b^\intercal\right]. \tag{2.106}$$

Lastly, we can take the derivative and set it to zero to obtain the optimal $\mathcal{Z}_{f,t}$ as

$$0 \overset{!}{=} \frac{\partial \operatorname{Tr}\mathcal{C}_{s,t}}{\partial \mathcal{Z}_{f,t}} = 2\mathbb{E}\left[\mathcal{Z}_{f,t}\left(\epsilon_f\epsilon_f^\intercal + \epsilon_b\epsilon_b^\intercal\right) - \epsilon_b\epsilon_b^\intercal\right] \tag{2.107}$$

$$0 \overset{!}{=} \mathcal{Z}_{f,t}\left(\mathbb{E}\left[\epsilon_f\epsilon_f^\intercal\right] + \mathbb{E}\left[\epsilon_b\epsilon_b^\intercal\right]\right) - \mathbb{E}\left[\epsilon_b\epsilon_b^\intercal\right] \tag{2.108}$$

$$0 \overset{!}{=} \mathcal{Z}_{f,t}\left(\mathcal{C}^+_{f,t} + \mathcal{C}^-_{b,t}\right) - \mathcal{C}^-_{b,t} \tag{2.109}$$

$$\mathcal{Z}_{f,t} = \mathcal{C}^-_{b,t}\left(\mathcal{C}^+_{f,t} + \mathcal{C}^-_{b,t}\right)^{-1}. \tag{2.110}$$

From the condition on the weighting operators stated in Equation 2.100, it furthermore follows that $\mathcal{Z}_{b,t} = \mathcal{C}^-_{f,t}\left(\mathcal{C}^+_{f,t} + \mathcal{C}^-_{b,t}\right)^{-1}$.

### Smoothing the Covariance Embedding Operator

Taking the representation of the smoothed covariance in Equation 2.106 and substituting the covariance operators and the optimal weighting operator $\mathcal{Z}_{f,t}$ gives us the following smoothed covariance operator

$$\mathcal{C}_{s,t} = \mathcal{C}^-_{b,t} - \mathcal{C}^-_{b,t}\left(\mathcal{C}^+_{f,t} + \mathcal{C}^-_{b,t}\right)^{-1}\mathcal{C}^-_{b,t} \tag{2.111}$$

$$= \mathcal{C}^-_{b,t} - \left(\mathcal{I} - \mathcal{C}^+_{f,t}\left(\mathcal{C}^+_{f,t} + \mathcal{C}^-_{b,t}\right)^{-1}\right)\mathcal{C}^-_{b,t} \tag{2.112}$$

$$= \mathcal{C}^+_{f,t}\left(\mathcal{C}^+_{f,t} + \mathcal{C}^-_{b,t}\right)^{-1}\mathcal{C}^-_{b,t} \tag{2.113}$$

From the optimal solution of the weighting operator, we can now see that the smoothing update of the covariance embedding operator can be expressed as

$$\mathcal{C}_{s,t} = \mathcal{Z}_{b,t}\mathcal{C}^{-}_{b,t} = \mathcal{Z}_{f,t}\mathcal{C}^{+}_{f,t} \tag{2.114}$$

In the following section, we will show how the smoothing update can be expressed with finite samples using vector/matrix operations.

## The Empirical Kernel Forward-Backward Smoother

We assume that we are given the weight vectors and weight matrices from the forward and the backward pass as

$$\{(\boldsymbol{m}^{+}_{f,1}, \boldsymbol{S}^{+}_{f,1}), \ldots, (\boldsymbol{m}^{+}_{f,T}, \boldsymbol{S}^{+}_{f,T})\} \qquad \text{and} \qquad \{(\boldsymbol{m}^{-}_{b,1}, \boldsymbol{S}^{-}_{b,1}), \ldots, (\boldsymbol{m}^{-}_{b,T}, \boldsymbol{S}^{-}_{b,T})\},$$

respectively. Since the weighting operator $\hat{\mathcal{Z}}_{b,t}$ can be expressed by $\hat{\mathcal{Z}}_{f,t}$ and vice versa, we only need to compute one of the weighting operators which we choose to be $\hat{\mathcal{Z}}_{f,t}$. We add the identity operator with a small scalar $\gamma$ in the inverse to improve the numerical stability and obtain

$$\begin{aligned}
\hat{\mathcal{Z}}_{f,t} &= \hat{\mathcal{C}}^{-}_{b,t}\left(\hat{\mathcal{C}}^{+}_{f,t} + \hat{\mathcal{C}}^{-}_{b,t} + \gamma\mathcal{I}\right)^{-1} && (2.115)\\
&= \boldsymbol{\Upsilon}_x \boldsymbol{S}^{-}_{b,t} \boldsymbol{\Upsilon}_x^{\intercal}\left(\boldsymbol{\Upsilon}_x\left(\boldsymbol{S}^{+}_{f,t} + \boldsymbol{S}^{-}_{b,t}\right)\boldsymbol{\Upsilon}_x^{\intercal} + \gamma\mathcal{I}\right)^{-1} && (2.116)\\
&= \boldsymbol{\Upsilon}_x \boldsymbol{S}^{-}_{b,t}\left(\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Upsilon}_x\left(\boldsymbol{S}^{+}_{f,t} + \boldsymbol{S}^{-}_{b,t}\right) + \gamma\boldsymbol{I}_n\right)^{-1}\boldsymbol{\Upsilon}_x^{\intercal} && (2.117)\\
&= \boldsymbol{\Upsilon}_x \underbrace{\boldsymbol{S}^{-}_{b,t}\left(\boldsymbol{K}_{xx}\left(\boldsymbol{S}^{+}_{f,t} + \boldsymbol{S}^{-}_{b,t}\right) + \gamma\boldsymbol{I}_n\right)^{-1}}_{Z_{f,t}}\boldsymbol{\Upsilon}_x^{\intercal}, && (2.118)
\end{aligned}$$

where we use the matrix identity $\boldsymbol{A}(\boldsymbol{B}\boldsymbol{A}+\boldsymbol{I})^{-1} = (\boldsymbol{A}\boldsymbol{B}+\boldsymbol{I})^{-1}\boldsymbol{A}$ and defined the finite weighting matrix $\boldsymbol{Z}_{f,t}$. With this weighting matrix we can now combine the mean maps of the forward and the backward pass as

$$\begin{aligned}
\hat{\mu}_{s,t} &= \hat{\mathcal{Z}}_{f,t}\hat{\mu}^{+}_{f,t} + \left(\mathcal{I} - \hat{\mathcal{Z}}_{f,t}\right)\hat{\mu}^{-}_{b,t} && (2.119)\\
&= \boldsymbol{\Upsilon}_x \boldsymbol{Z}_{f,t}\boldsymbol{\Upsilon}_x^{\intercal}\hat{\mu}^{+}_{f,t} + \left(\mathcal{I} - \boldsymbol{\Upsilon}_x \boldsymbol{Z}_{f,t}\boldsymbol{\Upsilon}_x^{\intercal}\right)\hat{\mu}^{-}_{b,t} && (2.120)\\
&= \boldsymbol{\Upsilon}_x \boldsymbol{Z}_{f,t}\boldsymbol{K}_{xx}\boldsymbol{m}^{+}_{f,t} + \boldsymbol{\Upsilon}_x\boldsymbol{m}^{-}_{b,t} - \boldsymbol{\Upsilon}_x \boldsymbol{Z}_{f,t}\boldsymbol{K}_{xx}\boldsymbol{m}^{-}_{b,t} && (2.121)\\
\boldsymbol{\Upsilon}_x\boldsymbol{m}^{s}_{t} &= \boldsymbol{\Upsilon}_x\left(\boldsymbol{m}^{-}_{b,t} + \boldsymbol{Z}_{f,t}\boldsymbol{K}_{xx}\left(\boldsymbol{m}^{+}_{f,t} - \boldsymbol{m}^{-}_{b,t}\right)\right). && (2.122)
\end{aligned}$$

And similarly we can obtain the smoothed estimate of the covariance operator as

$$\begin{aligned}
\hat{\mathcal{C}}^{s}_{t} &= \hat{\mathcal{Z}}_{f,t}\hat{\mathcal{C}}^{+}_{f,t} && (2.123)\\
&= \boldsymbol{\Upsilon}_x \boldsymbol{Z}_{f,t}\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Upsilon}_x\boldsymbol{S}^{+}_{f,t}\boldsymbol{\Upsilon}_x^{\intercal} && (2.124)\\
\boldsymbol{\Upsilon}_x\boldsymbol{S}^{s}_{t}\boldsymbol{\Upsilon}_x^{\intercal} &= \boldsymbol{\Upsilon}_x \boldsymbol{Z}_{f,t}\boldsymbol{K}_{xx}\boldsymbol{S}^{+}_{f,t}\boldsymbol{\Upsilon}_x^{\intercal} && (2.125)
\end{aligned}$$

A concise description of the kernel forward-backward smoothing algorithm can be found in Algorithm 3.

### Initialization of the Backward Kernel Kalman Filter

A critical aspect of the classical forward-backward smoothing algorithm is the initialization of the belief state for the backward pass. Often the distribution over the initial state is well known but not a distribution over the terminal state, where it is often not even clear how a terminal state is defined. For the backward kernel Kalman filter, two approaches can be used to initialize the belief state. The first approach assumes that we have multiple episodes in the training data, where each episode terminates in a terminal state of the system. We can then compute the initialization for the backward pass analogously to the initialization for the KKF described in Section 2.5.1. The second approach simply assumes that the system has a stationary distribution which is covered by the training data. The initialization is then the embedding of the distribution over all the samples in the training set.

### 2.5.5 The Subspace Kernel Forward-Backward Smoother

If we use the subspace kernel Kalman filter to perform the forward and the backward pass, we obtain as outcome the subspace projections $\boldsymbol{n}_t$ of the mean map and $\boldsymbol{P}_t$ covariance embedding instead of the weight vectors $\boldsymbol{m}_t$ and weight matrices $\boldsymbol{S}_t$, respectively. To perform smoothing on these subspace projections, we need to find the weighting matrices for the smoothing update analog to Equation 2.101. Though, as the representation is already in a finite domain, we can directly apply the optimal solution found in Equation 2.110 to the subspace projections of the covariance operator. Hence, the weighting matrix for the subspace kernel forward-backward smoother (subKFBS) becomes

$$\boldsymbol{Z}^S_{f,t} = \boldsymbol{P}^-_{b,t}\left(\boldsymbol{P}^+_{f,t} + \boldsymbol{P}^-_{b,t}\right)^{-1} \tag{2.126}$$

From here, we can obtain the equations for the smoothing update of the subspace kernel forward-backward smoother easily by

$$\boldsymbol{n}_{s,t} = \boldsymbol{Z}^S_{f,t}\boldsymbol{n}^+_{f,t} + \left(\boldsymbol{I} - \boldsymbol{Z}^S_{f,t}\right)\boldsymbol{n}^-_{b,t}, \qquad \text{and} \tag{2.127}$$

$$\boldsymbol{P}_{s,t} = \boldsymbol{Z}^S_{f,t}\boldsymbol{P}^+_{f,t}. \tag{2.128}$$

Algorithm 4 gives a compact description of the subKFBS.

### 2.5.6 Experimental Evaluation of the Kernel Forward Backward Smoother

We evaluate the kernel forward backward smoother with two experiments. In the first experiment on data from a simulated pendulum, we show the performance gain of the KFBS over the KKF. In the second experiment, we apply the KFBS on data of a table tennis ball recorded with a camera-based tracking system and show how the KFBS and subKFBS

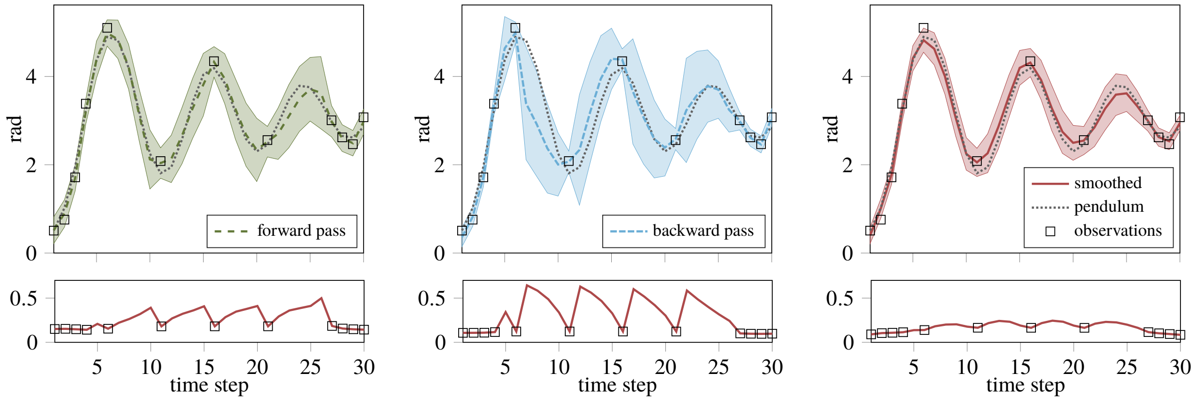


**Figure 2.13:** Qualitative comparison of the forward and the backward pass to the smoothed estimates of the KFBS on a simulated pendulum. The upper plots show the mean and variance output of the filter/smoother, the lower plots show the profiles of the standard deviation. While the forward pass already yields good estimates in the first half of the time series, the smoother incorporates the good estimates from the backward pass in the second half and outperforms the filters. In addition, the smoother yields a more confident about its estimate.

are able to restore the full trajectory of the ball while only having observations at the first four and at the last time step.

## Pendulum

We simulate a pendulum similar to the one from Section 2.5.3, however we initialize the pendulum in the range $[-0.25\pi, 0.25\pi]$ and with a angular velocity sampled from the range $[-2\frac{\pi}{s}, 2\frac{\pi}{s}]$. During the simulation, we apply Gaussian process noise with $\sigma = 0.01$ and as observations we use the angular displacement and add Gaussian observation noise with $\sigma = 0.2$. To learn the KFBS models, we sample 100 episodes with each 30 steps where a step corresponds to 0.1 second. The training samples are 200 windows of four observations, which we select by the kernel activation heuristic explained in Section 2.3.1. To find the optimal parameters, we apply CMA-ES [35] where we use the negative log-likelihood of the ground-truth to the smoothed estimate as optimality criterion. During the optimization, we use a test data set of 10 episodes, where we still observe at each time step. Later, we evaluate the smoothing performance on an evaluation data set where we do not observe at each time step but only at $t = [1-4, 6, 11, 16, 21, 27-30]$. This optimization procedure yielded better results than directly optimizing with only partial observations.

In Figure 2.13, we show a qualitative comparison of the forward and the backward pass to the smoother. The results are as expected: the forward pass yields better results in the first half of the episode, and the backward pass yields better results in the second half. The smoother combines both estimates and outperforms the filter results. The smoothing can also be observed in the profiles of the standard deviation. While the variance from the filters increases at each time step without observation until the next measurement, the variance of the smoother is much smaller and only rises slightly between the observations.

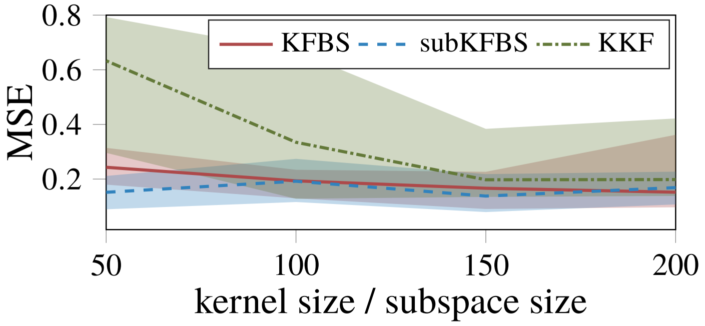


**Figure 2.14:** The KFBS and the sub KFBS outperform the KKF clearly for small kernel sizes but also with more samples in the gram matrices. The task was to estimate the state of a pendulum from noisy partial observations. Depicted are the median and the $[0.15, 0.85]$-quantiles of the MSE over 20 repetitions.

In Figure 2.14, we compare the performance of a standard KKF to the KFBS and the subKFBS for different kernel sizes on the same state estimation task for a simulated pendulum. The subKFBS has been learned with 300 samples in the full data set. Depicted are the median and the $[0.15, 0.85]$-quantiles of the MSE over 20 repetitions. The KFBS and the subKFBS clearly outperform the KKF for small kernel sizes (50, 100) and also yield better results for larger kernel sizes (i.e., 150 and 200 samples). The subKFBS yields slightly better results than the KFBS. In addition, we see from the quantiles of the MSE that the KFBS and the subKFBS have a more stable behavior in the optimization process than the KKF.

### Tabletennis

In a second experiment, we perform smoothing on observations of a table tennis ball [30]. The data set contains 54 trajectories of a table tennis ball tracked with a camera system, where each trajectory contains 51 observations which are recorded with a frequency of 100 Hz. We train the subKFBS with the data of 34 trajectories and use 10 trajectories for optimizing the parameters using CMA-ES [35]. The remaining 10 trajectories are used for evaluating the results. For the smoothing task, the ball has been observed at the first five time steps and then again at the last time step.

Figure 2.15 shows qualitative examples of smoothed trajectories using the subKFBS in comparison the output of the subKKF. Here, we used data windows of size 4 and learned the models with 300 samples in the training data set and 100 samples in the subset. We optimized the regularization parameters and all bandwidths with CMA-ES [35]. The plot shows how the subKFBS can estimate accurately the path of the ball only from observations at the beginning and at the end of the trajectory, while the subKKF diverges from the actual trajectory over time. Especially the impact position of the ball on the table can be estimated much better by the subKFBS than by the subKKF.

We also compare the KFBS to the subKFBS for different kernel sizes with the same smoothing task on recorded table tennis ball data. Figure 2.16 shows a comparison of the MSE, depicting the median and the $[0.05, 0.95]$-quantiles over 20 repetitions. The KFBS has been learned with a varying kernel size of 50, 100, 150, and 200 samples. The subKFBS uses the same number of samples to span the subspace but learns the models

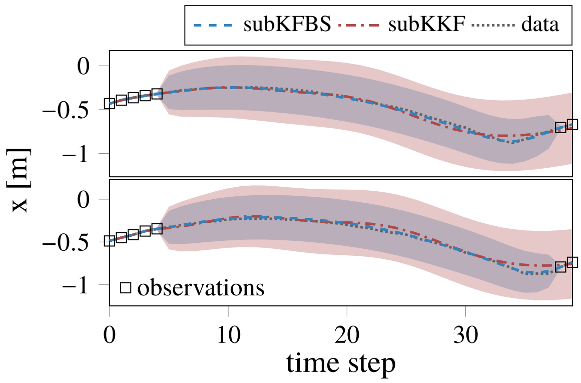


**Figure 2.15:** In comparison to the KKF, the KFBS is able to reconstruct the trajectories of a table tennis ball from observations at the first five and at the last two time steps. The plot shows the z-coordinate of two trajectories of a table tennis ball recorded with a camera-based tracking system. We learned the subKFBS/subKKF with 100 samples in the subspace and 300 points in the training set.

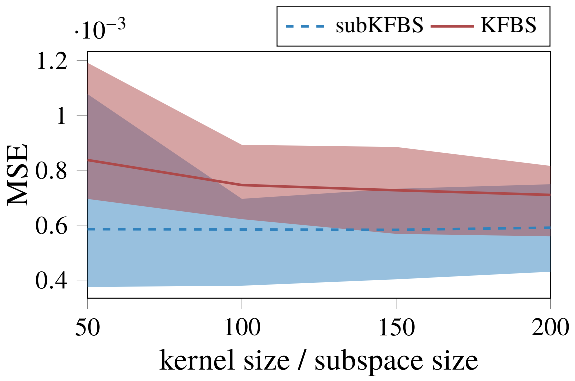


**Figure 2.16:** The subKFBS performs better than the KFBS in the table tennis ball smoothing task. This difference in the MSE between the estimates and the noisy recorded data is more prevalent for small kernel sizes and decreases with the number of samples in the Gram matrices. Depicted is the median and the $[0.05, 0.95]$-quantiles of the MSE over 20 repetitions.

always with 400 samples in the full training set. While the subKFBS outperforms the KFBS for all kernel sizes, the KFBS achieves a similar performance to the KFBS when learned with 200 samples.

## 2.6 Conclusion & Future Work

In this paper, we have presented the kernel Kalman rule (KKR) as an alternative to the kernel Bayes' rule (KBR) in the framework for nonparametric inference [103]. In contrast to the KBR, the KKR is computationally more efficient, numerically more stable and follows from a clear optimization objective. We have further combined the KKR as Bayesian update with the kernel sum rule to formulate the kernel Kalman filter (KKF). The kernel Kalman filter can be applied to nonlinear state estimation tasks as it learns the probabilistic transition and observation dynamics as linear functions on embeddings of the belief state in high-dimensional Hilbert spaces from data. In difference to existing kernel Kalman filter formulations, the KKF also provides a more general formulation that is much closer to the original Kalman filter equations and can also be applied to partially observable systems.

While the KKF can be applied to state estimation and prediction based on past observations, we extend this work by introducing the kernel forward backward smoother (KFBS) which infers the belief state from current, past, and future information. We have shown in an experimental evaluation how this additional information leads to a performance gain of the KFBS over the KKF. As kernel methods typically scale poorly with the number of data points in the kernel matrices, we have introduced a sparsification technique that leverages from the full training set while representing the embeddings only with a small subset of the data. This technique leads to significant gains of the computational efficiency while yielding similar or even slightly better results than whithout the sparsification.

We have shown that it is possible to learn the kernel Kalman rule and other kernelized inference methods also from partial observations if sliding windows of the time series provide sufficient statistics. However in future work, we want to concentrate on learning the transition dynamics in the RKHS with an expectation-maximization algorithm in case of missing information about the latent state as we think that this leads to better models of the dynamics and also improves the accuracy of the estimated variance.

```
Algorithm 3: The Kernel Forward-Backward Smoother
input: data set $\mathcal{D}_{\tilde{X}XX'Y} = \{(\tilde{\boldsymbol{x}}_1, \boldsymbol{x}_1, \boldsymbol{x}'_1, \boldsymbol{y}_1), \dots, (\tilde{\boldsymbol{x}}_n, \boldsymbol{x}_n, \boldsymbol{x}'_n, \boldsymbol{y}_n)\}$,
       kernel functions $k$ and $g$, regularization parameters $\lambda$ and $\kappa$,
       let $\boldsymbol{X}$ be the matrix of all $x_i$ as columns, $\tilde{\boldsymbol{X}}$, $\boldsymbol{X}'$ and $\boldsymbol{Y}$ analogously,
       let $\boldsymbol{X}_0$ be the matrix of all data points used to initialize the forward pass
       let $\boldsymbol{X}_T$ be the matrix of all data points used to initialize the backward pass
learn the forward filter
See Algorithm 1 with data matrices $\boldsymbol{X}_0$, $\tilde{\boldsymbol{X}}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$.
learn the backward filter
See Algorithm 1 with data matrices $\boldsymbol{X}_T$, $\boldsymbol{X}$, $\boldsymbol{X}'$, and $\boldsymbol{Y}$, note that you need to learn
  the transition model from $\boldsymbol{X}'$ to $\boldsymbol{X}$.
apply forward filter
Compute filtered estimates $\left\{\left(\boldsymbol{m}^+_{f,1}, \boldsymbol{S}^+_{f,1}\right), \dots, \left(\boldsymbol{m}^+_{f,T}, \boldsymbol{S}^+_{f,T}\right)\right\}$
apply backward filter and compute smoothed estimates
loop
    if observation $\boldsymbol{y}_t$ available then
        compute kernel Kalman gain
        $\boldsymbol{Q}_{b,t} = \boldsymbol{S}^-_{b,t} \boldsymbol{O}^\intercal (\boldsymbol{G}_{yy} \left(\boldsymbol{O}\boldsymbol{S}^-_{b,t}\boldsymbol{O}^\intercal + \boldsymbol{R}_b\right) + \kappa \boldsymbol{I})^{-1}$
        innovation update
        $\boldsymbol{m}^+_{b,t} = \boldsymbol{m}^-_{b,t} + \boldsymbol{Q}_{b,t}(\boldsymbol{g}_{y_t} - \boldsymbol{G}_{yy}\boldsymbol{O}\boldsymbol{m}^-_t)$
        $\boldsymbol{S}^+_{b,t} = \boldsymbol{S}^-_{b,t} - \boldsymbol{Q}_{b,t}\boldsymbol{G}_{yy}\boldsymbol{O}\boldsymbol{S}^-_{b,t}$
    transition update
    $\boldsymbol{m}^-_{b,t-1} = \boldsymbol{T}_b \boldsymbol{m}^+_{b,t}, \quad \boldsymbol{S}^-_{b,t-1} = \boldsymbol{T}_b \boldsymbol{S}^+_{b,t} \boldsymbol{T}^\intercal_b + \boldsymbol{V}_b$
    compute smoothed estimate
    $\boldsymbol{Z}_{f,t} = \boldsymbol{S}^-_{b,t} \left(\boldsymbol{K}_{xx} \left(\boldsymbol{S}^+_{f,t} + \boldsymbol{S}^-_{b,t}\right) + \gamma \boldsymbol{I}_n\right)^{-1}$
    $\boldsymbol{m}^s_t = \boldsymbol{m}^-_{b,t} + \boldsymbol{Z}_{f,t}\boldsymbol{K}_{xx}\left(\boldsymbol{m}^+_{f,t} - \boldsymbol{m}^-_{b,t}\right)$
    $\boldsymbol{S}^s_t = \boldsymbol{Z}_{f,t}\boldsymbol{K}_{xx}\boldsymbol{S}^+_{f,t}$
    project into state space
    $\boldsymbol{\eta}^s_t = \boldsymbol{X}\boldsymbol{O}\boldsymbol{m}^s_t, \quad \boldsymbol{\Sigma}^s_t = \boldsymbol{X}\boldsymbol{O}\boldsymbol{S}^s_t\boldsymbol{O}^\intercal\boldsymbol{X}^\intercal$
```

**Algorithm 4:** The Subspace Kernel Forward-Backward Smoother

```
input: data set $\mathcal{D}_{\tilde{X}XX'Y} = \{(\tilde{\boldsymbol{x}}_1, \boldsymbol{x}_1, \boldsymbol{x}'_1, \boldsymbol{y}_1), \dots, (\tilde{\boldsymbol{x}}_n, \boldsymbol{x}_n, \boldsymbol{x}'_n, \boldsymbol{y}_n)\}$,
       kernel functions $k$ and $g$, regularization parameters $\lambda$ and $\kappa$,
       let $\boldsymbol{X}$ be the matrix of all $x_i$ as columns, $\tilde{\boldsymbol{X}}$, $\boldsymbol{X}'$ and $\boldsymbol{Y}$ analogously,
       let $\boldsymbol{X}_0$ be the matrix of all data points used to initialize the forward pass
       let $\boldsymbol{X}_T$ be the matrix of all data points used to initialize the backward pass

learn forward and backward filter
See Algorithm 1 with data matrices $\boldsymbol{X}_0$, $\boldsymbol{X}$, $\boldsymbol{X}'$, and $\boldsymbol{Y}$ for the forward filter and data
  matrices $\tilde{\boldsymbol{X}}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$ for the backward filter. Note that you need to learn the
  transition model for the backward filter from $\boldsymbol{X}$ to $\tilde{\boldsymbol{X}}$.

apply forward filter
Compute filtered estimates $\left\{\left(\boldsymbol{n}^+_{f,1}, \boldsymbol{P}^+_{f,1}\right), \dots, \left(\boldsymbol{n}^+_{f,T}, \boldsymbol{P}^+_{f,T}\right)\right\}$

apply backward filter and compute smoothed estimates
loop
    if observation $\boldsymbol{y}_t$ available then
        compute subspace kernel Kalman gain
        $\boldsymbol{Q}^S_t = \boldsymbol{P}^-_{b,t} \left(\boldsymbol{O}^S\right)^\intercal \left(\boldsymbol{K}^\intercal_{x\tilde{x}} \boldsymbol{G}_{yy} \boldsymbol{K}_{x\tilde{x}} \boldsymbol{O}^S \boldsymbol{P}^-_{b,t} \left(\boldsymbol{O}^S\right)^\intercal + \kappa \boldsymbol{I}_m\right)^{-1} \boldsymbol{K}^\intercal_{x\tilde{x}}$
        innovation update
        $\boldsymbol{n}^+_{b,t} = \boldsymbol{n}^-_{b,t} + \boldsymbol{Q}^S_t \left(\boldsymbol{g}_{\boldsymbol{y}_t} - \boldsymbol{G}_{yy} \boldsymbol{K}_{x\tilde{x}} \boldsymbol{O}^S \boldsymbol{n}^-_{b,t}\right), \quad \boldsymbol{P}^+_{b,t} = \boldsymbol{P}^-_{b,t} - \boldsymbol{Q}^S_t \boldsymbol{G}_{yy} \boldsymbol{K}_{x\tilde{x}} \boldsymbol{O}^S \boldsymbol{P}^-_{b,t}$
    transition update
    $\boldsymbol{n}^-_{b,t-1} = \boldsymbol{T}^S \boldsymbol{n}^+_{b,t}, \quad \boldsymbol{P}^-_{b,t-1} = \boldsymbol{T}^S \boldsymbol{P}^+_{b,t} \left(\boldsymbol{T}^S\right)^\intercal + \boldsymbol{V}^S$
    compute smoothed estimate
    $\boldsymbol{Z}^S_{f,t} = \boldsymbol{P}^-_{b,t} \left(\boldsymbol{P}^+_{f,t} + \boldsymbol{P}^-_{b,t}\right)^{-1}$
    $\boldsymbol{n}_{s,t} = \boldsymbol{Z}^S_{f,t} \boldsymbol{n}^+_{f,t} + \left(\boldsymbol{I}_m - \boldsymbol{Z}^S_{f,t}\right) \boldsymbol{n}^-_{b,t}, \quad \boldsymbol{P}_{s,t} = \boldsymbol{Z}^S_{f,t} \boldsymbol{P}^+_{f,t}$
    project into state space
    $\boldsymbol{\eta}_{s,t} = \boldsymbol{X} \boldsymbol{K}_{x\tilde{x}} \boldsymbol{O}^S \boldsymbol{n}_{s,t}, \quad \Sigma_{s,t} = \boldsymbol{X} \boldsymbol{K}_{x\tilde{x}} \boldsymbol{O}^S \boldsymbol{P}_{s,t} \boldsymbol{O}^S \boldsymbol{K}^\intercal_{x\tilde{x}} \boldsymbol{X}^\intercal$
```

# 3 Learning Swarm Policies for Pushing Objects

In the previous chapter, I have proposed novel methods for state estimation that learn the required distributions from data. These methods are based on non-parametric Hilbert space embeddings of probability distributions. A finite set of samples from the probability distributions can be used to compute empirical estimators of the embeddings. Furthermore, the kernel trick allows to compute the inner product of two embeddings by only evaluating the kernel function on such finite sample sets. Both of these insights allow to develop algorithms based on infinite dimensional feature mappings which eventually only require manipulations of finite vectors and matrices.

In this chapter, I investigate the use of RKHS embeddings as a representation for a set of homogeneous observations. A set of homogeneous observations can be seen as a set of samples from a generative distribution. Using an RKHS embedding to represent this generative distribution abstracts from the representation by the individual samples. Such an abstraction is beneficial since the representation by the individual samples is not trivial without assuming an order of the samples and a specific cardinality of the set. Obtaining estimators for the Hilbert space embeddings from the set of samples is straight-forward. Furthermore, the squared difference of two RKHS embeddings can be computed by inner products and hence kernel evaluations of the samples. I use the squared difference as a distance between two sets of samples and use this distance to define a kernel function.

In particular, I target the problem of learning policies for a swarm of robots. The state of a swarm with identical agents can be considered as such a set of homogeneous observations. Using the swarm kernel which I present in this chapter, allows to learn policies that can deal with a variable number of agents in the swarm. Furthermore, the state space in which the policy is learned is significantly smaller as the order of the agents in the observation is not important.

## 3.1 Introduction

An important characteristic of swarms is that the abilities of the collective are usually much larger than the abilities of the individual. Examples are large structures built by ants or termites or the defense behavior of fish schools or bees. This synergy effect which is also called superadditivity (‘the entire team should be able to achieve much more than individual robots working alone’ [74]), is a main principle that swarm robotics aims to exploit. The field of nano-robotics which makes use of biological or synthetic molecular machines is one particularly interesting area of research. An example for such biological molecular machines are flagellated bacteria, i.e., bacteria that are propelled by a filament attached to a rotating molecular motor [64]. While the flagella motors provide propulsion, magnetotactic bacteria, for example, have a chain of ferromagnetic particles in their body that serves as an interface for controlling their orientation using a weak magnetic field [62].

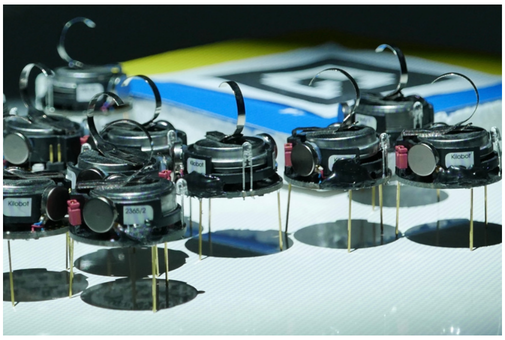


**Figure 3.1:** Kilobots pushing an object in an assembly task. The robots have a diameter of 3.3cm, the object in the background is a square of 15cm width.

In contrast to traditional robotics, which usually is based on robust machines with sufficient sensory equipment, swarm robots are usually simple agents with limited actuation and sensors. Instead, robotic swarms leverage from the high redundancy and the distributed nature of their hardware. Because state-of-the-art learning algorithms usually rely on computational powerful machines, their application to learn a behavior directly on swarm robots is limited. Learning a policy that externally guides a swarm to achieve a complex behavior is a feasible way to overcome this limitation. In this paper, we consider the task of autonomous object assembly with robot swarms. A swarm of Kilobots [83] has been used in an object assembly experiment [85], where a human operator controls a light source as global input signal to the swarm. Formulating the control rules to automate the assembly process is, however, a hard task.

Motivated by this application, we present an approach based on policy search to find a control strategy for the external signal. We split the assembly process in two subtasks: generating a top-level assembly plan using simple planning strategies, and learning a low-level object movement policy. The assembly plan encodes way points for each object while the object movement policy controls the trajectory execution by guiding the Kilobots with the light source. In this study, we treat the assembly plan as given and only learn object movement policies through policy search.

Learning to push an object is a complex task as we have to coordinate a large number of agents, which results in many state variables. While we need information about the configuration of the swarm (e.g., the positions of the agents) in our state representation, it is of no importance which individuals of the swarm are at which positions. Furthermore, our policy should be independent of the number of agents participating in pushing the object. Hence, instead of representing the state by the positions of every agent, we represent it as a distribution over the agent locations which we embed into a reproducing kernel Hilbert space [100]. This allows us to compare the swarm configurations independently of the number of individuals and of their specific locations. Our reinforcement learning algorithm is based on the recently introduced actor-critic relative entropy policy search (AC-REPS) algorithm [116] and learns a non-parametric Gaussian Process (GP) policy for controlling the light source. We evaluate our approach in simulation on different assembly tasks with different object shapes. Additionally, we demonstrate that the learned policies can be transferred to the real Kilobots which shows that the learning process is robust enough to allow a direct transfer from simulation to the real world. Parts of this work have been presented by the authors in [27, 25].

## 3.2 Related Work

While swarm robotics have been studied over the last three to four decades, using machine learning techniques to control robot swarms is a very recent field of research. In contrast to our approach, related work often directly learns the policies of the agents instead of a policy for a common control signal. For example, [51, 55] both learn actor and critic functions based on feature mappings using fuzzy-nets. The authors assume that the state is fully observable to the critic and the actor. In contrast, [60] proposes a multi-agent learning approach based on deep Q-learning in which only the critic has access to the full information about the state, while the actor has local observations. In [79], a method for multi-robot learning based on particle swarm optimization is presented. Each robot acts as a particle that rolls out a certain set of controller parameters. After each iteration the best performing parameters are shared with the robots in the neighborhood. The particle swarm approach is furthermore compared to genetic algorithms in [78]. However, this approach requires that each agent is able to asses the quality of the action it has performed and communicate the result with its neighbors.

The significant difference of our method is that we do not learn the policies for the individual agents. Instead, we assume that the same behavior runs on each agent and a desired swarm behavior is achieved by learning the controller for the global input. This setup—simple policy on the robots, complex control using an external signal—allows to use a much simpler hardware for the agents. The Kilobots, for example, can only sense the ambient light and communication is limited to robots in the close neighborhood. Evaluating a policy on the agent or communicating values between a global critic and the agents would be very difficult. In [61], a swarm of flagellated magnetotactic bacteria was used to to build a pyramid of building blocks in the micrometer range. The bacteria have a flagatella-based propulsion motor and their direction is controlled by a magnetic field that acts on nanoparticles in the cellular body. In the pyramid-building experiment the magnetic lines are controlled according to a planned trajectory to move the swarm. In [92] a control method for a swarm of phototactic agents is presented. The paper proposes a set of different PD controllers for manipulating an object with different goals (i.e., rotating, or translating the object, or a combination of both).

## 3.3 Preliminaries

This section provides a short discussion of the policy search method we use for learning the low-level object movement policy and gives a quick overview of the embeddings of distributions which we use to represent the state of the swarm. Additionally, we will depict the planning strategies for generating swarm and object trajectories from the high-level assembly policy.

### 3.3.1 Actor-Critic Relative Entropy Policy Search

We use actor-critic relative entropy policy search (AC-REPS) [116] to learn a continuous, non-parametric, probabilistic policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ from samples. This model-free reinforcement learning algorithm is based on relative entropy policy search (REPS) [76] and consists of three steps:

1. Least-squares temporal policy iteration (LSPI) is used to estimate the Q-function from the observed samples [56, 9, 40].

2. REPS optimizes the policy with respect to the expected Q-function while staying close to the old policy [76]. For the sample-based case this results in a discrete distribution over the state-action samples.

3. A sparse weighted Gaussian process learns the new continuous policy by regressing from states to actions using the weights obtained from REPS [41].

Additionally, these three steps can be iterated until convergence of the resulting policy for a given SARS sample set similar to a policy iteration approach. In the next paragraphs, we will discuss these three steps in more detail. Note that they assume some feature mapping of either a state or of a state-action pair. While AC-REPS is applicable to a wide range of problems, naturally, the feature mapping is very specific to the problem domain. We will discuss the feature mapping that we have used for the domain of swarm robotics later in Section 3.4.1.

### Least-Squares Policy Iteration

We want to estimate the Q-function from a data set of tuples $\mathcal{D} = \{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t)\}_{t=0}^{T}$ sampled from the environment. These tuples consist of the state $\boldsymbol{s}_t$ and the action $\boldsymbol{a}_t$ taken by the agent at time $t$ which results in a reward $r_t$ and the transition to the next state $\boldsymbol{s}'_t$. Given a feature mapping $\boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})$, the Q-function can be approximated as a linear function in feature space $Q(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})^\intercal \boldsymbol{\theta}$, where $\boldsymbol{\theta}$ are the parameters of the function. We write the features of the state-action pairs as matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\boldsymbol{s}_0, \boldsymbol{a}_0), \dots, \boldsymbol{\phi}(\boldsymbol{s}_T, \boldsymbol{a}_T)]^\intercal$ and the rewards $r_t$ as vector $\boldsymbol{R} = [r_0, \dots, r_T]^\intercal$. We can obtain the parameters $\boldsymbol{\theta}$ as [9, 56]

$$\boldsymbol{u} = \left(\boldsymbol{\Phi}^\intercal(\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}')\right)^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{R} \tag{3.1}$$

$$\boldsymbol{\theta} = (\boldsymbol{\Phi}^\intercal\boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal \left(\boldsymbol{R} + \gamma\boldsymbol{\Phi}'\boldsymbol{u}\right) \tag{3.2}$$

with discount factor $\gamma$ and the feature matrix $\boldsymbol{\Phi}'$ that consists of the expected feature mappings $\mathbb{E}_{a'_t \sim \pi(s'_t)}[\boldsymbol{\phi}(\boldsymbol{s}'_t, a'_t)]$ with the next states $\boldsymbol{s}'_t$ from the samples and the policy $\pi$. In practice, estimating the expected feature mapping with a single sample from the policy or, in case of a Gaussian process policy, with the mean action is usually sufficient. Here, Equation 3.1 is the fixed-point equation of the Bellman operator and Equation 3.2 is an orthogonal projection into the space spanned by the features.

However, as this solution often overfits to ‘the noise of the system rather than the underlying system itself’, [40] propose to regularize both, the fixed-point equation and the orthogonal projection. Using $l_2$ regularization on both terms, they arrive at

$$\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X} + \beta' \boldsymbol{I})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}, \tag{3.3}$$

$$\boldsymbol{X} = \boldsymbol{\Phi} - \gamma \boldsymbol{\Sigma} \boldsymbol{\Phi}', \tag{3.4}$$

$$\boldsymbol{y} = \boldsymbol{\Sigma} \boldsymbol{R}, \tag{3.5}$$

$$\boldsymbol{\Sigma} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi} + \beta \boldsymbol{I})^{-1} \boldsymbol{\Phi}^\intercal. \tag{3.6}$$

Here, $\boldsymbol{\Sigma}$ is the orthogonal projection into the space spanned by the features, regularized by $\beta$, and Equation 3.3 is the fixed-point equation of the Bellman operator regularized by $\beta'$.

## Policy Improvement

In the policy improvement step, we want to optimize the policy such that the Q-function is maximized. However at the same time, large changes in the policy might lead to loss of information [76]. Inspired by the episodic REPS algorithm [54], AC-REPS solves this problem by using information-theoretic constraints. AC-REPS updates the policy by optimizing the expected Q-values of the samples with the constraint that the new policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ is close to the old policy $q(\boldsymbol{a}|\boldsymbol{s})$ in terms of the Kullback-Leibler divergence (KL). Assuming a state distribution $\mu(\boldsymbol{s})$, we want to maximize the expected Q-value

$$\mathbb{E}_{\mu,\pi}[Q(\boldsymbol{s},\boldsymbol{a})] = \int \mu(\boldsymbol{s}) \int \pi(\boldsymbol{a}|\boldsymbol{s}) Q(\boldsymbol{s},\boldsymbol{a})\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a}. \tag{3.7}$$

However, instead of optimizing the policy for each state independently, we resort to maximize over the joint state-action distribution $p(\boldsymbol{s},\boldsymbol{a}) = p(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$. As we don’t want to optimize the state distribution, we need to ensure that the optimized marginal state distribution $p(\boldsymbol{s}) = \int p(\boldsymbol{s},\boldsymbol{a}) \mathrm{d}\boldsymbol{a}$ is the same as the state distribution $\mu(\boldsymbol{s})$ that has generated the data. This constraint is implemented by matching feature averages of the distributions $p(\boldsymbol{s})$ and $\mu(\boldsymbol{s})$ as

$$\int p(\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s})\, d\boldsymbol{s} = \hat{\boldsymbol{\phi}}, \tag{3.8}$$

where $\hat{\boldsymbol{\phi}}$ is the average feature vector of all state samples. Similarly, we bound the KL between the new and the old joint state-action distributions instead of the new and old policy, i.e.,

$$\mathrm{KL}[p(\boldsymbol{s},\boldsymbol{a}) || q(\boldsymbol{s},\boldsymbol{a})] \leq \epsilon \tag{3.9}$$

With a final constraint that ensures that $p(\boldsymbol{s},\boldsymbol{a})$ is a probability distribution, the resulting constraint optimization problem is

$$\begin{gathered}\arg\max_p \iint p(\boldsymbol{s},\boldsymbol{a}) Q(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{s}\;\mathrm{d}\boldsymbol{a},\\ \text{s.t. } \mathrm{KL}\left[p(\boldsymbol{s},\boldsymbol{a})||q(\boldsymbol{s},\boldsymbol{a})\right]\le\epsilon,\\ \int p(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s})\,d\boldsymbol{s}=\hat{\boldsymbol{\phi}},\\ \iint p(\boldsymbol{s},\boldsymbol{a})\,d\boldsymbol{s}\;d\boldsymbol{a}=1.\end{gathered} \tag{3.10}$$

The upper bound $\epsilon$ for the KL divergence is a parameter of REPS that controls the exploration-exploitation trade-off by restricting the greediness of the method. This parameter is usually chosen heuristically. The constraint optimization problem can be solved in closed form with the method of Lagrange multipliers, yielding

$$p(\boldsymbol{s},\boldsymbol{a})=q(\boldsymbol{s},\boldsymbol{a})\exp\left[\frac{Q(\boldsymbol{s},\boldsymbol{a})-V(\boldsymbol{s})}{\eta}\right]Z^{-1} \tag{3.11}$$

where $V(\boldsymbol{s})=\boldsymbol{\nu}^{\intercal}\boldsymbol{\phi}(\boldsymbol{s})$ is a state dependent baseline similar to a value function, $\boldsymbol{\nu}$, $\eta$, and $\lambda$ are the Lagrangian multipliers, and $Z=\exp[(\eta+\lambda)/\eta]$ is a normalizing therm. Using $Z=\iint p(s,a)\mathrm{d}s\mathrm{d}a$, we can obtain the dual function by substituting $p(s,a)$ in the Lagrangian with Equation 3.11. The Lagrangian multipliers $\eta$ and $v$ can then be obtained efficiently by minimizing the dual

$$g(\boldsymbol{\nu},\eta)=\eta\log\left(\frac{1}{N}\sum_{i=1}^{N}\exp\left(\frac{Q(\boldsymbol{s}_i,\boldsymbol{a}_i)-\boldsymbol{\nu}^{\intercal}\phi(s_i)}{\eta}\right)+\eta\epsilon+\boldsymbol{\nu}^{\intercal}\hat{\phi}\right) \tag{3.12}$$

using the Broyden-Fletcher-Goldfarb-Shannon algorithm.

### Matching a Continuous Stochastic Policy

Solving the optimization problem obtained from REPS gives the desired probabilities $p(\boldsymbol{s}_i,\boldsymbol{a}_i)=p(\boldsymbol{s}_i)\hat{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i)$ only for the discrete samples in $\mathcal{D}$. For sample $(\boldsymbol{s}_i,\boldsymbol{a}_i)$ we want to take action $\boldsymbol{a}_i$ in state $\boldsymbol{s}_i$ with probability

$$\hat{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i)\propto\exp\left(\frac{Q(\boldsymbol{s}_i,\boldsymbol{a}_i)-\boldsymbol{\nu}^{\intercal}\boldsymbol{\phi}(\boldsymbol{s}_i)}{\eta}\right) \tag{3.13}$$

We can ignore the old policy $q(\boldsymbol{a}|\boldsymbol{s})$ here because the actions in $\mathcal{D}$ have been samples from $q(\boldsymbol{a}|\boldsymbol{s})$ and the state distributions $p(\boldsymbol{s})$ and $q(\boldsymbol{s})$ are equal due to the constraint in Equation 3.8.

By fitting a weighted linear model in the feature space of the states, we obtain a continuous policy for the whole state space. With the assumptions of Gaussian exploration noise and of a Gaussian prior over the actions we arrive at a weighted linear Gaussian model which turns into a weighted Gaussian process by using a kernel function to compute the inner product of the features, i.e.,

$$\begin{aligned} \pi(\boldsymbol{a}|\boldsymbol{s}) &= \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{s}), \Sigma(\boldsymbol{s})), \quad \text{where} \\ \boldsymbol{\mu}(\boldsymbol{s}) &= \boldsymbol{k}(\boldsymbol{s})^\intercal (\boldsymbol{K} + l\boldsymbol{W})^{-1} \boldsymbol{A}, \\ \Sigma(\boldsymbol{s}) &= k(\boldsymbol{s},\boldsymbol{s}) + \boldsymbol{k}(\boldsymbol{s})^\intercal (\boldsymbol{K} + l\boldsymbol{W})^{-1} \boldsymbol{k}(\boldsymbol{s}) + l. \end{aligned} \tag{3.14}$$

Here, $k(\boldsymbol{s}_i, \boldsymbol{s}_j)$ denotes the kernel function of the two states $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$, $\boldsymbol{k}(\boldsymbol{s})$ is the kernel vector of the states $\boldsymbol{s}_i \in \mathcal{D}$ and state $\boldsymbol{s}$ and $\boldsymbol{K}$ is the kernel matrix of the states in $\mathcal{D}$. Note that the prior over the actions is a scalar in the kernel function $k(\boldsymbol{s}_i, \boldsymbol{s}_j) = bK(\boldsymbol{s}_i, \boldsymbol{s}_j)$, we will discuss the kernel function $K$ in Section 3.4.1. Furthermore, $l$ is the variance of the exploration noise, $\boldsymbol{W}$ is a diagonal matrix of the weights $W_{ii} = \hat{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i)$, and $\boldsymbol{A}$ is a matrix with the actions as row vectors.

A common drawback of kernel functions is that they do not scale well with the number of samples used for training. Several sparsification approaches have been proposed to overcome this issue, e.g., the method of projected latent variables by [91], the sparse Gaussian processes (SPGP) by [101], or the sparse greedy Gaussian process regression by [99]. They use a set of *pseudo inputs* (also called latent variables, inducing inputs or active set) to approximate the full covariance of the Gaussian process. Effectively, given a set of $m$ pseudo inputs, the kernel function in the predictive mean and covariance of a sparse GP is substituted by

$$k(\boldsymbol{s}_i, \boldsymbol{s}_j) = \boldsymbol{k}_m(\boldsymbol{s}_i)^\intercal \boldsymbol{K}_{mm}^{-1} \boldsymbol{k}_m(\boldsymbol{s}_j), \tag{3.15}$$

where $\boldsymbol{k}_m(\boldsymbol{s}_i)$ is a kernel vector of state $\boldsymbol{s}_i$ with the pseudo inputs and $\boldsymbol{K}_{mm}$ is the kernel matrix of the pseudo inputs. Following the derivations in [41], this allows to define a sparse, weighted Gaussian processes as

$$\boldsymbol{\mu}(\boldsymbol{s}) = \boldsymbol{k}_m(\boldsymbol{s})^\intercal \boldsymbol{Q}^{-1} \boldsymbol{K}_{mn} (l\boldsymbol{W} + \boldsymbol{\Lambda})^{-1} \boldsymbol{A} \tag{3.16}$$

$$\Sigma(\boldsymbol{s}) = k(\boldsymbol{s},\boldsymbol{s}) - \boldsymbol{k}_m(\boldsymbol{s})^\intercal \left(\boldsymbol{K}_{mm}^{-1} + \boldsymbol{Q}^{-1}\right) \boldsymbol{k}_m(\boldsymbol{s}) + l \tag{3.17}$$

$$\boldsymbol{Q} = \left(\boldsymbol{K}_{mm} + \boldsymbol{K}_{mn} (l\boldsymbol{W} + \boldsymbol{\Lambda})^{-1} \boldsymbol{K}_{nm}\right), \tag{3.18}$$

with diagonal matrix $\Lambda_{ii} = k(\boldsymbol{s}_i, \boldsymbol{s}_i) - \boldsymbol{k}_m(\boldsymbol{s}_i)^\intercal \boldsymbol{K}_{mm}^{-1} \boldsymbol{k}_m(\boldsymbol{s}_i)$.

### 3.3.2 Kernel Embeddings of Distributions

We will represent the configuration of the swarm as a distribution where each agent refers to a single sample of that distribution. Using moments of the distribution as representation, e.g., the mean and the variance as in [92], allows for a compact representation of the swarm. However, this representation is also quite limited if the swarm is not distributed similar to a Gaussian distribution. For example, if the swarm is separated into two or more groups, this representation is not able to distinguish between quite different configurations.

The recent technique of kernel embeddings [100] allows a nonparametric representation of distributions with arbitrary shapes. A reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ of functions is uniquely defined by a positive definite kernel function $k(\boldsymbol{x},\boldsymbol{x}') := \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{x}')\rangle_{\mathcal{H}}$ [3]. Here, the feature mappings $\boldsymbol{\psi}(\boldsymbol{x})$ are often intrinsic to the kernel functions and might map into an infinite dimensional feature space (e.g., the squared exponential kernel.) The embedding of a marginal distribution $p(X)$ is defined as the expected feature mapping of its random variable $\boldsymbol{\mu}_X := \mathbb{E}_p[\boldsymbol{\psi}(X)] = \int_\Omega \boldsymbol{\psi}(\boldsymbol{x})\,\mathrm{d}p(\boldsymbol{x})$ [100]. In practice we estimate the embedding using samples from $p(X)$ as

$$\hat{\boldsymbol{\mu}}_X = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{\psi}(\boldsymbol{x}_i) = \frac{1}{m}\sum_{i=1}^{m}k(\boldsymbol{x}_i,\cdot). \tag{3.19}$$

We will use the mean embedding as representation later to define a kernel function for the state of the swarm. Given infinite dimensional features in the kernel function, such mean embeddings are infinite dimensional as well and cannot be represented explicitly. Still, we can asses the discrepancy of two distributions $p(X)$ and $q(Y)$ using the *maximum mean discrepancy* (MMD) as

$$\begin{aligned} \mathrm{MMD}(\boldsymbol{\mu}_X,\boldsymbol{\mu}_Y) &= \langle \boldsymbol{\mu}_X-\boldsymbol{\mu}_Y, \boldsymbol{\mu}_X-\boldsymbol{\mu}_Y\rangle & (3.20)\\ &= \mathbb{E}_{p,p}\left[k(x_i,x_j)\right] - 2\mathbb{E}_{p,q}\left[k(x_i,y_j)\right] + \mathbb{E}_{q,q}\left[k(y_i,y_j)\right] & (3.21)\\ \widehat{\mathrm{MMD}}(\boldsymbol{\mu}_X,\boldsymbol{\mu}_Y) &= \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}k(x_i,x_j) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}k(x_i,y_j) & \\ &\quad + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}k(y_i,y_j) & (3.22) \end{aligned}$$

Note, that the MMD can also be seen as the squared error between two mean embeddings.

### 3.3.3 Planning Strategies

A* is a heuristic search algorithm commonly applied for graph search problems [36]. The algorithm selects which node $n_s$ to expand by minimizing the cost $f(n_s) = g(n_s) + h(n_s)$, where $g(n_s)$ is the cost for reaching node $n_s$ from the start and $h(n_s)$ is a heuristic that provides a lower bound to the costs from $\boldsymbol{s}$ to the goal state $\boldsymbol{s}_G$. The cost $g(n_s)$ can be computed by

$$g(n_s) = g(\mathrm{pred}(n_s)) + c(\mathrm{pred}(n_s), n_s), \tag{3.23}$$

where $\mathrm{pred}(n_s)$ is the parent node of $n_s$ and $c(n_{s_1}, n_{s_2})$ is the cost to get from $n_{s_1}$ to $n_{s_2}$.

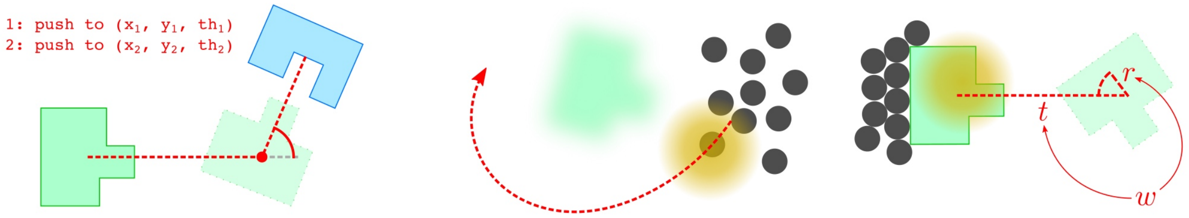


**Figure 3.2:** The three components of our approach. Left: the assembly policy defines way points for the objects; middle: a path planning strategy computes collision free paths for the objects but is also used to position the Kilobots for the next push; right: the object movement policy controls the light source when the swarm is pushing the objects.

Potential fields [52] are a fast planning method for mobile robots. The robots move along a hypothetical force field, being attracted to the goal position and repulsed from the obstacles. The repulsive potential for an object $\boldsymbol{o}$ is defined as

$$U_{\text{rep}}(\boldsymbol{s},\boldsymbol{o}) = \begin{cases} \frac{1}{2}\chi\left(\frac{1}{d(\boldsymbol{s},\boldsymbol{o})}-\frac{1}{d_o}\right)^2 & \text{if } d(\boldsymbol{s},\boldsymbol{s}_G) < d_o \\ 0 & \text{else} \end{cases}, \tag{3.24}$$

respectively. Here, $d(\boldsymbol{a},\boldsymbol{b})$ is a measure for the distance between $\boldsymbol{a}$ and $\boldsymbol{b}$, $d_o$ is the maximum distance to the obstacle, and $\chi$ is a scaling factor. In our approach, we use the repulsive potential in the cost term for the path segments $c(n_{s_1}, n_{s_2})$ of A* (see Section 3.4.2 for more details).

## 3.4 Learning Control Policies for Object Assembly

We split the task of object assembly into three components (an overview is given in Figure 3.2): (1) an *assembly policy* that describes how the individual objects should move, (2) a *path planning strategy* to guide the swarm around the objects and to arrange them for the next pushing task, and (3) an *object movement policy* that realizes basic movements of an object by controlling a light source that guides the robot swarm.

### 3.4.1 The Object Movement Policy

The object movement policy controls the global input signal, e.g., the position of the light source such that the swarm, which follows this signal, pushes the object along a given trajectory. We reduce the search space for the object movement policies by considering only pushes in positive $x$-direction or counterclockwise rotations. Later, we apply the learned policies to arbitrary movements by rotating and flipping the state representation accordingly. We further introduce a trade-off parameter $\rho \in [0,1]$ that weighs between translational and rotational movements. This trade-off is achieved by the design of the reward function which we introduce in Section 3.4.1. In our experiments we usually learned object movement policies for three settings of $\rho$, i.e., $\rho = 0.0$, $\rho = 0.5$, $\rho = 1.0$.

## The Reward Function

The reward function reflects the setting of the trade-off parameter $\rho \in [0,1]$. The function rewards only rotational movements for $\rho = 1$ and only translational movement for $\rho = 0$. For values in between, $\rho$ trades off the rotational and the translational term. Given the translational movements $d_x$ in $x$-direction, $d_y$ in $y$-direction and the rotational movement $d_\theta$, we define the reward as

$$r(\rho) = \rho\, r_{\text{rot}} + (1-\rho) r_{\text{trans}} - c_y d_y, \tag{3.25}$$

with the translational and rotational reward terms

$$r_{\text{trans}} = d_x - c_\theta\, d_\theta, \quad \text{and} \tag{3.26}$$

$$r_{\text{rot}} = d_\theta - c_x\, d_x, \tag{3.27}$$

respectively. The weights $c_x$, $c_y$, and $c_\theta$ scale the costs for undesired translational or rotational movements.

## States and Actions

We define our state relative to the center of the object part that we want to push. Given the relative light position $\boldsymbol{l} = (x_l, y_l)$ and a swarm configuration with $n$ agents, where agent $i$ has the relative position $\boldsymbol{b}_i = (x_i, y_i)$, the state vector is defined as $\boldsymbol{s} := [\boldsymbol{l}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_n]$. The action vector $\boldsymbol{a} = (a_x, a_y)$ is the desired displacement of the light source in $x$- and $y$-direction.

## Features and Kernels

To learn an object movement policy that generalizes to different swarm sizes, we need to employ a feature mapping that abstracts from the number of individuals in the swarm and also from the allocation of the single robots to their positions (i.e., which agent is at which position). Therefore, we represent the state of the swarm as a distribution embedded into a RKHS [100] where we treat each agent as a sample of that distribution, i.e.,

$$\mu_{\boldsymbol{b}}(\cdot) = \frac{1}{n}\sum_{i=1}^{n} k(\boldsymbol{b}_i, \cdot) = \frac{1}{n}\sum_{i=1}^{n} \psi(\boldsymbol{b}_i), \tag{3.28}$$

where $k$ is a kernel function (e.g., the Gaussian kernel) and $\psi$ is the intrinsic feature mapping of $k$. This representation is invariant to both, the allocation of the individual agent to the position as well as to the number of agents in the swarm. We can compute the difference between two swarm distributions independently from the number of agents by computing the squared difference of their embeddings

$$d_{\text{b}}(\boldsymbol{b}, \boldsymbol{b}') = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k(\boldsymbol{b}_i, \boldsymbol{b}_j) - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} k(\boldsymbol{b}_i, \boldsymbol{b}'_j) + \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} k(\boldsymbol{b}'_i, \boldsymbol{b}'_j). \tag{3.29}$$

Here, $\boldsymbol{b}$ and $\boldsymbol{b}'$ are two swarm configurations with $n$ and $m$ individuals, respectively. In addition to the state of the swarm, we also need to represent the current relative position of the light $\boldsymbol{l}$ and the desired displacement of the light $\boldsymbol{a}$ (i.e., the action) in the feature vector. For both, we can obtain the squared distance simply by

$$d_{\mathrm{v}}(v, v') = -0.5(v - v')^{\intercal} \operatorname{diag}\left(\sigma_{\mathrm{v}}^{-2}\right)(v - v'), \tag{3.30}$$

where $v$ can be either the composition of $\boldsymbol{l}$ and $\boldsymbol{a}$ or only the light position $\boldsymbol{l}$, depending if we need a feature function for state-action pairs or only states. We can now combine these two distance measures into a kernel function

$$K(\boldsymbol{s}, \boldsymbol{s}') = \exp\left(-\frac{\alpha}{2} d_{\mathrm{v}}(v, v') - \frac{1-\alpha}{2} d_{\mathrm{b}}(\boldsymbol{b}, \boldsymbol{b}')\right), \tag{3.31}$$

where $\alpha \in [0, 1]$ weighs the importance of the non-agent dimensions $v$ and the agent dimensions $\boldsymbol{b}$ of the state $\boldsymbol{s}$.

At each learning iteration of the AC-REPS algorithm, we select a kernel reference set $\mathcal{D}_{\mathrm{ref}} = (\boldsymbol{s}_i, \boldsymbol{a}_i)_{i=1}^{N}$ randomly from the SARS samples. With this, we can define the feature vector $\boldsymbol{\phi}(\boldsymbol{s}, \boldsymbol{a})$ for approximating the Q-function, where the $i$-th entry of the feature vector

$$\phi(\boldsymbol{s}, \boldsymbol{a})_i = K((\boldsymbol{s}_i, \boldsymbol{a}_i), (\boldsymbol{s}, \boldsymbol{a})), \quad i = 1, \ldots, N \tag{3.32}$$

is the kernel function evaluated at the reference sample $(\boldsymbol{s}_i, \boldsymbol{a}_i)$. For the policy improvement step, we need a state-dependent feature function which we define as

$$\varphi(\boldsymbol{s})_i = K(\boldsymbol{s}_i, \boldsymbol{s}), \quad i = 1, \ldots, N. \tag{3.33}$$

### 3.4.2 Assembly Policy and Path Planning Strategy

The assembly policy contains the construction information stored as a list of oriented way points with required accuracies for each object. These way points are processed consecutively by applying either the object movement policy or the path planning strategy. When the object movement strategy is applied, we have to minimize the translational error $e_{\mathrm{trans}}$ and the rotational error $e_{\mathrm{rot}}$ until the next way point is reached. We compute the desired translation-rotation ratio as $\rho_{\mathrm{des}} = \eta e_{\mathrm{rot}} / (e_{\mathrm{trans}} + \eta e_{\mathrm{rot}})$ and choose the object movement policy with closest ration $\rho$. The parameter $\eta$ scales the rotational error with the translational error, usually a value of 0.1 leads to good results.

For guiding the swarm from one object to the next, we use a path planning strategy to obtain a collision free path to the target. We use A* in combination with the repulsive potential of the potential fields in the cost term $c(\boldsymbol{s}_1, \boldsymbol{s}_2)$. Naively, we could also simply follow the gradient of the potential field. However, this approach is prone to issues such as local minima, narrow passages, or oscillations around obstacles [53]. Instead, we define the cost function $c(\boldsymbol{s}_1, \boldsymbol{s}_2) = d(\boldsymbol{s}_1, \boldsymbol{s}_2) + U_{\mathrm{rep}}(\boldsymbol{s}_2)$, where $d(\boldsymbol{s}_1, \boldsymbol{s}_2)$ is the distance between $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, and $U_{\mathrm{rep}}(\boldsymbol{s}_2)$ is the repulsive potential. As heuristic $h(\boldsymbol{s})$ we use the Euclidean distance to the target state.

## 3.5 Experimental Setup & Results

We evaluate the proposed learning method in simulation as well as on a robotic platform. As robotic platform, we chose the Kilobot platform [83]. The Kilobots are an affordable and open source platform developed specifically for the evaluation of algorithms on large swarms of robots. Each robot is approximately 3 cm in diameter, 3 cm tall and moves up to 1 cm/s by using two vibration motors.

We have implemented a 2D simulator of the Kilobot platform in Python[1] compatible with the OpenAI gym[10]. To simulate the interactions of the agents and the objects in the world, we use the physics engine Box2D[2]. We use this simulation to evaluate the learning algorithm and to learn the policies that we will later apply directly to the real Kilobots. The simulator internally runs at 10Hz, but only takes action and returns state and rewards at 1Hz.

### 3.5.1 Evaluation of the Learning Algorithm

We learn the object movement policy for six object types, i.e., square ($w = 0.15$), rectangle ($w = 0.05, h = 0.3$), triangle ($w = 0.14, h = 0.21$), L-shape, T-shape, and C-shape (each with overall $w = 0.14, h = 0.21$) (c.f. Table 3.1) and for three ratios $\rho \in [0.0, 0.5, 1.0]$. The object is initialized at $(0, 0)$ with a random orientation uniformly sampled from $[\pi, -\pi]$. To simulate the light source, we use a circular gradient with radius $r = 0.2$. If an agent is within this radius, it senses the gradient towards the center. The initial position of the light and the swarm is sampled normally around the worlds center with standard deviation $\max(w_{world}, h_{world})/3$. The agents are sampled normally around the light position with standard deviation $r/3$.

We learn the object movement policies with 10 agents over 60 iterations. In each iteration we sample 200 episodes with 60 steps/episode. Afterwards, we keep a set of 10000 SARS tuples which we choose randomly from the new samples and the old SARS tuples. To define the feature function for LSTD, we select 1000 samples from the SARS data randomly. We choose 1000 inducing inputs for the sparse GP later by importance sampling using the weights obtained from REPS. After each learning iteration, we evaluate the learned policy on 50 episodes of length 125.

Tables 3.1–3.3 show the learned policies, the learned value function, as well as the light and object trajectories of the learning episodes and of the evaluation episodes for the ratios $\rho = 0.0, \rho = 0.5, \rho = 1.0$. The depicted results are taken from the iteration with the highest mean reward of the evaluation episodes. Note that we use artificial configurations in which all agents and the light are at the same position $(x, y)$ to visualize the policies and value functions.

Figure 3.3 shows the learning curves for each object shape and ratios $\rho \in [0.0, 0.5, 1.0]$. Note the differently scaled y-axes which depicts the different difficulties in learning the policies for different object types. It can be seen that the relation of the different policy types varies strongly between the object shapes. This relates to the object geometries which make it harder for some objects to be pushed without rotational movement (e.g.

---

[1] The Kilobot Gym, `https://github.com/gregorgebhardt/gym-kilobots`

[2] Box2D – A 2D Physics Engine for Games, `http://box2d.org/`

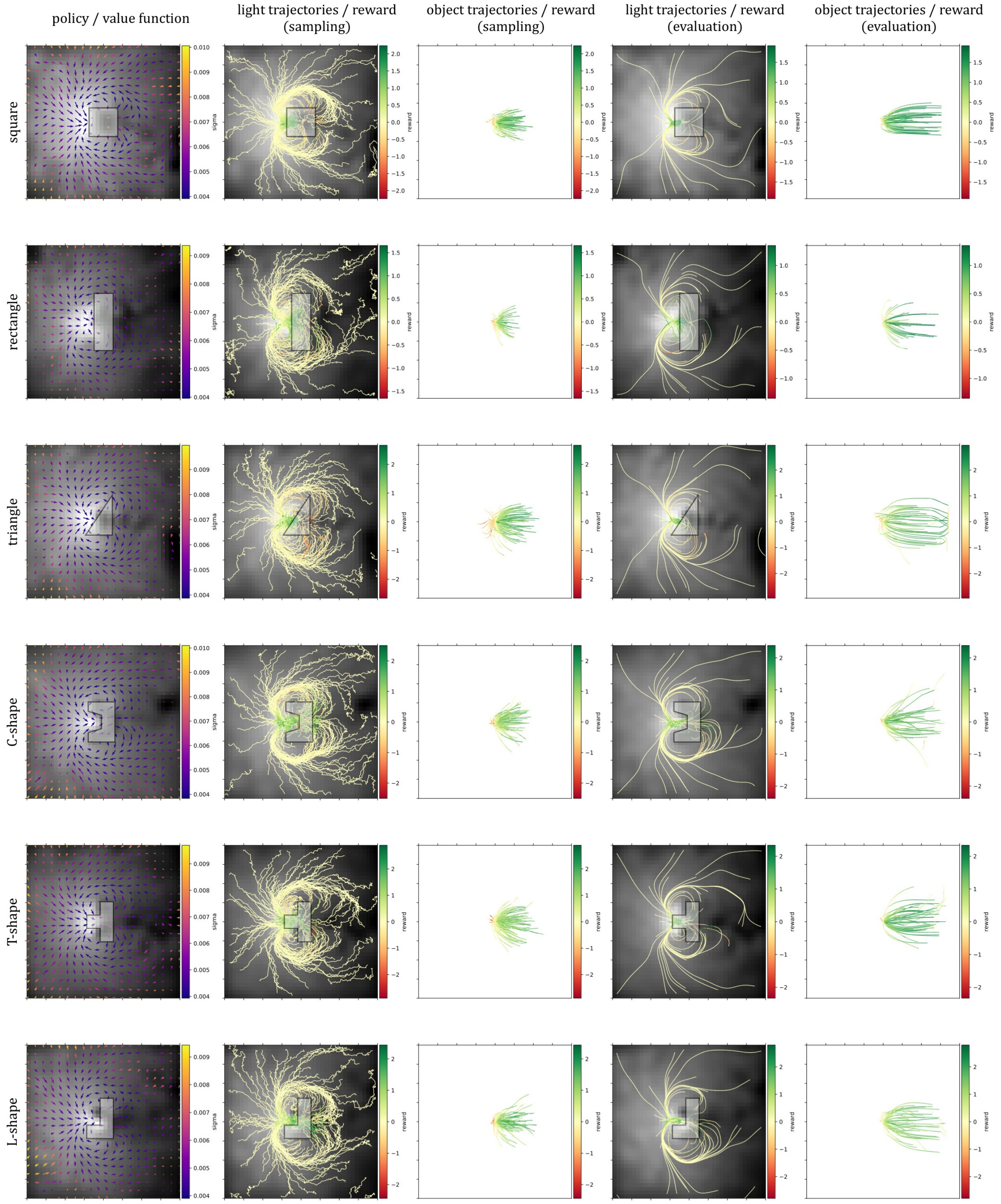


**Table 3.1:** Results from learning the object movement policy for $\rho = 0.0$. The first column depicts the learned policy and the value function. The policy is shown as quiver plot where the arrows denote the mean action and the color denotes the variance of the GP. The second column shows the trajectories of the light center during the sampling episodes relative to the object. The third column shows the object trajectories during the sampling episodes. The color of the trajectories denotes the reward. Likewise, the fourth and fifth column depict the trajectories of the light source and the object, respectively, during the evaluation episodes.

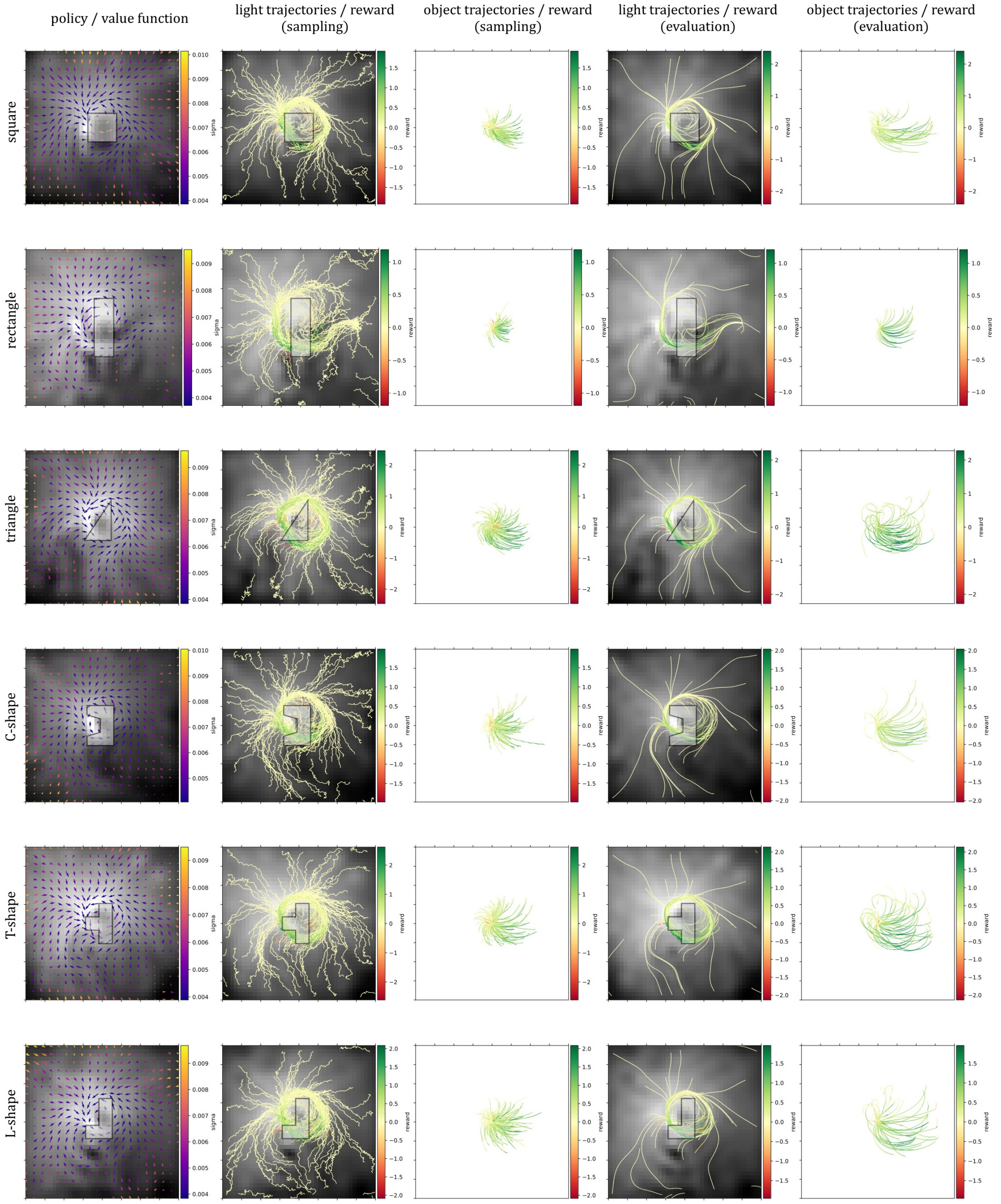


**Table 3.2:** Results from learning the object movement policy for $\rho = 0.5$. The first column depicts the learned policy and the value function. The policy is shown as quiver plot where the arrows denote the mean action and the color denotes the variance of the GP. The second column shows the trajectories of the light center during the sampling episodes relative to the object. The third column shows the object trajectories during the sampling episodes. The color of the trajectories denotes the reward. Likewise, the fourth and fifth column depict the trajectories of the light source and the object, respectively, during the evaluation episodes.

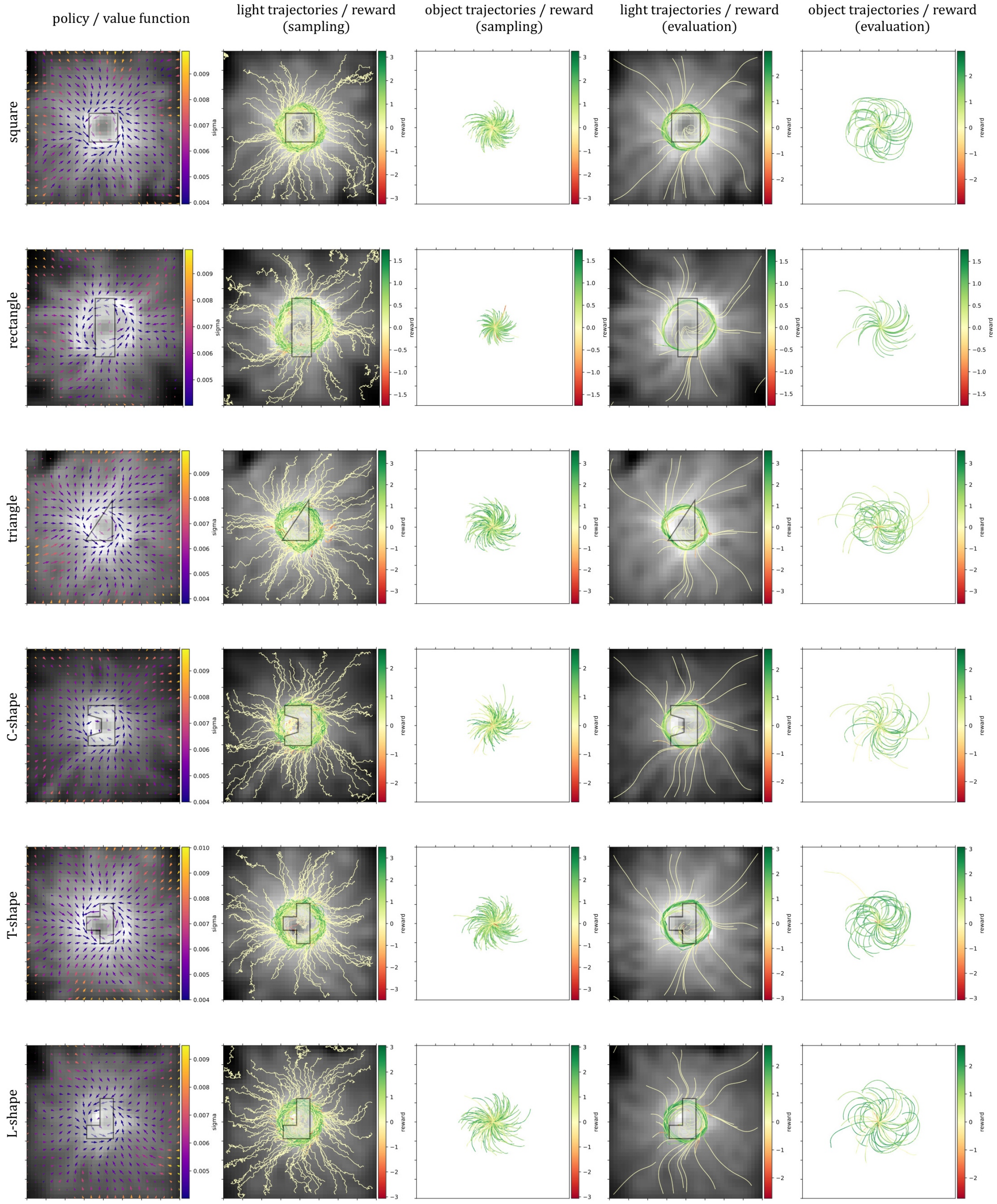


**Table 3.3:** Results from learning the object movement policy for $\rho = 1.0$. The first column depicts the learned policy and the value function. The policy is shown as quiver plot where the arrows denote the mean action and the color denotes the variance of the GP. The second column shows the trajectories of the light center during the sampling episodes relative to the object. The third column shows the object trajectories during the sampling episodes. The color of the trajectories denotes the reward. Likewise, the fourth and fifth column depict the trajectories of the light source and the object, respectively, during the evaluation episodes.

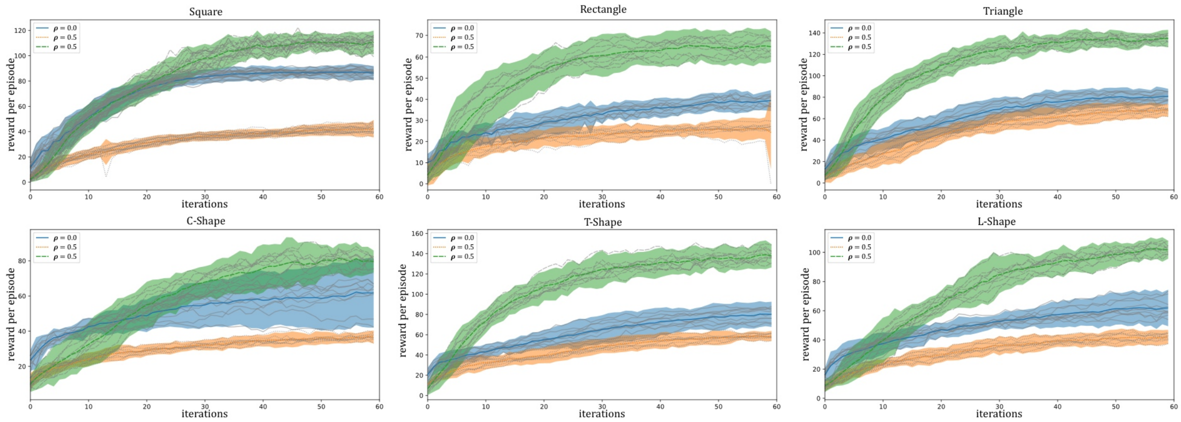


**Figure 3.3:** Learning curves for square, rectangle, triangle, C-shape, T-shape, and L-shape for $\rho \in [0.0, 0.5, 1.0]$. Note the different scalings of the y-axis.

triangle), or which make it easier to be rotated with only little translational movement (e.g. rectangle).

We have evaluated how well each learned set of policies can pushing the objects along a straight line and along a circular path. In addition, we have compared how well a policy learned on the square would generalize to the other shapes. Figure 3.4 shows the trajectories obtained from the straight-line-task. While we get good results for the square, the rectangle and the T-shape using policies learned for the respective shapes, the task is harder to solve with the triangle, the L-shape and the C-shape. The policies learned for the square perform worse for the rectangle, the L-shape, and the C-shape. For the triangle, the square policies yield similar results as the triangle policies. Except for two outliers, the square policies also yield similar results as the learned policies for the T-shape.

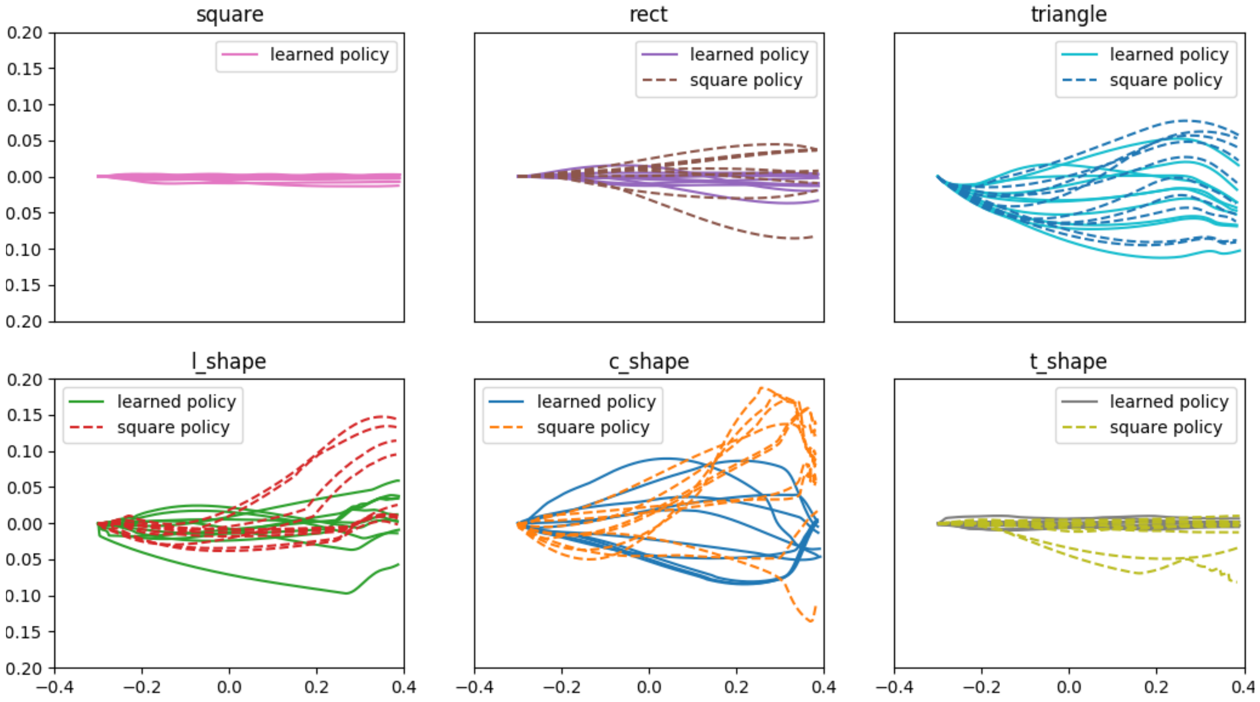


**Figure 3.4:** Evaluation of pushing the objects along a straight path with a policy learned for the specific object shape and with a policy learned for the square object.

Figure 3.5 shows the trajectories for the circular-path-task with radii 0.2, 0.4, 0.6, and 0.8. The circular paths are defined by 10 way points in equiangular distance with an orientation accuracy of 1.5 (roughly $\pm\frac{\pi}{2}$) and a position accuracy of 0.1. Except for the last way point which has a position accuracy of 0.05 only in the y coordinate. While the square tracks the circular path nicely, the rectangle only succeeds in tracking the circular paths with radii 0.4, 0.6, and 0.8 adequately. The triangle manages to follow the paths although with a much larger variance than square or rectangle. The L-shape policies seem to be able to track the paths somehow, however, they often fail to meet the quite broad orientation accuracy of the way points. This results in maneuvering around the way points and thus in the loopy trajectories. The same can be observed for the T-shape. In contrast, the C-shape policies manage to follow the circular paths with errors but stay inside the given accuracy windows.

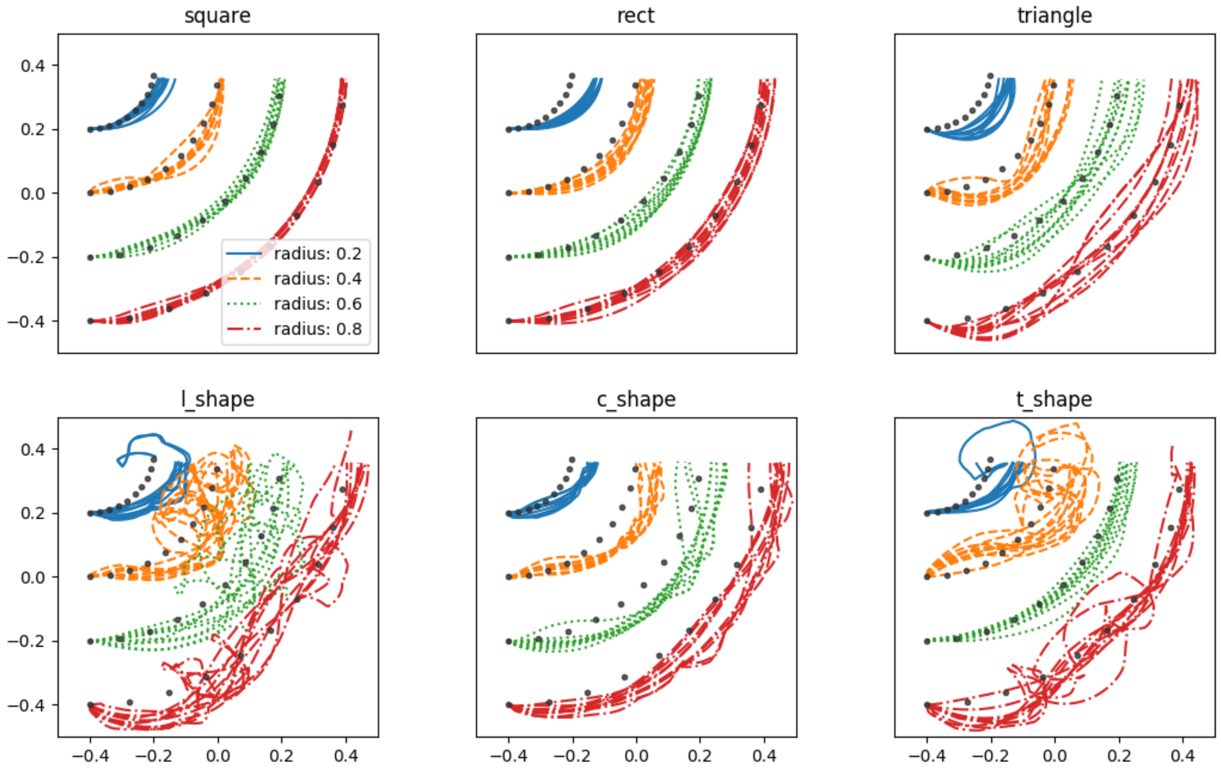


**Figure 3.5:** Evaluation of pushing the objects along circular paths with radii $r \in [0.2m, 0.4m, 0.6m, 0.8m]$ with a policy learned for the specific object shape and with a policy learned for the square object. The black dots denote the way points that define the circular path.s Note that some of the trajectories overshoot the target of the circular path since we set the orientation accuracy at the target to $\pm 1.5$ (roughly $\pm\pi/2$) which was not met in these trajectories.

A *pose controller* (PC) for a swarm of simple agents has been proposed in [92]. We compare against this PC by pushing the rectangle object to three target poses. Figure 3.6, shows the results of this comparison. While both approaches successfully push the object to the first target pose in each of the 10 trials, the PC produces a much longer trajectory as it first pushes the object towards a line through the target pose, before it starts pushing the object towards its final pose. In the second experiment, the PC only succeeds in 4 out of 10 trials to position the object in the target pose. In the third experiment, the PC fails all 10

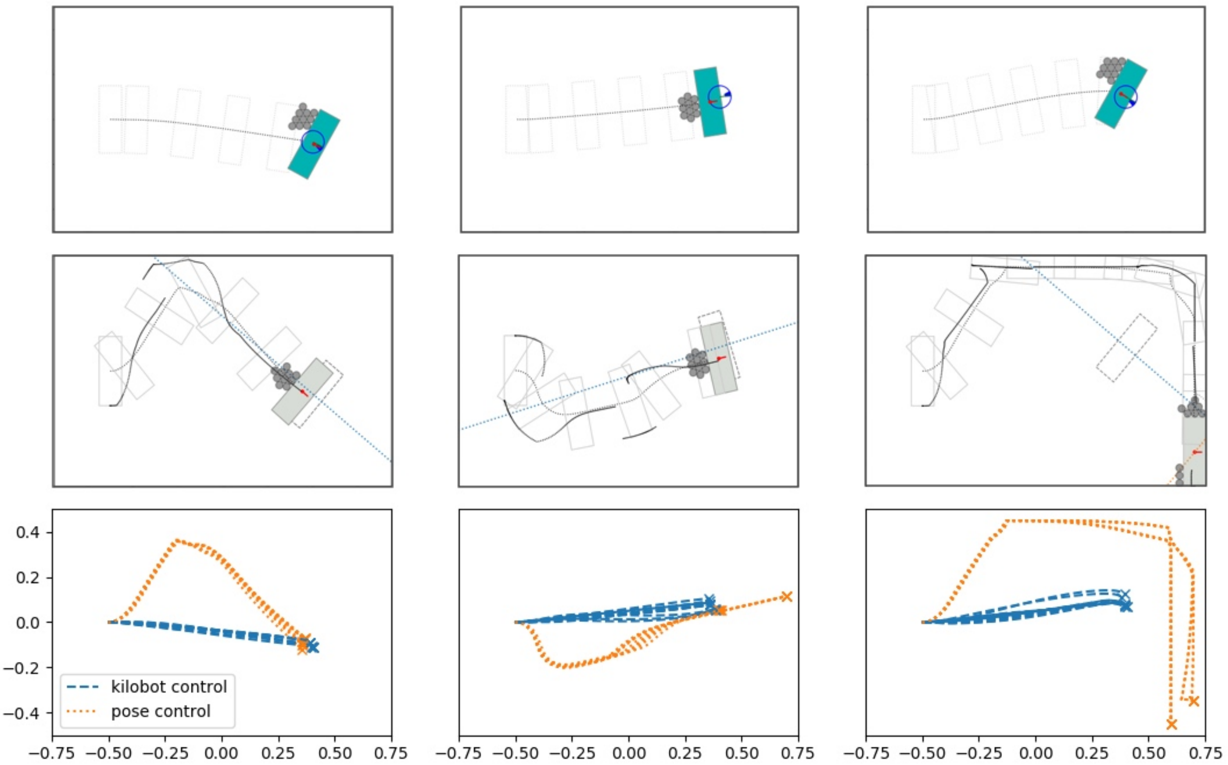


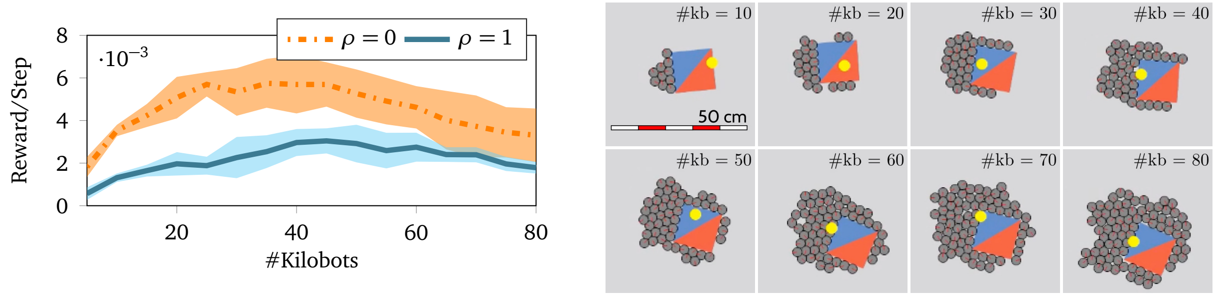
**Figure 3.6:** Evaluation of the object movement policy learned on the rectangular object against the pose controller proposed in [92]. The top row shows exemplary runs with the object movement policy for the three target locations. The second row shows exemplary runs with the pose controller [92]. The bottom row shows a comparison of the object trajectories with 10 runs for each controller and each target object position.

trials since the target point on the auxiliary line is located outside of the environment. In general, the PC is not able to recover from situations in which the object has been pushed into an undesired position as maneuvering of the object has not been considered in the algorithm.

To evaluate how well our approach generalizes to different swarm sizes, we have applied policies learned with 15 agents on the square object to swarms with 5 to 80 agents. Figure 3.7a shows the average reward per step for $\rho = 0$ and for $\rho = 1$. Until a swarm size of about 40 agents the reward increases. The more agents are able to push the object the higher is the combined force and, hence, the object moves faster. However, from a swarm size of roughly 50 agents on, the average reward starts to decline. With too many agents in the swarm, the swarm distributes around the object so that the agents push from opposing directions and obstruct the desired motion. Figure 3.7b depicts this evaluation.

### 3.5.2 The Assembly Task in Simulation

We have evaluated the learned policies for triangle, L-shape, C-shape and T-shape on three object assembly tasks in simulation. Furthermore, we have also executed these tasks with



**(a)** Reward for different swarm sizes.

**(b)** Swarms of different sizes pushing an object.

**Figure 3.7:** Comparison of the performance of a policy on swarms of different sizes. The policies have been learned with 15 agents and evaluated with 5 to 80 agents. (a) shows the average reward per time-step of a policy with $\rho = 0$ and a policy with $\rho = 1$. With a size of 50 agents and more, the swarm distributes around the object and obstructs the intended push as it can be seen from (b). A video including these evaluations is available at `https://youtu.be/kuU8wsR9dD4`.

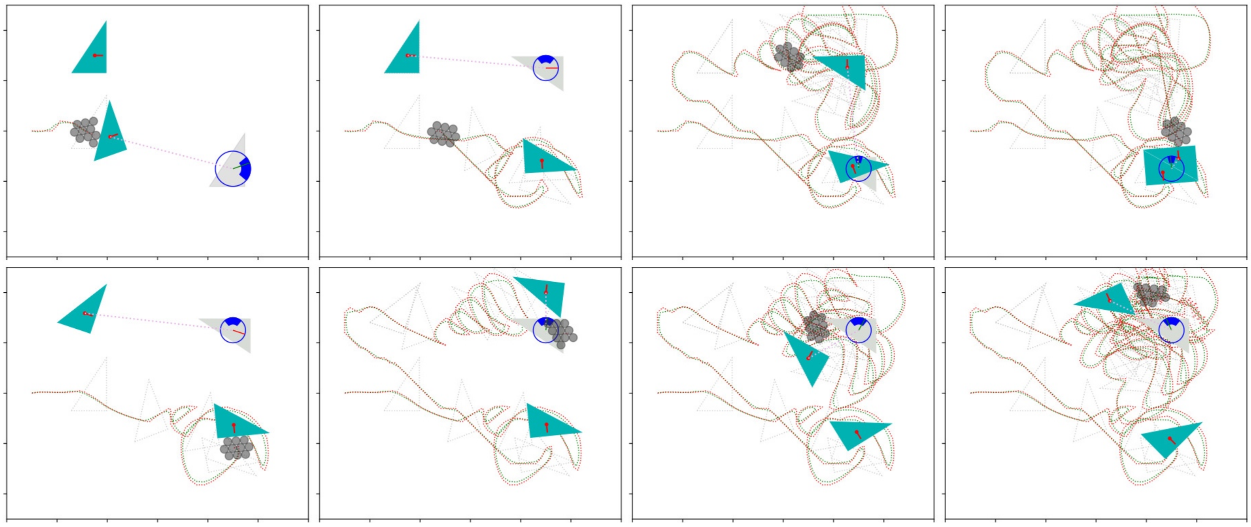


**Figure 3.8:** Assembly task with two triangle objects. In the top row, the task is executed successfully with a policy learned on the triangle shape, in the bottom row the task fails when executed with a policy learned with a square object. The red line is the trace of the light source, the green line is the trace of the swarm. The blue circles depict the way points with the required accuracy for position and orientation, the green/red line inside of the way points depicts the current orientation of the object.

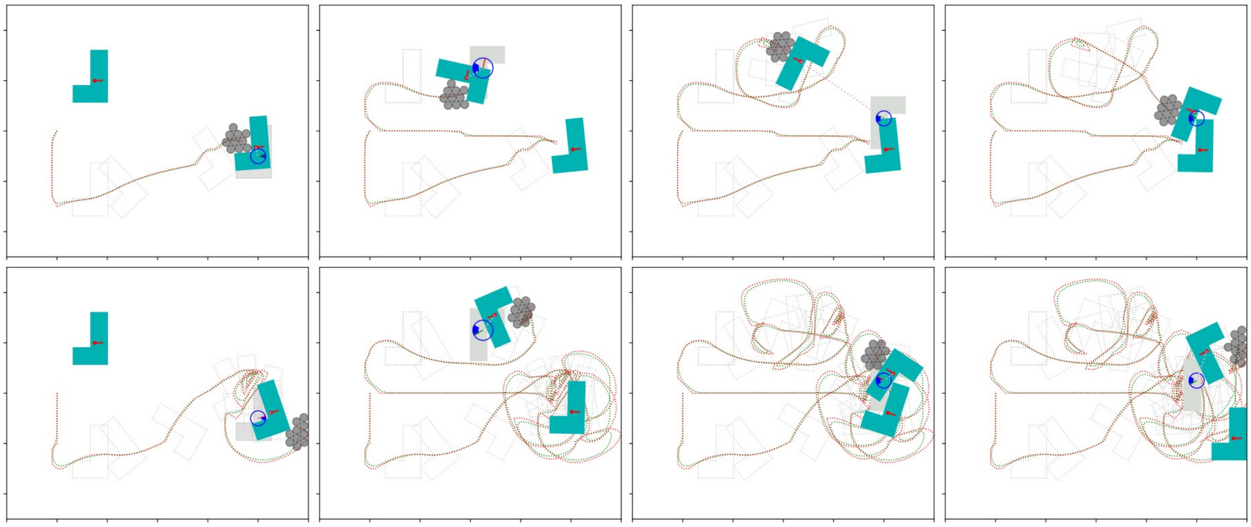


**Figure 3.9:** Assembly task with two L-shape objects objects. In the top row, the task is executed successfully with a policy learned on the L-shape, in the bottom row the task fails when executed with a policy learned with a square object.

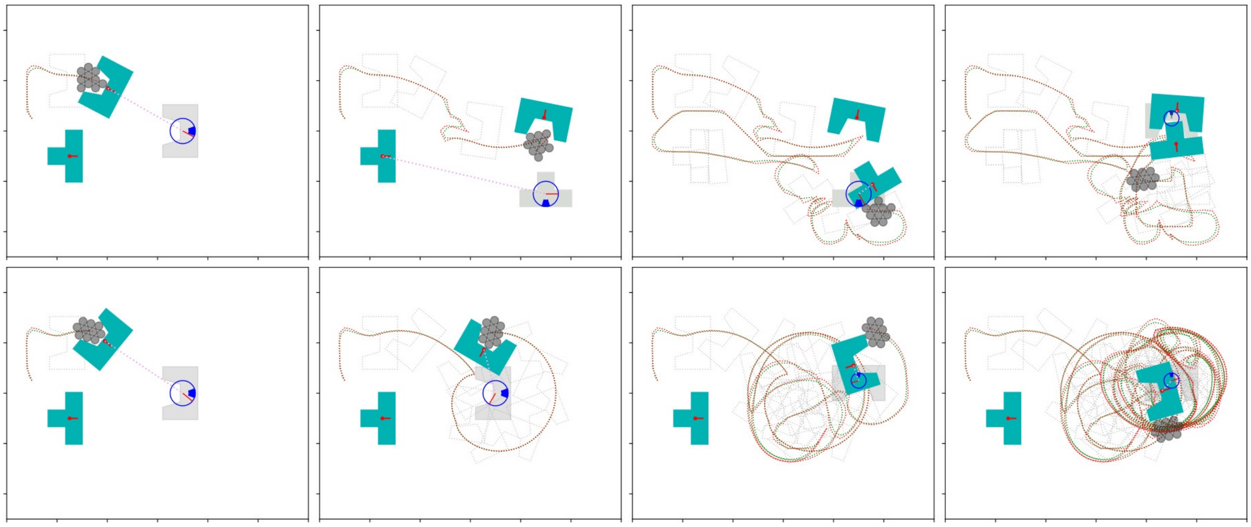


**Figure 3.10:** Assembly task of a C-shape with a T-shape. In the top row, the task is executed successfully with polices learned on C- and T-shape. In the bottom row the task fails when executed with a policy learned with a square object.

policies learned on the square object to asses how good policies learned on a simple shape generalize to more complex shapes.

The first task is to assemble two triangular objects. The assembly policy contains two way points for each of the triangular objects, where the first ensures that the objects are positioned well before they are pushed into the target position. An exemplary execution of the assembly is shown in Figure 3.8. The swarm pushes the first object to its target position passing through the intermediate way point. The swarm is then guided to the second object along a path obtained from the path planning strategy. The positioning of the second object at the intermediate way point requires maneuvering which is done by the learned object movement policy. Finally, the second object is pushed to the target position to assemble it with the first object. In our experiments, the assembly process succeeded in 4 of 5 trials when using the policy learned for the triangle shape and in 0 of 5 trials when using the policies learned with the square shape.

In the second task, the goal is to assemble two L-shapes, exemplary executions of the task are depicted in Figure 3.9 The first L-shape is pushed directly to the target position. The second L-shape is first rotated to an intermediate way point before pushed against the first L-shape at the target position. The depicted assembly process with square policies nearly succeeds but eventually fails at positioning the second L-shape at the target position. A frequent issue during the execution of this task for both, L-shape policies and square policies, was that the second L-shape could not be positioned adequately at the target way point. During the maneuvering of the object the swarm then pushed the first L-shape away from its target pose and thus breaks the assembly process. In our experiments the assembly process succeeded in 5 of 5 trials for the L-shape policy, however in 2 trials the first L-shape was pushed away during maneuvering the second L-shape. With the square policy, the assembly task succeeded in 2 of 5 trials, although in both successful trials the first L-shape was pushed away from its target position.

The third task is the assembly of a C-shape with a T-shape as depicted in Figure 3.10. First, the C-shape has to be pushed through a way point to guide the rotation into the final pose. Then the swarm is repositioned at the T-shape which is pushed to an intermediate way point with the right orientation for pushing it into the final position. In our experiments, the CT-shape-assembly task succeeded in 3 of 5 trials with policies learned on C- and T-shapes, respectively. With the square policy, the CT-shape-assembly task succeeded in 0 of 5 trials.

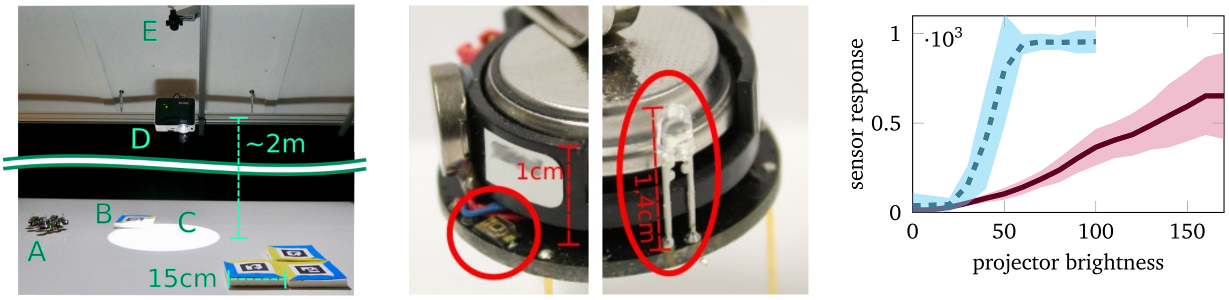
### 3.5.3 The Kilobot Setup

We use a horizontally mounted 2m × 1.5m whiteboard as environment for the Kilobots. The whiteboard provides a reflective surface with low friction which is beneficial for the slip-stick motion of the robots. We further emulate a light source using a projector mounted vertically to the ceiling. To control the swarm, we project a circular gradient with radius 0.2m and use the phototaxis algorithm on the Kilobots [5]. Figure 3.11a depicts the setup.

In contrast to the original design developed at Harvard [84], the commercially manufactured Kilobots[3] have a surface-mounted device (SMD) light sensor at the side of the battery instead of the through hole (TH) diode at the back. However, this change in the

---

[3] Distributed by K-Team, `http://www.k-team.com/`



**(a)** Experimental Scene **(b)** Hardware Modifications **(c)** Sensor Response Curves

**Figure 3.11:** (a) The Kilobot swarm (A) pushes the assembly objects (B) on a $2\text{m} \times 1.5\text{m}$ whiteboard. The circular light gradient (C) is projected onto the table by a video projector (D). The scene is observed with an RGB camera (E). (b) Modification of the Kilobot hardware, to achieve a good phototaxis behavior. Left: commercially available Kilobot with an SMD light sensor. Right: modified Kilobot with a through-hole diode as in the original design. (c) Sensor response curves of SMD sensor and TH diode. The TH diode has a much greater dynamic range. The plots show mean and average over 5 and runs for SMD sensor and TH diode, respectively.

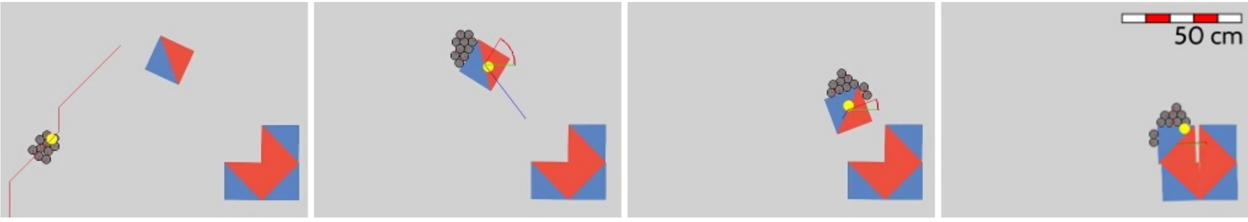


**Figure 3.12:** The assembly task of four squares into a big square in simulation. The Kilobots are depicted by gray circles and the light position by a yellow circle. A video of both experiments is available at `https://youtu.be/kuU8wsR9dD4`.

design significantly decreases the performance of the phototaxis algorithm. Additionally, the chosen SMD sensor has a roughly three-times-reduced dynamic range in comparison to the TH sensor which we chose as replacement (see Figure 3.11c).

To obtain the positions of the Kilobots and the objects in the scene, we apply simple detection and tracking algorithms. However, the low illumination of the scene (which is required for the phototaxis behavior of the Kilobots) and the bright circular gradient projected onto the table exceeds the dynamic range of the RGB camera. To overcome this problem, we generate HDR images from images with different exposure times.

To achieve a stable and robust tracking of the pose of arbitrary objects, we mark the objects with Chilitags [7]. Chilitags are precise, reliable and illumination tolerant 2D fiducial markers and thus are well suited for the experimental setup. We track the Kilobots using a Hough circle transform (HCT) which is well suited for the round geometry of the robots. HCT is not as precise and robust as the Chilitag tracking, but since the policy uses a distribution-based state representation, it is less sensitive to noise in the Kilobot state.

### 3.5.4 The Assembly Task on the Kilobots

We have evaluated the assembly task on the real Kilobot platform with the modifications described in the previous section. We have learned the object movement policies with ratios $\rho = 0$, $\rho = 0.25$, $\rho = 0.5$, $\rho = 0.75$, and $\rho = 1$ in simulation with a swarm size of 15 Kilobots and evaluated on the real Kilobots using swarms of 12, 15, and 24 agents. For the experiment with the real Kilobots, the swarm size is limited as the area of the circular gradient is limited and the robots outside of the gradient are not controllable anymore. Still, the phototaxis performance is not sufficient to keep all robots reliably in the area of the gradient. We apply policies learned in simulation directly to the Kilobot platform. No further optimization on the real robots is required. Figure 3.12 depicts a simulation of the assembly task that we later applied to the real Kilobot platform. The experiment on the real Kilobots is depicted in Figure 3.13.

With 12 Kilobots, our approach was able to push the fourth square close to the remaining three squares. Yet, only around half of the swarm remains in an area of the gradient when approaching the final position. Consequently, the swarm is not able to finish the assembly by correcting the orientation of the square. With a swarm size of 15 agents the assembly task has succeeded. Although again many robots fail to follow the light source, the number of Kilobots that remain in the area of the gradient is sufficiently large to finish the assembly task. With 24 Kilobots in the swarm, the assembly task has been completed successfully as well and also the time consumption of the task could be reduced to ca. $700s$ in contras to ca. $950s$ that were required by the assembly process with 15 Kilobots.

## 3.6 Conclusions

In this paper, we have presented a novel method for solving the assembly task using a common input signal to a swarm of simple agents. Our method learns policies for the input signal such that the swarm, by following this input signal, pushes an object into a given direction or rotational movement. For this learning method, we have introduced a swarm representation that is invariant to the number of agents in the swarm and their specific locations. This representation simplifies not only the search space for the learning method, it also allows to transfer the learned policy to different swarm sizes. We could show that a policy learned on a simple shape generalizes to a certain extend, still, policies specifically learned for a certain object shape outperforms a general object movement policy. We applied the learned object movement policies in a hierarchical Kilobot controller. We could show in simulation and on the real Kilobot platform, that the Kilobot controller is able to solve the object assembly task. The learned policies could be transfered directly to the real robots without any additional learning.

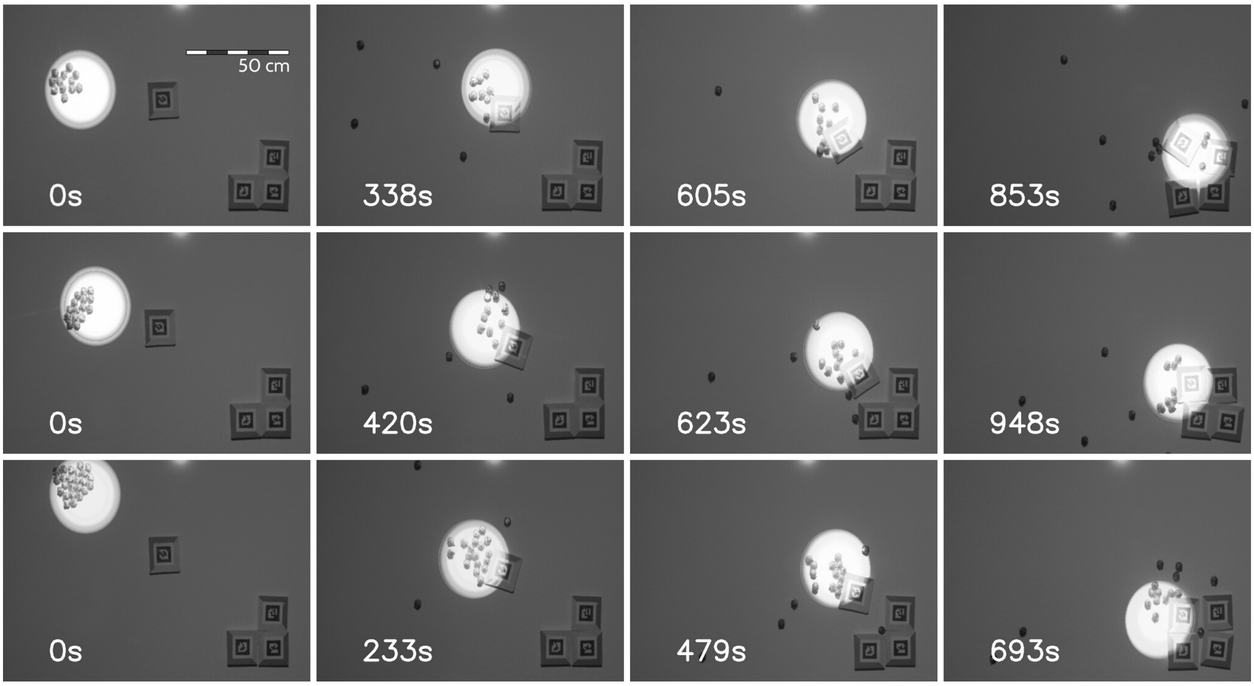


**Figure 3.13:** Assembly of a square part with three similar parts into a big square with different swarm sizes. In the first row: 12 Kilobots, in the second row: 15 Kilobots, in the third row: 24 Kilobots Multiple robots are lost during the run, larger swarm sizes lead to better performances and faster execution. A video is available at `https://youtu.be/kuU8wsR9dD4`.

# 4 Using M-Embeddings to Learn Control Strategies for Robot Swarms

In the previous chapter, I have presented the swarm kernel which uses mean embeddings as representation for sets of homogeneous observations. The idea is to represent the generative distribution of the samples instead of the sample set itself. This representation abstracts from the specific samples and is invariant to the order of the samples and the cardinality of the set. We have employed the swarm kernel in a reinforcement learning algorithm to find a policy for a robotic swarm.

In recent years, the resurgence of neural networks in classification and regression problems has also led to the advent of deep reinforcement learning (DRL). In DRL, value functions and policies are approximated by neural networks. However, neural networks usually have a predefined structure which requires that the number of inputs and outputs is known in advance. For sets of observations without order and of variable cardinality, this is a severe limitation. Such sets could be taken as input by assuming a maximum number of samples, concatenating the individual observations, and filling missing observations with values that do not influence the outcome. However, handling the samples individually would require to learn redundant feature mappings for each input, where neural networks are already by itself a highly redundant structure that adheres many local minima.

In this chapter, I investigate how the idea of kernel mean embeddings can be transferred to neural networks. The recently presented deep mean embeddings [44] emulate the kernel mean embeddings by learning the feature function and using a mean reduction to compute a single feature vector from the feature mappings of the samples. Intuitively, the mean reduction computes a distribution over the activation of the feature dimension similar to a histogram. In this chapter, I generalize this structure by introducing a max- and a softmax-reduction as alternatives. The intuition behind the max-reduction is to collect the activations of the individual sample features into a single feature vector. The softmax reduction allows to scale between mean- and max-embedding per feature dimension, where the scaling parameter can be learned.

I employ the deep M-embedding for learning policies for swarm agents that observe each, the other agents and the objects in the environment, as such a homogeneous set. In experimental evaluations, I demonstrate that the generalization of the deep mean embeddings to the deep M-embeddings is necessary to learn policies in such settings.

## 4.1 Introduction

Swarms in nature are groups of often simple animals which, as a collective, exposes a complex behavior. Examples are bees, ants and termites, or fish schools. As a collective they achieve much higher goals than an individual would be able to, such as building large structures, defending against predators, or foraging in environments of sparse nutrition

[6]. Swarm robotics aims to build similar collectives of simple agents which can achieve higher goals if they act as a collective. Such agent are usually quite limited in their sensory and motor skills. Also the computational power of these agents is often restricted to a small micro-processor and little memory. A strength, however, is their inherent redundancy which adds robustness against failures and a high parallelization of the tasks. For the application of learning control policies, these limitations pose challenges such as an upper limit on the complexity of the learned controller. However, learning also profits from the large number of agents which allows for parallel exploration by either running slightly different control routines on each agent or by gathering different experiences for the same policy.

In this work, we want to address the problem of learning swarm policies for object manipulation using reinforcement learning (RL). Learning policies for swarms is a hard task because it usually implies a large state space and also a large action space. Both spaces need to be represented and explored in RL to find a good policy for controlling the agents. Recent development in reinforcement learning which leverages from the power of deep neural networks in learning compact feature representations allow for learning controllers in such high dimensional settings with large state and action spaces. Deep reinforcement learning (DRL) has proven to successfully learn polices directly from large dimensional observations such as images and for high-dimensional action spaces such as humanoid robots. In DRL, neural networks have been applied to either approximate a state(-action) value functions [67, 68, 37] and/or the policy itself [88, 90, 4, 59]. In this paper, we use an actor-critic variant of trust region policy optimization (TRPO) [89] to learn the swarm policy, which we discuss in more detail in Section 4.2.1.

Learning policies for swarms of robots to act in environments cluttered with objects using deep networks as function approximators is challenging because of two reasons. First, networks usually have a predefined structure which requires that the number of inputs and outputs is known in advance. For swarms this would require that we know the exact number of agents in the swarm before we learn the policy and later we can also apply the learned policy only to swarms with exactly the same size. Second, while a neural network itself is already highly redundant structure (the nodes in a layer are exchangeable), the state space of a swarm introduces a lot more redundancy into the problem, i.e., having agent $a$ at position $p_1$ and agent $b$ at position $p_2$ is for homogeneous agents equivalent to $a$ being at $p_2$ and $b$ at $p_1$. Furthermore, both of these issues also apply for other observations such as the objects in the environment.

In this work, we use a network structure that allows to compute a fixed size representation from a set of observations. With this network structure, we can learn functions that can take a variable number of observations as inputs. In this structure, each observation is processed individually by an arbitrary feature network, before the outputs of all observations are combined into a single feature representation. This combination of feature activations is inspired by the kernel mean embeddings of distributions [100]. In prior work, [44] have presented the deep mean embeddings on which we build this work. We present different variants which involve a *mean*, a *max*, or a *softmax* operation and, hence, we call them the *Deep M-Embeddings*.

While the deep M-embeddings solve the issue of a variable number of agents in the swarm or objects in the environment, we still need to compute an action for each agent.

To this end, we centrally learn a decentralized policy that acts locally on the agent using a parameter sharing approach [34]. That is, we learn a single policy for all agents which takes the local observations of an agent as input and outputs the local action for the agent. Thus, the learned policy could be run locally on the individual swarm agents. The reward signal for learning this policy, however, is a global reward signal to the whole swarm. A centralized critic is learned from this reward signal and used for updating the local policy.

In experimental evaluations, we compare the variants of the deep M-embeddings with each other and to a standard multi-layer perceptron which has been used in state-of-the-art RL literature. We further show, that using the proposed network structure of the deep M-embeddings enables us to learn complex control strategies for swarms including multi-modal problems (sorting objects of different types).

### 4.1.1 Related Work

After the latest successes of DRL in single-agent learning problems, also the application of deep learning methods to multi-agent reinforcement learning (MARL) has become an area of more and more interest in the recent years. The prevalent problems of MARL is the non-stationarity of the environment which often leads to instabilities in the learning algorithms that prevent the learner to find good solutions, as well as, depending on the problem, the partial observability of the system. While most of the contributions in multi-agent DRL focus on these problems, they usually neglect the problem of the growing dimensionality and the interchangeability in the state space.

The deep mean embeddings—on which we build our work—are presented in [44]. The authors successfully employ the deep mean embeddings in the policy and value function networks that are learned with TRPO in the rendezvous and pursuit evasion task. In [27] a swarm kernel based on the kernel mean embedding of the swarm state is presented. The kernel is used with actor-critic REPS [54] to learn policies for guiding the swarm with a common input signal to manipulate objects.

[119] present a simulation environment for massive multi-agent RL which uses images as representation with multiple layers for storing different entities of information. The environment is a grid world with discrete states and, thus, a representation of continuous states, which would be a limitation of images, is not necessary. [118] apply mean-field theory to approach the problem of increasing dimensionality in settings with many agents. Instead of modeling the interactions with all other agents, each individual agent only considers the average effect of its local neighborhood. The derived mean-field Q-function is applied to actor-critic learning with deterministic policy gradients [96].

Many approaches aim for robustly learning the Q-function in the multi-agent setting. [57] introduce distributed Q-learning where optimistic agents only update their Q-values for positive TD-errors. [63] follow this direction but instead of neglecting negative updates of the Q-function, they introduce a hysteresis to the update. [71] approach the problem of multi-task multi-agent reinforcement learning by combining hysteretic Q-learning with deep recurrent Q-networks (DRQN) using concurrent experience replay memory. DRQNs [38] extend the deep Q-networks [68, 67] to the partially observable setting by adding a recurrent LSTM [39] layer to the network architecture. By further applying policy distillation [86], [71] can combine expert Q-networks of each agent into a multi-task policy. A problem of this approach is that the trajectories stored in the experience replay memory get outdated because of the changing behavior of the other agents. To approach this problem, [72] introduces leniency [73] for controlling the influence of negative policy updates from the experience replay memory.

[108] learns agent policies with joint reward signal using a decomposition of the swarm value function into agent value functions. The centralized Q-function is the additive composition of the agent Q-functions. Thus, by learning the centralized Q-function, the agent Q-functions are learned. [82]: builds on this work, but instead of additive composition only requires monotonicity in the selection of the optimal action. Thus allowing for more complex agent value functions.

[34] investigate the application of prominent DRL methods (DQN, DDPG, A3C, TRPO) to the multi-agent setting by introducing parameter sharing. In this work, we use this approach with TRPO, but since we assume homogeneous agents, we omit the agent index. [60] present the multi-agent deep deterministic policy gradient (MADDPG), an extension of the actor-critic DDPG [59] to the multi-agent scenario by learning a centralized Q-function as critic, while updating the policies locally. Similarly, [18] introduce the counterfactual multi-agent (COMA) policy gradients. COMA uses a centralized critic that takes the joint action and uses a counterfactual baseline. This baseline is separate for each agent and uses counterfactuals in which only the agents action changes to improve the assessment of the impact of the agents action on the reward signal. [75] introduce a multiagent bidirectionally-coordinated network (BiCNet) with a recurrent structure that allows for information sharing between agents. However, the proposed learning method requires individual rewards instead of a global reward signal.

[32] present a framework for learning representations of policies in a multi-agent setting using an encoder-decoder structure. The representations are learned from observed trajectories and are used to characterize, imitate, and adapt to the other agent's behavior. Similarly, [94] introduce a gradient-based meta-learning algorithm based on [17] to quickly adapt the agent's policy to the opponent's behavior in a non-stationary competitive scenario. Other approaches use genetic algorithms such as particle swarm optimization to learn the policies for robot swarms [79, 78].

## 4.2 Preliminaries

We use deep reinforcement learning for obtaining the control policies. In the following paragraphs we shortly discuss the policy gradient method we applied, and the network structure we used for approximating policy and value function.

### 4.2.1 Trust Region Policy Optimization

In reinforcement learning (RL), the problems which we want to optimize are usually given as a Markov decision process (MDP). An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P$ is the state transition model (usually defined as a probability distribution $P(s'|s,a)$), $r$ is the reward function, $\rho_0$ is the initial state distribution, and $\gamma$ is the discount factor. In each state $s_t \in \mathcal{S}$, an agent choses an action according to a (stochastic) policy $\pi(a_t|s_t)$. Applying this action results in a transition from state $s_t$ to $s_{t+1}$ according to $P(s_{t+1}|s_t,a_t)$ for which the agent receives the reward $r(s_t,a_t,s_{t+1})$. Reinforcement learning (RL) aims to find the policy $\pi^*$ that maximizes the expected discounted reward of the agent

$$\mathbb{E}_{a_{0:T},s_{0:T}}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t,s_{t+1})\right]. \tag{4.1}$$

If we assume a policy function parameterized by the parameter vector $\boldsymbol{\theta}$, we can find the optimal policy by applying a policy gradient method. Policy gradient methods try to optimize a parametric policy by following the gradient of the expected return with respect to the policy parameters. Directly following the gradient, however, can lead to disastrous behavior as small changes in the parameter space might lead to large changes in the state-action distribution $p(s,a)$. To circumvent this problem, the update step is usually subject to constraints on the divergence of the policy to ensure that the policy changes slowly and gracefully [48, 77]. In this work, we use Trust Region Policy Optimization (TRPO) [88]. By applying a constraint on the Kullback-Leibler (KL) divergence between the old policy $\pi_{\theta_{\text{old}}}$ and the new policy $\pi_\theta$ and doing further approximations to the optimization problem, TRPO optimizes the objective

$$\begin{aligned} J(\theta) &= \mathbb{E}\left[\frac{\pi_\theta(s,a)}{\pi_{\theta_{\text{old}}}(s,a)}\hat{A}(s,a)\right] \\ \text{s.t.}\quad & \mathbb{E}\left[D_{\text{KL}}\left(\pi_\theta(s,a)||\pi_{\theta_{\text{old}}}(s,a)\right)\right] \leq \delta. \end{aligned} \tag{4.2}$$

Here, $\pi_{\theta_{\text{old}}}(s,a)$ is the policy with the old set of parameters, $\hat{A}(s,a)$ denotes the advantage function and $D_{\text{KL}}$ is the KL divergence which is bound by a hyper-parameter $\delta$.

The advantage of an action $a$ in state $s$ is the difference between the expected future return for taking $a$ in state $s$ and from then following the policy $\pi$ over directly following $\pi$ from state $s$. It can be expressed as the difference between the state-action value function $Q^\pi(s,a)$ and the state value function $V^\pi(s)$, i.e.,

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{4.3}$$

The estimator of the advantage function $\hat{A}(s,a)$ can be obtained from samples collected during roll-outs of the policy with the old parameters using *generalized advantage estimation* (GAE) [89]. Similar to TD($\lambda$), a parameter $\lambda$ allows to scale between temporal-difference (TD) ($\lambda = 0$) and Monte-Carlo ($\lambda = 1$) estimates. While TD updates have less variance, they are highly biased by the current estimate of the value function. In contrast, Monte-Carlo estimates are unbiased but have a large variance [110, 109].

TRPO solves the constraint optimization problem by linearizing the objective and uses the Hessian of the KL as approximation to the covariance of the gradients. As neural networks are considered as approximator of the policy function, the parameter vector $\boldsymbol{\theta}$ is usually very high dimensional. Using the conjugate gradient algorithm alleviates this problem. Additionally, a line search along the found gradient ensures that the objective improves and that the constraints are met. GAE requires the estimation of a value function $V(s)$ for estimating the advantages. This value functions is optimized using a similar objective as in (4.2) [89].

### 4.2.2 Neural Networks as Function Approximator for Policy and Value

We use neural networks (NN) to approximate the policy and the value function. Neural networks consist of multiple layers of linear mappings, with non-linear activations, i.e.,

$$\boldsymbol{h}_i = \sigma(\boldsymbol{W}\boldsymbol{h}_{i-1} + \boldsymbol{b}), \qquad i = 1, \ldots, m \tag{4.4}$$

where $h_i \in \mathbb{R}^{d_i}$ is the output of the $i$-th layer, $\boldsymbol{W} \in \mathbb{R}^{d_i \times d_{i-1}}$ is a weight matrix, $\boldsymbol{b} \in \mathbb{R}^{d_i}$ a bias vector, $\sigma$ a non linear activation function, and $d_i$ is the number of neurons in the layer (or the size of the layer). The input $\boldsymbol{h}_0$ to the first layer is the input to the neural network, in our case the observation of the environment. The choice of activation functions is wide, however most DRL applications use the *tanh* function or the (leaky) rectified linear unit (RELU).

In TRPO and similar approaches [90], policy and value approximator usually use the same network structure up to the last layer as *feature network*. Note however, that in TRPO the parameters of the networks are not shared. To obtain a value function, the output of the feature network is mapped by the last layer to a scalar value. In contrast, the policy is usually represented as a parametric distribution, where the last layer of the policy network maps to the parameters of the distribution. Hence, for a Gaussian distribution we would get

$$\pi(a_t|s_t) = \mathcal{N}(a_t; \mu(s_t), \Sigma(s_t)), \tag{4.5}$$

where $\mu(s_t), \Sigma(s_t)$ is given by the output of the policy network.

### 4.2.3 Mean Embeddings of Distributions

A general issue with neural networks is that the structure needs to be fixed in advance before optimizing the parameters of the network. A variable dimensionality of the inputs during or after training is hard to implement into the structure of the network. In the case of learning policies for swarms this is a critical issue since the actual number of agents in the swarm is not important as long as there are enough agents to solve the task. Moreover, as a key argument of swarm robotics is the robustness against failures due to the high redundancy, a policy should be able to deal with changes in the number of swarm agents. Furthermore, with homogeneous agents, the allocation of the specific agents to the position in the swarm is arbitrary, switching the position of two agents does not change the state of the swarm. Thus, if the representation respects this invariance to permutation (i.e. allocation to positions) and to the size of the swarm, the search space for policy and value function is drastically reduced.

In prior work [27], we have leveraged the embeddings of probability distributions into reproducing kernel Hilbert spaces (RKHS) [100] to construct a kernel function that enables this invariant representation of the swarm. An RKHS $\mathcal{H}$ is uniquely defined by a positive definite kernel function $k(\boldsymbol{x}, \boldsymbol{x}') := \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{x}') \rangle_{\mathcal{H}}$, where the feature function $\varphi(\boldsymbol{x})$ is usually intrinsic to the kernel function and maps the vector $\boldsymbol{x}$ into a potentially infinite dimensional space. We can embed a marginal distribution $p(X)$ as the expected feature mapping $\mathbb{E}_X [\varphi(\boldsymbol{x})]$. In practice, we use a sample based estimator

$$\hat{\mu}_X = \frac{1}{N} \sum_{i=1}^{m} \varphi(\boldsymbol{x}_i) = \frac{1}{N} \sum_{i=1}^{m} k(\boldsymbol{x}_i, \cdot). \tag{4.6}$$

In [27], we have used such embeddings to represent the swarm as a distribution, where each agent is a sample from the distribution. While we cannot represent the embedding explicitly (due to the infinite dimensionality), we can construct a kernel function which compares two swarm configurations. [44] have introduced the deep mean embeddings, which are inspired by the kernel embeddings of distributions but use a neural network to explicitly compute the feature mappings. In this work we build on the deep mean embeddings, but introduce a more broad formulation: the deep M-embeddings.

## 4.3 Deep M-embeddings

As input to the deep M-embeddings (DME), we assume $N$ observations $o_i \in \mathbb{R}^{d_o}$ from an environment, where each observation has the same nature (e.g., each observation is the position of an agent or each observation is the location of an obstacle). Note that the number of observations, $N$, does not need to be fixed or predefined, as we will see later on. The DMEs allow to compute a compact representation of a variable set of homogeneous inputs to a neural network.

We consider three different types of deep DMEs: deep mean embeddings, deep max embeddings, and deep soft-max embeddings. All variants of the DMEs have in common that they map each observation $\boldsymbol{o}_i$ through a feature network $\Phi : \mathbb{R}^{d_o} \to \mathbb{R}^{d_\phi}$, yielding a feature vector $\phi(\boldsymbol{o}_i)$ as output.

The **deep mean embedding** then combines the feature vectors $\phi(\boldsymbol{o}_i)$ by computing the mean, i.e.,

$$m_j = \frac{1}{N} \sum_{i=1}^{N} \phi_j(\boldsymbol{o}_i), \qquad j = 1, \dots, d_\phi, \tag{4.7}$$

where $j$ is the index of the output vector. This embedding is the direct translation of the mean embeddings discussed in Section 4.2.3 using the feature vector defined by the neural network.

Instead of averaging, the **deep max embedding** takes the element wise max of the feature vectors $\phi(\boldsymbol{o}_i)$, i.e.,

$$m_j = \max_i \phi_j(\boldsymbol{o}_i), \qquad j = 1, \dots, d_\phi. \tag{4.8}$$

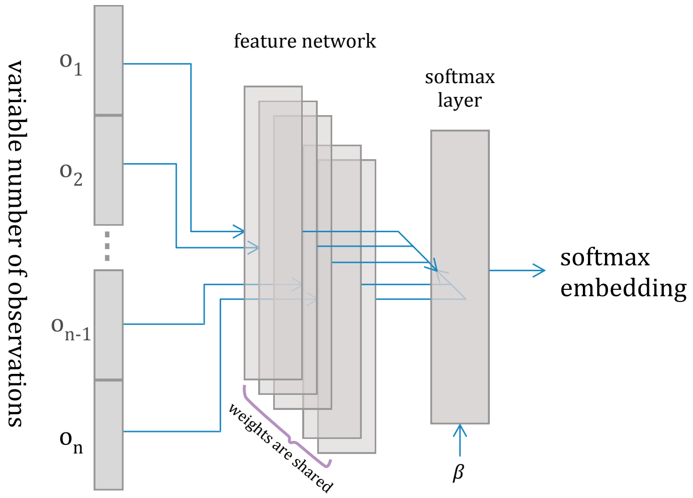


**Figure 4.1:** Structure of the soft-max embedding network. The network takes a variable number of observations as input. Each observation is mapped by the same feature network (which can be an arbitrary network structure) to obtain feature activations for each observation. The soft-max layer combines the feature activations of all observations into a single representation. The temperature parameter $\boldsymbol{\beta}$ of the soft-max embedding is a variable of the network which can be optimized alongside the other network parameters.

The intuition behind this embedding is that the feature network could learn to activate certain entries of the feature vector to express details of the inputs. Rather than averaging over these activations, we might want to combine the activations similar to an operation.

The **deep soft-max embedding** computes the element-wise soft-max of the feature vectors. Using a temperature vector $\boldsymbol{\beta}$, which is a variable of the network and can be learned alongside the other network variables, the soft-max embedding allows to scale between the characteristics of the deep mean embedding and the deep max embedding for each element of the feature vector. The output of the deep soft-max embedding is given by

$$m_j = \sum_{i=1}^{N} w_{ij}\phi_j(o_i), \qquad w_{ij} = \frac{\exp\left(\beta_j \phi_j(o_i)\right)}{\sum_i \exp\left(\beta_j \phi_j(o_i)\right)}, \qquad j = 1, \ldots, d_\phi. \tag{4.9}$$

The deep soft-max embedding is at least theoretically preferable over the mean- and the max-embedding as it can represent both characteristics. Moreover, these characteristics can be scaled per element of the feature vector and can be learned along the other network parameters.

The network structure of a deep soft-max embedding is depicted in Figure 4.1. The deep mean embedding and the deep max embedding have the same structure but use a different reduction for the output of the feature network.

## 4.4 Learning Swarm Policies

We consider the problem of object manipulation and object assembly with a swarm of *homogeneous agents*. The approach we present in this paper learns a policy that acts locally, i.e., the policy takes the *local observations* of the agent as input and returns an action for the agent, similar to [34]. The problem settings in which each agent acts individually are often formulated as partially observable Markov decision process (POMDP) as the state and intentions of the other agents are unknown. Solutions to POMDPs usually require a stateful policy, where the state maintains a belief about the state of the other agents. Learning such policies requires advanced techniques with recurrent neural networks. To circumvent this problem, we assume in this paper that each agent get *full observations* of the state of the system.

These assumptions, i.e., homogeneous agents and local observations of the full state, are strong, however they provide some benefits compared to the approach of a single actor that computes joint actions for the entire swarm. First, the policies that are learned with our approach could be applied later to robots without the need for a central control unit. Second, the evaluation of the policy on the agents is parallelized by the swarm. From each step in simulation, we get multiple samples of state, action, reward and next state, depending on the number of agents in the swarm. In the next paragraphs, we present the agent model, the policy network, and the reward functions we have used for learning the swarm policies in detail.

### 4.4.1 Swarm Agents

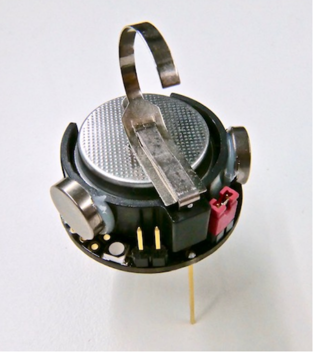


**Figure 4.2:** A Kilobot.

We assume homogeneous, disc-shaped agents with single or double integrator dynamics. Our agents are inspired by the Kilobots (see Figure 4.2 for an example), however, the dynamics and the perceptual abilities of our agents differ substantially. While the Kilobots are actuated via two vibration motors that lead to a rotational or linear movement via the slip-stick principle [83], we simulate dynamics that take directly linear and rotational velocities for the single integrator or accelerations for the double integrator. With the single integrator dynamics, the state of the agent is the position as $(x, y)$ coordinates and the orientation as angle $\alpha$. The action is the two dimensional vector $a = (v_l, v_\alpha)$ with the linear and angular velocity of the agent. For the double integrator dynamics, the state of the agent is the position $(x, y)$, the orientation $\alpha$, and the velocity $(v_l, v_\alpha)$. The action is in this case the linear and angular acceleration $(a_l, a_\alpha)$.

The agents in our setup observe the full state of the environment from a local perspective. The relative positions of the other agents and of the objects are observed as polar coordinates $(r, \rho)$, where we transform the angle to $\sin(\rho), \cos(\rho)$. The orientations of other agents and objects are perceived relative to the own orientation of the agent. If we use agents with double integrator dynamics, we also observe the velocities of the other agents. To distinguish between different object types, we added the color of the objects to the observations of the agents. Finally, the agents have a proprioception of their own location, their own orientation as sin/cos transformation and in the case of double integrator dynamics also their own velocity. Figure 4.3 summarizes the local observations of the swarm agents. In difference to our model, the Kilobots are not able to perceive the

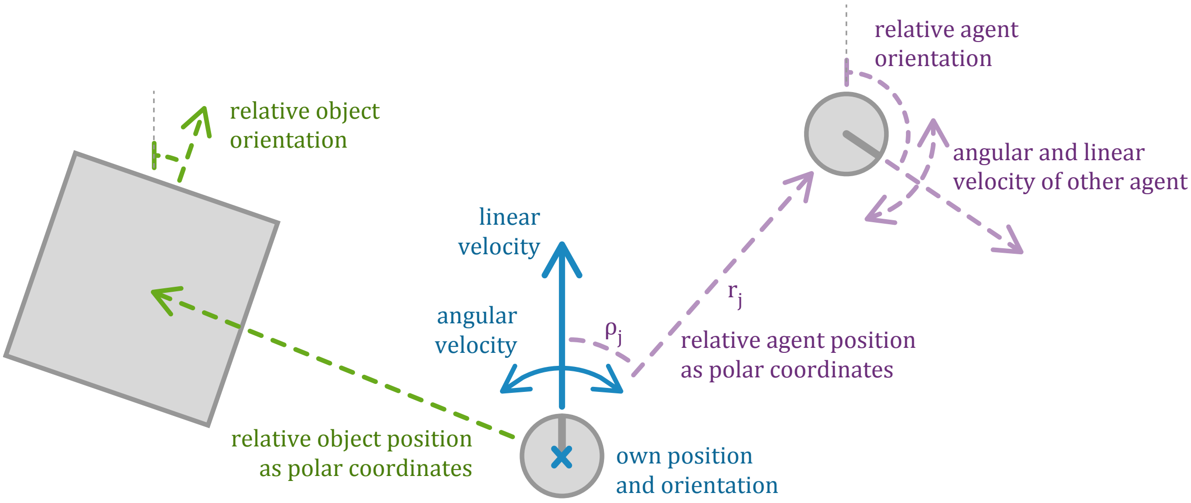

relative agent orientation
relative object orientation
angular and linear velocity of other agent
linear velocity
$r_j$
angular velocity
$\rho_j$
relative agent position as polar coordinates
relative object position as polar coordinates
own position and orientation

**Figure 4.3:** Perception model of the swarm agents. The agent observes the other agent's relative positions in polar coordinates, the relative orientation (using a sin/cos transformation for the angle), and the angular and linear velocity (purple). The positions and orientations are also observed relative to the agent's state (green). Lastly, the agent observes its own position, orientation, and velocities (blue).

full state of their environment, but only light intensities and an estimate of the distance to other agents in their local neighborhood.

### 4.4.2 Policy and Value Function Network

To learn policies and value function approximators in scenarios with a variable number of homogeneous observations, we propose a network structure that uses an M-embedding for each type of homogeneous observation. For example, in the scenario of object manipulation with robot swarms, the observations of the swarm agents form a set of homogeneous inputs and, similarly, the observations of the objects form another set of homogeneous inputs. We feed the remaining inputs (e.g., the proprioceptive observations of the swarm agent) through a set of fully connected layers. The fixed sized outputs of the M-embeddings are then concatenated together with the activations of the remaining inputs and fed through another set of fully connected layers. A schematic diagram of the network architecture is depicted in Figure 4.4.

The proposed network structure can then be used in any deep reinforcement learning method for approximating a value function and/or the policy with sets of homogeneous inputs. In our experimental evaluations, which we discuss in the next section, we have used an actor-critic variant of TRPO which estimates a value function from TD($\lambda$) errors estimated using generalized advantage estimation [89].

## 4.5 Experimental Setup and Evaluation

In the following paragraphs, we want to present our experimental setup and discuss the results we have obtained from the evaluations of the proposed algorithm. With our experimental evaluations, we address the following questions:

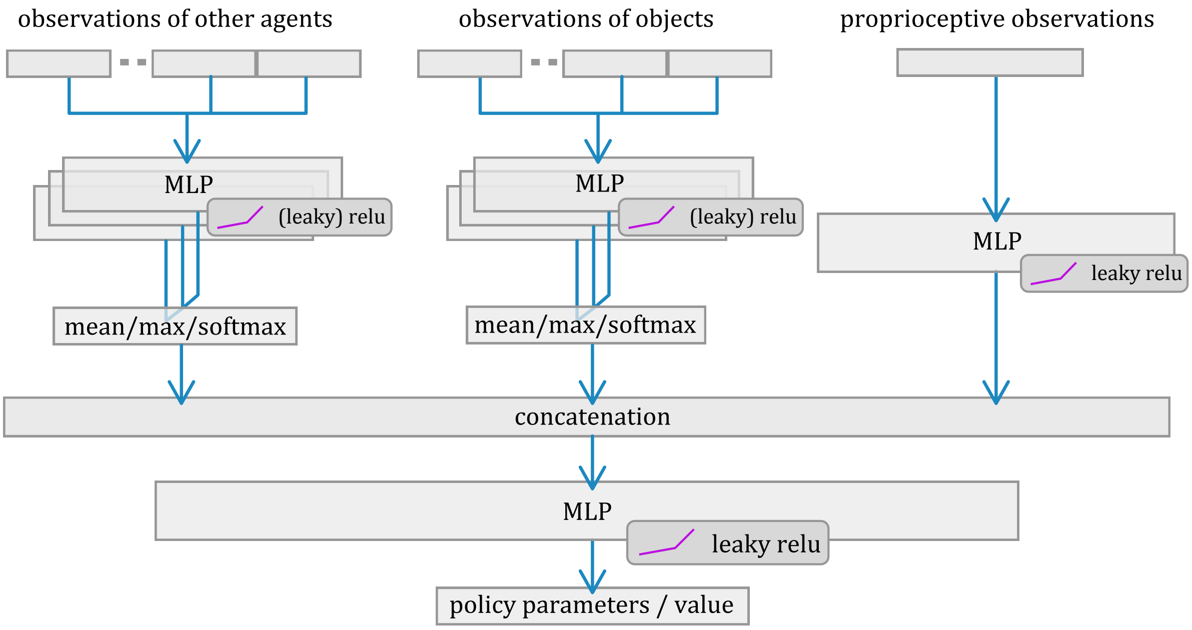


**Figure 4.4:** Structure of the policy and value function network. The observations of the other agents and the observations of the objects are processed by an M-embedding. The proprioceptive observations are processed by an MLP. The output of the embeddings and the MLP are concatenated and mapped by another MLP to the parameters of the policy distribution or to a value of the observations.

1. How do the different deep M-embeddings perform in comparison to each other?
2. How does our proposed network structure perform against a simple multi-layer perceptron which has been used in state-of-the-art robot learning applications?
3. Can we solve challenging tasks of swarm robotics such as the assembly of multiple objects, or the segregation of different object types?
4. How does the $\beta$ variable in the softmax-embedding change during learning?
5. Can we transfer the learned policies to different swarm sizes and a different number of objects?

Before addressing these questions in Section 4.5.3, we present the simulation environment, in which we have conducted our experiments, and give a short overview of the tasks and reward functions we have used for the evaluation.

### 4.5.1 The Kilobot Gym

The Kilobot Gym[1] is a simulation framework based on the OpenAI Gym [10] which allows to evaluate reinforcement learning algorithms for swarm robotics. The framework offers two modes of operation: either the agents follow a simple, hard-coded logic, e.g., a phototactic behavior, where the swarm is controlled via a global input signal such as a light

---

[1] Code available at `https://github.com/gregorgebhardt/gym-kilobots`

source; or the agents receive individual actions based on their local state and observation. While we have used the former mode in prior work [27], we use the latter in this work to learn a policy that acts locally for the individual agent.

We simulate the physics using the 2D simulation framework Box2d [11] at a rate of 10 Hz. At each simulation step, the velocities for all agents are computed (either we take directly the velocity as action or we integrate the acceleration at each time step) and set to the respective bodies in the simulation environment. The simulation is stepped 20 times for each call to the environments step function which results in a time step of 2 seconds per action. Since the Kilobots are a very slow system, taking such a long time step is not an issue compared to highly dynamic systems.

### 4.5.2 Tasks and Reward Functions

We evaluate the proposed network structure on three tasks of object manipulation with robot swarms. The first task is simply to push the objects through the environment, the second task is to assemble a set of objects, and the third task is the segregation of two types of objects.

A critical part of reinforcement learning is the credit assignment of the reward to the decisive actions for the obtained reward. Techniques such as reward shaping [69] have been applied to alleviate this problem. In the multi-agent setting this problem becomes even more prevalent, since we assume global reward signals that might be induced by any of the agents actions. Thus, giving credit to the correct action selection of the individual agent becomes an even harder task. In our setting, we want to achieve a certain manipulation of the objects, hence the reward is only indirectly coupled with the action of the agent. While we use techniques such as GAE [89] to learn a baseline that removes a lot of variance from the estimation of the expected reward of a trajectory, we still noticed that the learning is very sensible to the selection of the reward function. For example, using an absolute reward, e.g., the distance of the object to a goal, or a sparse reward, e.g., giving a reward of 1 if the object is close enough to a goal pose, did prevent the learner from finding a good solution. Instead, we had to use relative rewards between the current state $\boldsymbol{s}_t$ and the next state $\boldsymbol{s}_{t+1}$.

**Push Objects.**

The first task is to push the objects in the environment. To evaluate a state-action pair, we first compute the difference $d(o_{i,t}, o_{i,t+1})$ between the current and the next position of all objects and then take the sum

$$r_t = \sum_i d(o_{i,t}, o_{i,t+1})$$

as reward. Thus, the more an object is pushed, the higher is the reward.

**Assemble Objects.**

In the second task, the swarm has to assemble all objects in the scene. First, we compute the point-wise distances $d(o_i, o_j)$ between all object positions at the current time step $t$ and the next time step $t+1$, respectively. We then take the sum over the difference between the distances as reward

$$r_t = \sum_{i,j>i} d(o_{i,t}, o_{j,t}) - d(o_{i,t+1}, o_{j.t+1}).$$

Hence, if the objects approach each other this difference is positive and so is the reward.

**Segregate Objects.**

In this last task, the goal is to segregate the objects in the scene into groups of the same kind. The reward is composed of two parts: one for assembling each group of objects with the same type, and one for separating the object groups. The first part is computed similar to the reward in the previous task, but for each group individually, i.e.,

$$r_{g,t} = \frac{1}{|g|} \sum_{i,j>i} \delta(i \in g)\delta(j \in g)(d(o_{i,t}, o_{j,t}) - d(o_{i,t+1}, o_{j,t+1})),$$

where $\delta(i \in g)$ is the indicator if object $o_i$ belongs to group $g$ and $|g|$ is the cardinality of group $g$. Similar to the previous task, this reward reflect if the objects of each group are approaching each other or not. The second part is computed using the differences of the point-wise distances of the mean positions of all groups, i.e.,

$$r_{\mu,t} = \frac{1}{n} \sum_{g,h>g} d(\mu_{g,t+1}, \mu_{h,t+1}) - d(\mu_{g,t}, \mu_{j,t}),$$

with groups $g$ and $h$ and the number of groups $n$. This reward reflects if the group center diverge. The total reward of this task is then computed as weighted sum $r_t = 1.5\frac{1}{n}\sum_g r_{g,t} + r_{\mu,t}$.

### 4.5.3 Experimental Evaluation

**The swarm scenario requires different characteristics of the Deep M-Embeddings.**

In a first experiment, we want to evaluate how the different combinations of DMEs perform in our swarm scenario. The task was to simply move the objects and accordingly the reward function sums over the distances of the object positions between state $s$ and $s'$. The learning curves of the task for different combinations of DMEs are depicted in Figure 4.5. Most interestingly, using a mean embedding for the object observations prevents the learning of the policy. Choosing a softmax embedding for the objects together with a mean embedding or a max embedding for the swarm leads to worse performance compared to the remaining combinations (including taking a softmax-embedding for both observations) which all yield approximately similar results on this task.

**Deep M-Embeddings outperform standard multi-layer perceptrons.**

In a second experiment, we want to compare the proposed algorithm to learning a simple multi-layer perceptron (MLP) which has been used for example to learn a policy

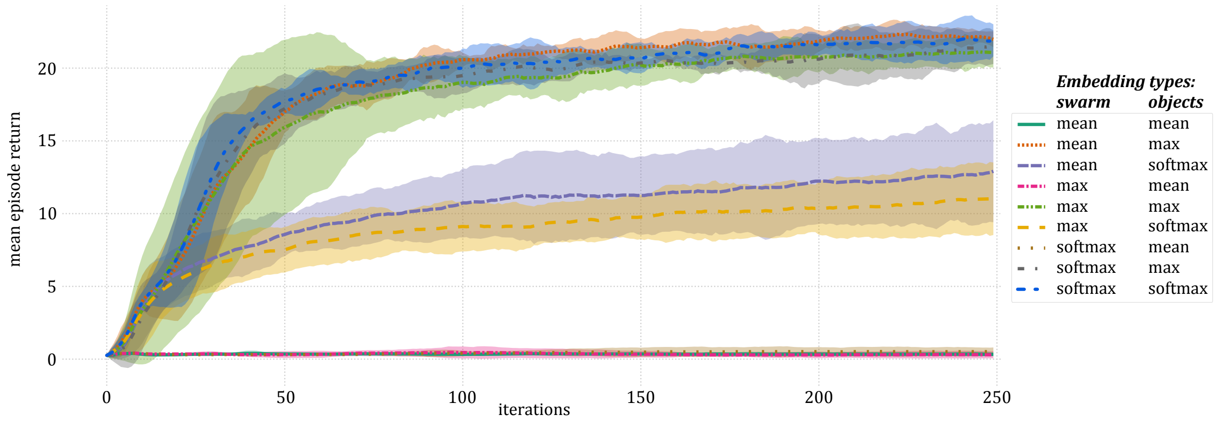


**Figure 4.5:** Learning curves for the 'moving objects' task for all combinations of DMEs for the swarm observations and the object observations. The plot shows the mean ± 2*std of the mean episode return over ten runs of the learning algorithm.

for a humanoid in [89]. In this experiment, the task is to assemble the objects in the environment. The reward function computes the point-wise distance between the objects and returns the sum over the differences between the distances in state $s$ and next state $s'$. We have learned the policies for this task with the combination of DMEs that have showed to yield good results in the previous experiment and with an MLP with 100, 50, and 25 neurons in three layers using tanh activations. Figure 4.8 depicts the learning curves for all policy types. In Figure 4.7, we show an exemplary animation of the task with a learned policy.

### The Deep M-Embeddings can discern multiple object types.

In the third experiment, we wanted to investigate if a learned policy was capable to distinguish between two object types and treat them differently. The task of this experiment is to segregate two types of objects and assemble each group individually. The agents observe the type of the object as a one-hot-encoding with the other object observations. The reward should be positive if the swarm moves objects of the same type closer together and separates them from objects of the other type. To this end, the reward function computes for each object type the point-wise distances of the objects and the mean object position. From there, the reward is computed is then computed based on the differences in the point wise distances and the mean positions from the current to the next state. Learning curves for this experiment can be found in Figure 4.9. An animation of the task can be found in Figure 4.10.

We can see, that the policies are capable to successfully segregate the two object types. However, the agents always prefer to segregate one of the two object types and neglect the other. In general, such a task would require an hierarchical approach in which an upper hierarchy decides which object type should be segregated or a recurrent approach in which the policy is able to make long term decisions. We leave such an approach for future work.

### The $\beta$-values of the soft-max embedding change during learning.

In addition, we have inspected the $\beta$-values of the softmax-embedding in this experiment. Figure 4.11 depicts the $\beta$-values of the value function network and the policy

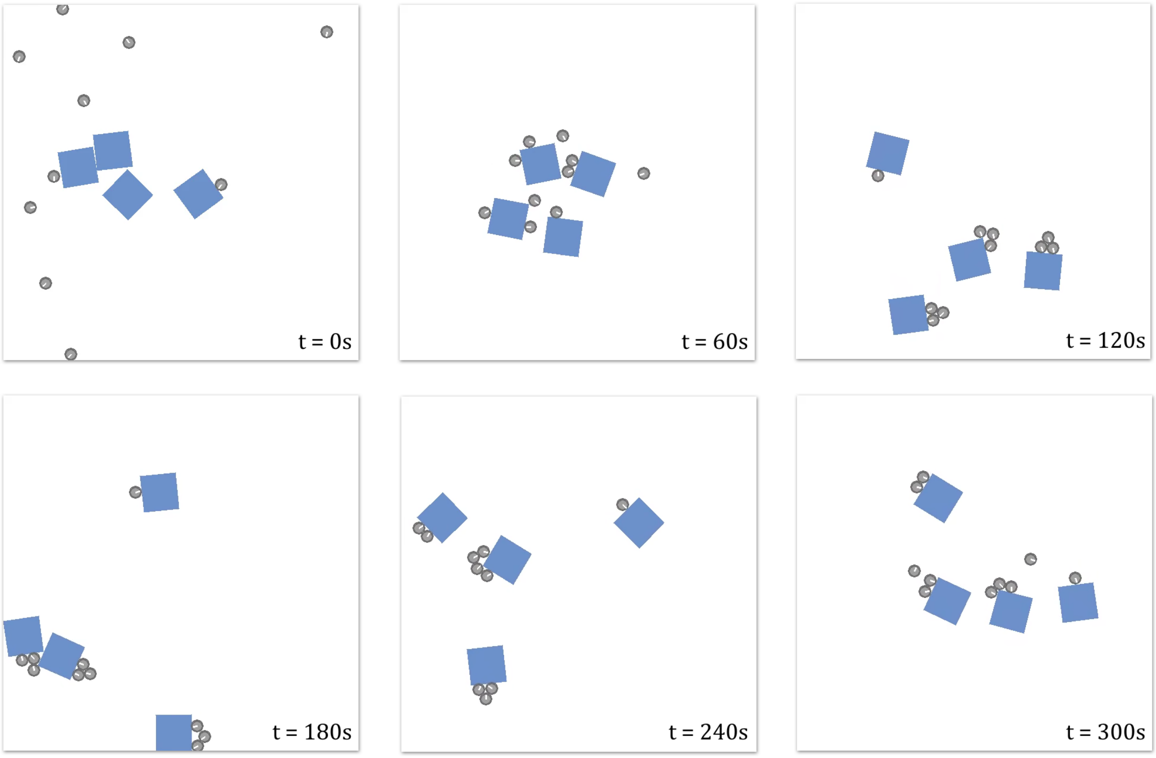


**Figure 4.6:** Exemplary animation of the moving objects task. With the learned policy, the agents collect the objects and move them in circular trajectories across the environment. For this animation, we used a policy network with the mean-embedding for the swarm observations and a max-embedding for the object observations.

network, where we have used a softmax embedding for both, the swarm observations and the object observations. Interestingly, the $\beta$-values did change only slightly in the policy network, while they changed to much larger extend in the value function network. The changes in the policy network are too small to have an effect on the characteristics of the softmax-embedding. In the softmax-embedding for the swarm observations in the value function network, the changes of the $\beta$-values are roughly equally distributed into positive and negative changes. Thus, some of the features have developed a more max-like characteristic (positive changes), while others have developed a more mean-like characteristic (negative changes). In contrast, the changes of the $\beta$-values in the softmax-embedding for the object observations in the value function network are nearly all negative. Hence, all these features were updated towards a more mean-like embedding. The latter finding is a bit contradicting to the results of the first experiment in which mean-embeddings for the object observations did lead to very poor performance. An explanation to this could be that for the value function network it is more important to have a distribution over the object features while for the policy network the exact locations of the objects are crucial.

#### Learned policies are transferable to other swarm sizes and numbers of objects.

Finally, we have investigated the transferability of the learned policies to different swarm sizes and different numbers of objects in the environment. We evaluated this ability with the policies learned on the object assembly task and the object segregation task. In the

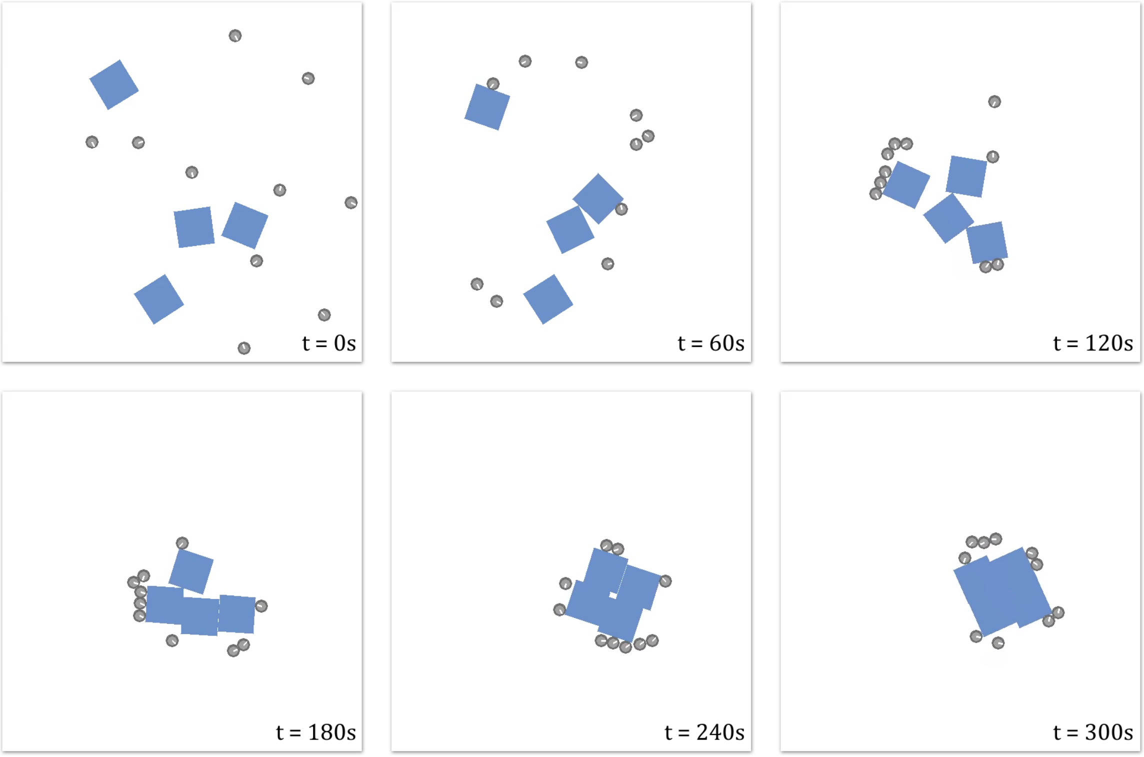


**Figure 4.7:** Exemplary animation of the assembly task. The agents successfully assemble the objects in a rotary movement. For this animation, we used a policy network with the mean-embedding for the swarm and a max-embedding for the object observations.

object assembly task we have used a policy with mean embedding for the swarm observations and with max embedding for the object observations. In the object segregation task with used a softmax embedding for both. Figure 4.12 depicts the average episode return for different settings of the number of agents and objects in both tasks. We can clearly see that the policy generally can be transferred to scenarios with more and less agents in the swarm as well as more and less objects in the environment. While more agents seem to have a positive impact on the outcome, changes in the number of objects tend to have a negative effect on the results. This can be partly explained by the limited space in the environment. Note however, that the reward function and thus the return is not completely independent of the number of objects as we compute the mean over the point-wise object distances. The more objects we have in the environment, the smaller this distance will be already in the beginning and, thus, the smaller the return we can obtain. With too many agents in the environment (i.e., 25 agents), the return tends to decrease because the agents start to obstruct the efforts of other agents by pushing objects from different sides. While this experiment should demonstrate that the learned policies can be transferred to different swarm sizes and different numbers of objects, using a variable number of observations during the learning of the policies could further improve the results. Videos of all experiments can be found at `https://tinyurl.com/dme-videos`.

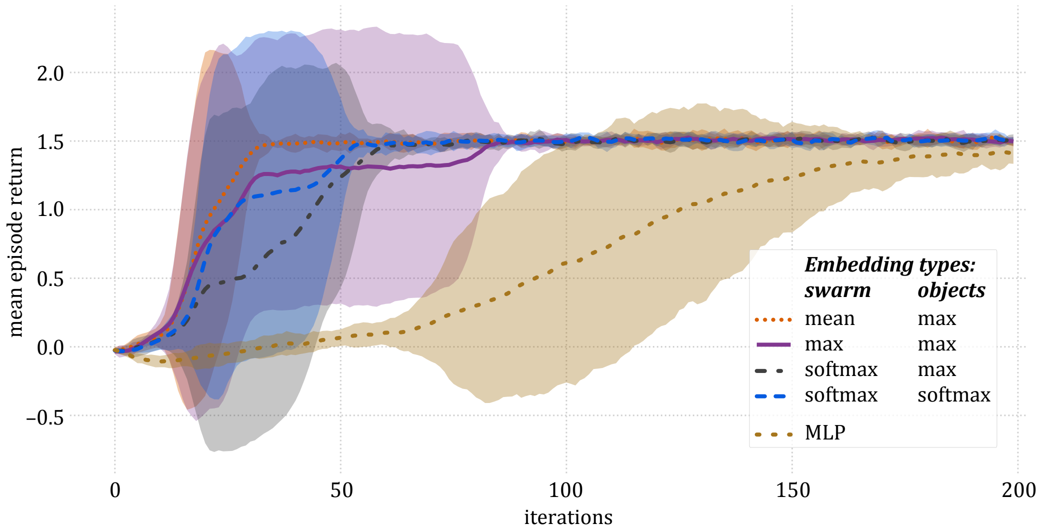


**Figure 4.8:** Learning curves of the object assembly task for different combinations of the DMEs for swarm and object observations in comparison to a standard MLP as policy network. The plot shows the mean ± 2*std of the mean episode return over the seven best of ten runs of the learning algorithm.

## 4.6 Conclusions

In this paper, we present a method to learn policies for manipulating objects with a swarm of homogeneous agents. The policy for the agents is learned from a common reward signal using a centralized critic and distributed actors. Each agent uses the identical policy to compute its actions based on the local observations. To allow for variable number of observations (of other agents but also of objects in the environment) and to reduce the search space of the parameters, we introduce the deep M-embeddings. The deep M-embeddings are inspired by the kernel mean embeddings and provide a network structure that computes a fixed size feature representation from a variable number of inputs. We show in experimental evaluations, that the deep M-embeddings can be employed to learn complex policies for object manipulation with robot swarms in which they outperform classical multi-layer perceptrons in terms of the achieved return but also considering the sample complexity.

The different forms of the deep M-embeddings—mean embedding, max embedding and soft-max embedding—allow to learn embeddings of different characteristics. Our experiments justify, that these characteristics are necessary to learn the policies in the swarm settings. While mean and max embeddings can be used to learn a representation for the swarm state, the object state needs to be represented by a max or a soft-max embedding. While the $\beta$-values in the soft-max embedding are learned with the other parameters of the network, it seems that the learning is rather slow and not sufficient find the best characteristics as the use of soft-max embeddings for both, the swarm and the object observations, does not yield good results. In future work, these issues need to be investigated more thoroughly to understand why certain characteristics are necessary for certain types of observation and to leverage the adaptability of the soft-max embeddings.

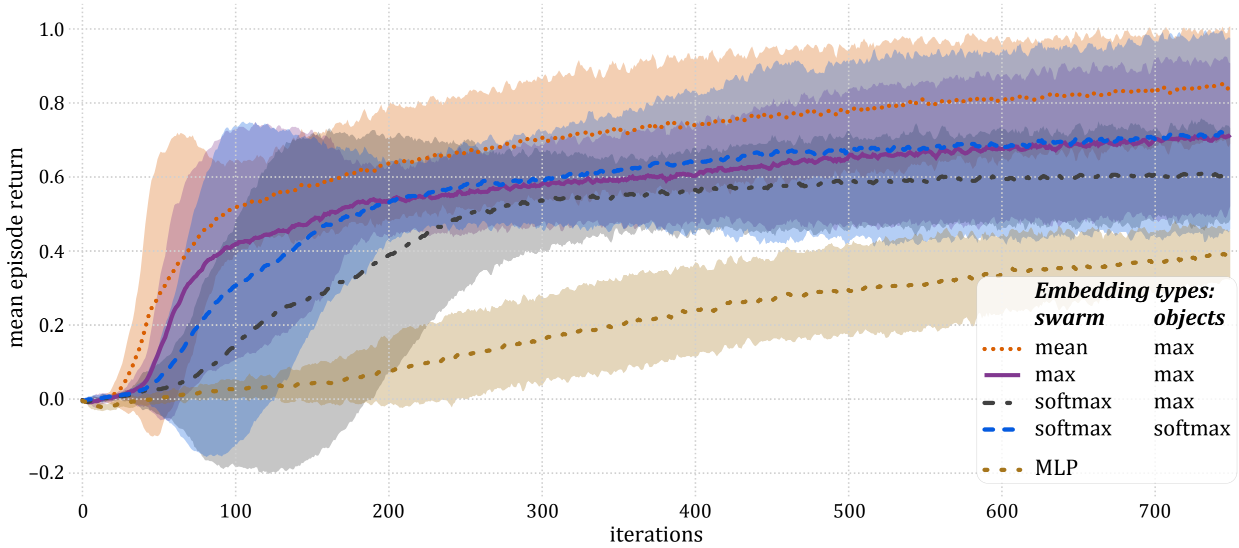


**Figure 4.9:** Learning curves for the segregation task. The colored lines and shaded areas show the mean and two times the standard deviation of the best four out of five trials. The trials that were not included into the statistics are depicted in gray. The plot shows the mean ± 2*std of the mean episode return over the seven best of ten runs of the learning algorithm.

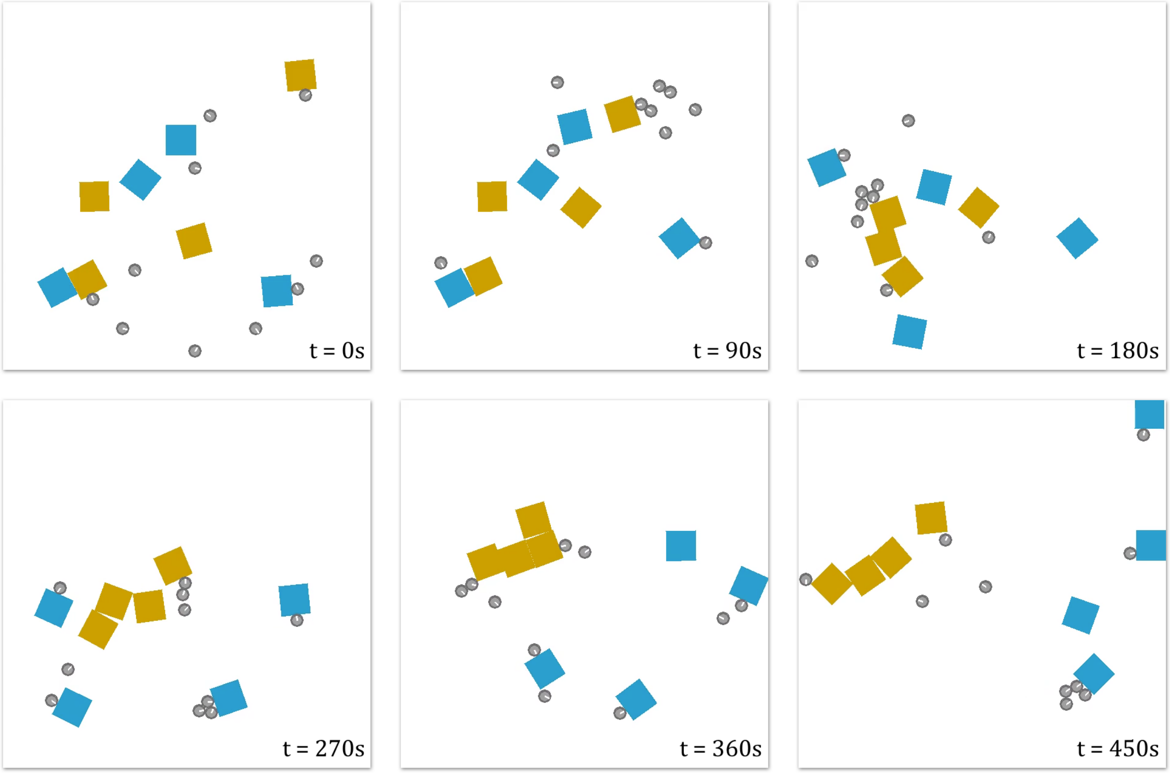


**Figure 4.10:** Exemplary animation of the segregation task. The agents successfully separate the objects into two groups. For this animation, we used a policy network with the softmax-embedding for the swarm observations and a softmax-embedding for the object observations.

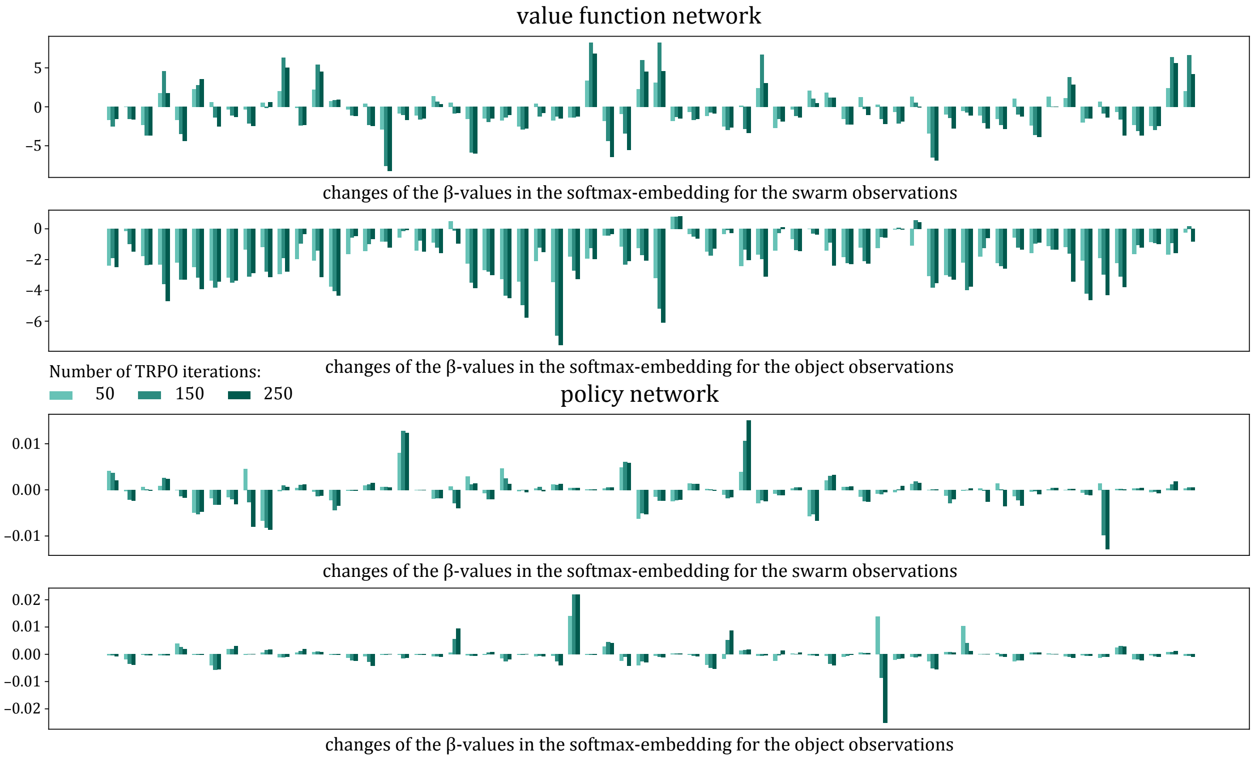


**Figure 4.11:** Changes of the $\beta$-values in the softmax-embeddings of the policy network and the value function network after 50, 150, and 250 learning iterations. While the optimization changes the $\beta$-values in the value function network, the parameters in the policy network do not change.

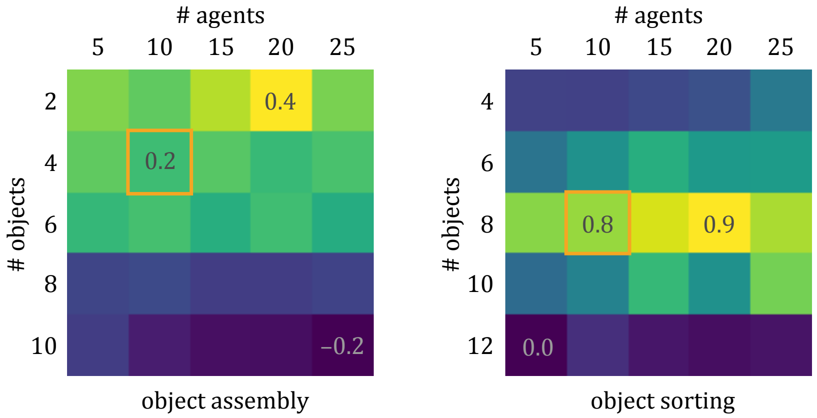


**Figure 4.12:** Evaluation of the transferability to different swarm sizes and different numbers of objects in the environment. Depicted is the mean return over 10 roll outs. The orange squares denote the settings in which the evaluated policy has been learned. Numbers are given for min/max results and for the learning setting.

# 5 Conclusion

The research presented in this thesis alleviates the learning of state representations, models and policies for complex, high-dimensional systems. The proposed methods improve the computational complexity of the model or reduce the search space for the parameters and thus enable a more efficient learning. The increased efficiency of the learning process allows to apply optimization to learn the parameters of the models instead of cumbersome hand tuning which would require expert knowledge. In the following paragraphs, I want to recapitulate the individual contributions before I will give an outlook on possible future work.

In Chapter 2, I have presented the kernel Kalman rule as an addition to the framework for nonparametric inference. This framework is based on the embeddings of distributions into reproducing kernel Hilbert spaces (RKHS) and provides inference rules to manipulate the embedded distributions entirely in the Hilbert space. The kernel Kalman rule follows from the clear optimization objective of recursive least squares and is inspired by the Bayesian update of the classical Kalman filter. In addition, I have introduced the subspace conditional embedding operator which allows to represent embeddings in a subspace spanned by a small representative sample set while using the full data set for learning the embedding operators of the conditional distribution. Both presented contributions improve the computational complexity and by that allow to optimize the hyper parameters of the models using black box optimization. With my experiments, I have demonstrated that these contributions improve the computational efficiency and the numerical stability in comparison to the kernel Bayes' rule.

Furthermore, I have presented the kernel Kalman filter for state estimation and state prediction. The kernel Kalman filter uses RKHS embeddings to represent the belief state and updates these embeddings with the kernel Kalman rule and the kernel sum rule entirely in the Hilbert space. The forward and observation models for these updates can be learned from data without requiring any domain knowledge. My experiments have shown that the kernel Kalman filter is able to filter the state of complex systems such as human motion data. I have also introduced the kernel forward-backward smoother (KFBS), a post-processing routine for estimating the state of a system from past and future observations. The KFBS uses a forward and a backward kernel Kalman filter and combines the estimates of both filters with a smoothing update in Hilbert space. The experiments demonstrate how the KFBS improves the accuracy of the state estimates over purely filtered estimates.

In Chapter 3, I have investigated the use of kernel mean embeddings to represent the complex state of a swarm. To this end, I have introduced the swarm kernel which represents the state of a swarm by its generative distribution embedded into an RKHS. Hereby, each agent is considered a sample of that distribution. The swarm kernel provides a similarity measure between two swarm configurations by computing the squared difference between the embeddings of the distributions. I have applied the swarm kernel to learn a policy for controlling the swarm via a common external input signal. Since this representation is invariant to the number and the permutation of the agents in the swarm, the swarm kernel furthermore reduces the search space for the policy search method. Additionally, the swarm kernel allows to apply the learned policies to swarms of different sizes without further modifications or learning. The experimental evaluations have shown that the policies learned with the proposed swarm kernel can be used for object assembly tasks with complex object shapes. I could furthermore directly transfer the learned policies directly to a robotic platform.

In Chapter 4, I have investigated how the idea of kernel mean embeddings can be transferred to neural networks. Neural networks usually have a predefined structure which requires that the number of inputs and outputs is known in advance. The deep M-embeddings provide a structure that alleviates this problem by computing a feature mapping with a fixed size for a set of homogeneous inputs. I have used these embeddings to propose a network structure for learning a policy for the agents of a swarm in object manipulation tasks. For the observations of the agents and for the observations of the objects a feature representation is learned via a deep M-embedding. In my experiments, I compare how the different variants of the deep M-embeddings affect the learning of the policies. Furthermore, I demonstrate that complex control strategies can be learned with the proposed network structure where a competitive structure from the recent literature fails.

To summarize, the core contributions of my thesis are twofold: the additions to the framework for non-parametric inference, namely the kernel Kalman rule and the subspace conditional operator, enable computational more efficient and numerically more stable inference with kernel embeddings of distributions; The swarm kernel and the deep M-embeddings leverage the idea of kernel mean embeddings to provide a compact representation of a variable set of homogeneous observations which enables the learning of policies for a swarm of agents.

## 5.1 Future Work

With the kernel Kalman rule and the subspace conditional operator, I have presented two methods to increase the computational efficiency of non-parametric inference. While this has largely improved the applicability of non-parametric inference methods to real-world problems, a bottleneck is still the optimization of the hyper-parameters. In the experiments I have conducted for this thesis, I have optimized the hyper-parameters in most experiments via black-box optimization methods and for some experiments with heuristics and hand-tuning. Black-box optimizers, such as CMA-ES which has been used in this thesis, take samples from a distribution over the parameters to find a gradient. This gradient is the used to update the parameter distribution. However, this requires to evaluate the model with many parameter hypotheses. If we would have access to the true gradient of the model, we could drastically reduce this number of evaluations. Thus, adding gradient-based optimization methods to the framework for non-parametric inference, similar to using gradients in sparse Gaussian processes [101, 91] for optimizing the hyper-parameters but also the inducing inputs, would be an interesting direction of future research.

The subspace approximations for the non-parametric inference methods provide an approach for scaling kernel methods to large training sets. However, kernel methods still do not scale well with the complexity of the system. While they can easily deal with high-dimensional inputs, kernel methods are not able to learn models in high-dimensional manifolds. An approach to transfer kernel methods to even more complex scenarios could be to learn local models similar to the approach presented in [114]. In this thesis, I have neglected actuated systems which have a control variable as input. Future research could extend the formulations of the kernel Kalman filter to controlled systems which take the control as additional input to the transition model. Such local models for actuated systems could then be used in combination with the kernel forward-backward smoother to learn planners for complex robotics tasks.

The swarm kernel provides a similarity measure for sets of homogeneous inputs. In my experiments, I have shown that this kernel can be used to learn policies for an external, global input signal. These policies have been learned in an object-relative frame and the input to these policies was given by a high-level assembly policy. So far, this assembly policy has been defined by hand as a set of oriented way points. To achieve a fully autonomous system, hierarchical learning approaches [111, 14] could be applied to this task in future research. I have furthermore shown that the learned policies can be transferred to a real robotic system without any further modifications. Still, the robotic platform I have used in my experiments (i.e., the Kilobots) is just a testbed for the evaluation of swarm algorithms. The Kilobots can not be used for any productive application. Thus, applying the proposed algorithm on a system that aims at solving real world problems would be a challenging task of future research. An example for such a system could be a nano-robotics platform for the transportation of medication in blood vessels [62].

With the deep M-embeddings, I have demonstrated that the idea of the kernel mean embeddings can be transferred to neural networks. The deep softmax-embeddings with the temperature variable $\beta$ provide the structure for learning the characteristics of the embedding as mean-, max-, or min-embedding per feature dimension. Which of these characteristics is beneficial in which cases is however still unclear. Future research which would focus on a thorough evaluation of these characteristics and investigate their impact of the information in the feature vector could lead to valuable insights. Furthermore, as I have only done experimental evaluations in simulation, an evaluation on a robotic platform could further support the proposed algorithm. However, current platforms for swarm robotics do not have the sensory capabilities we assume in the model of our agents. Hence, an approach that shares information via communication of the agents and assumes partial observability [43] would be necessary. Alternatively, evaluating the deep M-embeddings in a setting with global input signals—similar to the problem addressed in Chapter 3—would avoid the necessity of swarm agents with global observations. With such global input signals, the approach could be furthermore evaluated on a nano-robotics platform.

# List of Figures

1.1 Binomial distributions . . . . . 4
1.2 Multinomial distributions . . . . . 4
1.3 Normal distributions . . . . . 5
1.4 Gaussian mixture model . . . . . 6
1.5 Samples of a Gaussian mixture model . . . . . 7
1.6 Histogram representation . . . . . 7
1.7 Kernel density estimation . . . . . 8

2.1 Comparison of subspace KBF to KBF . . . . . 29
2.3 Comparison of KKR and KBR on conditioning task . . . . . 36
2.2 Graphical model of conditioning task . . . . . 36
2.4 Comparison of the performances of the KKR, subKKR, KBR(b), and subKBR . 37
2.5 Graphical model of pendulum task . . . . . 42
2.6 Comparison of KKF, KBF, KKF-CEO, EKF, and UKF on pendulum task . . . . . 44
2.8 Exemplary trajectories of the quadlink task . . . . . 44
2.7 Graphical model of quadlink task . . . . . 44
2.9 Prediction performance in the quadlink task . . . . . 45
2.10 Exemplary postures and observed muscle activities in the HuMoD task . . . . 46
2.11 Impact of kernel size on performance for subKKF and subKBF . . . . . 47
2.12 Performance over optimization iterations for subKKF . . . . . 47
2.13 Qualitative comparison of forward and backward pass to estimates of KFBS . 52
2.14 Evaluation of kernel size for KFBS, subKFBS and KKF on pendulum task . . . 53
2.15 Comparison of subKFBS and subKKF on table tennis ball trajectory . . . . . 54
2.16 Performance of subKFBS and KFBS on table tennis task . . . . . 54

3.1 Kilobot swarm with object . . . . . 60
3.2 Schematic of approach to solve the object assembly task . . . . . 67
3.3 Learning curves for object shapes . . . . . 74
3.4 Straight path evaluation of pushing policy . . . . . 74
3.5 Circular path evaluation of the pushing policy . . . . . 75
3.6 Comparison of learned policy against pose controller . . . . . 76
3.7 Evaluation of learned policy on different swarm sizes . . . . . 77
3.8 Assembly of two triangular shapes . . . . . 77
3.9 Assembly of two L-shapes . . . . . 78
3.10 Assembly task of a C-shape with a T-shape . . . . . 78
3.11 Kilobot setup, hardware modifications, and sensor response curves . . . . . 80
3.12 Assembly task in simulation . . . . . 80
3.13 Assembly task on robotic platform . . . . . 82

4.1 Structure of the soft-max embedding network . . . . . . . . . . . . . . . . . . . . 90
4.2 A Kilobot. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
4.3 Perception model of the swarm agents . . . . . . . . . . . . . . . . . . . . . . . . 92
4.4 Structure of the policy and value function network . . . . . . . . . . . . . . . . . 93
4.5 Learning curves for the ‘moving objects’ task . . . . . . . . . . . . . . . . . . . . 96
4.6 Exemplary animation of the moving objects task . . . . . . . . . . . . . . . . . . . 97
4.7 Exemplary animation of the assembly task . . . . . . . . . . . . . . . . . . . . . . 98
4.8 Learning curves of the object assembly task . . . . . . . . . . . . . . . . . . . . . 99
4.9 Learning curves for the segregation task . . . . . . . . . . . . . . . . . . . . . . 100
4.10 Exemplary animation of the segregation task . . . . . . . . . . . . . . . . . . . 100
4.11 Changes of the $\beta$-values in the softmax-embeddings during learning . . . . 101
4.12 Evaluation of the learned policy on different swarm sizes . . . . . . . . . . . 101

# List of Tables

1.1 Challenges addressed in this thesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.1 Time consumption of KKR and KBR in conditioning task . . . . . . . . . . . . . . . 36
2.2 Comparison of subKKF, subKBF, and SGP in HuMoD task . . . . . . . . . . . . . 47

3.1 Leaned policies, trajectories, and value function for $\rho = 0.0$ . . . . . . . . . . . 71
3.2 Leaned policies, trajectories, and value function for $\rho = 0.5$ . . . . . . . . . . . 72
3.3 Leaned policies, trajectories, and value function for $\rho = 1.0$ . . . . . . . . . . . 73

# List of Algorithms

1 The Kernel Kalman Filter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
2 The Subspace Kernel Kalman Filter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
3 The Kernel Forward-Backward Smoother . . . . . . . . . . . . . . . . . . . . . . . . . 56
4 The Subspace Kernel Forward-Backward Smoother . . . . . . . . . . . . . . . . . . . . 57

# Publications

[22] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, and G. Neumann. *Learning Policies for Object Manipulation with Robot Swarms*. In: *Advanced Robotics* (submitted).

[23] G. H. W. Gebhardt, M. Hüttenrauch, and G. Neumann. *Using M-Embeddings to Learn Control Strategies for Robot Swarms*. In: *Swarm Intelligence* (submitted).

[24] G. H. W. Gebhardt, A. Kupcsik, and G. Neumann. *The Kernel Kalman Rule*. In: *Machine Learning Journal* (submitted).

[25] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, A. Hendrich, D. Kauth, and G. Neumann. *Learning to Assemble Objects with a Robot Swarm (Extended Abstract)*. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. 2017.

[26] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, A. Hendrich, D. Kauth, and G. Neumann. *Learning to Assemble Objects with Robot Swarms (ARMS 2017)*. Sao Paulo, 2017.

[27] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, and G. Neumann. *Learning Robust Policies for Object Manipulation with Robot Swarms*. In: *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*. Brisbane, 2018.

[28] G. H. W. Gebhardt, A. Kupcsik, and G. Neumann. *Learning Subspace Conditional Embedding Operators*. Lille, 2015.

[29] G. H. W. Gebhardt, A. Kupcsik, and G. Neumann. *The Kernel Kalman Rule - Efficient Nonparametric Inference with Recursive Least Squares*. In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. 2017.

# Bibliography

[1] D. Achlioptas and F. McSherry. *On Spectral Learning of Mixtures of Distributions*. In: *Learning Theory*. Springer, Berlin, Heidelberg, 2005, pp. 458–469.

[2] A. Agresti. *Deriving Standard Errors with the Delta Method*. In: *Categorical data analysis*. Wiley series in probability and statistics. New York: Wiley-Interscience, 2002, 73ff.

[3] N. Aronszajn. *Theory of reproducing kernels*. In: *Transactions of the American Mathematical Society* 68.3 (1950), pp. 337–404.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. *A Brief Survey of Deep Reinforcement Learning*. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[5] A. Becker, G. Habibi, J. Werfel, M. Rubenstein, and J. McLurkin. *Massive uniform manipulation: Controlling large populations of simple robots with a common input signal*. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013.

[6] E. Bonabeau, M. Dorigo, and G. Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. OUP USA, 1999.

[7] Q. Bonnard, S. Lemaignan, G. Zufferey, A. Mazzei, S. Cuendet, N. Li, A. Özgür, and P. Dillenbourg. *Chilitags 2: Robust Fiducial Markers for Augmented Reality and Robotics.* 2013.

[8] B. Boots, A. Gretton, and G. J. Gordon. *Hilbert Space Embeddings of Predictive State Representations*. In: *Proceedings of the 29th International Conference on Uncertainty in Artificial Intelligence (UAI)*. 2013.

[9] J. A. Boyan. *Least-squares temporal difference learning*. In: *Proceedings of the 16th International Conference on Machine Learning*. 1999.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. *OpenAI Gym*. In: *arXiv:1606.01540 [cs]* (2016).

[11] E. Catto. *Box2D is a 2D physics engine for games*. 2018.

[12] Y. Chen, M. Welling, and A. Smola. *Super-samples from Kernel Herding*. In: *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press, 2010.

[13] L. Csató and M. Opper. *Sparse on-line gaussian processes.* In: *Neural computation* 14.3 (2002), pp. 641–668.

[14] C. Daniel, H. van Hoof, J. Peters, and G. Neumann. *Probabilistic inference for determining options in reinforcement learning*. In: *Machine Learning* 104.2-3 (2016), pp. 337–357.

[15] G. G. Denton. *The Use Made of the Speedometer as an Aid to Driving*. In: *Ergonomics* 12.3 (1969), pp. 447–452.

[16] P. Drineas and M. W. Mahoney. *On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning*. In: *Journal of Machine Learning Research* 6 (2005), p. 23.

[17] C. Finn, P. Abbeel, and S. Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. In: *arXiv:1703.03400 [cs]* (2017).

[18] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. *Counterfactual Multi-Agent Policy Gradients*. In: *arXiv:1705.08926 [cs]* (2017).

[19] K. Fukumizu, L. Song, and A. Gretton. *Kernel Bayes' Rule*. In: *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc., 2011, pp. 1737–1745.

[20] K. Fukumizu, L. Song, and A. Gretton. *Kernel Bayes' Rule: Bayesian Inference with Positive Definite Kernels*. In: *Journal of Machine Learning Research* 14.1 (2013), pp. 3683–3719.

[21] C. F. Gauss. *Theoria combinationis observationum erroribus minimis obnoxiae*. H. Dieterich, 1823.

[22] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, and G. Neumann. *Learning Policies for Object Manipulation with Robot Swarms*. In: *Advanced Robotics* (submitted).

[23] G. H. W. Gebhardt, M. Hüttenrauch, and G. Neumann. *Using M-Embeddings to Learn Control Strategies for Robot Swarms*. In: *Swarm Intelligence* (submitted).

[24] G. H. W. Gebhardt, A. Kupcsik, and G. Neumann. *The Kernel Kalman Rule*. In: *Machine Learning Journal* (submitted).

[25] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, A. Hendrich, D. Kauth, and G. Neumann. *Learning to Assemble Objects with a Robot Swarm (Extended Abstract)*. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. 2017.

[26] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, A. Hendrich, D. Kauth, and G. Neumann. *Learning to Assemble Objects with Robot Swarms (ARMS 2017)*. Sao Paulo, 2017.

[27] G. H. W. Gebhardt, K. Daun, M. Schnaubelt, and G. Neumann. *Learning Robust Policies for Object Manipulation with Robot Swarms*. In: *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*. Brisbane, 2018.

[28] G. H. W. Gebhardt, A. Kupcsik, and G. Neumann. *Learning Subspace Conditional Embedding Operators*. Lille, 2015.

[29] G. H. W. Gebhardt, A. Kupcsik, and G. Neumann. *The Kernel Kalman Rule - Efficient Nonparametric Inference with Recursive Least Squares*. In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. 2017.

[30] S. Gomez-Gonzalez, G. Neumann, B. Schölkopf, and J. Peters. *Using probabilistic movement primitives for striking movements*. In: *IEEE-RAS International Conference on Humanoid Robots* (2016), pp. 502–508.

[31] L. S. Gottfredson. *Mainstream science on intelligence: An editorial with 52 signatories, history, and bibliography*. In: *Intelligence* 24.1 (1997), pp. 13–23.

[32] A. Grover, M. Al-Shedivat, J. K. Gupta, Y. Burda, and H. Edwards. *Learning Policy Representations in Multiagent Systems*. In: *arXiv:1806.06464 [cs, stat]* (2018).

[33] S. Grünewälder, G. Lever, L. Baldassarre, S. Patterson, A. Gretton, and M. Pontil. *Conditional mean embeddings as regressors*. In: *Proceedings of the 29th International Conference on Machine Learning*. 2012.

[34] J. K. Gupta, M. Egorov, and M. Kochenderfer. *Cooperative Multi-agent Control Using Deep Reinforcement Learning*. In: *Autonomous Agents and Multiagent Systems*. Lecture Notes in Computer Science. Springer International Publishing, 2017, pp. 66–83.

[35] N. Hansen. *The CMA Evolution Strategy: A Comparing Review*. In: *Towards a New Evolutionary Computation. Advances on estimation of distribution algorithms*. Berlin/Heidelberg: Springer, 2006, pp. 75–102.

[36] P. E. Hart, N. J. Nilsson, and B. Raphael. *A formal basis for the heuristic determination of minimum cost paths*. In: *IEEE Transactions on Systems Science and Cybernetics* (1968).

[37] H. v. Hasselt, A. Guez, and D. Silver. *Deep Reinforcement Learning with Double Q-Learning*. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[38] M. Hausknecht and P. Stone. *Deep Recurrent Q-Learning for Partially Observable MDPs*. In: *2015 AAAI Fall Symposium Series*. 2015.

[39] S. Hochreiter and J. Schmidhuber. *Long Short-Term Memory*. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[40] M. W. Hoffman, A. Lazaric, M. Ghavamzadeh, and R. Munos. *Regularized least squares temporal difference learning with nested l2 and l1 penalization*. In: *European Workshop on Reinforcement Learning*. 2011.

[41] H. van Hoof, G. Neumann, and J. Peters. *Non-parametric Policy Search with Limited Information Loss*. In: (2017).

[42] D. Hsu, S. M. Kakade, and T. Zhang. *A Spectral Algorithm for Learning Hidden Markov Models*. In: *Journal of Computer and System Sciences* 78.5 (2012), pp. 1460–1480.

[43] M. Hüttenrauch, A. Šošić, and G. Neumann. *Local Communication Protocols for Learning Complex Swarm Behaviors with Deep Reinforcement Learning*. In: *arXiv:1709.07224 [cs, stat]* (2017).

[44] M. Hüttenrauch, A. Šošić, and G. Neumann. *Deep Reinforcement Learning for Swarm Systems*. In: *Journal of Machine Learning Research* (2019).

[45] T. Jaakkola, M. Diekhans, and D. Haussler. *Using the Fisher kernel method to detect remote protein homologies.* In: *Proceedings of the International Conference on Intelligent Systems for Molecular Biology; ISMB.* (1999).

[46] H. Jaeger. *Observable Operator Models for Discrete Stochastic Time Series*. In: *Neural Computation* 12.6 (June 2000), pp. 1371–1398.

[47] S. J. Julier and J. K. Uhlmann. *A New Extension of the Kalman Filter to Nonlinear Systems*. In: *Int. symp. aerospace/defense sensing, simul. and controls*. 1997.

[48] S. Kakade. *A Natural Policy Gradient*. In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Cambridge, MA, USA: MIT Press, 2001, pp. 1531–1538.

[49] R. E. Kalman. *A new approach to linear filtering and prediction problems*. In: *Journal of Fluids Engineering* 82.1 (1960), pp. 35–45.

[50] Y. Kawahara, T. Yairi, and K. Machida. *A Kernel Subspace Method by Stochastic Realization for Learning Nonlinear Dynamical Systems*. In: *Advances in Neural Information Processing Systems* 19 (2007), pp. 665–672.

[51] T. Kawakami, M. Kinoshita, M. Watanabe, N. Takatori, and M. Furukawa. *An actor-critic approach for learning cooperative behaviors of multiagent seesaw balancing problems*. In: *IEEE International Conference on Systems, Man and Cybernetics*. 2005.

[52] O. Khatib. *Real-time obstacle avoidance for manipulators and mobile robots*. In: *The International Journal of Robotics Research* (1986).

[53] Y. Koren and J. Borenstein. *Potential field methods and their inherent limitations for mobile robot navigation*. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 1991.

[54] A. Kupcsik, M. P. Deisenroth, J. Peters, L. Ai Poh, P. Vadakkepat, and G. Neumann. *Model-based Contextual Policy Search for Data-Efficient Generalization of Robot Skills*. In: *Artificial Intelligence* (2015).

[55] T. Kuremoto, M. Obayashi, K. Kobayashi, H. Adachi, and K. Yoneda. *A reinforcement learning system for swarm behaviors*. In: *IEEE International Joint Conference on Neural Networks*. 2008.

[56] M. G. Lagoudakis and R. Parr. *Least-squares policy iteration*. In: *Journal of Machine Learning Research* (2003).

[57] M. Lauer and M. Riedmiller. *An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems*. In: *In Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann, 2000, pp. 535–542.

[58] S. Levine, C. Finn, T. Darrell, and P. Abbeel. *End-to-end Training of Deep Visuomotor Policies*. In: *J. Mach. Learn. Res.* 17.1 (2016), pp. 1334–1373.

[59] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. *Continuous control with deep reinforcement learning*. In: *arXiv:1509.02971 [cs, stat]* (2015).

[60] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 6379–6390.

[61] S. Martel and M. Mohammadi. *Using a swarm of self-propelled natural microrobots in the form of flagellated bacteria to perform complex micro-assembly tasks*. In: (2010).

[62] S. Martel, M. Mohammadi, O. Felfoul, Z. Lu, and P. Pouponneau. *Flagellated Magnetotactic Bacteria as Controlled MRI-trackable Propulsion and Steering Systems for Medical Nanorobots Operating in the Human Microvasculature*. In: *The International Journal of Robotics Research* 28.4 (2009), pp. 571–582.

[63] L. Matignon, G. J. Laurent, and N. Le Fort-Piat. *Hysteretic Q-Learning : an algorithm for decentralized reinforcement learning in cooperative multi-agent teams.* In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'07.* Vol. sur CD ROM. San Diego, CA., United States, 2007, pp. 64–69.

[64] C. Mavroidis, A. Dubey, and M. Yarmush. *Molecular Machines*. In: *Annual Review of Biomedical Engineering* 6.1 (2004), pp. 363–395.

[65] L. Mccalman, S. O 'callaghan, and F. Ramos. *Multi-Modal Estimation with Kernel Embeddings for Learning Motion Models*. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. Karlsruhe: IEEE, 2013, pp. 2845–2852.

[66] B. A. McElhoe. *An Assessment of the Navigation and Course Corrections for a Manned Flyby of Mars or Venus*. In: *IEEE Transactions on Aerospace and Electronic Systems* AES-2 (1966).

[67] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. In: *arXiv:1312.5602 [cs]* (2013), p. 9.

[68] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. *Human-level control through deep reinforcement learning*. In: *Nature* 518.7540 (2015), pp. 529–533.

[69] A. Y. Ng, D. Harada, and S. J. Russell. *Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping*. In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287.

[70] Y. Nishiyama, A. Afsharinejad, S. Naruse, B. Boots, and L. Song. *The Nonparametric Kernel Bayes Smoother*. In: *International Conference on Artificial Intelligence and Statistics*. Vol. 41. 2016, pp. 547–555.

[71] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. *Deep Decentralized Multitask Multi-Agent Reinforcement Learning under Partial Observability*. In: *International Conference on Machine Learning*. 2017, pp. 2681–2690.

[72] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani. *Lenient Multi-Agent Deep Reinforcement Learning*. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 443–451.

[73] L. Panait, K. Sullivan, and S. Luke. *Lenient Learners in Cooperative Multiagent Systems*. In: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '06. New York, NY, USA: ACM, 2006, pp. 801–803.

[74] L. E. Parker. *Multiple Mobile Robot Systems*. In: *Springer Handbook of Robotics*. Ed. by B. Siciliano and O. Khatib. Springer Berlin Heidelberg, 2008, pp. 921–941.

[75] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang. *Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games*. In: *arXiv:1703.10069 [cs]* (2017).

[76] J. Peters, K. Mülling, and Y. Altun. *Relative Entropy Policy Search*. In: *Proceedings of the 24th AAAI Conference on Artificial Intelligence*. 2010.

[77] J. Peters, S. Vijayakumar, and S. Schaal. *Natural Actor-Critic*. In: *Machine Learning: ECML 2005*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 280–291.

[78] J. Pugh and A. Martinoli. *Parallel learning in heterogeneous multi-robot swarms*. In: *2007 IEEE Congress on Evolutionary Computation*. 2007, pp. 3839–3846.

[79] J. Pugh and A. Martinoli. *Multi-robot Learning with Particle Swarm Optimization*. In: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS ’06. New York, NY, USA: ACM, 2006, pp. 441–448.

[80] A. Rahimi and B. Recht. *Random features for large-scale kernel machines*. In: *Advances in Neural Information Processing Systems*. 2007, pp. 1–8.

[81] L. Ralaivola and F. d’Alche-Buc. *Time series filtering, smoothing and learning using the kernel kalman filter*. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* 3 (2005), pp. 1449–1454.

[82] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson. *QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning*. In: *arXiv:1803.11485 [cs, stat]* (2018).

[83] M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera, and R. Nagpal. *Kilobot: A low cost robot with scalable operations designed for collective behaviors*. In: *Robotics and Autonomous Systems* 62.7 (2014), pp. 966–975.

[84] M. Rubenstein, C. Ahler, and R. Nagpal. *Kilobot: A Low Cost Scalable Robot System for Collective Behaviors*. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2012.

[85] M. Rubenstein, A. Cabrera, J. Werfel, G. Habibi, J. McLurkin, and R. Nagpal. *Collective Transport of Complex Objects by Simple Robots: Theory and Experiments*. In: *Proceedings of the International Conferenece on Autonomous Agents and Multi-Agent Systems*. 2013.

[86] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. *Policy Distillation*. In: *arXiv:1511.06295 [cs]* (2015).

[87] B. Schölkopf, R. Herbrich, and A. J. Smola. *A Generalized Representer Theorem*. In: *Computational Learning Theory*. 2001.

[88] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. *Trust Region Policy Optimization*. In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[89] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015.

[90] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. *Proximal Policy Optimization Algorithms*. In: *arXiv:1707.06347 [cs]* (2017), p. 12.

[91] M. Seeger, C. K. Williams, and N. D. Lawrence. *Fast forward selection to speed up sparse Gaussian process regression*. In: *Workshop on AI and Statistics 9* 9 (2003).

[92] S. Shahrokhi and A. T. Becker. *Object manipulation and position control using a swarm with global inputs*. In: (2016).

[93] M. Shannon, H. Zen, and W. Byrne. *Autoregressive models for statistical parametric speech synthesis*. In: *IEEE Transactions on Audio, Speech and Language Processing* 21.3 (2013), pp. 587–597.

[94] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel. *Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments*. In: *International Conference on Learning Representations*. 2018.

[95] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. *Mastering the game of Go with deep neural networks and tree search*. In: *Nature* 529.7587 (2016), pp. 484–489.

[96] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. *Deterministic Policy Gradient Algorithms*. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, pp. 387–395.

[97] D. Simon. *Optimal State Estimation*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2006.

[98] G. L. Smith, S. F. Schmidt, and L. A. McGee. *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*. National Aeronautics and Space Administration, 1962.

[99] A. J. Smola and P. P. Bartlett. *Sparse Greedy Gaussian Process Regression*. In: *Advances in Neural Information Processing Systems 13* 13 (2001), pp. 619–625.

[100] A. J. Smola, A. Gretton, L. Song, and B. Schölkopf. *A Hilbert Space Embedding for Distributions*. In: *Proceedings of the 18th international conference on Algorithmic Learning Theory*. Vol. 4754. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 13–31.

[101] E. Snelson and Z. Ghahramani. *Sparse Gaussian Processes using Pseudo-inputs*. In: *Advances in Neural Information Processing Systems 18*. MIT Press, 2006, pp. 1257–1264.

[102] L. Song, B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola. *Hilbert Space Embeddings of Hidden Markov Models*. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 991–998.

[103] L. Song, K. Fukumizu, and A. Gretton. *Kernel Embeddings of Conditional Distributions: A Unified Kernel Framework for Nonparametric Inference in Graphical Models*. In: *IEEE Signal Processing Magazine* 30.4 (July 2013), pp. 98–111.

[104] L. Song, J. Huang, A. Smola, and K. Fukumizu. *Hilbert Space Embeddings of Conditional Distributions with Applications to Dynamical Systems*. In: *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*. New York, New York, USA: ACM Press, 2009, pp. 1–8.

[105] H. W. Sorenson. *Least-squares estimation: from Gauss to Kalman*. In: *Spectrum, IEEE* 7.7 (1970), pp. 63–68.

[106] W. Sun, R. Capobianco, G. J. Gordon, J. A. Bagnell, and B. Boots. *Learning to Smooth with Bidirectional Predictive State Inference Machines*. In: *The Conference on Uncertainty in Artificial Intelligence (UAI 2016)*. 2016.

[107] W. Sun, A. Venkatraman, B. Boots, and J. A. Bagnell. *Learning to Filter with Predictive State Inference Machines*. In: *International Conference on Machine Learning*. 2016, pp. 1197–1205.

[108] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. *Value-Decomposition Networks For Cooperative Multi-Agent Learning*. In: *arXiv:1706.05296 [cs]* (2017).

[109] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, MA: The MIT Press, 2018.

[110] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. *Policy Gradient Methods for Reinforcement Learning with Function Approximation*. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, pp. 1057–1063.

[111] R. S. Sutton, D. Precup, and S. Singh. *Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning*. In: *Artificial Intelligence* 112.1 (1999), pp. 181–211.

[112] T. J. Triggs and J. S. Berenyi. *Estimation of Automobile Speed under Day and Night Conditions*. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 24.1 (1982), pp. 111–114.

[113] E. A. E. Wan and R. Van Der Merwe. *The unscented Kalman filter for nonlinear estimation*. In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*. IEEE, 2000, pp. 153–158.

[114] M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller. *Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images*. In: *arXiv* (2015), pp. 1–18.

[115] C. K. I. Williams and M. Seeger. *Using the Nyström Method to Speed Up Kernel Machines*. In: *Proceedings of the 13th International Conference on Neural Information*. 2000, pp. 661–667.

[116] C. Wirth, J. Fürnkranz, and G. Neumann. *Model-Free Preference-Based Reinforcement Learning*. In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. 2016.

[117] J. Wojtusch and O. von Stryk. *HuMoD - A versatile and open database for the investigation, modeling and simulation of human motion dynamics on actuation level*. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015.

[118] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang. *Mean Field Multi-Agent Reinforcement Learning*. In: *arXiv:1802.05438 [cs]* (2018).

[119] L. Zheng, J. Yang, H. Cai, W. Zhang, J. Wang, and Y. Yu. *MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence*. In: *arXiv:1712.00600 [cs]* (2017).

[120] P. Zhu, B. Chen, and J. C. Principe. *Learning Nonlinear Generative Models of Time Series With a Kalman Filter in RKHS*. In: *IEEE Transactions on Signal Processing* 62.1 (Jan. 2014), pp. 141–155.

# Appendix

## A Derivations for the Subspace Conditional Operator

We define the subspace conditional operator $\mathcal{C}^{S}_{Y|X}$ as the mapping from an embedding $\varphi(\boldsymbol{x}) \in \mathcal{H}_X$ to the mean embedding $\mu_{Y|\boldsymbol{x}} \in \mathcal{H}_Y$ of the conditional distribution $P(Y|\boldsymbol{x})$ conditioned on a certain variate $\boldsymbol{x}$. To obtain this subspace conditional operator, we first introduce an auxiliary conditional operator $\mathcal{C}^{\mathrm{aux}}_{Y|X}$ which maps from the subspace projection of the embedding $\boldsymbol{\Gamma}^{\intercal}\varphi(\boldsymbol{x})$ to the mean map of the conditional distribution, i.e.,

$$\hat{\mu}_{Y|\boldsymbol{x}} = \hat{\mathcal{C}}^{\mathrm{aux}}_{Y|X}\boldsymbol{\Gamma}^{\intercal}\varphi(\boldsymbol{x}). \tag{A.1}$$

We can then derive this auxiliary conditional operator by minimizing the squared error on the full data set

$$\begin{aligned}
\hat{\mathcal{C}}^{\mathrm{aux}}_{Y|X} &= \arg\min_{\mathcal{C}} \left\|\boldsymbol{\Phi} - \mathcal{C}\boldsymbol{\Gamma}^{\intercal}\boldsymbol{\Upsilon}_x\right\|_2 && \text{(A.2)}\\
0 &= \frac{\partial}{\partial\hat{\mathcal{C}}^{\mathrm{aux}}_{Y|X}} \left\|\boldsymbol{\Phi} - \hat{\mathcal{C}}^{\mathrm{aux}}_{Y|X}\boldsymbol{\Gamma}^{\intercal}\boldsymbol{\Upsilon}_x\right\|_2 && \text{(A.3)}\\
0 &= -2\left(\boldsymbol{\Phi} - \hat{\mathcal{C}}^{\mathrm{aux}}_{Y|X}\boldsymbol{\Gamma}^{\intercal}\boldsymbol{\Upsilon}_x\right)\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Gamma} && \text{(A.4)}\\
\hat{\mathcal{C}}^{\mathrm{aux}}_{Y|X}\boldsymbol{\Gamma}^{\intercal}\boldsymbol{\Upsilon}_x\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Gamma} &= \boldsymbol{\Phi}\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Gamma} && \text{(A.5)}\\
\hat{\mathcal{C}}^{\mathrm{aux}}_{Y|X} &= \boldsymbol{\Phi}\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Gamma}\left(\boldsymbol{\Gamma}^{\intercal}\boldsymbol{\Upsilon}_x\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Gamma} + \lambda\boldsymbol{I}\right)^{-1}. && \text{(A.6)}
\end{aligned}$$

We can then substitute this result for the auxiliary conditional operator in Equation 2.23 and obtain the subspace conditional operator as

$$\begin{aligned}
\hat{\mathcal{C}}^{S}_{Y|X} &= \hat{\mathcal{C}}^{\mathrm{aux}}_{Y|X}\boldsymbol{\Gamma}^{\intercal} && \text{(A.7)}\\
&= \boldsymbol{\Phi}\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Gamma}\left(\boldsymbol{\Gamma}^{\intercal}\boldsymbol{\Upsilon}_x\boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Gamma} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Gamma}^{\intercal} && \text{(A.8)}\\
&= \boldsymbol{\Phi}\boldsymbol{K}_{x\bar{x}}\left(\boldsymbol{K}_{x\bar{x}}^{\intercal}\boldsymbol{K}_{x\bar{x}} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Gamma}^{\intercal}, && \text{(A.9)}
\end{aligned}$$

where $\boldsymbol{K}_{x\bar{x}} = \boldsymbol{\Upsilon}_x^{\intercal}\boldsymbol{\Gamma} \in \mathbb{R}^{n\times m}$ is the kernel matrix of the sample feature set $\boldsymbol{\Upsilon}_x$ and its subset $\boldsymbol{\Gamma}$.

## B Derivations for the Kernel Kalman Rule and its Applications

## B.1 The Residual of the Observation Operator is Unbiased

We can easily show that the residual of the observation operator is unbiased by taking the expectation

$$\begin{aligned}\mathbb{E}_Y[\boldsymbol{\zeta}_t] &= \mathbb{E}_Y[\phi(\boldsymbol{y}_t) - \mathcal{C}_{Y|X}\varphi(\boldsymbol{x}_t)] && \text{(B.10)}\\ &= \mathbb{E}_{Y,X}[\phi(\boldsymbol{y}_t) - \mathcal{C}_{Y|X}\varphi(\boldsymbol{x}_t)] && \text{(B.11)}\\ &= \mathbb{E}_{Y,X}[\phi(\boldsymbol{y}_t)] - \mathcal{C}_{Y|X}\mu_{X,t} && \text{(B.12)}\\ &= \mathbb{E}_X[\mathbb{E}_{Y|X}[\phi(\boldsymbol{y}_t)]] - \mathbb{E}_X[\mathbb{E}_{Y|X}[\phi(\boldsymbol{y}_t)]] = 0. && \text{(B.13)}\end{aligned}$$

Here, we used the definition of the conditional embedding operator found in [103].

## B.2 Derivation of the Optimal Kalman Gain Operator

We want to find the kernel Kalman gain operator $\mathcal{Q}_t$ which minimizes the expected squared loss $\mathbb{E}\left[\left(\boldsymbol{\varepsilon}_t^+\right)^\intercal \boldsymbol{\varepsilon}_t^+\right]$ or equivalently the variance of the estimator. The objective for minimizing the variance can also be reformulated as minimizing the trace of the a-posteriori covariance operator $\mathcal{C}_{XX,t}^+$ of the state $\boldsymbol{x}_t$ at time $t$, i.e.,

$$\begin{aligned}\min_{\mathcal{Q}_t} \mathbb{E}\left[\left(\boldsymbol{\varepsilon}_t^+\right)^\intercal \boldsymbol{\varepsilon}_t^+\right] &= \min_{\mathcal{Q}_t} \operatorname{Tr} \mathbb{E}\left[\boldsymbol{\varepsilon}_t^+ \left(\boldsymbol{\varepsilon}_t^+\right)^\intercal\right] && \text{(B.14)}\\ &= \min_{\mathcal{Q}_t} \operatorname{Tr} \mathbb{E}\left[\left(\varphi(\boldsymbol{x}_t) - \mu_{X,t}^+\right)\left(\varphi(\boldsymbol{x}_t) - \mu_{X,t}^+\right)^\intercal\right] && \text{(B.15)}\\ &= \min_{\mathcal{Q}_t} \operatorname{Tr} \mathcal{C}_{XX,t}^+. && \text{(B.16)}\end{aligned}$$

By substituting the posterior error with $\boldsymbol{\varepsilon}_t^+ = (\mathcal{I} - \mathcal{Q}_t\mathcal{C}_{Y|X})\boldsymbol{\varepsilon}_t^- - \mathcal{Q}_t\boldsymbol{\zeta}_t$, we can now rewrite this a-posteriori covariance operator as

$$\begin{aligned}\mathcal{C}_{XX,t}^+ &= \mathbb{E}\left[\boldsymbol{\varepsilon}_t^+ \left(\boldsymbol{\varepsilon}_t^+\right)^\intercal\right] && \text{(B.17)}\\ &= \mathbb{E}\left[\left(\left(\mathcal{I} - \mathcal{Q}_t\mathcal{C}_{Y|X}\right)\boldsymbol{\varepsilon}_t^- - \mathcal{Q}_t\boldsymbol{\zeta}_t\right)\left(\left(\mathcal{I} - \mathcal{Q}_t\mathcal{C}_{Y|X}\right)\boldsymbol{\varepsilon}_t^- - \mathcal{Q}_t\boldsymbol{\zeta}_t\right)^\intercal\right] && \text{(B.18)}\\ &= \left(\mathcal{I} - \mathcal{Q}_t\mathcal{C}_{Y|X}\right)\mathbb{E}\left[\boldsymbol{\varepsilon}_t^-(\boldsymbol{\varepsilon}_t^-)^\intercal\right]\left(\mathcal{I} - \mathcal{Q}_t\mathcal{C}_{Y|X}\right)^\intercal - && \text{(B.19)}\\ &\quad \mathcal{Q}_t\mathbb{E}\left[\boldsymbol{\zeta}_t(\boldsymbol{\varepsilon}_t^-)^\intercal\right]\left(\mathcal{I} - \mathcal{Q}_t\mathcal{C}_{Y|X}\right)^\intercal - \left(\mathcal{I} - \mathcal{Q}_t\mathcal{C}_{Y|X}\right)\mathbb{E}\left[\boldsymbol{\varepsilon}_t^-\boldsymbol{\zeta}_t^\intercal\right]\mathcal{Q}_t^\intercal + && \text{(B.20)}\\ &\quad \mathcal{Q}_t\mathbb{E}\left[\boldsymbol{\zeta}_t\boldsymbol{\zeta}_t^\intercal\right]\mathcal{Q}_t^\intercal && \text{(B.21)}\end{aligned}$$

Since the residual of the observation operator $\boldsymbol{\zeta}_t$ is assumed to be independent from the estimation error and since we assume the a-priori estimate to be zero mean we get

$$\mathbb{E}\left[\boldsymbol{\zeta}_t(\boldsymbol{\varepsilon}_t^-)^\intercal\right] = \mathbb{E}\left[\boldsymbol{\zeta}_t\right]\mathbb{E}\left[(\boldsymbol{\varepsilon}_t^-)^\intercal\right] = \mathbb{E}\left[(\boldsymbol{\varepsilon}_t^-)^\intercal\right]\mathbb{E}\left[\boldsymbol{\zeta}_t\right] = \mathbb{E}\left[\boldsymbol{\varepsilon}_t^-\boldsymbol{\zeta}_t^\intercal\right] = 0. \qquad \text{(B.22)}$$

With this insight the posterior covariance operator can be reformulated as

$$\mathcal{C}^{+}_{XX,t} = \left(\mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X}\right) \mathcal{C}^{-}_{XX,t} \left(\mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X}\right)^{\mathsf{T}} + \mathcal{Q}_t \mathcal{R} \mathcal{Q}^{\mathsf{T}}_t, \tag{B.23}$$

where $\mathcal{R} = \mathbb{E}[\zeta_t \zeta_t^{\mathsf{T}}]$ is the covariance of the residual of the observation operator. Taking the derivative of the trace of the covariance operator and setting it to zero leads to the solution for the kernel Kalman gain operator as

$$0 = 2\left(\mathcal{I} - \mathcal{Q}_t \mathcal{C}_{Y|X}\right) \mathcal{C}^{-}_{XX,t} \left(-\mathcal{C}^{\mathsf{T}}_{Y|X}\right) + 2\mathcal{Q}_t \mathcal{R} \tag{B.24}$$

$$\mathcal{Q}_t \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} + \mathcal{Q}_t \mathcal{R} = \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \tag{B.25}$$

$$\mathcal{Q}_t \left(\mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} + \mathcal{R}\right) = \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \tag{B.26}$$

$$\mathcal{Q}_t = \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \left(\mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} + \mathcal{R}\right)^{-1}. \tag{B.27}$$

## B.3 Simplifying the Update of the Covariance Operator

Following the derivations in [97], we can find a simpler formulation for the update of the covariance operator in Equation 2.63. First, we substitute the kernel Kalman gain from Equation 2.64 using the notation $\mathcal{U} = \left(\mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} + \mathcal{R}\right)$ which yields

$$\mathcal{C}^{+}_{XX,t} = \left(\mathcal{I} - \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{C}_{Y|X}\right) \mathcal{C}^{-}_{XX,t} \left(\dots\right)^{\mathsf{T}} + \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{R} \left(\dots\right)^{\mathsf{T}} \tag{B.28}$$

By expanding both terms we obtain

$$\mathcal{C}^{+}_{XX,t} = \mathcal{C}^{-}_{XX,t} - \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} - \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} + \tag{B.29}$$

$$\mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} + \tag{B.30}$$

$$\mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{R} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \tag{B.31}$$

If we now combine the second and third as well as the last two terms in this equation, we arrive at

$$\mathcal{C}^{+}_{XX,t} = \mathcal{C}^{-}_{XX,t} - 2\mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} + \tag{B.32}$$

$$\mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \left(\mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} + \mathcal{R}\right) \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \tag{B.33}$$

$$= \mathcal{C}^{-}_{XX,t} - 2\mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} + \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{U} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \tag{B.34}$$

$$= \mathcal{C}^{-}_{XX,t} - \mathcal{C}^{-}_{XX,t} \mathcal{C}^{\mathsf{T}}_{Y|X} \mathcal{U}^{-1} \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \tag{B.35}$$

$$= \mathcal{C}^{-}_{XX,t} - \mathcal{Q}_t \mathcal{C}_{Y|X} \mathcal{C}^{-}_{XX,t} \tag{B.36}$$

With this update equation for the estimator covariance, which is more concise and more similar to the original equations of the Kalman filter, we can summarize the kernel Kalman rule as follows: I. compute the kernel Kalman gain operator $\mathcal{Q}_t$ (Equation 2.64), II. update the estimator of the mean embedding $\mu^{+}_{X,t}$ (Equation 2.55), III. update the estimator of the covariance embedding $\mathcal{C}^{+}_{XX,t}$ (Equation 2.65).

## B.4 Derivations for the Sample-based Kernel Kalman Rule

The Kalman gain operator can be rewritten with the sample estimators of mean embedding, covariance embedding and conditional embedding operator as

$$\hat{\mathcal{Q}}_t = \boldsymbol{\Upsilon}_x \boldsymbol{S}_t^- \boldsymbol{O}^\intercal \boldsymbol{\Phi}^\intercal \left(\boldsymbol{\Phi} \boldsymbol{O} \boldsymbol{S}_t^- \boldsymbol{O}^\intercal \boldsymbol{\Phi}^\intercal + \kappa \mathcal{I}\right)^{-1}, \tag{B.37}$$

However, this formulation still has the inversion of a potentially infinite dimensional operator. To this end, we can apply the matrix identity $\boldsymbol{A}(\boldsymbol{B}\boldsymbol{A}+\boldsymbol{I})^{-1} = (\boldsymbol{A}\boldsymbol{B}+\boldsymbol{I})^{-1}\boldsymbol{A}$ to the kernel Kalman gain matrix as follows

$$\begin{aligned}
\hat{\mathcal{Q}}_t &= \boldsymbol{\Upsilon}_x \boldsymbol{S}_t^- \boldsymbol{O}^\intercal \underbrace{\boldsymbol{\Phi}^\intercal}_{\text{A}} \Big[\underbrace{\boldsymbol{\Phi} \boldsymbol{O} \boldsymbol{S}_t^- \boldsymbol{O}^\intercal}_{\text{B}} \underbrace{\boldsymbol{\Phi}^\intercal}_{\text{A}} + \kappa \mathcal{I}\Big]^{-1} && \text{(B.38)}\\
&= \boldsymbol{\Upsilon}_x \boldsymbol{S}_t^- \boldsymbol{O}^\intercal (\underbrace{\boldsymbol{\Phi}^\intercal}_{\text{A}} \underbrace{\boldsymbol{\Phi} \boldsymbol{O} \boldsymbol{S}_t^- \boldsymbol{O}^\intercal}_{\text{B}} + \kappa \boldsymbol{I})^{-1} \underbrace{\boldsymbol{\Phi}^\intercal}_{\text{A}} && \text{(B.39)}\\
&= \boldsymbol{\Upsilon}_x \underbrace{\boldsymbol{S}_t^- \boldsymbol{O}^\intercal (\boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{S}_t^- \boldsymbol{O}^\intercal + \kappa \boldsymbol{I})^{-1}}_{\boldsymbol{Q}_t} \boldsymbol{\Phi}^\intercal, && \text{(B.40)}
\end{aligned}$$

where we defined $\boldsymbol{Q}_t = \boldsymbol{S}_t^- \boldsymbol{O}^\intercal (\boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{S}_t^- \boldsymbol{O}^\intercal + \kappa \boldsymbol{I})^{-1} \in \mathbb{R}^{n \times n}$ with the Gram matrix of the observations $\boldsymbol{G}_{yy} = \boldsymbol{\Phi}^\intercal \boldsymbol{\Phi}$. Based on this formulation of the kernel Kalman gain operator, we can now derive finite vector/matrix formulations for the update equations of the estimator of the mean embedding (Equation 2.55) and the estimator of the covariance operator (Equation 2.65). For the estimator of the mean embedding, we obtain

$$\begin{aligned}
\hat{\mu}^+_{X,t} &= \hat{\mu}^-_{X,t} + \hat{\mathcal{Q}}_t \left(\phi(\boldsymbol{y}_t) - \hat{\mathcal{C}}_{Y|X} \hat{\mu}^-_{X,t}\right) && \text{(B.41)}\\
\boldsymbol{\Upsilon}_x \boldsymbol{m}_t^+ &= \boldsymbol{\Upsilon}_x \boldsymbol{m}_t^- + \boldsymbol{\Upsilon}_x \boldsymbol{Q}_t \boldsymbol{\Phi}^\intercal \left(\phi(\boldsymbol{y}_t) - \boldsymbol{\Phi} \left(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_m\right)^{-1} \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Upsilon}_x \boldsymbol{m}_t^-\right), && \text{(B.42)}\\
&= \boldsymbol{\Upsilon}_x \boldsymbol{m}_t^- + \boldsymbol{\Upsilon}_x \boldsymbol{Q}_t \left(\boldsymbol{\Phi}^\intercal \phi(\boldsymbol{y}_t) - \boldsymbol{\Phi}^\intercal \boldsymbol{\Phi} \left(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_m\right)^{-1} \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Upsilon}_x \boldsymbol{m}_t^-\right), && \text{(B.43)}\\
\boldsymbol{m}_t^+ &= \boldsymbol{m}_t^- + \boldsymbol{Q}_t \left(\boldsymbol{g}_{:\boldsymbol{y}_t} - \boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{m}_t^-\right), && \text{(B.44)}
\end{aligned}$$

where $\boldsymbol{g}_{:\boldsymbol{y}_t} = \boldsymbol{\Phi}^\intercal \phi(\boldsymbol{y}_t)$ is the embedding of the measurement at time $t$. And the estimator for the covariance operator gets

$$\begin{aligned}
\hat{\mathcal{C}}^+_{XX,t} &= \hat{\mathcal{C}}^-_{XX,t} - \boldsymbol{\Upsilon}_x \boldsymbol{Q}_t \boldsymbol{\Phi}^\intercal \hat{\mathcal{C}}_{Y|X} \hat{\mathcal{C}}^-_{XX,t} && \text{(B.45)}\\
\boldsymbol{\Upsilon}_x \boldsymbol{S}_t \boldsymbol{\Upsilon}_x^\intercal &= \boldsymbol{\Upsilon}_x \boldsymbol{S}_t^- \boldsymbol{\Upsilon}_x^\intercal - \boldsymbol{\Upsilon}_x \boldsymbol{Q}_t \boldsymbol{\Phi}^\intercal \boldsymbol{\Phi} \left(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_m\right)^{-1} \boldsymbol{\Upsilon}_x^\intercal \boldsymbol{\Upsilon}_x \boldsymbol{S}_t^- \boldsymbol{\Upsilon}_x^\intercal && \text{(B.46)}\\
&= \boldsymbol{\Upsilon}_x \boldsymbol{S}_t^- \boldsymbol{\Upsilon}_x^\intercal - \boldsymbol{\Upsilon}_x \boldsymbol{Q}_t \boldsymbol{G}_{yy} \left(\boldsymbol{K}_{xx} + \lambda \boldsymbol{I}_m\right)^{-1} \boldsymbol{K}_{xx} \boldsymbol{S}_t^- \boldsymbol{\Upsilon}_x^\intercal && \text{(B.47)}\\
\boldsymbol{S}_t &= \boldsymbol{S}_t^- - \boldsymbol{Q}_t \boldsymbol{G}_{yy} \boldsymbol{O} \boldsymbol{S}_t^-. && \text{(B.48)}
\end{aligned}$$

## B.5 Derivation of the Subspace Kernel Kalman Gain Operator/Matrix

To derive the gain operator of the subspace kernel Kalman rule, we apply the subspace conditional operator

$$\hat{\mathcal{C}}^S_{Y|X} = \boldsymbol{\Phi} \boldsymbol{K}_{x\bar{x}} \left( \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{\Gamma}^\top, \tag{B.49}$$

to the kernel Kalman gain from Equation 2.64 and obtain

$$\begin{aligned} \hat{\mathcal{Q}}^S_t &= \hat{\mathcal{C}}^-_{XX,t} \left( \hat{\mathcal{C}}^S_{Y|X} \right)^\top \left( \hat{\mathcal{C}}^S_{Y|X} \hat{\mathcal{C}}^-_{XX,t} \left( \hat{\mathcal{C}}^S_{Y|X} \right)^\top + \kappa \mathcal{I} \right)^{-1} && \text{(B.50)} \\ &= \boldsymbol{\Upsilon}_x \boldsymbol{S}^-_t \boldsymbol{K}_{x\bar{x}} (\boldsymbol{O}^S)^\top \underbrace{\boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{\Phi}^\top}_{A} \Big( \underbrace{\boldsymbol{\Phi} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{S}^-_t \boldsymbol{K}_{x\bar{x}} (\boldsymbol{O}^S)^\top}_{B} \underbrace{\boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{\Phi}^\top}_{A} + \kappa \mathcal{I} \Big)^{-1}. && \text{(B.51)} \end{aligned}$$

Here, we denote $\boldsymbol{O}^S := \left( \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{K}_{x\bar{x}} + \lambda \boldsymbol{I} \right)^{-1}$. To obtain a finite dimensional matrix in the inverse, we apply again the matrix identity $\boldsymbol{A}(\boldsymbol{B}\boldsymbol{A} + \boldsymbol{I})^{-1} = (\boldsymbol{A}\boldsymbol{B} + \boldsymbol{I})^{-1}\boldsymbol{A}$ and arrive at

$$\begin{aligned} \hat{\mathcal{Q}}^S_t &= \boldsymbol{\Upsilon}_x \boldsymbol{S}^-_t \boldsymbol{K}_{x\bar{x}} (\boldsymbol{O}^S)^\top \Big( \underbrace{\boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{\Phi}^\top}_{A} \underbrace{\boldsymbol{\Phi} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{S}^-_t \boldsymbol{K}_{x\bar{x}} (\boldsymbol{O}^S)^\top}_{B} + \kappa \boldsymbol{I} \Big)^{-1} \underbrace{\boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{\Phi}^\top}_{A} && \text{(B.52)} \\ &= \boldsymbol{\Upsilon}_x \boldsymbol{S}^-_t \boldsymbol{K}_{x\bar{x}} (\boldsymbol{O}^S)^\top \Big( \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{G}_{yy} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{S}^-_t \boldsymbol{K}_{x\bar{x}} (\boldsymbol{O}^S)^\top + \kappa \boldsymbol{I} \Big)^{-1} \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{\Phi}^\top. && \text{(B.53)} \end{aligned}$$

Using the projecting the subspace kernel Kalman gain into the subspace spanned by the features $\boldsymbol{\Gamma}^\top$ leads to

$$\begin{aligned} \boldsymbol{\Gamma}^\top \hat{\mathcal{Q}}^S_t &= \boldsymbol{\Gamma}^\top \boldsymbol{\Upsilon}_x \boldsymbol{S}^-_t \boldsymbol{K}_{x\bar{x}} (\boldsymbol{O}^S)^\top \Big( \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{G}_{yy} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{S}^-_t \boldsymbol{K}_{x\bar{x}} (\boldsymbol{O}^S)^\top + \kappa \boldsymbol{I} \Big)^{-1} \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{\Phi}^\top && \text{(B.54)} \\ &= \underbrace{\boldsymbol{P}^-_t (\boldsymbol{O}^S)^\top \Big( \boldsymbol{K}^\top_{x\bar{x}} \boldsymbol{G}_{yy} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{P}^-_t (\boldsymbol{O}^S)^\top + \kappa \boldsymbol{I} \Big)^{-1} \boldsymbol{K}^\top_{x\bar{x}}}_{\boldsymbol{Q}^S_t} \boldsymbol{\Phi}^\top, && \text{(B.55)} \end{aligned}$$

where we define the subspace kernel Kalman gain matrix $\boldsymbol{Q}^S_t$. Using this representation, we can obtain the update equations for the subspace projections of mean embedding as

$$\begin{aligned} \boldsymbol{\Gamma}^\top \hat{\mu}^+_{X,t} &= \boldsymbol{\Gamma}^\top \left( \hat{\mu}^-_{X,t} + \hat{\mathcal{Q}}^S_t \left( \phi(\boldsymbol{y}_t) - \hat{\mathcal{C}}^S_{Y|X} \hat{\mu}^-_{X,t} \right) \right) && \text{(B.56)} \\ \boldsymbol{n}^+_{X,t} &= \boldsymbol{n}^-_{X,t} + \boldsymbol{Q}^S_t \left( \boldsymbol{g}_{:\boldsymbol{y}_t} - \boldsymbol{G}_{yy} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{n}^-_t \right), && \text{(B.57)} \end{aligned}$$

and similarly for the covariance operator as

$$\begin{aligned} \boldsymbol{\Gamma}^\top \hat{\mathcal{C}}^+_{XX,t} \boldsymbol{\Gamma} &= \boldsymbol{\Gamma}^\top \left( \hat{\mathcal{C}}^-_{XX,t} - \hat{\mathcal{Q}}^S_t \hat{\mathcal{C}}^S_{Y|X} \hat{\mathcal{C}}^-_{XX,t} \right) \boldsymbol{\Gamma} && \text{(B.58)} \\ \boldsymbol{P}^+_t &= \boldsymbol{P}^-_t - \boldsymbol{Q}^S_t \boldsymbol{\Phi}^\top \boldsymbol{\Phi} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{\Gamma}^\top \hat{\mathcal{C}}^-_{XX,t} \boldsymbol{\Gamma} && \text{(B.59)} \\ &= \boldsymbol{P}^-_t - \boldsymbol{Q}^S_t \boldsymbol{G}_{yy} \boldsymbol{K}_{x\bar{x}} \boldsymbol{O}^S \boldsymbol{P}^-_t && \text{(B.60)} \end{aligned}$$

# Curriculum Vitae

**Gregor H. W. Gebhardt**
Technische Universität Darmstadt
Hochschulstr. 10
64289 Darmstadt
Tel.: +49-6151-16-25372
e-mail: gebhardt@ias.tu-darmstadt.de

## Current Position

| | |
|---|---|
| 02/2015–now | Ph.D Student, Research Assistant, Teaching Assistant<br>Supervisor: Prof. Dr. Gerhard Neumann<br>Referee: Prof. Dr. Jan Peters<br>Computational Learning for Autonomous Systems (CLAS),<br>Technische Universität Darmstadt |

## Education

| | |
|---|---|
| 2011–2014 | **Technische Universität Darmstadt**<br>Master of Science *(Autonomous Systems)*,<br>Specialized masters program at the department of computer science,<br>Thesis: *Embedding Kalman Filters into Reproducing Hilbert Spaces*,<br>Supervisors: Prof. Dr. Gerhard Neumann, Prof. Dr. Jan Peters |
| 2008–2011 | **Freie Universität Berlin**<br>Bachelor of Science *(Computer Science)*,<br>Focus on artificial intelligence, machine learning, and robotics.<br>Thesis: *Learning Symbols for Hierarchical Control from Interaction: Controllability Control*,<br>Supervisors: Prof. Dr. Marc Toussaint, Dr. Tobias Lang,<br>Referee: Prof. Dr. Oliver Brock |
| 2006–2007 | **Universität Potsdam**<br>Geowissenschaften |
| 1997–2006 | **Ottheinrich-Gymnasium Wiesloch**<br>University-entrance diploma (Abitur) with major in mathematics, physics, and geography. |

## Professional Experience

| | |
|---|---|
| 2012–2014 | Student Assistant, Bionic Robotics GmbH, Darmstadt<br>Contribution to the development of the control software for a compliant, tendon-driven robot arm. |

2011 Internship, Carmeq GmbH, Berlin
Development of a tool for automatic generation of test cases for a hardware-in-the-loop-system.

2010–2011 Student Assistant, AutonomOS Project, Freie Universität Berlin
Participation in the software development for autonomous cars.

## Teaching Experience

2015 Teaching Assistant for *Probabilistic Graphical Models*
Prof. Dr. Gerhard Neumann, Technische Universität Darmstadt

2015 Teaching Assistant for *Intelligent Multi-Agent Systems*
Prof. Dr. Gerhard Neumann, Technische Universität Darmstadt

2014 Lecture Assistant for *Machine Learning - Statistical Approaches I*
Prof. Dr. Jan Peters, Technische Universität Darmstadt

2013 Lecture Assistant for *Foundations of Robotics*
Prof. Dr. Oskar von Stryk, Technische Universität Darmstadt

## Student Supervision

2017 S. Venugopal, D. Singh Mohan. Integrated Project.
*Learning Hand Kinematics*

2017 P. Becker. Integrated Project.
*Learning Deep Feature Spaces for Inference*

2017 K. Daun, M. Schnaubelt. Integrated Project.
*Learning an Assembling Task with Swarm Robots*

2016 P. Becker. Bachelor Thesis.
*Learning Deep Feature Spaces for Nonparametric Inference*

2016 A. Hendrich, D. Kauth. Integrated Project.
*Learning to Control the Kilobots with a Light Source*

2015 F. Boschert, M. Holz, S. Kreutzer, M. Willig. Bachelor Praktikum.
*Entwicklung einer Policy Search Toolbox in Python*

## Extracurricular Activities

2013 Participation at the RoboCup in Eindhoven, The Netherlands
with Team Hector Darmstadt
Winner of Best in Class Autonomy Award (Rescue Robot League)

2013 Participation at the RoboCup German Open Magdeburg, Germany
with Team Hector Darmstadt
1st place and winner of Best in Class Autonomy Award (Rescue Robot League)

## Invited Talks

| | |
|---|---|
| Aug. 2015 | Embedding Kalman Filters into Reproducing Kernel Hilbert Spaces<br>Lab of Prof. Dr. Kenji Fukumizu, Institute for Statistical Mathematics<br>Tokyo, Japan |
| Aug. 2015 | Embedding Kalman Filters into Reproducing Kernel Hilbert Spaces<br>Lab of Prof. Dr. Sugiyama, University of Tokyo<br>Tokyo, Japan |

## Contributions to the Scientific Community

| | |
|---|---|
| 2019 | Reviewer for the<br>IEEE Transactions on Signal Processing (T-SP) |
| 2018 | Reviewer for the<br>Conference on Robot Learning (CoRL)<br>Wrokshop on Robot Learning under Partial Observability (at NeurIPS)<br>IEEE International Conference on Intelligent Robots and Systems (IROS)<br>European Workshop on Reinforcement Learning (EWRL) |
| 2017 | Reviewer for the<br>IEEE International Conference on Humanoid Robots (Humanoids)<br>IEEE International Conference on Intelligent Robots and Systems (IROS) |
| 2016 | Reviewer for the<br>AAAI Conference on Artificial Intelligence (AAAI)<br>Journal of Machine Learning Research (JMLR) |
| 2015 | Reviewer for the<br>AAAI Conference on Artificial Intelligence (AAAI)<br>IEEE International Conference on Intelligent Robots and Systems (IROS) |
| 2014 | Reviewer for the<br>Workshop on Autonomously Learning Robots (at NIPS) |