Koopman-based Learning and Control of Agile Robotic Systems

Thesis by

Carl A. A. Folkestad

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy, Control and Dynamical Systems

Caltech

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2022
Defended October 8, 2021

ACKNOWLEDGEMENTS

I am very grateful to my advisor, Joel Burdick, for being a reliable and caring advisor. He has supported me to pursue the ideas I've found most interesting and given me the freedom to make my PhD-journey interesting and enjoyable, both professionally and personally. I would also like to thank Richard Murray, Aaron Ames, and Yisong Yue for being great mentors and sources of inspiration during my PhD and for serving on my committee.

I have been lucky to work with many great graduate students, postdocs, and professors during my years at Caltech. I would especially like to thank Daniel Pastor, Yuxiao Chen, and Skylar Wei for great research collaborations. I would also like to thank Igor Mezic, Ryan Mohr, and Maria Fonoberova for their mentorship and interesting discussions in the early stages of my PhD research. You have all challenged me intellectually, filled in my blind spots, and made the research more interesting, fun, and productive. Finally, I would like to thank everyone in the Burdick and Ames research groups for inspiring me and helping me become a better researcher.

I would also like to thank the Aker Scholarship foundation for their guidance and financial support for pursuing my PhD. In particular, Bjørn Blindheim has been invaluable to help me navigate the possibilities for graduate studies in the US and to help me succeed. Their financial support has also enabled me to pursue research ideas more freely, which has been a great benefit throughout my PhD.

I am deeply grateful to all the great friends I have made during my time at Caltech, who make both work and time off fun, interesting, and entertaining. I am also convinced that I would not have gotten through the first year if it hadn't been for all the brilliant people in CMS and CDS fighting in the trenches besides me. I must especially thank my office mates in both Annenberg and Gates Thomas for making the hours spent in the office enjoyable.

Finally, I would like to thank my family for their unconditional support and for giving me a strong foundation at home that makes it easier to pursue my dreams even though it means being far away. Last but not least, I must thank my partner-in-crime, Maria, for always dreaming with me, and for making life exciting and colorful.

ABSTRACT

Learning methods to enable high performance control systems have recently shown promising results in selected environments and applications. These advances promote the next generation of autonomous robots capable of significantly improving efficiency, cost, and safety in their respective domains. Importantly, these systems are *safety-critical* and operate in proximity to humans in diverse and uncertain environments. As a result, operational failures may cause significant material and societal losses. Additionally, robot learning and control are further complicated by requiring fast controller update rates and operational constraint satisfaction.

To address these challenges, this thesis presents multiple methods based on Koopman operator theory. The first approach develops algorithms to learn lifted-dimensional models of nonlinear systems and leverages the models in model predictive control (MPC) design. Koopman-based methods typically employ hand-crafted observable functions to "lift" the state variables to the higher dimensional space. For most systems, this leads to poor prediction performance and inefficient use of data and computational resources. Instead, I present methods that generate observable functions from data, both based on underlying theory and by incorporating the observable functions and model structure in a neural network model. This allows lower dimensional models, important for real-time control, and enables the nonlinearities of control-affine dynamics to be captured, crucial to describing many robotic systems. I use quadrotor drones to experimentally demonstrate that the learned models combined with MPC can achieve close to optimal behavior while respecting important operational constraints.

The last part of the thesis is concerned with endowing systems with an arbitrary nominal control policy with safety guarantees. Control barrier functions (CBFs) are a powerful tool to achieve this, yet they rely on the computation of control invariant sets, which is notoriously difficult. To avoid this, a backup strategy can be used to implicitly define a control invariant set. However, this requires forward integration of the system dynamics under a backup controller, which is prohibitively expensive for realistic systems. I present a method that replaces the expensive integration using learned Koopman operators of the closed-loop dynamics. As a result, the online computation time required to evaluate the controller is drastically reduced, enabling real-time use. I also derive an error bound on the unmodeled dynamics in order to robustify the CBF controller and demonstrate the method on multi-agent collision avoidance for wheeled robots and quadrotors.

PUBLISHED CONTENT AND CONTRIBUTIONS

Folkestad, Carl, Skylar X. Wei, and Joel W. Burdick (2022). "Quadrotor Trajectory Tracking with Learned Dynamics: Joint Koopman-based Learning of System Models and Function Dictionaries". In: *2022 International Conference on Robotics and Automation (ICRA) (submitted).*
Contribution: Algorithm development, code implementation, experiment design and execution, article writing. Content is presented in Chapter 4.

Folkestad, Carl and Joel W. Burdick (May 2021). "Koopman NMPC: Koopman-based Learning and Nonlinear Model Predictive Control of Control-affine Systems". In: *2021 IEEE International Conference on Robotics and Automation (ICRA).* URL: `http://arxiv.org/abs/2105.08036`.
Contribution: Algorithm development and theoretical analyses, code implementation, experiment design and execution, article writing. Content is presented in Chapter 4.

Folkestad, Carl, Yuxiao Chen, et al. (Dec. 2021). "Data-Driven Safety-Critical Control: Synthesizing Control Barrier Functions with Koopman Operators". In: *IEEE Control Systems Letters* 5.6, pp. 2012–2017. ISSN: 24751456. DOI: `10.1109/LCSYS.2020.3046159`.
Contribution: Algorithm development and theoretical analyses, code implementation, experiment design and execution, article writing. Content is presented in Chapter 5.

Folkestad, Carl, Daniel Pastor, and Joel W. Burdick (May 2020). "Episodic Koopman Learning of Nonlinear Robot Dynamics with Application to Fast Multirotor Landing". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., pp. 9216–9222. ISBN: 9781728173955. DOI: `10.1109/ICRA40945.2020.9197510`.
Contribution: Algorithm development, simulation and hardware code implementation, experiment design and execution, article writing. Content is presented in Chapter 3.

Folkestad, Carl, Daniel Pastor, Igor Mezic, et al. (July 2020). "Extended Dynamic Mode Decomposition with Learned Koopman Eigenfunctions for Prediction and Control". In: *Proceedings of the American Control Conference*. Vol. 2020-July. Institute of Electrical and Electronics Engineers Inc., pp. 3906–3913. ISBN: 9781538682661. DOI: `10.23919/ACC45564.2020.9147729`.
Contribution: Algorithm development and theoretical analyses, code implementation, experiment design and execution, article writing. Content is presented in Chapter 3.

Pastor, Daniel, Carl Folkestad, and Joel W. Burdick (Dec. 2020). "Ensemble Model Predictive Control: Learning and Efficient Robust Control of Uncertain Dynamical Systems". In: *Proceedings of the IEEE Conference on Decision and Control*

2020-Decem, pp. 1254–1259. DOI: `10.1109/CDC42340.2020.9304442`. Contribution: Algorithm development and theoretical analyses, article writing. Content is not part of this thesis.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

NOMENCLATURE

CBF.	Control barrier function.
DMD.	Dynamic mode decomposition.
$EDMD$.	Extended dynamic mode decomposition.
GP.	Gaussian processes.
GPR.	Gaussian processes regression.
ML.	Machine learning.
MPC.	Model predictive control.
$NMPC$.	Nonlinear model predictive control.
QP.	Quadratic program.
RL.	Reinforcement learning.

Chapter 1

INTRODUCTION

Over the last few decades, an increasing number of agile autonomous robotic systems have been conceptualized and demonstrated. Dynamic autonomous robot applications are currently being developed for a wide range of commercial and research applications with examples including autonomous driving (Cruise, 2018), drone delivery (O'Brien, 2018), mobility recovery of paraplegic patients (Gurriet, Finet, et al., 2018), and planetary exploration (Balaram et al., 2018). These applications have the potential to revolutionize the sectors they operate in, but require systems and technologies that can perform a wide range of tasks while operating in close proximity to humans in highly diverse and uncertain environments. A further complication is that robots for these tasks typically are *safety critical*, where any failure can lead to catastrophic societal or economic consequences in addition to harm or loss of human life. As a result, significant academic and industrial research and development efforts are aimed at developing systems capable of navigating complex tasks and environments to realize the benefits of widespread robot adoption while maintaining strict safety and reliability standards.

The current progress and optimism around autonomous systems rely on advances in a range of core technologies that together lay the foundation for the future of high performance autonomous robots. First, better mechanical designs, materials and sensor hardware enable lighter, more durable, and longer operating duration robots at reasonable cost. Second, sensors are not only becoming smaller, lighter, and cheaper, but the quality and type of data that can be captured are also changing. One important example is how progress in image classification using machine learning (ML) and fast graphical processing units enable rich data to be collected and interpreted using inexpensive cameras, providing important information about the context and environment (Schmidhuber, 2015; Lecun, Bengio, and Hinton, 2015). Third, control algorithms have improved to better take advantage of the improved sensing and model information resulting in more efficient and higher performing systems. For example, model predictive control that is capable of optimizing a performance criterion while satisfying state and actuation constraints can now be solved in real-time even with nonlinear dynamics models (Kouzoupis et al., 2018; Gros et al., 2020; Grandia et al., 2020). Also, nonlinear control methods based on Lyapunov and control barrier functions can be efficiently implemented using quadratic programs (Ames and M. Powell, 2013; Ames, Xu, et al., 2017). In addition to the improvements in pure control methods, ML-inspired methods have significantly improved and are intensely researched. In particular, reinforcement learning (RL), concerned with improving the future operation of a dynamical system based on past data, has demonstrated impressive performance in simulated and selected laboratory experiments. A large number of methods and architectures are currently explored, ranging from model-free methods, which directly learn a control policy based on past data and a performance measure, and model-based methods, which explicitly learn a model of the underlying system dynamics and use a control algorithm exploiting the learned model as a policy (Recht, 2019; Kober, Bagnell, and Peters, 2013).

This thesis is focused on model-based methods and explores how ideas based on Koopman operator theory (Koopman and Neumann, 1932) can be used for model learning and control of agile autonomous robotic systems. Whereas a system's behavior conventionally is characterized via its state space flows, Koopman-based approaches study the evolution of *observables*, which are functions over the state-space. In this space, the system can be represented by a *linear* (but possibly infinite dimensional) operator (Lan and Mezić, 2013; Mauroy and Goncalves, 2016).

Although many methods have been proposed to use data and learning to improve control performance of autonomous robots, many challenges are still unresolved. In particular, many modern learning architectures rely on a very large number of learnable parameters to achieve accurate models, requiring very large amounts of data which can be infeasible or prohibitively expensive to collect from a physical robot. Furthermore, large models may take too long to evaluate at runtime when used for controllers which are typically updated at a rate of 10-1000 hz. Secondly, few methods allow safety and performance behavior to be interpreted and/or guaranteed, prohibiting their deployment for safety critical systems. Finally, all real-world robots have constraints on their actuation, e.g. the maximum force that can be exerted by an actuator, and operation, e.g. avoiding collision with other agents, and therefore need control methods that can incorporate these constraints into the design such that constraint satisfaction can be guaranteed and close to optimal behavior realized.

To address these challenges, this thesis presents methods that solve these problems using two different approaches. The first approach develops learning algorithms inspired by Koopman spectral theory to learn higher-dimensional lifted models of nonlinear control systems and leverage the learned models in model predictive control design. Two main challenges are identified in contemporary Koopman-based learning methods. First, hand-crafted observable functions are typically used to "lift" the state variables to the higher dimensional space. For most systems, this will lead to poor prediction performance over longer time horizons and inefficient use of data and computational resources. Chapter 3 presents a systematic approach to generate observable functions from data based on theory of the existence of a diffeomorphism between the nonlinear dynamics and their linearization around a fixed point. Furthermore, Chapter 4 presents a more flexible approach to observable design by incorporating the model structure of Koopman-based models in a deep neural network architecture. Second, most existing methods rely on identifying lifted linear models, which are not capable of capturing nonlinear control-affine dynamics, a crucial model class for robotic systems to capture important nonlinear model effects of how actuation enters the system. Chapter 4 therefore shows how to utilize Koopman-based learning underpinned by a *bilinear* model, that theoretically can capture control-affine dynamics, and how to use the learned model in a nonlinear model predictive control framework.

The second approach focuses on how to endow a system with an arbitrary nominal control policy with safety guarantees. This can for example be used to ensure that a system remains safe during the training process of a model-free reinforcement learning policy (Cheng et al., 2019). Multiple approaches use Control barrier functions (CBFs) to choose the action closest possible to the nominal policy while ensuring safety. However, when the system has constrained actuation, as is often the case for robotic systems, a control invariant set must be calculated, which is notoriously difficult. The expensive offline computation can be avoided by designing a backup controller stabilizing the system to a safe state, and implicitly defining the control invariant set by integrating the system dynamics under the backup controller. Unfortunately, the integration is prohibitively expensive for high dimensional systems, and inaccurate in the presence of unmodelled dynamics. Chapter 5 presents a method to learn discrete-time Koopman operators of the closed-loop dynamics under a backup strategy. This approach replaces forward integration by a simple matrix multiplication, which can mostly be computed offline. Furthermore, an error bound on the unmodeled dynamics is derived in order to robustify the CBF controller.

In the remainder of this section, I will first do a survey of existing and ongoing work on model-based learning and control before outlining the main contributions and structure of the thesis.

1.1 Related Work

Currently, many different approaches are pursued with the common goal of merging learning and control to achieve high performance controllers with provable guarantees and adaptive behavior. In this section, a selection of this work is highlighted to give an overview of the current state-of-the-art. This is not meant to be an exhaustive literature review, but rather a discussion of examples of progress in various recent directions relevant to this thesis.

I focus on model-based approaches and learning methods motivated by structure resulting from specific control strategies, while excluding direct policy learning methods. This is strongly motivated by the need for data efficiency in learning for robotic control strategies to be able to realize these methodologies on physical systems. Furthermore, model-based methods typically yield more compact function representations, whose evaluation time is crucial for real-time control. Lastly, deriving usable model error bounds can be easier by exploiting prior knowledge and structure of the dynamics which is more naturally encoded in model-based approaches.

Methods that Learn Full State Space Dynamics

Methods that learn full state space models of the dynamics are typically independent of specific task or control objectives, and can therefore be used for both prediction and control with any compatible control design approach. Many works have shown that using machine learning techniques coupled with traditional control design methods can yield significant performance improvements over first-principles modeling for systems with hard to model dynamics effects or unknown model parameters. A large portion of the algorithms that learn full state space models can be classified into three categories: (1) neural network-based methods, (2) Gaussian process-based methods, and (3) Koopman-based methods.

Neural Network-based Methods: Inspired by the widespread adoption of neural network architectures in other domains, many researchers have explored model learning using neural network-based methods in the setting of controlled dynamical systems. Although neural network architectures certainly are capable of capturing dynamics models, two important challenges for robotics application are obtaining a reliable model with a limited amount of collected data, and how to design a controller utilizing the learned model. One approach to address these challenges is to leverage prior knowledge of the system dynamics to constrain the learning process such that stable control using the learned model can be guaranteed. G. Shi, X. Shi, et al., 2019 proposed a spectrally normalized neural network to learn the ground effect to improve landing of a multirotor. Interestingly, the spectral normalization is shown to both improve model generalization and enables control design with stability guarantees. However, the stability guarantee requires the unknown part of the dynamics to be small relative to the nominal model, and state and actuation constraints cannot be incorporated into the control design.

Manek and Kolter take a different approach and use the stability of the true system (if known) as a prior to learn neural network-based dynamical models (Manek and Kolter, 2020). To achieve this, a model for the dynamics and a Lyapunov function guaranteeing the stability of the learned dynamics model are inferred simultaneously. The Lyapunov function is then used to project the learned dynamics into the space of stable dynamics if needed, ensuring that the learned dynamics model is stable throughout the training process. Although an interesting idea, controlled dynamical systems are not yet considered, and as such, stability guarantees of the true dynamical system when it is controlled based on the learned dynamics and/or Lyapunov function are not developed.

Gaussian Process-based Methods: *Gaussian process regression* (GPR) models the dynamical system as a Gaussian process and uses data and a prior on the state transition distribution to estimate a posterior mean and variance. If a good prior can be designed, GPR provides a practical and theoretically appealing tool for inference of dynamical models because structure and prior knowledge can be encoded, and a measure of uncertainty is obtained as part of the inference process (Rasmussen and C. K. I. Williams, 2018). Deisenroth, Fox, and Rasmussen, 2015 demonstrate that GPR can efficiently be used to learn dynamics to improve sample efficiency and performance in a model-based RL framework. GPR has also been used in combination with a nominal model to learn residual dynamics (Chang et al., 2017; Beckers, Kulić, and Hirche, 2019). If a reasonable nominal model is known, this can improve sample efficiency and aid theoretical guarantees on safety and stability.

More recently, Wenk et al., 2020 proposed a GP-based learning framework leveraging knowledge of the ODE parametric form of the system dynamics being learned. The knowledge of the parametric forms significantly reduces the number of parameters to be learned resulting in high data efficiency. When the prior information is accurate, the method outperforms other state-of-the-art methods based on GPs. Although the assumptions of their current work is restrictive in many settings, the idea to encode all available prior knowledge in GP type learning is attractive and demonstrates the flexibility of GPR. However, computational complexity is high, restricting practical use of GPR to certain applications. Furthermore, design of controllers that are robust to model uncertainties in this approach while respecting state and actuation bounds, is not evident.

GPR has also been extensively used to learn models or partial models to be utilized for safety filtering. This will be discussed in the end of this section.

Koopman-based Methods: Koopman-inspired modelling and identification techniques have received substantial recent attention (Rowley et al., 2009; Budišić, Mohr, and Mezić, 2012). In particular, the Dynamic Mode Decomposition (DMD) and extended DMD (EDMD) methods have emerged as efficient numerical algorithms to identify finite dimensional approximations of the Koopman operator associated with the system dynamics (Schmid, 2010; M. O. Williams, Kevrekidis, and Rowley, 2015). The methods are easy to implement, mainly relying on least squares regression, and computationally and mathematically flexible, enabling numerous extensions and applications (S. L. Brunton and Kutz, 2019). For example, DMD-based methods have been successfully used in the field of fluid mechanics to capture low-dimensional structure in complex flows (Taira et al., 2017), in robotics for external perturbation force detection (Berger et al., 2015), and in neuroscience to identify dynamically relevant features in ECOG data (B. W. Brunton et al., 2016). Typically, EDMD-methods employ a dictionary of functions that are used to lift the state variables to a space where the dynamics are approximately linear. A practical nonlinear sparse regression method was proposed by S. L. Brunton, Proctor, et al., 2016. Based on a dictionary of nonlinear transformations of the state variables (e.g. monominal functions), a l_1-regularized regression problem over the nonlinear transformations can be posed resulting in the key functional forms needed for good prediction performance to be identified. This method is shown to be data efficient and achieve good prediction performance if a good library of transformations is known.

It is especially relevant for this thesis that Koopman-style modeling has previously been extended to *controlled* nonlinear systems (Kaiser, Kutz, and S. L. Brunton, 2018; Proctor, S. L. Brunton, and Kutz, 2018). This is particularly interesting as EDMD can be used to approximate nonlinear control systems by a lifted state space model. As a result, well-developed linear control design methods such as robust, adaptive, and model predictive control (MPC) can be utilized to design nonlinear controllers. Korda and Mezić, 2018b use EDMD to learn a high-dimensional linear model and linear MPC for control design . Furthermore, it is shown that the explicit relationship between the state and control input in the dynamics update equation enables the MPC to be implemented in a way that eliminates the state from the optimization problem, thereby removing the dependence on the lifting dimension. Consequently, the MPC can be implemented very efficiently. This is one of the key results that have spurred a lot of interest in Koopman-based learning and control and is indeed part of the inspiration for the methods presented in Chapters 3-4.

As discussed in the introduction, one of the big challenges with current Koopman methods is how to choose a good function dictionary to "lift" the state variables of the system. If not chosen carefully, the time evolution of the dictionary functions cannot be described by a linear combination of the other functions in the dictionary. This results in error accumulation when the model is used for prediction, potentially causing significant prediction performance degradation. Previous works have attempted to principally construct observables for certain model classes (e.g. (Korda and Mezić, 2018a)), but typically rely on assumptions that are problematic for robotic systems, e.g. that the data must be collected while the system is operating under open loop controls, which can lead to catastrophic system damage. This is addressed in Chapter 3, which introduces an approach for principled eigenfunction construction for lifted linear models that is usable for robotic systems. Alternatively, multiple researchers have proposed to parametrize the function dictionary using neural networks and learn the lifting functions alongside the dynamics model (Li et al., 2017; Lusch, Kutz, and S. L. Brunton, 2018). Although these methods have demonstrated promising results for certain dynamical systems, none have focused on algorithm development and the unique requirements of controlled robotic systems, such as how to design controllers using the learned models.

The second significant challenge of current methods is that many rely on model structures that cannot capture nonlinear control-affine dynamics, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}$, which are an important model class in robotics that allows a wide class of aerial and ground robots to be characterized. Most current methods approach this problem by learning a lifted *linear* model, which only allows the control vector fields, $\mathbf{g}(\mathbf{x})$, to take a constant state-invariant form. This is a significant limitation, as many useful robotic systems, e.g. systems where input forces enter the system dynamics through rotation matrices, are best described by nonlinear control-affine dynamics. Theoretically, the *Koopman canonical transform* (KCT) (Surana, 2016) allows a large class of nonlinear control-affine dynamic models to be *lifted* to a higher-dimensional space where the system evolution can be described by a *bilinear* (but possibly infinite dimensional) dynamical system. Learning and control design with bilinear models is sparsely explored, but connections between Koopman bilinear system descriptions and classical control concepts such as reachability and control Lyapunov functions have been presented (Goswami and Paley, 2018; Huang, Ma, and Vaidya, 2019). Very recently, bilinear Koopman models that are linearized at the current state of the system were used in MPC (Bruder, Fu, and Vasudevan, 2021). Another approach uses the bilinear model structure to simplify the construction of a control Lyapunov function enforced as a constraint in a nonlinear MPC method to obtain stability guarantees (Narasingam and Kwon, 2020). Chapter 4 develops a method for joint dictionary and model learning underpinned by the bilinear model structure that make these emerging ideas practically useful for robotic systems and model predictive control design.

Methods that Learn Partial Models Motivated by Controller Structure

Although learning the full state space model leads to a model that can be used for both prediction and control design for various tasks, increased data efficiency could be achieved by posing the learning problem in the setting of a specified objective. This motivates learning and control frameworks that exploit structure that is available when a dynamics model is combined with a specific control design approach to achieve some objective.

Several methods approach learning and control in the context of MPC. This is a natural approach as most robotic control problems can be formulated as a MPC problem. Aswani *et al.* formulate a MPC problem that employs a nominal linear model with additive disturbance capable of capturing unmodeled dynamics of a nonlinear system for robustness guarantees and non-parametric learning for performance improvement (Aswani et al., 2013). As such, the problem of robustness and performance is decoupled, and the model is also robust against mislearning.

However, not updating the belief of the disturbance as more data is collected can lead to overly conservative behavior. Rosolia, X. Zhang, and Borrelli, 2018 developed an episodic learning framework dubbed Learning MPC (LMPC) that overcomes this issue. Based on an initial experiment execution, an initial safe set, and known dynamics, the system iteratively executes an experiment, updates the safe set based on the states visited, and improves the control performance as more of the state space is deemed safe. The framework does not explicitly learn the system dynamics, but learns the safe set of the system and the effect of disturbances on the dynamics through a sampling-based method. MPC-motivated approaches also exist in the more traditional approximate dynamic programming methods of RL. For example, Amos et al., 2018 formulate a MPC problem such that it can be differentiated and thereby integrated in a neural network architecture and backpropagated on. This is shown to significantly improve the sample complexity and performance of the controller based on learned (MPC) cost and dynamics compared to model-free reinforcement learning methods.

Several methods are also based on control Lyapunov (CLFs) and control barrier functions (CBFs) to guide the learning and synthesize controllers (Ames and M. Powell, 2013). Controllers with stability and safety guarantees can be synthesized by including CLF and/or CBF constraints in a quadratic program. Because stability/safety can be evaluated by a scalar function, learning improved CLF/CBF derivatives can significantly reduce the learned model dimension aiding data efficiency and generalization. Taylor, Dorobantu, Le, et al., 2019 proposed to learn a reduced dimensional projection of the state in the form of Lyapunov function derivatives, which effectively project the dynamics to a scalar quantity that can be used to promote stability (the dynamics are implicitly described through the Lyapunov function derivative). The method is integrated in an episodic learning framework and subsequent works developed state-dependent uncertainty bounds and guarantees on *Projection to State Stability*, a novel concept similar to Input-to-State Stability for projected dynamics (Taylor, Dorobantu, Krishnamoorthy, et al., 2019).

Similar ideas have also been presented to learn control barrier function derivatives (Taylor, Singletary, et al., 2020) and simultaneous learning of both control Lyapunov and control barrier functions (Choi et al., 2020). Additionally, methods that can explicitly account for the model error in the CLF/CLF constraints based on Lipschitz continuity arguments (Taylor, Dorobantu, Dean, et al., 2020) and confidence bounds of GPR (Castaneda et al., 2021) allow model uncertainty to be accounted for in the theoretical guarantees. Finally, multiple approaches include CLF and/or CLF terms in the reward function of reinforcement learning approaches to promote stability and/or safety in learned policies. This has been shown to significantly reduce variance and speed up convergence as system knowledge can be encoded into the reward terms (Cohen and Belta, 2020; Zheng et al., 2020; Cheng et al., 2019).

As discussed above, methods that learn partial models motivated by controller structure have shown great performance for multiple simulated and physical environments. However, the improved performance and data efficiency comes at the cost of learning models/controllers that are task specific. Transferring a learned system between tasks can therefore be costly and time consuming.

Safety Filtering

A different philosophy to approach theoretical safety guarantees of dynamical systems is through various forms of safety filters similar to those created through *control barrier functions* (CBF) as introduced by Ames, Xu, et al., 2017. In this setting, full freedom is given to the design of high performance control policies, and the control actions suggested by these policies are executed as long as they don't violate the system safety specification. If safety is violated, a safety controller takes over when necessary and ensures safe operation of the system.

Many learning methods based safety filters use GPR for model and uncertainty estimates. L. Wang, Theodorou, and Egerstedt, 2018 utilize GPR and CBFs to endow general control policies with learned safety certificates. Their approach approximates the mean and variance of residual dynamics between observed data and a nominal control-affine dynamics model. Furthermore, safety guarantees based on samples from the system and an adaptive sampling algorithm are developed. Similarly, Turchetta, Berkenkamp, and Krause, 2019 and Berkenkamp et al., 2017 leverage a GP prior and observations of a function encoding safety through an oracle to learn a mean and variance estimate of the value of the safety function at unseen states. Then, smoothness assumptions on the dynamics are used to expand the set of safe actions from an initial (small) set of safe actions. A key feature of the method is that the safety filter only intervenes and/or expands if necessitated by the suggested control action of the controller, i.e. the safe set is only expanded in the directions relevant for high performance control, significantly improving sampling efficiency compared to other methods that tend to expand the safe operating region in all directions of the state space.

The GPR safety filter is extended with an online evaluation of the model reliability by Fisac, Akametalu, et al., 2019. Their key insight is that a safety controller should not only be utilized when the system is on the boundary of the control invariant set, but also if the prediction model used in control design is expected to be unreliable. This enables the system to retract to a safe region if, e.g., new uncertainties which are not captured by the model are introduced. The framework can also be modified to design a confidence-aware predictor of human behavior based on GPR to help robots navigate around humans (Fisac, Bajcsy, et al., 2018). Finally, safety filtering approaches have also used parametric learning methods such as neural networks (Richards, Berkenkamp, and Krause, 2018).

Safety filtering is a flexible tool to endow any control policy with safety guarantees. However, in the context of learning and control there are two main drawbacks to this approach. First, all methods reviewed only allow uncertainty to enter the model through the unactuated dynamics, limiting the unknowns that can be captured for many practical applications. Second, a high performance controller still needs to be designed and this controller should also take advantage of the collected data to improve its performance. As a result, treating performance and safety separately may lead to inefficient use of computational resources and unnecessary system complexity. In the work presented in Chapter 5, a computational and data efficient method to implement a safety filtering method based on partially learned dynamics to address this is presented. Furthermore, I discuss the potential to learn both a model for safety filtering and one for performance-oriented work in the future work section of Chapter 6.

1.2 Thesis Contribution and Organization

The main contributions of this thesis can be summarized as follows:

- I develop a learning framework to construct Koopman eigenfunctions for unknown, nonlinear dynamics using data gathered from experiments. The method exploits the learned Koopman eigenfunctions to learn a lifted linear state-space model. To the best of my knowledge, the method was the first to utilize Koopman eigenfunctions as lifting functions for EDMD-based methods. Furthermore, the learned model is utilized for linear MPC and demonstrated experimentally on a multirotor drone to learn aerodynamic ground effect. Furthermore, this is one of the first demonstrations of Koopman learning and control demonstrated on a physical robotic platform.

- I develop a method combining the process of learning control-affine dynamics with nonlinear MPC design. First, building on recent advances in NMPC, I design a controller for bilinear Koopman models that uses the bilinear model structure to improve computational efficiency, making real-time computation possible. Second, I show the advantages of learning lifted bilinear models over linear models and demonstrate that the completely data-driven Koopman NMPC method can match the performance of a NMPC controller with full a priori model knowledge on a simulated planar quadrotor.

- I extend the method to learn lifted bilinear models by encoding both the function dictionary and the bilinear lifted model in a single neural network architecture. This solves the problem of designing an appropriate function dictionary, which is typically difficult for real-world systems, and allows improved prediction performance and/or lower lifting dimension of the model compared to fixed function dictionaries. Reducing the lifting dimension is crucial to enable the NMPC based on the learned model to run in real time on physical robots. I demonstrate the learning and controller performance on a quadrotor drone. At medium altitude, the controller based on the learned model is able to match the performance of a NMPC based on a nominal nonlinear quadrotor model. At lower altitudes, where aerodynamic ground effect impacts the drone's dynamics, however, the controller based on the learned model maintains acceptable tracking performance whereas the nominal controller fails to achieve stable flight.

- I introduce a data-driven approach that combines Koopman-based learning and control barrier functions to achieve safety-critical control that guarantees safety under limited actuation and errors in the learned Koopman model. The method significantly reduces the online computation required allowing the safety filter to be used for high-dimensional systems. Furthermore, backup trajectories can be learned from data, improving the applicability of the approach in real-world scenarios where accurate models may be unavailable. Experiments and simulations show that the method can be incorporated in a decentralized framework for multi-agent control, further expanding the impact of the efficiency improvements.

Each of the methods in Chapters 3-5 includes demonstrations of the proposed methods on both simulated and laboratory experiments. Many of the experiments are carried out using quadrotor drones. These vehicles are well suited for such research, as they have sufficient complexity to study learning and control algorithms for robotic systems, such as bounded actuation, important nonlinear dynamical effects, and the ability to perform agile maneuvers that can lead to catastrophic failure, while being relatively inexpensive and easy to deploy. Furthermore, aerial vehicles are affected by aerodynamic effects that can be very hard to model from first principles using model classes that are practical for real-time control and therefore presents an attractive use case for learning. For example, all the chapters include examples where data is used to learn an unmodeled ground effect occuring when the vehicle flies close to the ground and the downwash from the propellers impact the motion of the vehicle. Even though the experimental focus of this thesis is on drones, all the methods are applicable to general robotic systems satisfying the assumptions described in each respective chapter.

The rest of the thesis is organized as follows. Chapter 2 introduces relevant background on Koopman operator theory and Koopman-based learning frameworks. Chapters 3 and 4 focus on simultaneous learning of function dictionaries and dynamics models based on Koopman theory. Chapter 3 is concerned with learning with models underpinned by lifted linear models and control using linear MPC. As a result, only linear actuated dynamics can be accurately captured. Chapter 4 develops algorithms to capture nonlinear control-affine actuated dynamics underpinned by lifted bilinear models and nonlinear MPC. Chapter 5 focuses on combining safety filtering with CBFs and Koopman-based learning resulting in a flexible, computationally efficient method to guarantee safety under any nominal controller. Finally, conclusions and future work are discussed in Chapter 6.

Chapter 2

LEARNING AND CONTROL FROM AN OPERATOR THEORETIC PERSPECTIVE

This chapter introduces the fundamental concepts of operator theoretic approaches to learning and control of dynamical systems. Section 2.1 discusses the basic theory of the Koopman operator for autonomous systems and Section 2.2 describes how the theory can be extended to controlled systems. Finally, Section 2.3 introduces the most widespread method to learn finite dimensional approximations of the Koopman operator, the extended dynamic mode decomposition, before practical control design with MPC is discussed in Section 2.4. These preliminaries serve as an introduction to the Koopman operator for dynamical systems and are the foundations of the research presented in Chapters 3-5.

2.1 Koopman Spectral Theory for Autonomous Dynamics

Koopman Operator of Continuous-time Dynamics

Consider the autonomous dynamical system:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \tag{2.1}$$

with state $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$ and $\mathbf{f}(\cdot)$ Lipschitz continuous on $\mathcal{X}$. The flow of this dynamical system is denoted by $S_t(\mathbf{x})$ and is defined as

$$\frac{d}{dt}S_t(\mathbf{x}) = \mathbf{f}(S_t(\mathbf{x})) \tag{2.2}$$

for all $\mathbf{x} \in \mathcal{X}$ and $t \geq 0$. The *Koopman operator semi-group* $(\mathcal{K}_t)_{t\geq 0}$, hereafter denoted as the *Koopman operator*, is defined as

$$\mathcal{K}_t \gamma = \gamma \circ S_t \tag{2.3}$$

for all $\gamma \in C(\mathcal{X})$, where $\circ$ denotes function composition. Each element of the Koopman operator maps continuous functions to continuous functions, $\mathcal{K}_t : C(\mathcal{X}) \to C(\mathcal{X})$. Crucially, each $\mathcal{K}_t$ is a *linear* operator that governs the evolution of scalar functions along trajectories of the associated nonlinear dynamical system.

Similar to matrix analysis using eigenvectors, Koopman eigenfunctions can be used to describe key characteristics of the underlying dynamical system. An *eigenfunction* of the Koopman operator associated to an eigenvalue $e^{\lambda} \in \mathbb{C}$ is any function $\varphi \in$

$C(\mathcal{X})$ that defines a coordinate evolving linearly along the flow of (2.1), satisfying

$$(\mathcal{K}_t\varphi)(\mathbf{x}) = \varphi(S_t(\mathbf{x})) = e^{\lambda t}\varphi(\mathbf{x}). \tag{2.4}$$

Furthermore, the infinitesimal generator of $\mathcal{K}_t$, $\lim_{t\to 0}\frac{\mathcal{K}_t - I}{t}$, can be shown to be $\mathbf{f}\cdot\nabla = L_{\mathbf{f}}$, where $L_{\mathbf{f}}$ is the Lie derivative with respect to $\mathbf{f}$ (Mauroy and Mezić, 2016). The infinitesimal generator satisfies the eigenvalue equation $L_{\mathbf{f}}\varphi = \lambda\varphi$. An infinite number of eigenfunctions of the Koopman operator can be constructed (Budišić, Mohr, and Mezić, 2012). In particular, if φ_1, φ_2 are eigenfunctions of $\mathcal{K}_t$ with eigenvalues λ_1, λ_2, respectively, then $\varphi_1^k\varphi_2^l$ is also an eigenfunction with eigenvalue $k\lambda_1 + l\lambda_2$, $k, l \in \mathbb{N}$ (Budišić, Mohr, and Mezić, 2012).

To gain more familiarity with the core concepts of the Koopman operator, Example 2.1 shows a constructed system that admits a finite dimensional Koopman operator. Very few real world systems allow a simple construction of observable functions resulting in a finite dimensional Koopman operator, but it is instructive to study the mechanics of the construction for a simple synthetic system regardless. In contrast, Example 2.2 shows an example of a system that does not allow a finite set of observables using a simple set of dictionary functions.

Example 2.1 (**System with finite dimensional Koopman operator**)**.** Certain systems have a structure that leads to a closed Koopman subspace if a correct set of observables is chosen. Consider the system

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}\mu x_1\\ \lambda(x_2 - x_1^2)\end{bmatrix}. \tag{2.5}$$

By choosing observables $\mathbf{y} = [x_1, x_2, x_1^2]^T$, (2.5) can be rewritten as an equivalent linear system

$$\begin{bmatrix}\dot{y}_1\\ \dot{y}_2\\ \dot{y}_3\end{bmatrix} = \underbrace{\begin{bmatrix}\mu & 0 & 0\\ 0 & \lambda & -\lambda\\ 0 & 0 & 2\mu\end{bmatrix}}_{\mathcal{K}}\begin{bmatrix}y_1\\ y_2\\ y_3\end{bmatrix}, \tag{2.6}$$

where $\mathcal{K}$ is the Koopman operator of the system. Note that since $\dot{y}_3 = \dot{\overline{x_1^2}} = 2x_1\dot{x}_1 = 2\mu x_1^2 = 2\mu y_3$, (x_1, x_2, x_1^2) form a closed subspace under differentiation resulting in a finite dimensional Koopman operator.

Example 2.2 (**System with infinite dimensional Koopman operator**)**.** Most systems do not have a structure that allows a finite dimensional Koopman operator. Consider instead the system

$$\dot{x} = x^2. \tag{2.7}$$

If we naively try to construct observables to obtain a Koopman operator for this system, a natural choice is to start with $\mathbf{y} = [x, x^2]^T$. However, when differentiating x^2, we get $\dot{\overline{x^2}} = 2x\dot{x} = 2x^3$. This will lead us to add $y_3 = x^3$, and for every observable we add, the differentiation will result in the power being increased by one and we get an infinite cascade of higher powers in order to fully describe the derivative of $\mathbf{y}$. I.e. an infinite number of observables is needed and the associated Koopman operator will be infinite dimensional.

This situation is common for many systems, and as a result, practical use of the Koopman operator relies on finding good finite dimensional approximations from data.

Discrete Time Dynamics

This thesis also utilizes discrete-time Koopman operators, which can be defined in a similar manner to the continous-time counterpart. Consider the discrete-time autonomous dynamical system

$$\mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k) \tag{2.8}$$

with state $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathbf{f}_d(\cdot)$ Lipschitz continuous on $\mathcal{X}$. Define a real-valued observable function $\gamma : \mathcal{X} \longrightarrow \mathbb{R}$. Then, the Koopman operator is defined as

$$\mathcal{K}\gamma = \gamma \circ \mathbf{f}_d \tag{2.9}$$

where $\circ$ denotes function composition such that $\mathcal{K}\gamma(\mathbf{x}_k) = \gamma(\mathbf{f}(\mathbf{x}_k)) = \gamma(\mathbf{x}_{k+1})$. Again, $\mathcal{K}$ is a *linear* operator. An *eigenfunction* of the discrete-time Koopman operator associated to an *eigenvalue* $\lambda \in \mathbb{C}$ is any function $\varphi : \mathcal{X} \longrightarrow \mathbb{C}$ that defines a coordinate evolving according to

$$\mathcal{K}\varphi = \lambda\varphi. \tag{2.10}$$

2.2 Theory and Application of Koopman Theory to Control Systems

Recently, extensions of the Koopman operator for control systems have been presented. If the control law is a state dependent feedback control law, $\mathbf{u}(\mathbf{x}) = \mathbf{q}(\mathbf{x})$, the Koopman operator of the controlled system under control reduces to the Koopman operator for the autonomous dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{q}(\mathbf{x}))$. Alternatively, if the dynamics are affected by a time-dependent external forcing, observables describing the forcing term can be defined and the Koopman operator needs to be defined such that both the time-dependent forcing and the observables associated with the dynamics are propagated by the operator (Proctor, S. L. Brunton, and Kutz, 2018).

More concretely, control-affine dynamics can be transformed to a *bilinear* form through the *Koopman canonical transform* (KCT) (Surana, 2016). Consider the control-affine dynamics

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \sum_{i=1}^{m} \mathbf{g}_i(\mathbf{x}) u_i, \tag{2.11}$$

where $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d, \mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^m$, and $\mathbf{f}, \mathbf{g}_i, i = 1, \dots, m$ are assumed to be Lipschitz continuous on $\mathcal{X}$. Let $\left(e^{\lambda_i}, \varphi_i(\mathbf{x})\right), i = 1, \dots, n$ be eigenvalue-eigenfunction pairs of the Koopman operator associated with the autonomous dynamics of (2.11), $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. The KCT relies on the assumption that the state vector can be described by a linear combination of a finite number of eigenfunctions, i.e. that $\mathbf{x} = \sum_{i=1}^{n} \varphi_i(\mathbf{x}) \mathbf{v}_i^{\mathbf{x}}$ for all $\mathbf{x} \in \mathcal{X}$, and where $\mathbf{v}_i^{\mathbf{x}} \in \mathbb{C}^d$. This is likely to hold if n is large. If not, the state may be well approximated by n eigenfunctions.

When $\varphi_i : \mathcal{X} \to \mathbb{R}$, the KCT is defined as (see Goswami and Paley, 2018 for the case when $\varphi_i : \mathcal{X} \to \mathbb{C}$)

$$\mathbf{x} = C^{\mathbf{x}} \mathbf{z}, \quad \dot{\mathbf{z}} = F\mathbf{z} + \sum_{i=1}^{m} L_{\mathbf{g}_i} T(\mathbf{x}) u_i, \tag{2.12}$$

where $\mathbf{z} = T(\mathbf{x}) = [\varphi_1(\mathbf{x}) \dots \varphi_n(\mathbf{x})]^T$, $C^{\mathbf{x}} = [\mathbf{v}_1^{\mathbf{x}} \dots \mathbf{v}_n^{\mathbf{x}}]$, and $F \in \mathbb{R}^{n \times n}$ is a diagonal matrix with entries $F_{i,i} = \lambda_i$.

Under certain conditions, the system (2.12) is bilinearizable in a countable, possibly infinite basis. The following theorem restates the conditions for the existence of a bilinear form in a *finite* basis, as this is of practical interest for the work in this thesis.

Theorem 2.3. *(Goswami and Paley, 2018) Suppose there exist Koopman eigenfunctions $\varphi_j, j = 1, \dots, n, n \in \mathbb{N}, n < \infty$ of the autonomous dynamics (2.11) whose span, $span(\varphi_1, \dots, \varphi_n)$, forms an invariant subspace of $L_{\mathbf{g}_i}, i = 1, \dots, m$. Then, the system (2.11), and in turn system (2.12), are bilinearizable with an n-dimensional state space.*

Although the conditions of Theorem 2.3 may be hard to satisfy in a given problem, an approximation of the true system (2.11) can be obtained with sufficiently small approximation error by including adequately many eigenfunctions in the basis. As a result, $L_{\mathbf{g}_i} = G_i$ and the system can be expressed as the *Koopman bilinear form* (KBF) (see Goswami and Paley, 2018 for details):

$$\dot{\mathbf{z}} = F\mathbf{z} + \sum_{i=1}^{m} G_i \mathbf{z} u_i, \quad \mathbf{z} \in \mathbb{R}^n, n < \infty. \tag{2.13}$$

In practice, most works on the use of Koopman models in control use a lifted linear model structure instead of a bilinear one when approximating Koopman operators from data (cf. (Korda and Mezić, 2018b; Kaiser, Kutz, and S. L. Brunton, 2018; Bruder, Gillespie, et al., 2019)). This is motivated by the abundance of control and analysis tools available for linear systems, and this simplification typically achieves good performance if the underlying dynamics have linear actuated dynamics ($\sum_{i=1}^{m} \mathbf{g}_i(\mathbf{x})u_i = B\mathbf{u}$), with B a constant matrix. Learning with extended dynamic mode decomposition and control with model predictive control for lifted linear models are introduced in the next sections. Methods that better capture nonlinear actuation effects is a central focus of this thesis and will be explored in depth using the KBF in Chapter 4.

To gain more familiarity with the basic mechanics of the Koopman bilinear form, Example 2.4 shows a modification of the constructed system of Example 2.1 that admits a finite dimensional Koopman bilinear form. As before, very few real world systems allow a simple construction of observable functions resulting in a finite dimensional KBF, but it is instructive to study the construction for a simple synthetic system to understand why reformulation to the KBF could be possible.

Example 2.4 (**System with exact Koopman bilinear form**)**.** Consider a modified version of the system introduced in Example 2.1, where two control inputs, u_1, u_2, are added with nonlinear actuation effects resulting in the control-affine dynamics:

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix} x_3 \\ x_4 \\ \lambda x_3 \\ \mu x_4 + (2\lambda - \mu) c x_3^2 \end{bmatrix}}_{\mathbf{f}_0} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{\mathbf{g}_1} u_1 + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ x_1 + 1 \end{bmatrix}}_{\mathbf{g}_2} u_2. \tag{2.14}$$

As a result of the careful construction of this system, there exists a Koopman canonical transform, $\mathbf{z} = T(\mathbf{x})$ that exactly transforms the control-affine dynamics into a bilinear system. Consider the transformation:

$$T(\mathbf{x}) = \begin{bmatrix}\phi_1(\mathbf{x})\\ \phi_2(\mathbf{x})\\ \phi_3(\mathbf{x})\\ \phi_4(\mathbf{x})\\ \phi_5(\mathbf{x})\\ \phi_6(\mathbf{x})\end{bmatrix} = \begin{bmatrix}1\\ x_1 - \frac{1}{\lambda}x_3\\ x_2 - \frac{1}{\mu}x_4 + \frac{(2\lambda-\mu)c}{2\lambda\mu}x_3^2\\ x_3\\ x_4 - cx_3^2\\ x_3^2\end{bmatrix}, \tag{2.15}$$

where $\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6$ are eigenfunctions of the Koopman operator associated with the drift vector field, $\mathbf{f}_0$. The matrix with the eigenvalue associated with the i-th eigenfunction on the i-th diagonal element is $F = \text{diag}(0, 0, \lambda, \mu, 2\lambda, 0)$. Then, to reformulate the dynamics, we have:

$$L_{\mathbf{g}_1}T(\mathbf{x}) = \begin{bmatrix}0\\ -\frac{1}{\lambda}\\ \frac{(2\lambda-\mu)c}{\lambda\mu}x_3\\ 1\\ -2cx_3\\ 2x_3\end{bmatrix}, \qquad L_{\mathbf{g}_2}T(\mathbf{x}) = \begin{bmatrix}0\\ 0\\ -\frac{1}{\mu}(x_1+1)\\ 0\\ x_1+1\\ 0\end{bmatrix}, \tag{2.16}$$

which can equivalently be expressed as a Koopman bilinear form (2.12) with $\mathbf{z} = T(\mathbf{x})$, $F = \text{diag}(0, 0, \lambda, \mu, 2\lambda, 0)$ and actuation matrices:

$$G_1 = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0\\ -\frac{1}{\lambda} & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & \frac{(2\lambda-\mu)c}{\lambda\mu} & 0 & 0\\ 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & -2c & 0 & 0\\ 0 & 0 & 0 & 2 & 0 & 0\end{bmatrix}, \quad G_2 = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ -\frac{1}{\mu} & -\frac{1}{\mu} & 0 & -\frac{1}{\lambda\mu} & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 1 & 1 & 0 & \frac{1}{\lambda} & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}. \tag{2.17}$$

2.3 Learning High Dimensional Linear Models to Approximate Nonlinear Dynamics

All EDMD-based learning frameworks essentially follow the same, simple concept. Consider that we have an unknown system of the form (2.11) and that our goal is to learn an estimated model for the system from data. For simplicity, I assume that we have access to state measurements in this exposition but the exact same method can be used to build a model for some output based on a set of measurements. With some nominal control signal, collect a data set of state, state derivative, and actuation snapshots $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T], \dot{X} = [\dot{\mathbf{x}}_1, \dot{\mathbf{x}}_2, \dots, \dot{\mathbf{x}}_T], U = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_T]$ where each $\mathbf{x}_t$, $\dot{\mathbf{x}}_t$ and $\mathbf{u}_t$ are the vectors of state measurements, state derivatives and control inputs at time t, respectively. If the state derivatives cannot be directly measured they can be estimated with numerical differentiation techniques. First, to learn a *linear* state space model, dynamic mode decomposition with control can be employed (Proctor, S. L. Brunton, and Kutz, 2016). In this case, we seek to learn state space matrices A and B by minimizing the least squares problem

$$\min_{A\in\mathbb{R}^{d\times d}, B\in\mathbb{R}^{d\times m}} ||AX + BU - \dot{X}|| \qquad \rightarrow \qquad \dot{\mathbf{x}} \approx A\mathbf{x} + B\mathbf{u} \tag{2.18}$$

where d is the state dimension and m is the number of control inputs. Then, to learn a high dimensional model to approximate the *nonlinear* dynamics, the state can be *extended* by defining a dictionary of n nonlinear transformations $\Phi(\mathbf{x}) = [\phi^{(1)}(\mathbf{x}), \phi^{(2)}(\mathbf{x}), \dots, \phi^{(n)}(\mathbf{x})]^T$ where each $\phi^{(i)}(x) : \mathbb{R}^d \rightarrow \mathbb{R}$ (Korda and Mezić, 2018b). Then, when all the state samples are transformed by the dictionary functions, the lifted state snapshots $Z = \Phi(X)$ are formed and a lifted dimensional model is obtained with EDMD by solving the following regression problems

$$\begin{aligned} \min_{A\in\mathbb{R}^{n\times n}, B\in\mathbb{R}^{n\times m}} ||AZ + BU - \dot{Z}|| \qquad &\rightarrow \qquad \dot{\mathbf{z}} \approx A\mathbf{z} + B\mathbf{u} \\ \min_{C^{\mathbf{x}}\in\mathbb{R}^{d\times n}} ||C^{\mathbf{x}}Z - X|| \qquad &\rightarrow \qquad \mathbf{x} \approx C^{\mathbf{x}}\mathbf{z} \end{aligned} \tag{2.19}$$

where $\dot{Z}$ can be calculated analytically or by numerical differentiation, and $C^{\mathbf{x}}$ is the *projection matrix*. The above regression problems can be solved analytically or by off-the-shelf solvers and result in the linear state/lifted state space models on the right-hand-side of Equations 2.18-2.19.

EDMD can similarly be defined to learn a discrete-time model. Instead of calculating the state derivative, a matrix of the state snapshots shifted one time step is constructed to get

$$\begin{aligned} X &= [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{T-1}], \\ X' &= [\mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_T], \\ U &= [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{T-1}]. \end{aligned} \tag{2.20}$$

Then, defining the lifted state snapshots, $Z = \Phi(X)$, as before, and the time shifted lifted state snapshots $Z' = \Phi(X')$, the following regression problem can be solved:

$$\begin{aligned} \min_{A_d\in\mathbb{R}^{n\times n}, B_d\in\mathbb{R}^{n\times m}} ||A_dZ + B_dU - Z'|| \qquad &\rightarrow \qquad \mathbf{z}_{k+1} = A_d\mathbf{z}_k + B_d\mathbf{u}_k \\ \min_{C^{\mathbf{x}}\in\mathbb{R}^{d\times n}} ||C^{\mathbf{x}}Z - X|| \qquad &\rightarrow \qquad \mathbf{x}_k = C^{\mathbf{x}}\mathbf{z}_k. \end{aligned} \tag{2.21}$$

One of the key challenges with EDMD methods is how to choose the number and type of nonlinear transformations in Φ. One possibility is to use knowledge of the system to pick suitable dictionary functions. This is known as *feature engineering* and can be successful if sufficient information about the system is available. However, to be able to predict the evolution of the chosen features in a linear fashion, i.e. as a linear combination of the other features in the dictionary, is not guaranteed which can lead to significant prediction performance degradation. Another approach that typically suffers from the same shortcomings is to choose a generic function class such as monomials of the state variables up to a certain order or radial basis functions. This method can work for certain problems, but has little rigorous motivation. A third approach, that has motivated my research, is to construct Koopman eigenfunctions and use them as nonlinear transformations. Importantly, these eigenfunctions are guaranteed to evolve linearly and are rigorously motivated by the Koopman operator. How to apply these ideas to improve learning is explored in Chapters 3 and 4.

2.4 Koopman Model Predictive Control

As discussed in Chapter 1, one of the results that has created significant interest in Koopman techniques for controlled dynamical systems is how approximated Koopman operators can be used to design effective controllers that leverage nonlinear model information with linear control design tools. For example, the Koopman operator can be used to transform the nonlinear optimization problem of MPC with nonlinear dynamics into an efficient quadratic program (QP) that is solved at each time step (Korda and Mezić, 2018b). This is achieved by formulating a quadratic objective function in the states and controls and assume that the state and control constraints are non-empty polytopes $\mathcal{X}, \mathcal{U}$. Then, the following optimization problem can be implemented:

$$\begin{aligned}
\min_{Z,U} \sum_{k=1}^{T-1} & \left[(C^{\mathbf{x}}\mathbf{z}_k - \boldsymbol{\tau}_k)^T Q (C^{\mathbf{x}}\mathbf{z}_k - \boldsymbol{\tau}_k) + \mathbf{u}_k^T R \mathbf{u}_k \right] + (C^{\mathbf{x}}\mathbf{z}_T - \boldsymbol{\tau}_T)^T Q_T (C^{\mathbf{x}}\mathbf{z}_T - \boldsymbol{\tau}_T), \\
& \mathbf{z}_{k+1} = A\mathbf{z}_k + B\mathbf{u}_k, \qquad k = 0, \dots, T-1, \\
& C^{\mathbf{x}}\mathbf{z}_k \in \mathcal{X}, \qquad k = 1, \dots, T, \\
& \mathbf{u}_k \in \mathcal{U}, \qquad k = 0, \dots, T-1, \\
& \mathbf{z}_0 = \Phi(\mathbf{x}_0),
\end{aligned} \tag{2.22}$$

where $Q, Q_N \in \mathbb{R}^{d\times d}$ and $R \in \mathbb{R}^{m\times m}$ are positive semidefinite cost matrices, $\boldsymbol{\tau} \in \mathbb{R}^{d\times T}$ is the reference trajectory, T is the number of timesteps to predict, $A_d \in \mathbb{R}^{n\times n}$ and $B_d \in \mathbb{R}^{n\times m}$ are the discrete time matrices (2.21), $C^{\mathbf{x}} \in \mathbb{R}^{d\times n}$ is the projection matrix, and $\Phi \in \mathbb{R}^n$ are the observable functions.

To speed up the solution of the optimization program, the dependency on the lifting dimension n in Eq. (2.22), the state can be eliminated via its explicit relation with the control input. This formulation is referred to as the *dense* form MPC (Mauroy, Mezic, and Susuki, 2020). This step greatly reduces the number of optimization variables, enabling real-time implementations regardless of the lifting dimension.

Chapter 3

LEARNING AND CONTROL OF SYSTEMS WITH LINEAR ACTUATED DYNAMICS USING LEARNED KOOPMAN EIGENFUNCTIONS

This chapter describes a systematic method to construct Koopman eigenfunctions from data and learn lifted linear models for systems with unknown dynamics. The contents of the chapter were previously published in two papers. The first paper was presented at the 2020 American Control Conference (ACC) (Folkestad, Pastor, Mezic, et al., 2020) and introduced the eigenfunction construction and model learning method. The second paper, presented at the 2020 International Conference on Robotic and Automation (ICRA) (Folkestad, Pastor, and Burdick, 2020), extended the basic method to an episodic learning framework that could better capture nonlinear actuation effects, and was one the early works to demonstrate Koopman-based learning and control on a physical robot.

3.1 Introduction

A key step in developing a high performance robotic application is the modeling of the robot's mechanics. Standard modelling and identification require extensive knowledge of the system and laborious system identification procedures (Lupashin et al., 2014). Moreover, although methods to show stability and safety of nonlinear systems exist (Khalil, 2002; Ames, Xu, et al., 2017), the process of control design that incorporates state and control limitations remains challenging.

The Dynamic Mode Decomposition (DMD) and extended DMD (EDMD) methods have emerged as efficient numerical algorithms to identify finite dimensional approximations of the Koopman operator associated with the system dynamics (Schmid, 2010; M. O. Williams, Kevrekidis, and Rowley, 2015). However, EDMD-methods typically employ a dictionary of functions used to lift the state variables to a space where the dynamics are approximately linear. However, if not chosen carefully, the time evolution of the dictionary functions cannot be described by a linear combination of the other functions in the dictionary. This results in error accumulation when the model is used for prediction, potentially causing significant prediction performance degradation. To mitigate this problem, I develop a learning framework that can extract spectral information from the full nonlinear dynamics by learning the eigenvalues and eigenfunctions of the associated Koopman operator. Limited attention has been given to constructing eigenfunctions from data. Sparse identification techniques have been used to identify approximate eigenfunctions (Kaiser, Kutz, and S. L. Brunton, 2021) but rely on defining an appropriate candidate function library. Other previous methods (e.g., (Korda and Mezić, 2018a)) depend upon assumptions that are problematical for robotic systems: the ID data is gathered while the robot operates under open loop controls, which can lead to catastrophic system damage.

This chapter presents a novel learning framework, *Koopman Eigenfunction Extended Dynamic Mode Decomposition* (KEEDMD), to construct Koopman eigenfunctions for unknown, nonlinear dynamics using a data gathered from experiments. The learned Koopman eigenfunctions are then exploited to learn a lifted linear state-space model and perform control design using model predictive control (MPC) (Mayne et al., 2000), thereby allowing control and state constraints to be satisfied during the learning process. After the details of the method are presented in a supervised learning setting, I extend the method to learn a model and control the system episodically. This is done by first allowing the method to gather data while the system operates under any *nonlinear* stabilizing controller. This enables input vector field nonlinearities to be captured, unlike prior Koopman-based model ID approaches. Second, I introduce an episodic learning procedure, by considering the closed-loop dynamics obtained with a nonlinear controller as the autonomous dynamics for the next episode. This feature increases sample efficiency (i.e., fewer learning trials) for improving specific tasks, and enables nonlinear actuation effects, which are important in robotics, to be captured in the Koopman eigenfunctions.

The rest of the chapter is organized as follows. Section 3.2 reviews relevant facts about Koopman eigenfunction construction for nonlinear systems. Then, Sections 3.3 and 3.4 describe eigenfunction and model learning for the KEEDMD, respectively, and Section 3.5 describes MPC design before simulated results on a inverted pendulum are considered in Section 3.6. Section 3.7 introduces the episodic learning framework before the details of the episodic KEEDMD are described in Section 3.8. Finally, Section 3.9 presents experimental results demonstrating that our method can be used to episodically learn the ground effect of a quadrotor drone to improve landing velocity before the chapter is concluded in Section 3.10.

3.2 Preliminaries on Construction of Eigenfunctions for Nonlinear Dynamics

For any sufficiently smooth autonomous dynamical system that is asymptotically stable to a fixed point, Koopman eigenfunctions can be constructed by first finding the eigenfunctions of the system linearization around the fixed point and then composing them with a diffeomorphism (Mohr and Mezić, 2014). To see this, consider asymptotically stable dynamics of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}). \tag{3.1}$$

The linearization of the dynamics around the origin is

$$\dot{\mathbf{y}} = \mathbf{Df}(0)\mathbf{y} = \hat{A}\mathbf{y}, \ \mathbf{y} \in \mathcal{Y}. \tag{3.2}$$

The following proposition describes how to construct eigenfunction-eigenvalue pairs for the linearized system (3.2).

Proposition 3.1. *Let $\hat{A}_1$ denote the linearization (3.2) of the nonlinear system (3.1) with $\mathcal{Y}$ scaled into the unit hypercube, $\mathcal{Y}_1 \subset \mathcal{Q}_1$, and let $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ be a basis of the eigenvectors of $\hat{A}_1$ corresponding to nonzero eigenvalues $\{\lambda_1, \ldots, \lambda_d\}$. Let $\{\mathbf{w}_1, \ldots, \mathbf{w}_d\}$ be the adjoint basis to $\{\mathbf{v}_1, \ldots, \mathbf{v}_d\}$ such that $\langle \mathbf{v}_j, \mathbf{w}_k \rangle = \delta_{jk}$, where δ_{jk} is the Kronecker delta, and $\mathbf{w}_j$ is an eigenvector of $\hat{A}_1^*$ at eigenvalue $\bar{\lambda}_j$. Then, the linear functional*

$$\psi_j(\mathbf{y}) = \langle \mathbf{y}, \mathbf{w}_j \rangle \tag{3.3}$$

is a nonzero eigenfunction of $\mathcal{K}_{\hat{A}_1}$, the Koopman operator associated to $\hat{A}_1$. Furthermore, for any tuple $(m_1, \ldots, m_d) \in \mathbb{N}_0^d$

$$\left(\prod_{j=1}^{d} e^{m_j \lambda_j}, \prod_{j=1}^{d} \psi_j^{m_j} \right) \tag{3.4}$$

is an eigenpair of the Koopman operator $\mathcal{K}_{\hat{A}_1}$.

Proof. A less formal description of the results in the proposition and associated proofs are described in (Mohr and Mezić, 2014), Example 4.6. By utilizing inner-product properties, ψ_j is an eigenfunction of $\mathcal{K}_{\hat{A}}$ as described in Chapter 2 since

$$\begin{aligned}(\mathcal{K}_t \psi_j)(\mathbf{y}) &= \mathcal{K}_t \langle \mathbf{y}, \mathbf{w}_j \rangle = \langle \mathbf{y}, \mathcal{K}_t^* \mathbf{w}_j \rangle = \langle \mathbf{y}, e^{\bar{\lambda}_j} \mathbf{w}_j \rangle \\ &= e^{\lambda_j} \langle \mathbf{y}, \mathbf{w}_j \rangle = e^{\lambda_j} \psi_j(\mathbf{y}).\end{aligned}$$

By scaling the state-space such that $\mathcal{Y}_1 \subset Q_1$, the linear eigenfunctions (3.3) form a vector space on $\mathcal{Y}_1$ that is closed under point-wise products. The construction of arbitrarily many eigenpairs (3.4) therefore follows from the semi-group property of eigenfunctions (see (Budišić, Mohr, and Mezić, 2012), Prop. 5). □

In the following, the linear functionals (3.3) are denoted as *principal eigenfunctions*. The eigenfunctions for the Koopman operator associated with the linearized dynamics can be used to construct eigenfunctions that are associated with the Koopman operator of the nonlinear dynamics through the use of a *conjugacy map*, as described in the following proposition.

Proposition 3.2. *(Budišić, Mohr, and Mezić, 2012) Assume that the nonlinear system (3.1) is topologically conjugate to the linearized system (3.2) via the diffeomorphism $h : \mathcal{X} \to \mathcal{Y}$. Let $B \in \mathcal{X}$ be a simply connected, bounded, positively invariant open set in $\mathcal{X}$ such that $h(B) \subset Q_r \subset \mathcal{Y}$, where Q_r is a cube in $\mathcal{Y}$. Scaling Q_r to the unit cube Q_1 via the smooth diffeomorphism $g : Q_r \to Q_1$ gives $(g \circ h)(B) \subset Q_1$. Then, if ψ is an eigenfunction for $\mathcal{K}_{\hat{A}_1}$ at e^{λ}, then $\psi \circ g \circ h$ is an eigenfunction for $\mathcal{K}_{\mathbf{f}}$ at eigenvalue e^{λ}, where $\mathcal{K}_{\mathbf{f}}$ is the Koopman operator associated with the nonlinear dynamics (3.1).*

The following extension of the Hartman-Grobman theorem guarantees the existence of the diffeomorphism, h described in Proposition 3.2, between the linearized and nonlinear systems in the entire basin of attraction of a fixed point, for sufficiently smooth dynamics.

Theorem 3.3. *(Lan and Mezić, 2013) Consider the system (3.1) with $\mathbf{f}(\mathbf{x}) \in C^2(\mathcal{X})$. Assume that matrix $A \in \mathbb{R}^{d\times d}$ is Hurwitz, i.e., all of its eigenvalues have negative real parts. So, the fixed point $\mathbf{x} = \mathbf{0}$ is exponentially stable and let Ω be its basin of attraction. Then $\exists h(\mathbf{x}) \in C^1(\Omega) : \Omega \to \mathbb{R}^d$, such that*

$$\mathbf{y} = c(\mathbf{x}) = \mathbf{x} + h(\mathbf{x}) \tag{3.5}$$

is a C^1 diffeomorphism with $\mathbf{D}c(\mathbf{0}) = I$ in Ω and satisfies $\dot{\mathbf{y}} = A\mathbf{y}$.

Example 3.4 (Eigenfunctions for system with finite dimensional Koopman operator). This example demonstrates how the theory presented in Section 3.2 can be used to construct eigenfunctions when the system dynamics are known and the diffeomorphism can analytically constructed as described in Theorem 3. Recall the system with a finite dimensional Koopman operator introduced in Chapter 2

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}\mu x_1\\ \lambda(x_2 - x_1^2)\end{bmatrix} \tag{3.6}$$

which has a finite dimensional Koopman operator. I first show how to construct three eigenfunctions that completely describe the evolution of the system by utilizing the Koopman modes associated with each eigenfunction (Budišić, Mohr, and Mezić, 2012). Then, I demonstrate how to arrive at the same eigenfunctions through the use of the diffeomorphism. This underpins our data-driven approach described in Section 3.3, using data to approximate the conjugacy map when the dynamics are unknown and/or a exact diffeomorphism cannot be derived.

Calculating Eigenfunctions from the Koopman Operator: By choosing observables $y = [x_1, x_2, x_1^2]^T$, (3.6) can be rewritten as an equivalent linear system

$$\begin{bmatrix}\dot{y}_1\\ \dot{y}_2\\ \dot{y}_3\end{bmatrix} = \underbrace{\begin{bmatrix}\mu & 0 & 0\\ 0 & \lambda & -\lambda\\ 0 & 0 & 2\mu\end{bmatrix}}_{\mathcal{K}} \begin{bmatrix}y_1\\ y_2\\ y_3\end{bmatrix} \tag{3.7}$$

where $\mathcal{K}$ is the Koopman operator of the system. From this, three Koopman eigenfunctions of (3.6) can be constructed. Let $\{\mathbf{v}_i\}_{i=1}^3$ be the eigenvectors of $\mathcal{K}$ and let $\{\mathbf{w}_i\}_{i=1}^3$ be the adjoint basis to $\{\mathbf{v}_i\}_{i=1}^3$ scaled such that $\langle \mathbf{w}_i, \mathbf{v}_j\rangle = \delta_{ij}$. Then, three eigenfunctions of the system are

$$\begin{aligned}\psi_1(\mathbf{y}) &= \langle \mathbf{y}, \mathbf{w}_1\rangle = y_1 = x_1\\ \psi_2(\mathbf{y}) &= \langle \mathbf{y}, \mathbf{w}_2\rangle = y_3 = x_1^2\\ \psi_3(\mathbf{y}) &= \langle \mathbf{y}, \mathbf{w}_3\rangle = y_2 + \frac{\lambda}{\lambda - 2\mu} y_3 = x_2 + \frac{\lambda}{\lambda - 2\mu} x_1^2.\end{aligned} \tag{3.8}$$

Calculating Eigenfunctions Based on the Diffeomorphism: I now show how the calculated eigenfunctions can be obtained through the diffeomorphism between the linearized and nonlinear dynamics. The linearization of the dynamics (3.6) around the origin is

$$\begin{bmatrix}\dot{\hat{x}}_1\\ \dot{\hat{x}}_2\end{bmatrix} = \underbrace{\begin{bmatrix}\mu & 0\\ 0 & \lambda\end{bmatrix}}_{A} \begin{bmatrix}\hat{x}_1\\ \hat{x}_2\end{bmatrix} \tag{3.9}$$

and principal eigenfunctions for the linearized system can be constructed, $\hat{\psi}_1(\mathbf{x}) = \langle \hat{\mathbf{w}}_1, \mathbf{x}\rangle = x_1$, $\hat{\psi}_2(\mathbf{x}) = \langle \hat{\mathbf{w}}_2, \mathbf{x}\rangle = x_2$, where $\hat{\mathbf{w}}_1, \hat{\mathbf{w}}_2$ are the eigenvectors of the adjoint of A. As described in Proposition 1, arbitrarily many eigenfunctions for the linearized system can be constructed by taking powers and products of the principal eigenfunctions, i.e. $\hat{\psi}_i(\mathbf{x}) = \hat{\psi}_1^{m_i^{(1)}}(\mathbf{x})\hat{\psi}_2^{m_i^{(2)}}(\mathbf{x}) = x_1^{m_i^{(1)}} x_2^{m_i^{(2)}}$ is an eigenfunction of the linearized system.

To get the eigenfunctions for the nonlinear system, it can be shown that

$$c(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\lambda}{\lambda-2\mu} x_1^2 \end{bmatrix} \tag{3.10}$$

is a diffeomorphism of the form described in Theorem 3. Then, ignoring the scaling function $g(\mathbf{x})$ for simplicity of exposition, the following eigenfunctions for the nonlinear dynamics are obtained

$$\begin{aligned} \phi_1(\mathbf{x}) &= \hat{\psi}_1(c(\mathbf{x})) = x_1 \\ \phi_2(\mathbf{x}) &= \hat{\psi}_2(c(\mathbf{x})) = x_2 + \frac{\lambda}{\lambda - 2\mu} x_1^2 \\ \phi_i(\mathbf{x}) &= \hat{\psi}_i(c(\mathbf{x})) = x_1^{m_i^{(1)}} \left(x_2 + \frac{\lambda}{\lambda - 2\mu} x_1^2 \right)^{m_i^{(2)}}, \ i = 3, \dots \end{aligned} \tag{3.11}$$

and by setting $m_3 = (2, 0)$, the analytic eigenfunctions of Equation (3.8) are recovered.

3.3 Data-driven Koopman Eigenfunctions for Unknown Nonlinear Dynamics

I now develop the data-driven approach to learn the diffeomorphism $h(x)$ described in Proposition 3.2 and Equation 3.5, resulting in a methodology for constructing Koopman eigenfunctions from data.

Modeling Assumptions

Consider the dynamical system

$$\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}) + B\mathbf{u} \tag{3.12}$$

where $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d,\ \mathbf{a}(\mathbf{x}) : \mathcal{X} \to \mathcal{X},\ \mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m,\ B \in \mathbb{R}^{d \times m}$, and where $\mathbf{a}(\mathbf{x})$ and B are unknown. Assume that we have access to a nominal linear model

$$\dot{\mathbf{x}} = A_{nom}\mathbf{x} + B_{nom}\mathbf{u} \tag{3.13}$$

where $\mathbf{x} \in \Omega \subset \mathcal{X} \subset \mathbb{R}^d,\ A_{nom} \in \mathbb{R}^{d \times d},\ B_{nom} \in \mathbb{R}^{d \times m},\ \mathbf{u} \in \mathcal{U}$ and an associated nominal linear feedback controller $\mathbf{u}^{nom} = K_{nom}\mathbf{x}$ that stabilizes the system (3.12) to the origin in a region of attraction Ω around the origin. The nominal model (3.13) can for example be obtained from first principles modeling or from parameter identification techniques and linearization of the constructed model around the fixed point if needed.

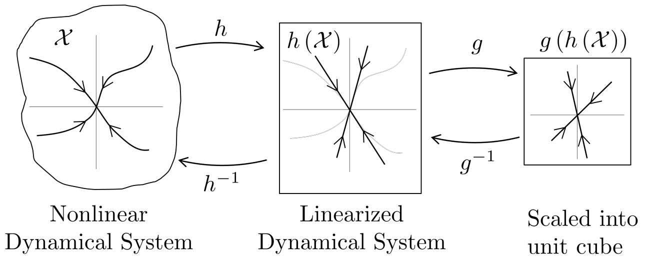


Figure 3.1: Chain of topological conjugacies used to construct eigenfunctions, adapted from (Mohr and Mezić, 2014).

Constructing Eigenfunctions from Data

Algorithm 1 constructs Koopman eigenfunctions from data, based on the foundations introduced in Section 3.2. M_t trajectories of fixed length T are executed from initial conditions $\mathbf{x}_0^j \in \Omega\ j = 1, \dots, M_t$, and are guided by the nominal control law $\mathbf{u}^{nom}$. The system's states and control actions are sampled at a fixed interval Δt, resulting in a data set

$$\mathcal{D} = \left(\left(\mathbf{x}_k^j, \mathbf{u}_k^j\right)_{k=0}^{M_s}\right)_{j=1}^{M_t} \tag{3.14}$$

where $M_s = T/\Delta t$. Variable length trajectories and sampling rates can be implemented with minor modifications.

Under the nominal control law, Koopman eigenfunctions for the nominal linearized model (3.13) can be constructed as in Proposition 3.1 using the eigenvectors and eigenvalues of the closed loop dynamics matrix $A_{cl} = A_{nom} + B_{nom}K_{nom}$. I.e. let Q_r be a hypercube centered at the origin with sides $2r$ such that $\mathcal{X} \subset Q_r$, a scaling function $g : Q_r \to Q_1$ can then be constructed (by scaling each coordinate) to get the scaled dynamics matrix $A_{cl,1}$. Furthermore, let $\{\mathbf{v_j}\}_{j=1}^d$ be a basis of eigenvectors of

Algorithm 1 Data-driven Koopman Eigenpair Construction

```
1: Input: Data set $\mathcal{D} = ((\mathbf{x}_k^j, \mathbf{u}_k^j)_{k=0}^{M_s})_{j=1}^{M_t}$, nominal model matrices $A_{nom}$, $B_{nom}$, nominal control gains $K_{nom}$, number of lifting functions $n$, $n$ power combinations $(m_1^{(i)}, \dots, m_d^{(i)}) \in \mathbb{N}_0^d, i = 1, \dots, n$
2: Construct principal eigenpairs for the linearized dynamics: $(\lambda_j, \psi_j(\mathbf{y})) \leftarrow (\lambda_j, \langle \mathbf{y}, \mathbf{w}_j \rangle),\ j = 1, \dots, n$
3: Construct $n$ eigenpairs: $(\tilde{\lambda}_i, \tilde{\psi}_i) \leftarrow \left(\prod_{j=1}^d e^{m_j^{(i)} \lambda_j}, \prod_{j=1}^d \psi_j^{m_j^{(i)}}\right), i = 1, \dots, n$
4: Fit diffeomorphism estimator: $h(\mathbf{y}) \leftarrow \text{ERM}(\mathcal{H}_h, \mathcal{L}_h, \mathcal{D})$
5: Construct scaling function: $g(\mathbf{y}) \leftarrow g : Q_r \to Q_1$
6: Construct $n$ eigenpairs for the nonlinear dynamics: $(\tilde{\lambda}_i, \phi_i) \leftarrow (\tilde{\lambda}_i, \tilde{\psi}_i(g(h(\mathbf{y})))),\quad i = 1, \dots, n$
7: Output: $\Lambda = \text{diag}(\tilde{\lambda}_1, \dots, \tilde{\lambda}_n),\quad \boldsymbol{\phi} = [\phi_1, \dots, \phi_n]^T$
```

$A_{cl,1}$ with corresponding eigenvalues $\{\lambda_j\}_{j=1}^d$ and let $\{\mathbf{w_j}\}_{j=1}^d$ be the adjoint basis to $\{\mathbf{v_j}\}_{j=1}^d$. Then $\psi_j(\mathbf{y}) = \langle \mathbf{y}, \mathbf{w}_j \rangle$ is an eigenfunction of $\mathcal{K}_{A_{cl,1}}$ with eigenvalue e_j^λ and an arbitrary number of eigenpairs can be constructed using the product rule (3.4).

The eigenfunction construction for the linearized system only relies on the nominal model. To construct Koopman eigenfunctions for the true nonlinear dynamical system, I aim to learn the diffeomorphism (3.5) between the linearized model (3.13) and the true dynamics (3.12), see Figure 3.1. This diffeomorphism is guaranteed to exist in the entire basin of attraction Ω by Theorem 3.3. Let $\mathcal{H}_h$ be a class of continuous nonlinear function mapping $\mathbb{R}^d$ to $\mathbb{R}^d$. The diffeomorphism is found by solving the following optimization problem:

$$\begin{aligned} \min_{h \in \mathcal{H}_h} & \sum_{k=1}^{M_t} \sum_{j=1}^{M_s} (\dot{\mathbf{x}}_k^j + \dot{h}(\mathbf{x}_k^j) - A_{cl}(\mathbf{x}_k^j + h(\mathbf{x}_k^j)))^2 \\ \text{s.t.} \quad & \mathbf{D}h(\mathbf{0}) = \mathbf{0} \end{aligned} \tag{3.15}$$

which is a direct transformation of Theorem 3.3 into the setting with unknown nonlinear dynamics. The form of problem (3.15) is found by minimizing the squared loss $\dot{\mathbf{y}}_k - A_{cl}\mathbf{y}_k$ over all data pairs, substituting $\mathbf{y} = \mathbf{x} + h(\mathbf{x})$, and then adding the constraint $\mathbf{D}c(\mathbf{0}) = I$ to yield the optimization problem (3.15).

Next, (3.15) is formulated as a general supervised learning problem. Consider the data set of input-output pairs $\mathcal{D}_h = \left\{(\mathbf{x}_k, \dot{\mathbf{x}}_k), \dot{\mathbf{x}}_k - A_{cl}\mathbf{x}_k\right\}_{k=1}^{M_s \cdot M_t}$, constructed from the state measurements (perhaps by calculating numerical derivatives $\dot{\mathbf{x}}_k^j$ as needed). The class $\mathcal{H}_h$ can be any function class suitable for supervised learning (e.g. deep neural networks) as long as the Jacobian of the function $h(\mathbf{x}) \in \mathcal{H}_h$ w.r.t. the input can be readily calculated. Assuming $h(\mathbf{x}) \in \mathcal{H}_h$ I define the loss function

$$\begin{aligned} \mathcal{L}_h(\mathbf{x}, \dot{\mathbf{x}}, A_{cl}\mathbf{x} - \dot{\mathbf{x}}) &= ||\dot{h}(\mathbf{x}) - A_{cl}h(\mathbf{x}) - (A_{cl}\mathbf{x} - \dot{\mathbf{x}})||^2 + \alpha||\mathbf{D}h(\mathbf{0})||^2 \\ &= ||\mathbf{D}h(\mathbf{x})\dot{\mathbf{x}} - A_{cl}h(\mathbf{x}) - (A_{cl}\mathbf{x} - \dot{\mathbf{x}})||^2 + \alpha||\mathbf{D}h(\mathbf{0})||^2 \end{aligned} \tag{3.16}$$

where parameter α penalizes the violation of constraint (3.15). The supervised learning goal is to select a function in $\mathcal{H}_h$ through empirical risk minimization (ERM):

$$h = \text{argmin}_{h \in \mathcal{H}_h} \frac{1}{M_s \cdot M_t} \sum_{k=1}^{M_s \cdot M_t} \mathcal{L}_h(\mathbf{x}_k, \dot{\mathbf{x}}_k, A_{cl}\mathbf{x}_k - \dot{\mathbf{x}}_k) \; . \tag{3.17}$$

Finally, with function h identified from ERM (3.17), Proposition 3.2 implies that the Koopman eigenfunctions for the unknown dynamics under the nominal control law can be constructed from the eigenfunctions of the linearized system by the function composition:

$$\phi_j(\mathbf{x}) = \tilde{\psi}_j(g(h(\mathbf{x}))) \tag{3.18}$$

where g is the scaling function ensuring that the basin of attraction Ω is scaled to lie within the unit hypercube Q_1 and $\tilde{\psi}_j$ is an eigenfunction for the linearized system with associated eigenvalue $\tilde{\lambda}_j$ constructed with (3.4).

Importantly, because the diffeomorphism is learned from data, it may not perfectly capture the underlying diffeomorphism over all of Ω, and thus the eigenfunctions for the unknown dynamics are approximate. The error arises from the fact that the ERM problem is underdetermined resulting in the possibility of multiple approximations with equal loss while failing to capture the underlying diffeomorphism. This is especially an issue when encountering states and state time derivatives not reflected in the training data and introduces a demand for exploratory control inputs to cover a larger region of the state space of interest. This can be achieved by introducing a random perturbation of the control action deployed on the system and is akin to persistence of excitation in adaptive control (Ljung, 1987). To understand these effects, state dependent model error bounds are needed, but they are a subject of future work.

3.4 Koopman Eigenfunction Extended Dynamic Mode Decomposition

To use the constructed Koopman eigenfunctions for prediction and control, an EDMD-based method to build a linear model in a lifted space is developed. Since this method exploits the structure of the Koopman eigenfunctions, it is dubbed *Koopman Eigenfunction Extended Dynamic Mode Decomposition* (KEEDMD). I construct n eigenfunctions $\{\phi_j\}_{j=1}^n$ with associated eigenvalues $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ as outlined in Section 3.3 and define the lifted state as

$$\mathbf{z} = [\mathbf{x}, \boldsymbol{\phi}(\mathbf{x})]^T \tag{3.19}$$

where $\boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \dots, \phi_n(\mathbf{x})]$. I seek to learn a model of the form

$$\dot{\mathbf{z}} = A\mathbf{z} + B\mathbf{u} \tag{3.20}$$

where matrices $A \in \mathbb{R}^{(n+d)\times(d+n)}, B \in \mathbb{R}^{(d+n)\times m}$ are unknown, and are to be inferred from the collected data. As discussed in Chapter 1, using a lifted linear model is a simplification that makes the learning and control formulation easier. Lifted bilinear models that can accurately capture control-affine dynamics, more appropriate to describe many robotic systems will be explored in Chapter 4.

I focus on systems governed by Lagrangian dynamics, whose state space coordinates consist of position, $\mathbf{p}$, and velocity $\mathbf{v}$: $\mathbf{x} = [\mathbf{p}, \mathbf{v}]^T$, with $\dot{\mathbf{p}} = \mathbf{v}$. The rows of A corresponding to the position states are known. Furthermore, by construction the eigenvalues Λ describe the evolution of the eigenfunctions under the nominal control law. Therefore, the rows of A corresponding to eigenfunctions are also known. As a result, the lifted state space model has the following structure:

$$\begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\mathbf{v}} \\ \dot{\boldsymbol{\phi}}\left(\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \end{bmatrix}\right) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I & 0 \\ A_{\mathbf{vp}} & A_{\mathbf{vv}} & A_{\mathbf{v}\boldsymbol{\phi}} \\ \multicolumn{2}{c}{-B_{\boldsymbol{\phi}}K_{nom}} & \Lambda \end{bmatrix}}_{A} \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \boldsymbol{\phi}\left(\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \end{bmatrix}\right) \end{bmatrix} + \underbrace{\begin{bmatrix} B_{\mathbf{p}} \\ B_{\mathbf{v}} \\ B_{\boldsymbol{\phi}} \end{bmatrix}}_{B} \mathbf{u} \tag{3.21}$$

where $0, I, \Lambda, K_{nom}$ are fixed matrices and $A_{\mathbf{vp}}, A_{\mathbf{vv}}, A_{\mathbf{v}\boldsymbol{\phi}}, B_{\mathbf{p}}, B_{\mathbf{v}}, B_{\boldsymbol{\phi}}$ are determined from data. The term $-B_{\boldsymbol{\phi}}K_{nom}$ accounts for the effect of the nominal controller on the evolution of the eigenfunctions.

To infer the different parts of (3.21), the data samples are first processed and aggregated into data matrices. Specifically, for each $\mathbf{x}_k^j$ in $\mathcal{D}$, the data set (3.14), the position and velocity, $\mathbf{p}_k^j, \mathbf{v}_k^j$ are extracted and the lifted state, $\boldsymbol{\phi}_k^j = \boldsymbol{\phi}(\mathbf{x}_k^j)$ calculated. Furthermore, the nominal and perturbed control signals are calculated as $\mathbf{u}_{k,nom}^j = K_{nom}\mathbf{x}_k^j$ and $\mathbf{u}_{k,pert}^j = \mathbf{u}_k^j - K_{nom}\mathbf{x}_k^j$, respectively. $\mathbf{u}_{k,nom}^j$ and $\mathbf{u}_{k,pert}^j$ are introduced to distinguish between the nominal control signal and the added random perturbation that is added to induce exploratory behavior as discussed in Section 3.3. The data matrices are then aggregated as follows

$$\begin{aligned} P &= [\mathbf{p}_1^1, \dots, \mathbf{p}_{M_s}^1, \dots, \mathbf{p}_1^{M_t}, \dots, \mathbf{p}_{M_s}^{M_t}]^T, \\ V &= [\mathbf{v}_1^1, \dots, \mathbf{v}_{M_s}^1, \dots, \mathbf{v}_1^{M_t}, \dots, \mathbf{v}_{M_s}^{M_t}]^T, \\ \Phi &= [\boldsymbol{\phi}_1^1, \dots, \boldsymbol{\phi}_{M_s}^1, \dots, \boldsymbol{\phi}_1^{M_t}, \dots, \boldsymbol{\phi}_{M_s}^{M_t}]^T, \\ U_{nom} &= [\mathbf{u}_{1,nom}^1, \dots, \mathbf{u}_{M_s,nom}^1, \dots, \mathbf{u}_{1,nom}^{M_t}, \dots, \mathbf{u}_{M_s,nom}^{M_t}]^T, \\ U_{pert} &= [\mathbf{u}_{1,pert}^1, \dots, \mathbf{u}_{M_s,pert}^1, \dots, \mathbf{u}_{1,pert}^{M_t}, \dots, \mathbf{u}_{M_s,pert}^{M_t}]^T. \end{aligned} \tag{3.22}$$

Additionally, $\dot{P}, \dot{V}, \dot{\Phi}$ are constructed, either by numerically differentiating P, V, Φ, respectively, or by directly measuring $\dot{\mathbf{x}} = [\dot{\mathbf{p}}, \dot{\mathbf{v}}]^T$ and analytically calculating $\dot{\boldsymbol{\phi}} = \nabla_{\mathbf{x}}\boldsymbol{\phi}\dot{\mathbf{x}}$ if available.

Second, the constructed data matrices are concatenated and the loss function for three separate ordinary least squares regression problems formulated, defined as follows

$$\min_{B_{\mathbf{p}} \in \mathbb{R}^{(d/2)\times m}} ||\mathbf{y_p} - X_{\mathbf{p}} B_{\mathbf{p}}^T||_2^2, \qquad X_{\mathbf{p}} = [U], \qquad \mathbf{y_p} = [\dot{P} - IV] \tag{3.23a}$$

$$\min_{\substack{A_{\mathbf{v}} \in \mathbb{R}^{(d/2)\times(d+n)} \\ B_{\mathbf{v}} \in \mathbb{R}^{(d/2)\times m}}} ||\mathbf{y_v} - X_{\mathbf{v}}[A_{\mathbf{v}}\ B_{\mathbf{v}}]^T||_2^2, \quad X_{\mathbf{v}} = [P\ V\ \Phi\ U], \quad \mathbf{y_v} = [\dot{V}] \tag{3.23b}$$

$$\min_{B_{\boldsymbol{\phi}} \in \mathbb{R}^{n\times m}} ||\mathbf{y}_{\boldsymbol{\phi}} - X_{\boldsymbol{\phi}} B_{\boldsymbol{\phi}}^T||_2^2, \qquad X_{\boldsymbol{\phi}} = [U - U_{nom}], \quad \mathbf{y}_{\boldsymbol{\phi}} = [\dot{\Phi} - \Lambda\Phi]. \tag{3.23c}$$

To reduce overfitting, different forms of regularization can be added to the objectives of the regression formulations. In particular, l_1-regularization, which promotes sparsity in the learned matrices, has been shown to perform well for dynamical systems (S. L. Brunton, Proctor, et al., 2016) when used in normal EDMD. This has also been the case in our numerical simulations, where l_1-regularization seem to improve the prediction performance and the stability of the results.

To avoid an ill-conditioned KEEDMD regression problem, Brownian noise is added to perturb the nominal controller (Kaiser, Kutz, and S. L. Brunton, 2021). Brownian noise is chosen in this instance because pure sampling from e.g. a Gaussian distribution leads to perturbations that have too high frequency to perturb the movement of the multirotor. This perturbation is also used by our episodic learning framework (Section 3.8). When the lifted state space model is identified, state estimates can be obtained as $\mathbf{x} = C^{\mathbf{x}}\mathbf{z}$, where $C^{\mathbf{x}} = [I \quad 0]$. As before, $C^{\mathbf{x}}$ is denoted the *projection matrix* of the lifted state space model.

Extensions for Trajectory-tracking Nominal Controller

In all of the above, a pure state feedback nominal control law is considered. This section discusses how to extend the methodology to allow linear trajectory-tracking feedback controllers of the form $\mathbf{u} = K_{nom}(\mathbf{x} - \boldsymbol{\tau}(t))$. Under such a controller, the closed loop linearized dynamics become $\dot{\mathbf{x}} = A_{nom}\mathbf{x} + B_{nom}K_{nom}(\mathbf{x} - \boldsymbol{\tau}(t))$. Let the definition of the closed loop dynamics matrix, $A_{cl} = (A_{nom} + B_{nom}K_{nom})$ and the principal eigenfunctions (the eigenfunctions associated with the Koopman operator of the linearized system), be as in Section 3.2. Then, the evolution of the principal eigenfunctions becomes

$$\begin{aligned}\dot{\psi}_j(\mathbf{y}) = \langle \dot{\mathbf{w}_j, \mathbf{y}} \rangle = \mathbf{w}_j^T \dot{\mathbf{y}} \\ &= \langle \mathbf{w}_j, A_{cl}\mathbf{y} - B_{nom}K_{nom}\boldsymbol{\tau}(t) \rangle \\ &= \lambda_j \langle \mathbf{w}_j, \mathbf{y} \rangle - \langle \mathbf{w}_j, B_{nom}K_{nom}\boldsymbol{\tau}(t) \rangle \\ &= \lambda_j \psi(\mathbf{y}) - \mathbf{w}_j^T B_{nom}K_{nom}\boldsymbol{\tau}(t)\end{aligned} \tag{3.24}$$

where λ_j and $\mathbf{w}_j$ are the j-th eigenvalue and adjoint eigenvector of A_{cl}, respectively. Notably, the principal eigenfunctions evolves as described in Section 3.3 but with an additional forcing term, $-\mathbf{w}_j^T B_{nom}K_{nom}\boldsymbol{\tau}(t)$.

Using the assumption that the dynamics considered have linear actuated dynamics (see Eq. 3.12), I show that the evolution of the eigenfunctions of the Koopman operator associated with the full dynamics is affine in the input signal.

Proposition 3.5. *Assume that B_{nom} in the linearized model of the dynamics (3.13) is equal to the actuation matrix of the true dynamics (3.12) and that the dynamics are controlled by a linear trajectory-tracking feedback controller of the form $\mathbf{u} = K_{nom}(\mathbf{x} - \boldsymbol{\tau}(t)$. Then, the time derivatives of the eigenfunctions of the Koopman operator associated with the dynamics (3.12) constructed as described in Proposition 3.1-3.2 are affine in the external forcing signal $\boldsymbol{\tau}(t)$.*

Proof. I first show that the diffeomorphism between the linearized and nonlinear dynamics is linear in the forcing signal. Consider the diffeomorphism described in Theorem 3.3 with an additional forcing term. Derived from the linearized dynamics, I seek to find a function $h(\mathbf{x})$ such that

$$\dot{\mathbf{y}} = A_{cl}\mathbf{y} - B_{nom}K_{nom}\boldsymbol{\tau}(t), \qquad \mathbf{y} = \mathbf{x} + h(\mathbf{x}). \tag{3.25}$$

By algebraic manipulations I get that

$$\begin{aligned}&\dot{\mathbf{y}} = \dot{\mathbf{x}} + \dot{h}(\mathbf{x}) = A_{cl}(\mathbf{x} + h(\mathbf{x}) - B_{nom}K_{nom}\boldsymbol{\tau}(t) \\ &\Rightarrow a(x) + BK_{nom}(\mathbf{x} - \boldsymbol{\tau}(t)) + \dot{h}(\mathbf{x}) \\ &\ = (A_{nom} + B_{nom}K_{nom})(\mathbf{x} + h(\mathbf{x})) - B_{nom}K_{nom}\boldsymbol{\tau}(t) \\ &\Rightarrow \dot{h}(\mathbf{x}) - A_{cl}h(\mathbf{x}) = A_{nom}\mathbf{x} - a(\mathbf{x}).\end{aligned} \tag{3.26}$$

Hence, the expression for $h(\mathbf{x})$ does not depend on the forcing signal $\boldsymbol{\tau}(t)$. As a result, the diffeomorphism $c(\mathbf{x})$ does not depend on the forcing signal and the eigenfunctions associated with the eigenfunctions of the nonlinear dynamics (3.12) evolve affinely in the forcing signal. □

Because the eigenfunctions evolve linearly in the forcing signal, the KEEDMD-framework can readily learn the effect of external forcing on the eigenfunctions with only minor modifications. First, the loss of the diffeomorphism empirical risk minimization (3.15) must be changed to account for the forcing term following the construction of (3.25) such that the new loss function becomes

$$\begin{aligned}&\mathcal{L}_h(\mathbf{x}, \dot{\mathbf{x}}, A_{cl}\mathbf{x} - \dot{\mathbf{x}}, \boldsymbol{\tau}(t)) = \\ &\qquad ||\dot{h}(\mathbf{x}) - A_{cl}h(\mathbf{x}) - (A_{cl}\mathbf{x} - \dot{\mathbf{x}}) + B_{nom}K_{nom}\boldsymbol{\tau}||^2 + \alpha||\mathbf{D}h(\mathbf{0})||^2\end{aligned} \tag{3.27}$$

where $\boldsymbol{\tau}$ is the vector of desired states corresponding to the time when $\mathbf{x}, \dot{\mathbf{x}}$ were sampled. Second, the data matrix $X_{\boldsymbol{\phi}}$ in the regression formulation (3.23) must be modified so the effect of the forcing on the eigenfunction evolution can be learned. This is achieved by setting

$$X_{\boldsymbol{\phi}} = [U - K_{nom}[P \quad V]]. \tag{3.28}$$

3.5 Model Predictive Control Design

The MPC design is based on the controller introduced in Chapter 2. The QP formulation requires us to discretize the previously learned linear continuous dynamics. I assume a known objective function that is solely a function of states and controls. For simplicity, a quadratic objective function is used but other objective functions are possible, and can be introduced by adding them to the lifting functions. I assume known control bounds $\mathbf{u}_{\min}, \mathbf{u}_{\max} \in \mathbb{R}^m$ and state bounds $\mathbf{x}_{\min}, \mathbf{x}_{\max} \in \mathbb{R}^d$. These assumptions define the following optimization problem, as introduced in Chapter 2:

$$\begin{aligned}\min_{Z,U} \sum_{k=1}^{T-1} &\Big[(C^{\mathbf{x}}\mathbf{z}_k - \boldsymbol{\tau}_k)^T Q (C^{\mathbf{x}}\mathbf{z}_k - \boldsymbol{\tau}_k) + \mathbf{u}_k^T R \mathbf{u}_k\Big] + (C^{\mathbf{x}}\mathbf{z}_T - \boldsymbol{\tau}_T)^T Q_T (C^{\mathbf{x}}\mathbf{z}_T - \boldsymbol{\tau}_T), \\ \text{s.t.} \quad & \mathbf{z}_{k+1} = A\mathbf{z}_k + B\mathbf{u}_k, \quad k = 0, \dots, T-1, \\ & \mathbf{x}_{\min} \le C^{\mathbf{x}}\mathbf{z}_k \le \mathbf{x}_{\max}, \; k = 1, \dots, T, \\ & \mathbf{u}_{\min} \le \mathbf{u}_k \le \mathbf{u}_{\max}, \quad k = 0, \dots, T-1, \\ & \mathbf{z}_0 = \Phi(\mathbf{x}_0)\end{aligned} \tag{3.29}$$

here $Q, Q_N \in \mathbb{R}^{d \times d}$ and $R \in \mathbb{R}^{m \times m}$ are positive semidefinite cost matrices, $\boldsymbol{\tau} \in \mathbb{R}^{d \times T}$ is the reference trajectory, $A_d \in \mathbb{R}^{n \times n}$ and $B_d \in \mathbb{R}^{n \times m}$ are the discrete time versions of (3.20), $C^{\mathbf{x}} \in \mathbb{R}^{d \times n}$ is the projection matrix, and $\boldsymbol{\phi} \in \mathbb{R}^n$ are the learned eigenfunctions.
To remove the dependency on the lifting dimension n in Eq. (3.29), the state is eliminated via the explicit relation with the control input. This formulation is referred as the *dense* form MPC (see Chapter 2). This step greatly reduces the number of optimization variables, which is beneficial as the MPC problem must be solved in real-time. In this form, the MPC is agnostic not only of the lifting dimension but of the whole Koopman formalism, *i.e.* the eigenfunctions $\boldsymbol{\phi}$ and linear matrices A_d, B_d and $C^{\mathbf{x}}$ do not directly appear in the formulation. In addition, the state constraints are relaxed while keeping hard control bounds in order to ensure there is always a solution to the quadratic program. The relaxation penalty can be tuned to have negligible violation of the constraints and to avoid numerical problems.

3.6 Supervised Learning and Control of Simulated Inverted Pendulum

To obtain an initial evaluation of the performance of the proposed framework, the canonical cart pole system with continuous dynamics is studied[1]:

$$\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{M+m}\left(ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2 + F\right) \\ \frac{1}{l}\left(g\sin\theta + \ddot{x}\cos\theta\right) \end{bmatrix} \tag{3.30}$$

where x, θ are the cart's horizontal position and the angle between the pole and the vertical axis, respectively, M, m are the cart's and pole tip's mass, respectively, l is the pole length, g the gravitational acceleration, and F the horizontal force input on the cart. The linearization of the dynamics around the origin is used as the nominal model. Starting with knowledge of the nominal model only, our goal is to learn a lifted state space model of the dynamics to improve the system's ability to track a trajectory designed based on the nominal model to move to the origin from a initial condition two meters away. Data is collected with a nominal controller, a lifted state space model learned and the learned model used to design an improved MPC.

To build the dataset used for training, 40 trajectories are simulated by sampling an initial point in the interval $(x, \theta, \dot{x}, \dot{\theta}) \in [-2.5, 2.5] \times [-0.25, 0.25] \times [-0.05, 0.05] \times [-0.05, 0.05]$, generating a two second long trajectory from the initial point to the origin with a MPC based on the nominal model, and simulating the system with

[1]Code available at `https://github.com/Cafolkes/keedmd`

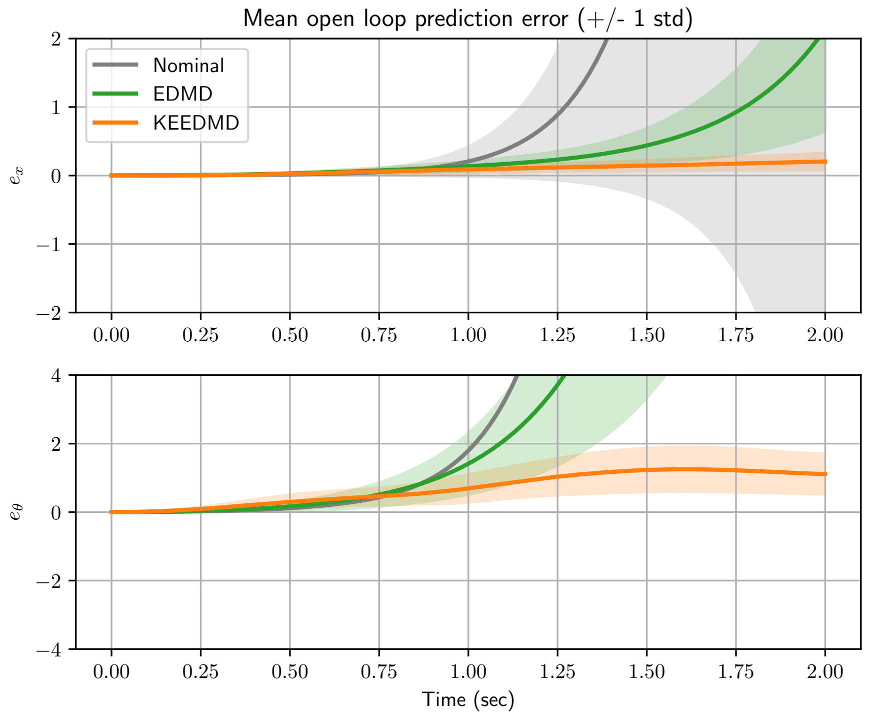


Figure 3.2: Performance comparison of the nominal model, EDMD, and KEEDMD applied to open loop prediction of the cart-pole system from 40 different initial conditions.

a PD controller stabilizing the system to the trajectory. Note that the system is underactuated and stabilizing the system to a set point under PD control is not possible. The PD controller is perturbed with white noise of variance 0.5 to aid the model fitting as described in Section 3.3, and state and control actions are sampled from the simulated trajectories at 100 hz. With the collected data, eigenfunctions are constructed as described in Algorithm 1, and a lifted state space model is identified according to (3.23).

To benchmark our results, I compare our prediction and control results against (1) the nominal model, and (2) a EDMD-model with the state and Gaussian radial basis functions as lifting functions. In both the EDMD and KEEDMD models, a lifting dimension of 85 is used and elastic net regularization is added with regularization parameters determined by cross validation. The diffeomorphism, h, is parameterized by a 3-layer neural network with 50 units in each layer and implemented with *PyTorch* (Paszke et al., 2019). The EDMD and KEEDMD regressions are implemented with *Scikit-learn* (Pedregosa et al., 2011).

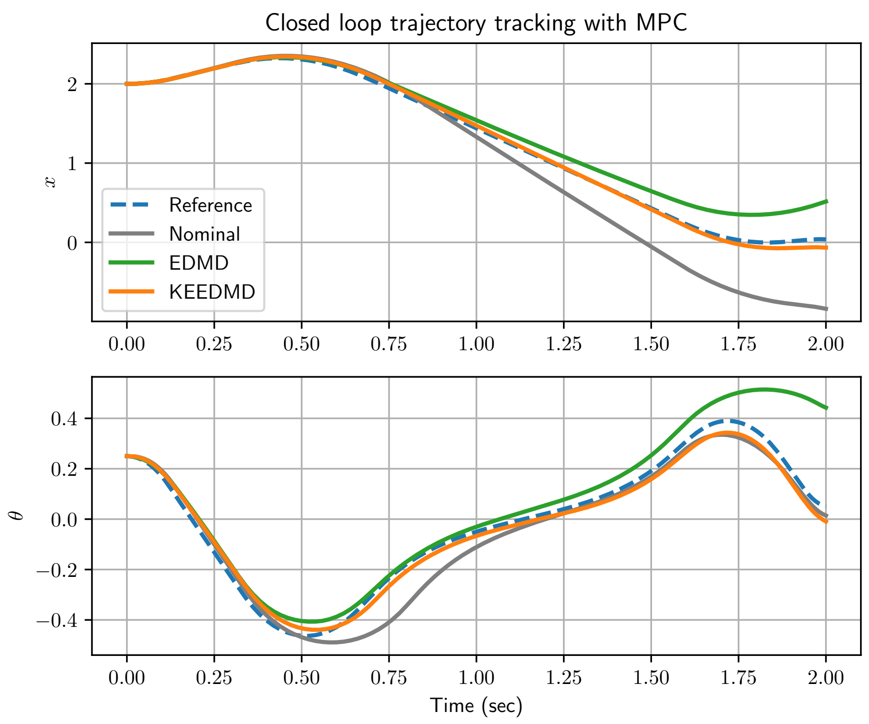


Figure 3.3: Performance comparison of the nominal model, EDMD, and KEEDMD used for closed loop control of the cart-pole system.

First, the open loop prediction performance is compared by sampling 40 points from the same intervals as the training data, and then stabilizing the system to the origin with a MPC based on the nominal model, using a 2-second prediction horizon. Then, the time evolution of the system is predicted from the sampled initial point and with the control sequence from the collected data for each trajectory with the nominal, EDMD, and the KEEDMD models. The mean error between the predicted evolution and the true system evolution over all the trajectories is depicted in Figure 3.2. Both the nominal and EDMD models are able to predict the evolution for the first second but then diverges. In contrast, KEEDMD is able to maintain good prediction performance over the entire duration of the trajectories with relatively low, constant standard deviation.

Table 3.1: Improvement in MPC cost with learned models

	Improvement over nominal model	**Improvement over EDMD-model**
EDMD	−68.00%	
KEEDMD	−96.75%	−89.84%

To evaluate the closed loop performance, I compare the behavior of the three different models on the task of moving from initial condition $(x_0, \theta_0, \dot{x}_0, \dot{\theta}_0) = (2, 0.25, 0, 0)$ in two seconds. The nominal model is used to generate a trajectory from the initial state to the origin. Then, a dense form MPC using the learned lifted state space model is implemented in Python using the QP solver *OSQP* (Stellato et al., 2018). The MPC costs on the trajectory tracking task are significantly improved when the lifted state space models are used, see Figure 3.3. It is important to note that the EDMD-based MPC regulates less towards the end of the trajectory causing large deviations but still outperforms the nominal model in terms of MPC cost by 62 percent. For the same penalty matrices Q, R, the KEEDMD-based MPC has significantly better trajectory tracking performance and further reduces the MPC cost by 90 percent.

3.7 Episodic KEEDMD Learning

KEEDMD (Section 3.4) requires batch training data to be collected from system executions under a nominal linear control law: $\mathbf{u}_{nom}(\mathbf{x}) = K_{nom}(\mathbf{x} - \boldsymbol{\tau}(t))$. The remainder of this chapter describes how to iteratively learn an improving sequence of eigenfunctions and nonlinear controllers in an episodic manner, i.e. by considering the closed-loop dynamics obtained with a nonlinear controller as the autonomous dynamics for the next episode. This approach enables the online learning process to capture nonlinear actuation effects in the learned Koopman eigenfunctions. Specifically, the lifted state-space model is iteratively used to design an MPC-controller to track learning trajectories (see Figure 3.4).

Assume that we have selected a fixed trajectory $\boldsymbol{\tau}$ to be tracked by the robot during episodic learning. Further assume a nominal controller $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$ that can stabilize the system to $\boldsymbol{\tau}$ within a region of attraction Ω around the trajectory. This controller might be the outcome of a previous learning episode (see below), or the simple linear nominal controller from KEEDMD. Finally, the system's governing dynamics are assumed to be *unknown*, and take the general form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{3.31}$$

where $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d, \mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$, and $\mathbf{f}(\mathbf{x}, \mathbf{u})$ is assumed to be Lipschitz continuous on $\mathcal{X} \times \mathcal{U}$. Note that the actuated dynamics are not required to be linear as in Section 3.3 as the episodic learning framework is designed to capture nonlinear actuation effects.

Learning with Arbitrary Stabilizing Control Laws

If a candidate nonlinear controller $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$ can stabilize system (3.31) to a given trajectory $\boldsymbol{\tau}$, the controlled system can be described by the autonomous dynamics

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)) = F_{\hat{\mathbf{u}},\boldsymbol{\tau}}(\mathbf{x}, t). \tag{3.32}$$

Importantly, for the autonomous dynamics (3.32), there exists an associated Koopman operator $\mathcal{K}_{F_{\hat{\mathbf{u}},\tau}}$ that depends on control law $\hat{\mathbf{u}}$ and trajectory $\boldsymbol{\tau}$. Therefore, approximate eigenpairs for $\mathcal{K}_{F_{\hat{\mathbf{u}},\tau}}$ can be constructed (see Section 3.2) from the gathered state and control samples. A lifted state-space model can be constructed from these eigenpairs.

However, unlike the framework presented in Section 3.4, I aim to learn a dynamical model that assumes that the system is already regulated by the *nominal controller* $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$. As a result, the A-matrix of the lifted state space model captures the autonomous dynamics under the nominal control law (Eq. 3.32), and the B-matrix captures the effect of control variations around the nominal controller:

$$\dot{\hat{\mathbf{z}}} = A\hat{\mathbf{z}} + B(\mathbf{u}(\mathbf{x}, \boldsymbol{\tau}, t) - \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)). \tag{3.33}$$

This model is used in an MPC framework below to design an *augmenting* control law that adds optimal control actions to the nominal controller. The augmenting controller leverages the improved system model to make corrections to sub-optimal actions taken by the nominal controller.

Modifications to Allow the Diffeomorphism to Capture Nonlinear Control and Dynamics Effects

To enable the learning framework to capture nonlinear effects caused by the nonlinear controller and actuated dynamics, a minor modification to the function approximator of h is necessary. Namely, since the diffeomorphism is affected by the forcing signal $\boldsymbol{\tau}(t)$ it must be included in the inputs of h. This is motivated by the form of the diffeomorphism loss function (3.15). In the case considered in Section 3.4 however, the actuated dynamics and controller are assumed to be linear. This causes the effect of the forcing signal $\boldsymbol{\tau}(t)$ to cancel out such that the diffeomorphism is independent of the desired trajectory. In the general nonlinear case however, the effect is not canceled out and must be captured by the diffeomorphism. As a result, the diffeomorphism is modified such that $h : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{Y}$.

3.8 Episodic Eigenfunction Construction and KEEDMD Inference

Overview of the Episodic Learning Algorithm

Algorithm 2 summarizes our episodic learning approach, which applies three key steps per episode. In each episode, e, the first key step starts when an initial condition is sampled from set X_0 and an experiment is executed with the controller that results from the previous episode $\mathbf{u}_{e-1}(\mathbf{x}, \boldsymbol{\tau}, t)$. The state $\mathbf{x}$, control actions $\mathbf{u}_{e-1}$, Brownian noise control perturbations $\tilde{\mathbf{u}}$, and the desired position dictated by the trajectory at the time associated with the i-th sample $\boldsymbol{\tau}_i$ are sampled. State data can be differentiated numerically to find estimates $\dot{\mathbf{x}}$. The resulting data set is:

$$\mathcal{D}_x^{(e)} = \left\{ \left(\mathbf{x}_i^{(e)}, \mathbf{u}_i^{(e)}, \tilde{\mathbf{u}}_i^{(e)}, \boldsymbol{\tau}_i \right), \dot{\mathbf{x}}_i^{(e)} \right\}_{i=1}^{T_s} \tag{3.34}$$

where $\mathbf{x}_i^{(e)}$ denotes the i-th timestep of the e-th episode and T_s denotes the number of samples in the episode. From $\mathcal{D}_x^{(e)}$, the diffeomorphism, h, is estimated and the eigenfunctions, $\boldsymbol{\phi}^{(e)}(\mathbf{x})$, with associated eigenvalues, $\Lambda^{(e)}$, constructed via Algorithm 1. Since changes in the control law between episodes are expected to be small, the learning algorithm is warm-started with model coefficients from the previous episode.

The second key step is to use the constructed eigenpairs to build a lifted data set $\mathcal{D}_z^{(e)}$:

$$\mathcal{D}_z^{(e)} = \left\{ \left(\mathbf{z}_i^{(e)}, \mathbf{u}_i^{(e)}, \tilde{\mathbf{u}}_i^{(e)}, \boldsymbol{\tau}_i \right), \dot{\mathbf{z}}_i^{(e)} \right\}_{i=1}^{T_s} \tag{3.35}$$

Algorithm 2 Episodic KEEDMD

```
1: Input: Desired trajectory $\boldsymbol{\tau}$, nominal controller $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$, diffeomorphism model class
   $\mathcal{H}_h$, diffeomorphism loss $\mathcal{L}_h$, number of lifting functions $n$, KEEDMD loss $\mathcal{L}_z$
2: $\mathcal{D}_z = \emptyset, \quad \mathbf{u}_0(\mathbf{x}, \boldsymbol{\tau}, t) = \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$
3:
4: for $e = 1, \dots, N_{ep}$ do
5:     Sample initial condition: $\mathbf{x_0} \leftarrow \text{sample}(X_0)$
6:     Execute experiment: $\mathcal{D}_x^{(e)} \leftarrow \text{run}(\mathbf{x}_0, \mathbf{u}^{(e-1)}(\mathbf{x}, \boldsymbol{\tau}, t))$
7:     Fit diffeomorphism estimator: $h(\mathbf{x}) \leftarrow \text{ERM}(\mathcal{H}_h, \mathcal{L}_h, \mathcal{D}_x^{(e)})$
8:     Construct eigenpairs: $(\boldsymbol{\phi}^{(e)}(\mathbf{x}), \Lambda^{(e)}) \leftarrow h(g(\boldsymbol{\psi}(\mathbf{x})))$
9:     Construct and aggregate lifted data set: $\mathcal{D}_z \leftarrow \mathcal{D}_z \cup \mathcal{D}_z^{(e)}$
10:    Fit KEEDMD model: $\dot{\mathbf{z}}^{(e)}(\mathbf{z}) \leftarrow \text{ERM}((\boldsymbol{\phi}^{(e)}, \Lambda^{(e)}), \mathcal{L}_z, \mathcal{D}_z)$
11:    Update controller: $\mathbf{u}^{(e)} \leftarrow \mathbf{u}^{(e-1)} + w^{(e)}\text{MPC}(\dot{\mathbf{z}}^{(e)}, \mathbf{u}^{(e-1)})$
12: end for
13: Output: Final control law $\mathbf{u}^{(N_{ep})}$
```

which is the same data as $\mathcal{D}_x^{(e)}$, but with the state and its derivative, $\mathbf{x}_i^{(e)}, \dot{\mathbf{x}}_i^{(e)}$, replaced with the lifted state and its derivative, $\mathbf{z}_i^{(e)}, \dot{\mathbf{z}}_i^{(e)}$. Next, data from the current and previous episodes is aggregated: $\bigcup_{j=1}^{e} \mathcal{D}_z^{(j)}$. The lifted state-space model is constructed from this data using the framework of Section 3.7. This results in a model of the form (3.33).

In the third and final step, an augmenting MPC is designed (see Section 3.8) for the lifted state-space model. The evaluation of the previous iteration's controllers is necessitated by the fact that the eigenfunctions depend on the dynamics under closed loop control with the controller deployed in the previous episodes. The controller augmentations are weighted and added to the previous episode's control law: $\mathbf{u}_e = \mathbf{u}_0 + \sum_{j=1}^{e} w_j \mathbf{u}_j$, where w_e is a weighting factor indicating the confidence in the augmenting controller. The weighting factors can be any monotonically increasing sequence on the interval $[0, 1]$ which allows the augmenting controller to have a bigger impact after a sufficiently rich data set has been collected.

Model Predictive Controller Details

This section shows how our framework yields, as a by-product of the learning process, an optimal controller that respects state and input constraints during both the learning and execution process. The controller structure is based on the MPC introduced in Section 2.4. Because the control input for each MPC problem refers to the change from the the previous controller, this change must be accounted for in the control bounds. All these assumptions define the following optimization problem that is solved at each time step:

$$
\begin{aligned}
\min_{Z,U} \sum_{k=1}^{T-1} \Big[(C^{\mathbf{x}}\mathbf{z}_k - \boldsymbol{\tau}_k)^T Q (C^{\mathbf{x}}\mathbf{z}_k - \boldsymbol{\tau}_k) + \mathbf{u}_k^T R \mathbf{u}_k \Big] + (C^{\mathbf{x}}\mathbf{z}_T - \boldsymbol{\tau}_T)^T Q_T (C^{\mathbf{x}}\mathbf{z}_T - \boldsymbol{\tau}_T), \\
\text{s.t.} \quad & \mathbf{z}_{k+1} = A\mathbf{z}_k + B\mathbf{u}_k, \quad k = 0, \dots, T-1, \\
& \mathbf{x}_{\min} \le C^{\mathbf{x}}\mathbf{z}_k \le \mathbf{x}_{\max}, \ k = 1, \dots, T, \\
& \mathbf{u}_{\min} \le \mathbf{u}_k - \sum_{i=1}^{j-1} w^{(i)} \mathbf{u}_k^{(i)} \le \mathbf{u}_{\max}, \quad k = 0, \dots, T-1, \\
& \mathbf{z}_0 = \Phi(\mathbf{x}_0).
\end{aligned}
\tag{3.36}
$$

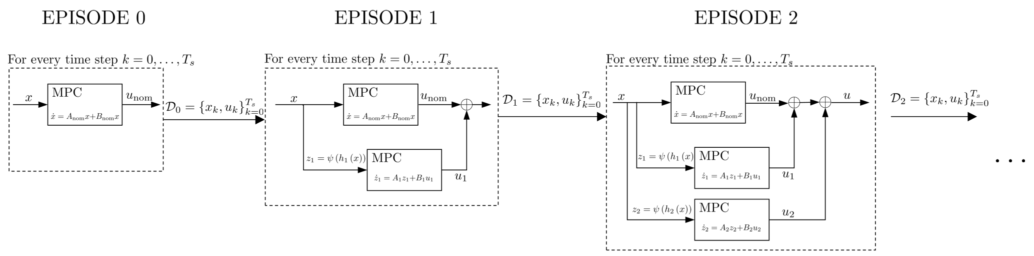

EPISODE 0
EPISODE 1
EPISODE 2
For every time step $k = 0, \dots, T_s$
x
MPC
$\dot{x} = A_{\text{nom}}x + B_{\text{nom}}x$
u_{nom}
$\mathcal{D}_0 = \{x_k, u_k\}_{k=0}^{T_s}$
$z_1 = \psi(h_1(x))$
$\dot{z}_1 = A_1 z_1 + B_1 u_1$
u_1
$\mathcal{D}_1 = \{x_k, u_k\}_{k=0}^{T_s}$
$z_2 = \psi(h_2(x))$
$\dot{z}_2 = A_2 z_2 + B_2 u_2$
u_2
u
$\mathcal{D}_2 = \{x_k, u_k\}_{k=0}^{T_s}$
...

Figure 3.4: Flow chart showing the different elements for each episode.

Figure 3.4 illustrates how each controller is augmented as more episodes are being executed. Additionally, a smoothing regularizer is added to avoid chatter that may arise from optimization-based controllers (Morris, M. J. Powell, and Ames, 2015).

3.9 Improving Fast multirotor Descent and Landing by Learning the Ground Effect

To validate the methodology, I apply it to fast descent and landing of a multirotor. Consider a multirotor drone, whose basic flight mechanics in open air are well understood (Hoffmann et al., 2007; Bangura and Mahoney, 2012). However, when multirotors fly close to the ground or a wall, or inside a narrow tunnel (e.g., for non-invasive inspection), the unmodeled effects of the complex vehicle-air-environment interaction can substantially reduce the drone's path tracking accuracy, and perhaps its stability. While ground effect models can be incorporated, their accuracy is limited, and their parameters must still be estimated in a slow process.

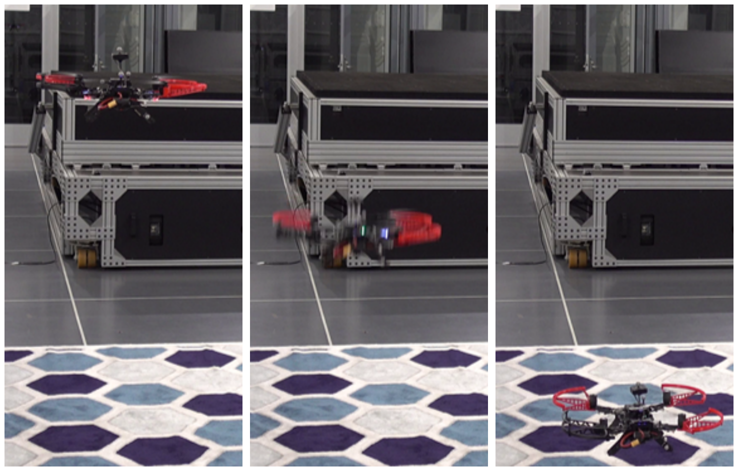


Figure 3.5: From left to right: hovering before the sequence start, high speed descent with learned dynamics, and soft landing.

Previous approaches have implemented MPC in real time on modest computational hardware (Zeilinger, 2011), on multirotors using an explicit solution in simulation (Liu, Lu, and W. H. Chen, 2015), and designed feedback linearizing controllers for multirotors in real experiments (Bangura and Mahony, 2014; Abdolhosseini, Y. M. Zhang, and Rabbath, 2013). Bouffard et al. (Bouffard, Aswani, and Tomlin, 2012) also used MPC to learn ground effects using an experimental multirotor, but used an Extended Kalman Filter (EKF) in combination with Learning-Based MPC (LBMPC). Shi et al. experimentally demonstrated using a spectrally-normalized neural network to learn the ground effect and improve drone landing by designing a feedback linearizing controller utilizing the learned model (G. Shi, X. Shi, et al., 2019). I introduce a new approach to solve this problem and aim to demonstrate that our method represents a first step towards practical Koopman-based learning and control of real-world robotic systems.

Modeling and Problem Statement

To simplify the discussion, consider a 1-dimensional *nominal* model of the multirotor's altitude dynamics, consisting of a point mass model having altitude and its derivative, $[p_z, \dot{p}_z]^T$, as states, mass m, and total thrust, F, as input:

$$\begin{bmatrix} \dot{p}_z \\ \ddot{p}_z \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_z \\ \dot{p}_z \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} F. \tag{3.37}$$

Using this model a nominal MPC is designed as described in Section 3.8 with the goal of reaching a fixed point of 0.05 m above ground at zero velocity. Furthermore, thrust constraints are added to reflect the multirotor's physical actuation limits.

A nominal MPC stabilizes the drone to a fixed point, but uses more control effort and time to reach that point as a result of its simplified model. Importantly, the nominal dynamics model does not capture the ground effect. Our goal is to iteratively learn a better dynamics model (and associated MPC) that will improve speed and tracking performance in both the air and near-ground regimes.

Table 3.2: Experiment Parameters

State error penalty, Q	$[10, 0.1]$	Min thrust, $u_{\min}$	0.3
Control penalty, R	1	Max thrust, $u_{\max}$	0.8
Min altitude, $x_{\min}$	0.05 m	Hover thrust, u_{hover}	0.66

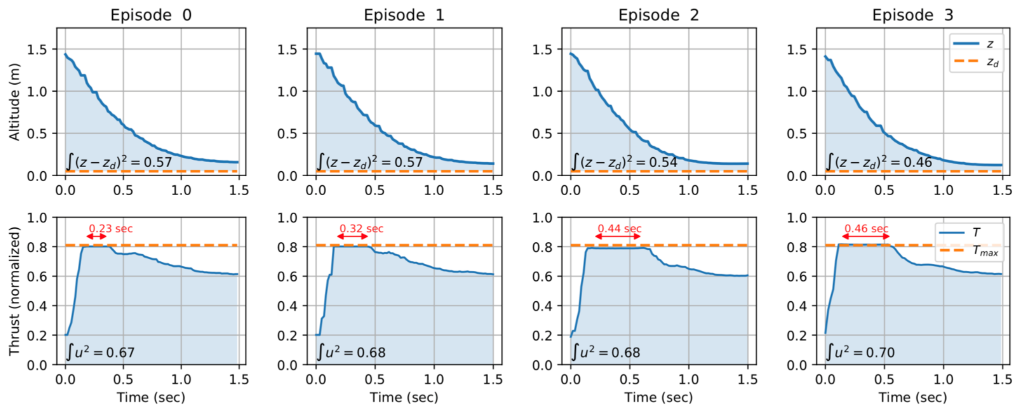


Figure 3.6: Evolution of drone altitude p_z with accumulated error and control effort after each episode. Episode 0: baseline controller, Episode 1-3: performance after each episode of learning. Red arrows: duration the thrust constraint is active.

Implementation and Experimental Details

Our experiments use the `Intel Aero RTF` Drone[2]. Drone position is measured using an *OptiTrack* motion capture system and is fused with the drone's IMU (stock PX4 v1.8) to estimate the state. The code for learning is implemented as discussed in Section 3.6. A dense form MPC-controller is implemented in Python using the QP solver *OSQP* (Stellato et al., 2018), and commands are sent to the PX4 flight controller via *ROS*. All computation for learning and control is done on board the drone. Each neural network and MPC evaluation takes 5 ms, limiting us to 5 episodes as update rates below 60 hz lead to poor performance on our hardware. The experiment's key parameters are summarized in Table 3.2[3].

Algorithm 2 is executed as discussed in Section 3.8 on the drone for three episodes in each campaign. Each episode starts with 3 repetitions of the following: (1) the drone takes off and moves to an initial point under PX4 control; (2) the lifted controller takes over to stabilize the fixed point and hovers at that point for a second. After 3 repetitions, the drone lands under lifted control, fits the diffeomorphism and KEEDMD models, and repeats the episode. An additional landing sequence is executed with no exploratory noise to evaluate the performance of the current episode controller.

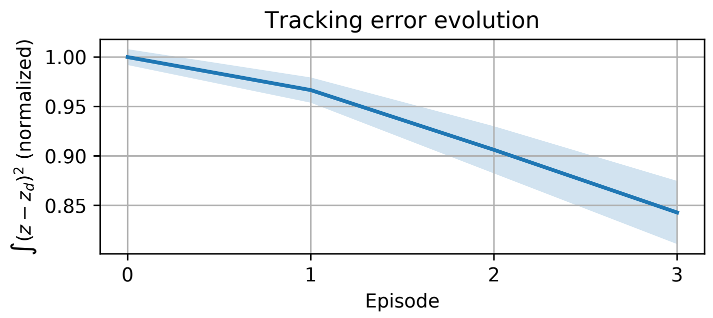


Figure 3.7: Mean ± 1 standard deviation of tracking performance after each episode over 5 independent campaigns.

Results and Discussion

Figure 3.6 depicts the drone's trajectory and control effort under the nominal controller (Episode 0), and then final landing for three episodes of a single learning campaign. Episode 0 represents the nominal performance before learning, while episodes 1-3 show the learning effect. Tracking error is reduced by 19.3 percent by the end of the last episode while the total control effort increases 4.5 percent as a consequence of the chosen MPC penalty matrices. Importantly, the thrust constraint is rigorously satisfied, and this constraint is active for longer duration. As the system learns more accurate dynamic models, it relies more on the open-loop bang-bang characteristic, as would be expected from an optimal solution, and less from closed loop control. Less control effort is needed towards the end of the trajectory, indicating that our methodology captures the ground effect.

The mean and standard deviation of five independent learning campaigns are reported in Fig. 3.7. The tracking performance improves in every episode. Furthermore, the methodology has low variance between campaigns.

3.10 Conclusions

I presented a novel method to learn nonlinear dynamics, using Koopman Eigenfunctions constructed from principal eigenfunctions and a nonlinear diffeomorphism as lifting functions for extended mode decomposition. I then used a MPC framework to obtain an optimal controller, while respecting state and control input bounds. I showed in simulation that the method drastically outperforms the linearization around the origin as well as the classical EDMD method with the same number of

[2] `https://github.com/intel-aero/meta-intel-aero/wiki`

[3] Code available at `https://github.com/Cafolkes/keedmd`

lifting functions in both prediction and closed loop control. These results indicate that focusing on the spectral properties of the Koopman Operator can allow for a more compact representation while achieving similar performance.

I then extended the batch learning process to an episodic learning framework to learn a robotic system's nonlinear dynamics, and learn a near optimal control strategy for given tasks. By using a Koopman approach, a real-time MPC framework for optimal system control can be implemented during the learning process. The approach improves performance as it gathers more data, augmenting the controller to avoid constant actuation matrix limitations, while respecting state and control input bounds. A current limitation is the addition of a controller in each episode leading to prohibitive computational complexity as the number of episodes grows. This limitation is addressed in Chapter 4 by modifying the underlying model structure to allow nonlinear actuation effects to be captured without the need of adding additional augmenting controllers each episode.

Chapter 4

LEARNING AND CONTROL OF SYSTEMS WITH CONTROL-AFFINE DYNAMICS

This chapter describes two methods that can learn lifted bilinear models that, unlike the linear lifted models discussed in Chapter 3, can capture control-affine dynamics. The first method was presented at the 2021 International Conference on Robotic and Automation (ICRA) (Folkestad and Burdick, 2021) and introduced a EDMD method to learn a bilinear model and utilize it in nonlinear MPC design. The second method has been submitted to the 2022 International Conference on Robotic and Automation (ICRA) (Folkestad, Wei, and Burdick, 2022). It extends the basic method by incorporating the model structure in a neural network architecture that can significantly reduce the lifting dimension of the learned model, enabling real time model predictive control based on the learned model.

4.1 Introduction

The process of designing high performance controllers for agile robotic systems that satisfy state and actuation constraints is challenging for systems with important nonlinear dynamical effects. Model predictive control (MPC) can capture appropriate performance objectives and constraints. Advances in optimization algorithms and computing power are enabling *nonlinear* MPC (NMPC) to be deployed on robotic systems in real-time if carefully implemented (Kouzoupis et al., 2018; Gros et al., 2020; Grandia et al., 2020). However, obtaining a sufficiently accurate dynamical model is *crucial* to achieve good performance with nonlinear MPC (NMPC). Many learning approaches have been proposed to capture a robots's complex mechanics and environmental interactions, reducing the need for time consuming system identification (cf. (Rasmussen and C. K. I. Williams, 2018; G. Shi, X. Shi, et al., 2019; Taylor, Dorobantu, Le, et al., 2019; Recht, 2019; Kaiser, Kutz, and S. L. Brunton, 2018)). However, NMPC design based on these models is typically impossible because of the model discretization and the intractable forward simulation needed to evaluate the nonlinear program. This chapter presents two methods that take a Koopman-centric approach to learn a *lifted bilinear model* of the dynamics that can be incorporated in NMPC to design close to optimal controllers that respect state and actuation constraints.

This chapter focuses on learning control-affine dynamics of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} \tag{4.1}$$

where $\mathbf{x}$ is the system state and $\mathbf{u}$ is a vector of control inputs, $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$, $\mathbf{f}(\mathbf{x}) : \mathcal{X} \to \mathcal{X}$, $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$, $\mathbf{g}(\mathbf{x}) : \mathcal{X} \to \mathbb{R}^{d\times m}$, $\mathbf{f}, \mathbf{g}$ assumed to be Lipschitz continuous on $\mathcal{X}$. This model characterizes a wide class of aerial and ground robots. As discussed in Chapters 1 - 3, the extended dynamic mode decomposition (EDMD) is a common indentification technique for Koopman-inspired models (Schmid, 2010; M. O. Williams, Kevrekidis, and Rowley, 2015; Korda and Mezić, 2018b; S. L. Brunton, Proctor, et al., 2016; Kaiser, Kutz, and S. L. Brunton, 2021; Proctor, S. L. Brunton, and Kutz, 2018). However, most current EDMD methods model the unknown control system dynamics by a lifted *linear* model, which implicitly restricts the control vector fields, $\mathbf{g}(\mathbf{x})$, to be state-invariant. This is a significant limitation, as many robotic systems, e.g. systems where input forces enter the dynamics through rotation matrices, are best described by nonlinear control-affine dynamics.

To overcome this limitation, I propose to learn a model motivated by the *Koopman canonical transform* (KCT) introduced in Chapter 2, which allows a large class of nonlinear control-affine dynamic models to be *lifted* to a higher-dimensional space where the system evolution can be described by a *bilinear* (but possibly infinite dimensional) dynamical system.

Prior work on Koopman-based control design has primarily focused on applying linear MPC to lifted linear models, and has been successfully implemented in both simulated and robotic experiments (Korda and Mezić, 2018b; Bruder, Gillespie, et al., 2019), and includes the work presented in Chapter 3. Design with bilinear models is less explored, but connections between Koopman bilinear system descriptions and classical control concepts such as reachability and control Lyapunov functions have been presented (Goswami and Paley, 2018; Huang, Ma, and Vaidya, 2019). Very recently, bilinear Koopman models linearized at the current state of the system were used in MPC (Bruder, Fu, and Vasudevan, 2021). Another approach uses the bilinear model structure to simplify the construction of a control Lyapunov function enforced as a constraint in a nonlinear MPC method to obtain stability guarantees (Narasingam and Kwon, 2020).

While a few works have addressed NMPC design for bilinear Koopman models (Bruder, Fu, and Vasudevan, 2021), (Narasingam and Kwon, 2020), little consideration has been given to practical real-time realization of these methods on robotic systems, which often require high control rates due to fast dynamics. Towards this goal, this chapter presents Koopman NMPC, combining the process of learning control-affine dynamics in Koopman bilinear form with NMPC design. I first present an EDMD-based learning method to learn a bilinear model from data. This is achieved by carefully designing the function dictionary and employing linear regression techniques on the lifted state (M. O. Williams, Kevrekidis, and Rowley, 2015; Korda and Mezić, 2018b). Then, building on recent advances in NMPC, I develop a controller for bilinear Koopman models that uses the bilinear model structure to improve computational efficiency, making real-time computation possible. The advantages of learning lifted bilinear models over linear models are highlighted, and this chapter demonstrates that the completely data-driven Koopman NMPC method can match the performance of a NMPC controller with full a priori model knowledge on a simulated planar quadrotor.

However, for realistic systems, such as real-world quadrotor drones, the dimension of the lifted model becomes too high when using a fixed function dictionary, resulting in the NMPC being intractable to implement in real-time. As a result, the second learning method incorporates the bilinear Koopman structure in a neural network model, allowing the function dictionary and bilinear model to be learned jointly from data. This enables similar or improved prediction performance with much fewer observable functions, thus allowing real-time use even for high-dimensional systems. Additionally, the resulting model maintains the bilinear structure, making it partially interpretable and possible to simulate its behavior forward in time by only evaluating the neural network at the initial state and propagating the lifted state forward using the bilinear system matrices.

Choosing a good function dictionary is a central challenge in Koopman-based methods, and using neural networks to jointly learn Koopman models and function dictionaries has been attempted previously. Multiple works have shown that using an encoder-decoder type architecture to parametrize the function dictionary allows many of the benefits of a Koopman-based model to be maintained while obtaining more compact models and/or improving prediction performance compared to fixed dictionaries (Li et al., 2017; Kaiser, Kutz, and S. L. Brunton, 2021; R. Wang, Han, and Vaidya, 2021). However, no existing method utilizes the bilinear models to accurately capture control-affine systems, nor has a view towards achieving high performance control for robotic systems. As such, the main contributions of this chapter are:

1. Development of two flexible learning methods underpinned by the Koopman canonical transformation to a *bilinear* form that can readily be used for NMPC design.
2. Hardware experiments with a quadrotor demonstrating that a controller based strictly on the data-driven model can outperform an NMPC based on a known, identified model, while needing limited training data.

The rest of this chapter is organized as follows. Preliminaries on NMPC are presented in Section 4.2. Then, the first learning method using fixed function dictionaries to learn lifted bilinear models is described in Section 4.3. Model predictive control design utilizing the learned model is described in Section 4.4 before demonstrations of the method are shown using a simulated planar quadrotor in Section 4.5. Subsequently, the second learning method that jointly learns both the function dictionary and bilinear model is presented in Section 4.6 and physical experiments validating the method on a quadrotor drone are presented in Section 4.7. Finally, concluding remarks are discussed in Section 4.8.

4.2 Preliminaries

Nonlinear model predictive control

When the exact continuous dynamics (4.1) are known, the general optimal control problem is intractable because there are infinitely many optimization variables. To reformulate the problem into a tractable finite-dimensional nonlinear program (NLP) the dynamics are discretized. Given a time horizon T, consider the time increment Δt and divide the time horizon [0,T] into $N = \frac{T}{\Delta t} + 1$ discrete subintervals $[t_k, t_{k+1}], t_k = k\Delta t, k = 0, \dots, N-1$. Replacing the continuous control signal $\mathbf{u}(t)$ with a zero-order-hold signal, the dynamics are integrated over each interval with an appropriate integration scheme to get a discrete-time representation of the dynamics $\mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k)$, where $\mathbf{x}_k = \mathbf{x}(t_k)$.

The quadratic objective NMPC problem is formulated as:

$$\begin{aligned}\min_{X,U} \quad & \sum_{k=0}^{N-1} \frac{1}{2} \begin{bmatrix} \mathbf{x}_k - \mathbf{x}_k^{\text{ref}} \\ \mathbf{u}_k - \mathbf{u}_k^{\text{ref}} \end{bmatrix}^T W_k \begin{bmatrix} \mathbf{x}_k - \mathbf{x}_k^{\text{ref}} \\ \mathbf{u}_k - \mathbf{u}_k^{\text{ref}} \end{bmatrix} \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k), \quad k = 0, \ldots, N-1 \\ & \mathbf{x}_0 = \hat{\mathbf{x}}, \quad \mathbf{h}_k(\mathbf{x}_k, \mathbf{u}_k) \leq 0, \quad k = 0, \ldots, N \end{aligned} \tag{4.2}$$

where $\hat{\mathbf{x}}$ is the current state, $X = [\mathbf{x}_0, \ldots, \mathbf{x}_N]^T$, $U = [\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}]^T$ are the stacked matrices of state and control vectors for each time step, W is the positive semi-definite cost matrix, and $\mathbf{h}_k$ is the constraint function encoding state and actuation constraints: both W and $\mathbf{h}_k$ can change at every timestep. In classical receding horizon fashion, at each timestep, a new state estimate $\hat{\mathbf{x}}$ is obtained, the optimization problem is solved, and the control signal solution corresponding to the first timestep, $\mathbf{u}_0$, is deployed to the system.

Sequential quadratic programming

NMPC problems (4.2) are primarily solved via interior point (IP) or sequential quadratic programming (SQP) methods. SQP-approaches can leverage the fact that the nonlinear programming problems (NLP) solved at adjacent timesteps are quite similar, so that the solution of the NMPC problem at the previous timestep can be used to *warm-start* the solution at the current timestep. This warm-start feature greatly reduces the real-time computational burden, and often a single SQP iteration is sufficient at each timestep to arrive at a close-to-optimal solution of the NMPC problem (Kouzoupis et al., 2018).

In the SQP algorithm, summarized in Algorithm 3, Eq. (4.2) is sequentially approximated by quadratic programs (QPs), whose solutions are Newton directions for performing steps toward the optimal solution of the NLP. The sequence is initialized at an initial guess of the solution, $(X_0^{\text{init}}, U_0^{\text{init}})$, at which the following QP is

Algorithm 3 (Gros et al., 2020) SQP for NMPC at discrete time i

```
1: Input: current state $\hat{\mathbf{x}}_i$, reference trajectory $(X_i^{\text{ref}}, U_i^{\text{ref}})$, initial guess $(X_i^{\text{init}}, U_i^{\text{init}})$
2:
3: while Not converged do
4:     Form $\mathbf{r}_{i,k}, \mathbf{h}_{i,k}, A_{i,k}, B_{i,k}, C_{i,k}, D_{i,k}, H_{i,k}, J_{i,k}$ by (4.4)
5:     Solve (4.3) to get the Newton direction $(\Delta X_i, \Delta U_i)$
6:     Update initial guess with the Newton step: $(X_i^{\text{init}}, U_i^{\text{init}}) \longleftarrow (X_i^{\text{init}} + \Delta X_i, U_i^{\text{init}} + \Delta U_i)$
7: end while
8: Return: NMPC solution $(X_i, U_i) = (X_i^{\text{init}}, U_i^{\text{init}})$
```

iteratively solved and the initial guess updated at each iteration i until convergence:

$$\begin{aligned}
\min_{\Delta X_i, \Delta U_i} \quad & \sum_{k=0}^{N} \begin{bmatrix} \Delta \mathbf{x}_{i,k} \\ \Delta \mathbf{u}_{i,k} \end{bmatrix}^T H_{i,k} \begin{bmatrix} \Delta \mathbf{x}_{i,k} \\ \Delta \mathbf{u}_{i,k} \end{bmatrix} + J_{i,k}^T \begin{bmatrix} \Delta \mathbf{x}_{i,k} \\ \Delta \mathbf{u}_{i,k} \end{bmatrix} \\
\text{s.t.} \quad & \Delta \mathbf{x}_{i,k+1} = A_{i,k} \Delta \mathbf{x}_{i,k} + B_{i,k} \Delta \mathbf{u}_{i,k} + \mathbf{r}_{i,k}, \quad k = 0, \dots, N-1, \\
& C_{i,k} \Delta \mathbf{x}_{i,k} + D_{i,k} \Delta \mathbf{u}_{i,k} + \mathbf{h}_{i,j} \leq 0, \qquad k = 0, \dots, N, \\
& \Delta \mathbf{x}_{i,0} = \hat{\mathbf{x}}_i - \mathbf{x}_{i,0}^{\text{init}},
\end{aligned} \tag{4.3}$$

where $H_{i,k}$ is the Hessian of the NLP Lagrangian (4.2) and

$$\begin{aligned}
& A_{i,k} = \left.\frac{\partial \mathbf{f}_d}{\partial \mathbf{x}}\right|_{\substack{X_i^{\text{init}} \\ U_i^{\text{init}}}}, \; B_{i,k} = \left.\frac{\partial \mathbf{f}_d}{\partial \mathbf{u}}\right|_{\substack{X_i^{\text{init}} \\ U_i^{\text{init}}}}, \; C_{i,k} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\substack{X_i^{\text{init}} \\ U_i^{\text{init}}}}, \; D_{i,k} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{u}}\right|_{\substack{X_i^{\text{init}} \\ U_i^{\text{init}}}}, \\
& \mathbf{r}_{i,k} = \mathbf{f}_d(\mathbf{x}_{i,k}^{\text{init}}, \mathbf{u}_{i,k}^{\text{init}}) - \mathbf{x}_{i,k+1}^{\text{init}}, \qquad \mathbf{h}_{i,k} = \mathbf{h}(\mathbf{x}_{i,k}^{\text{init}}, \mathbf{u}_{i,k}^{\text{init}}), \\
& J_{i,k} = W_{i,k} \begin{bmatrix} \mathbf{x}_{i,k}^{\text{init}} - \mathbf{x}_{i,k}^{\text{ref}} \\ \mathbf{u}_{i,k}^{\text{init}} - \mathbf{u}_{i,k}^{\text{ref}} \end{bmatrix}.
\end{aligned} \tag{4.4}$$

4.3 Learning Lifted Bilinear Dynamics

Modeling assumptions and data collection

EDMD is used to learn approximate finite dimensional lifted bilinear dynamics from data. The system's unknown dynamics are assumed to be control-affine, with $\mathbf{f}, \mathbf{g}$ in (4.1) unknown. I seek to learn a model and design a multi-purpose controller for the unknown system.

I assume that a nominal controller permits us to execute M_t data collection trajectories of length T_t from initial conditions $\mathbf{x}_0^j \in \Omega, j = 1, \dots, M_t$. From each trajectory, $M_s = (T_t/\Delta t)$ state and control actions are sampled at a fixed time interval Δt, resulting in a data set

$$\mathcal{D} = \left(\left(\mathbf{x}_{j,k}, \mathbf{x}'_{j,k}, \mathbf{u}_{j,k} \right)_{k=0}^{M_s-1} \right)_{j=1}^{M_t}, \tag{4.5}$$

where $\mathbf{x}'_{j,k} = \mathbf{x}_{j,k+1}$.

Since the NMPC design requires continuous-time models to be discretized, a discrete-time lifted bilinear model is learned, thereby avoiding potential numerical differentiation and discretization errors. This is further motivated by the existence of discretization procedures that maintain stability properties and the bilinear structure of the original system, such as the trapezoidal rule with zero-order-hold (Phan et al., 2012; Surana et al., 2018).

Supervised learning of unknown dynamics

Define a dictionary of n dictionary functions $\mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$, $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^n$. The choice of the functions can be based on system knowledge (i.e. feature engineering) or be a generic basis of functions such as monomials of the state up to a certain degree. Choosing dictionary functions is an ongoing area of research in Koopman-based learning methods. This will be further addressed in the second bilinear learning method in Section 4.6.

To learn a lifted bilinear dynamic model, the data $\mathcal{D}$ is organized into data matrices X, X', U, where each corresponding column of X, and X' are state samples recorded one sampling interval apart, see (4.6). Then, the lifted data matrix is created by applying $\boldsymbol{\phi}(\mathbf{x})$ to each column of X and X', denoted $Z = \boldsymbol{\phi}(X), Z' = \boldsymbol{\phi}(X')$ by slight abuse of notation. Finally, $Z_{\mathbf{u}}$ is constructed by applying $\boldsymbol{\phi}_{\mathbf{u}}(\mathbf{x}, \mathbf{u})$ to each corresponding pair of columns of X and U, where $\boldsymbol{\phi}_{\mathbf{u}}(\mathbf{x}, \mathbf{u}) = [\boldsymbol{\phi}(\mathbf{x}) \; \boldsymbol{\phi}(\mathbf{x})u_1 \; \dots \; \boldsymbol{\phi}(\mathbf{x})u_m]^T$. Learning can then be formulated as a linear regression problem (4.6).

$$
\begin{aligned}
&\min_{F, G_1, \dots, G_m \in \mathbb{R}^{n\times n}} ||Z' - \begin{bmatrix} F & G_1 & \dots & G_m \end{bmatrix} Z_{\mathbf{u}}||^2 \\
&\min_{C^x \in \mathbb{R}^{d\times n}} ||X - C^{\mathbf{x}} Z||^2
\end{aligned}
\tag{4.6}
$$

$$
\begin{aligned}
X &= \begin{bmatrix} \mathbf{x}_0^1 & \dots \mathbf{x}_{M_s-1}^1 & \dots & \mathbf{x}_0^{M_t} & \dots \mathbf{x}_{M_s-1}^{M_t} \end{bmatrix}, \\
X' &= \begin{bmatrix} \mathbf{x}_1^1 & \dots \mathbf{x}_{M_s}^1 & \dots & \mathbf{x}_1^{M_t} & \dots \mathbf{x}_{M_s}^{M_t} \end{bmatrix}, \\
U &= \begin{bmatrix} \mathbf{u}_0^1 & \dots \mathbf{u}_{M_s-1}^1 & \dots & \mathbf{u}_0^{M_t} & \dots \mathbf{u}_{M_s-1}^{M_t} \end{bmatrix}, \\
Z &= \boldsymbol{\phi}(X), \; Z' = \boldsymbol{\phi}(X'), \; Z_{\mathbf{u}} = \boldsymbol{\phi}_{\mathbf{u}}(X, U) \; .
\end{aligned}
$$

Regularization, such as sparsity-promoting l_1-regularization which has been shown to improve prediction performance and reduce overfitting (Kaiser, Kutz, and S. L. Brunton, 2018) can be added to the regression. Furthermore, learning $C^{\mathbf{x}}$ is not needed if the projection from the lifted space to the original space can be analytically computed for the chosen dictionary. For example, a monomial basis will typically include the state itself. This results in a lifted discrete-time bilinear model of the form

$$
\mathbf{x}_k = C^{\mathbf{x}} \mathbf{z}_k, \quad \mathbf{z}_{k+1} = F\mathbf{z}_k + \sum_{l=1}^{m} G_l \mathbf{z}_k u_{k,l}. \tag{4.7}
$$

Algorithm 4 Koopman NMPC (closed loop)

```
1: Input: reference trajectory $(X_i^{\text{ref}}, U_i^{\text{ref}})$, initial guess $(X_i^{\text{init}}, U_i^{\text{init}})$
2:
3: while Controller is running do
4:     Form $\mathbf{r}_{i,k}, A_{i,k}, B_{i,k}$ using (4.10)
5:     Get and lift current state, $\mathbf{z}_{i,0} = \boldsymbol{\phi}(\hat{\mathbf{x}})$
6:     Solve (4.8) to get the Newton direction $(\Delta X_i, \Delta U_i)$
7:     Update solution, $(X_i, U_i) \leftarrow (X_i^{\text{init}} + \Delta X_i, U_i^{\text{init}} + \Delta U_i)$
8:     Deploy first input $\mathbf{u}_0$ to the system
9:     Construct $(X_{i+1}^{\text{init}}, U_{i+1}^{\text{init}})$ using (4.9)
10: end while
```

4.4 Nonlinear Model Predictive Control Design

Design considerations

Based on Section 4.2, the NMPC problem (4.2) is first reformulated using the identified Koopman bilinear model:

$$\begin{aligned}
\min_{Z,U} \quad & \sum_{i=0}^{N} \begin{bmatrix} \mathbf{z}_{i,k}^{\text{init}} + \Delta \mathbf{z}_{i,k} \\ \mathbf{u}_{i,k}^{\text{init}} + \Delta \mathbf{u}_{i,k} \end{bmatrix}^T W_{i,k} \begin{bmatrix} \mathbf{z}_{i,k}^{\text{init}} + \Delta \mathbf{z}_{i,k} \\ \mathbf{u}_{i,k}^{\text{init}} + \Delta \mathbf{u}_{i,k} \end{bmatrix} \\
\text{s.t.} \quad & \mathbf{z}_{k+1} = F\mathbf{z}_k + \sum_{i=1}^{m} G_i \mathbf{z}_k u_k^{(i)}, \quad k = 0, \dots, N-1 \\
& \mathbf{c}_l \leq C^{\mathbf{x}} \mathbf{z}_k \leq \mathbf{c}_u, \quad \mathbf{d}_l \leq \mathbf{u}_k \leq \mathbf{d}_u, \quad k = 0, \dots, N, \\
& \mathbf{z}_0 = \boldsymbol{\phi}(\hat{\mathbf{x}}).
\end{aligned} \tag{4.8}$$

The optimization constraint set is generally non-convex due to the bilinear term between $\mathbf{z}$, in the lifted dynamic model and the control inputs, $u_1, \dots, u_m$. As a result, the optimization problem (4.8) is solved using sequential quadratic programming (SQP). The non-convex optimization problem (4.8) is sequentially approximated by quadratic programs (QPs), whose solutions are Newton directions for performing steps toward the optimal solution of the nonlinear program (NLP). The initialization and closed loop operation of the controller can be summarized as follows (see Algorithm 4). Before task execution, the SQP algorithm with the Koopman QP subproblem (4.8) is executed to convergence to obtain a good initial guess of the solution. Then, in closed loop operation, the Koopman bilinear model is linearized along the initial guess, the current state is obtained from the system, the current state is lifted using the function basis, and then the QP subproblem is solved only once. Finally, the first control input of the optimal control sequence is deployed to the system, and the full solution is shifted one timestep and used as an initial guess at the next timestep.

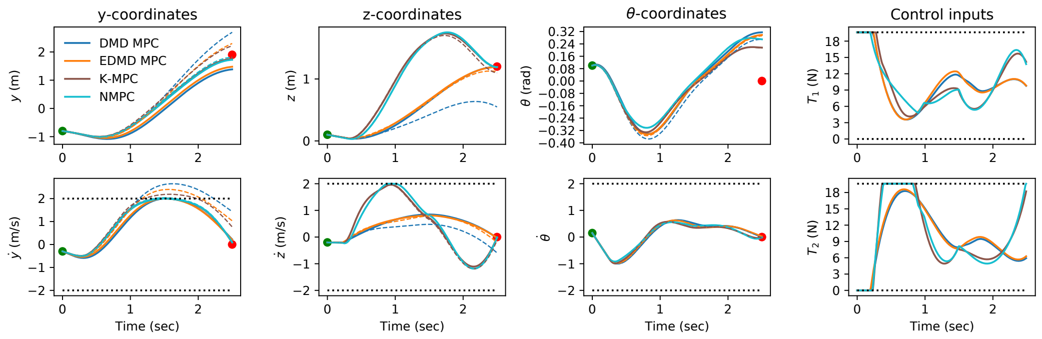


Figure 4.1: Trajectories generated with MPCs based on DMD, EDMD, and bEDMD models. True model-based NMPC used as benchmark. (Black dotted lines-state/actuation constraints, dashed lines-open loop simulation of generated trajectories).

Although I have restricted the objective to be quadratic and the state and actuation constraints to be linear (except for the evolution of the dynamics), nonlinear objective and constraint terms can be included by adding them to the lifted state $\mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$. For example, if it is desired to enforce the constraint $\cos(x_1) \leq 0$, $\phi_j = \cos(x_1)$ can be added to the lifted state and $z_k^{(j)} \leq 0$ enforced (Korda and Mezić, 2018b).

While not the main focus of this section, a discussion of how to achieve guaranteed closed loop stability of the proposed control strategy is in order. In the nominal case, with no model mismatch between the true dynamics and the Koopman bilinear model, closed loop stability of the controller for bilinear systems with a quasi-infinite method has been shown (see e.g. Bloemen, Cannon, and Kouvaritakis, 2002). More recently, some early stability results using Lyapunov MPC methods have been developed (Narasingam and Kwon, 2020). In particular, the bilinear model structure simplifies the construction of a Lyapunov function that is added as a constraint to the MPC. Lyapunov stability of the controller based on the KBF is then proved under the assumption that the prediction error of the learned Koopman model is finite. Although promising, further analysis of robustness and stability properties of the methodology is needed. In this work however, I focus on the practical implementation and defer further theoretical development to future research.

Warm-start of SQP at each timestep

As discussed in Section 4.2, the SQP algorithm requires an initial guess of the solution $X_i^{\text{init}}, U_i^{\text{init}}$. Selecting an initial guess that is sufficiently close to the true optimal solution is essential for the algorithm to converge fast and reliably (Gros et al., 2020). It is well known that the receding horizon nature of MPC can be exploited to obtain excellent initial guesses. At a time instant i, this can be achieved by *shifting* the NMPC solution from the previous timestep $i-1$ and by updating the guess of the final control input. Under certain conditions, a locally stable controller enforcing state and actuation constraints can be designed allowing feasibility of the initial guess to be guaranteed (Rawlings and Mayne, 2012). Typically, simpler approaches are taken such as simply adding a copy of the final control signal and calculating the implied final state using the dynamics model

$$\begin{aligned} \mathbf{u}_{i,k}^{\text{init}} &= \mathbf{u}_{i-1,k+1}, \quad k = 0, \dots, N-2, \\ \mathbf{x}_{i,k}^{\text{init}} &= \mathbf{x}_{i-1,k+1}, \quad k = 0, \dots, N-1, \\ \mathbf{u}_{i,N-1}^{\text{init}} &= \mathbf{u}_{i,N-2}^{\text{init}}, \quad \mathbf{x}_{i,N}^{\text{init}} = \mathbf{f}_d(\mathbf{x}_{i,N-1}^{\text{init}}, \mathbf{u}_{i,N-1}^{\text{init}}). \end{aligned} \tag{4.9}$$

If the previous solution X_{i-1}, U_{i-1} is feasible, the shifted solution will also be feasible for all but the last timestep.

Calculating the linearized system matrices

As a result of the bilinear structure of the dynamics model, the linearization can be efficiently computed for a given initial guess. The linearization at each timestep $k = 0, \dots, N-1$ of the initial guess is obtained by directly calculating the partial derivatives as described in (4.4)

$$\begin{aligned} A_{i,k} &= F + \sum_{j=1}^{m} G_j (u_{i,k}^{\text{init}})^{(j)}, \;\; B_{i,k} = \left[G_1 \mathbf{z}_{i,k}^{\text{init}} \dots G_m \mathbf{z}_{i,k}^{\text{init}} \right] \\ \mathbf{r}_{i,k} &= F \mathbf{z}_{i,k}^{\text{init}} + \sum_{j=1}^{m} G_j \mathbf{z}_{i,k}^{\text{init}} (u_{i,k}^{\text{init}})^{(j)} - \mathbf{z}_{i,k+1}^{\text{init}}. \end{aligned} \tag{4.10}$$

Consequently, the linearized dynamics matrices can be obtained by simple matrix multiplication and addition with the dynamics matrices of the Koopman model and the matrices containing the initial guesses of $Z_i^{\text{init}}, U_i^{\text{init}}$.

4.5 Simulated Quadrotor Learning and Control

System and data collection details

Consider a planar quadrotor with states $\mathbf{x} = [y\, z\, \theta\, \dot{y}\, \dot{z}\, \dot{\theta}]^T$,

$$\begin{bmatrix} \ddot{y} \\ \ddot{z} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} -(1/m)\sin\theta & -(1/m)\sin\theta \\ (1/m)\cos\theta & (1/m)\cos\theta \\ -l_{arm}/I_{xx} & l_{arm}/I_{xx} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}, \tag{4.11}$$

	DMD	EDMD	bEDMD
Mean squared error	8.71e-2	5.60e-2	7.53e-3
Improvement vs DMD		35.75 %	91.35 %
Improvement vs EDMD			86.54 %
Standard deviation	2.79e-1	2.36e-1	8.66e-2
Improvement vs DMD		15.27 %	68,94 %
Improvement vs EDMD			63.35 %

Table 4.1: Prediction error of DMD, EDMD, and bEDMD models.

where y, z describe the horizontal and vertical position in a fixed reference frame, θ is the orientation, T_1, T_2 are the propeller thrusts, g is the acceleration of gravity, m is the vehicle mass, l_{arm} is the distance from the vehicle's center of mass to the propeller axis, and I_{xx} is the rotational inertia.

To collect data, a nominal LQR controller is designed using the linearized dynamics, linearized around hover. Since the system is underactuated, learning trajectories are generated from a MPC based on the system's linearized dynamics. However, any controller can be used and the method does not need a known model linearization. Additionally, exploratory Gaussian white noise is added to aid sufficient excitation. The learning data set is collected as follows. First, an initial condition $\mathbf{x}_0$ and final condition $\mathbf{x}_f$ are sampled uniformly at random from the interval $y, z \in [-2, 2]^2, \theta \in [-\pi/3, \pi/3], \dot{y}, \dot{z}, \dot{\theta} \in [-1, 1]^3$. Then, 2-second long trajectories link $\mathbf{x}_0$ to $\mathbf{x}_f$, which are tracked via the LQR-controller. This process is repeated 100 times as state and actuation data is captured at 100 hz.

To compare the method against the state-of-the art of Koopman-based learning methods, I trained three separate models, dynamic mode decomposition (DMD) (Tu et al., 2014), extended DMD (EDMD) (Li et al., 2017), and the method of Section 4.3, denoted bilinear EDMD (bEDMD). Assuming that the input forces enter through rotation matrices, a simple dictionary of 27 functions was chosen for both the EDMD and bEDMD consisting of the state vector and monomials of the $\theta, \dot{\theta}$ up to the third order multiplied by $1, \cos(\theta), \sin(\theta)$, $\boldsymbol{\phi}(\mathbf{x}) = [1,\ y,\ z,\ \theta,\ \dot{y},\ \dot{z},\ \dot{\theta},\ \theta^2,\ \theta,\ \dot{\theta},\ \ldots,\ \dot{\theta}^3, \cos\theta]^T$. l_1-regularization tuned with cross-validation was also applied to each method. Code for learning and control is implemented in Python[1] and the dynamics are simulated using 5th order Runge-Kutta in `scipy`.

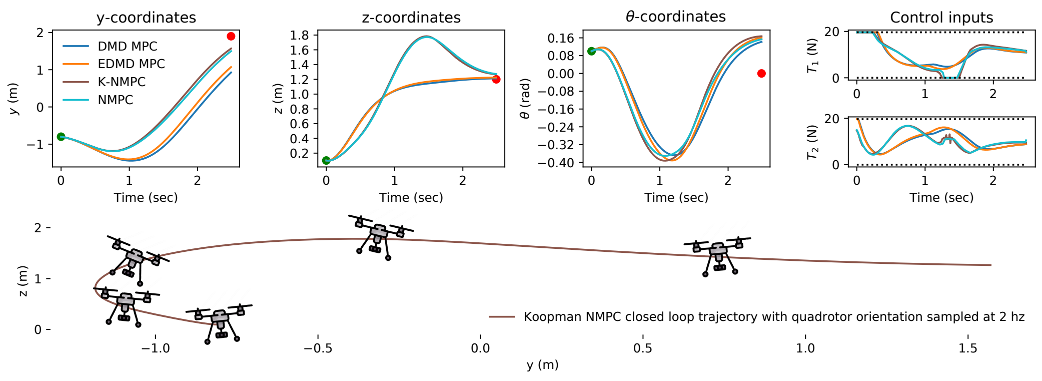


Figure 4.2: Closed loop control with MPCs based on DMD, EDMD, and bEDMD models. True model-based NMPC used as benchmark.

Open loop prediction

I first evaluate our method's prediction performance. A test data set is generated the same way as the training set. Then, the control sequence of each test trajectory is simulated forward with each of the learned models. The mean and standard deviation of the error between the true and predicted evolution over the trajectories are reported in Table 4.1. The experimental results support the theory: the mean and standard deviation of the error is reduced by 86-91 percent and 63 to 69 percent, respectively, compared to DMD and EDMD. bEDMD better captures the nonlinearities in the actuation matrix that drives the (y, z)-dynamics.

Trajectory generation and closed loop control

To study trajectory generation and control, MPCs for each of the learned models are designed. For the linear (DMD) and lifted linear (EDMD) models, a linear MPC is designed. Then, the Koopman NMPC (K-NMPC) is designed as described in Section 4.4. Finally, as a benchmark I implement NMPC using the true dynamics (4.11) based on Section 4.2. Each controller is based on a discrete-time model with sampling length 10 ms. All the optimization problems are solved with `OSQP` (Stellato et al., 2018). I initially study the ability of each controller to generate high quality trajectories. One hundred trajectories (2.5 second duration) are designed to move the system from $\mathbf{x}_0$ to $\mathbf{x}_f$, sampled uniformly at random from the interval $y, z \in [-2, 2]^2, \theta \in [-0.1, 0.1], \dot{y}, \dot{z}, \dot{\theta} \in [-1, 1]^3$. The Frobenius norm of the control inputs over the prediction horizon is minimized and a terminal state constraint

[1]Code available at `github.com/Cafolkes/koopman-learning-and-control`

is added to each of the controllers to ensure that the desired terminal position is reached. Finally, the velocities are constrained to have magnitude less than 2, $\dot{y}, \dot{z}, \dot{\theta} \in [-2, 2]$, and the thrust of each propeller is limited, $T_1, T_2 \in [0, 2T_{hover}]$.

The generated trajectories from one of the experiments (solid lines) along with the open-loop simulation of the true dynamics with the control sequence of each designed trajectory (dashed lines) are depicted in Figure 4.1. Table 4.2 presents summary statistics from 100 experiments: total control effort (as measured by the Frobenious norm and normalized by the NMPC control effort), the terminal state error (the Euclidean distance between the final open loop stimulation state and the desired state), and the number of SQP iterations needed by K-NMPC and NMPC. The open loop simulation reveals that the trajectories resulting from the DMD and EDMD models are not realizable, leading to significant mean terminal state errors of 2.24 and 2.46, respectively. K-NMPC has a significantly lower mean error of 0.70, and, more importantly, captures the idealized behavior of NMPC, even though it is completely data-driven.

Finally, I study closed loop control behavior of each control approach over the same 100 initial and terminal conditions. Each of the MPCs use a 0.5 second prediction horizon with sampling length 10 ms and a quadratic state penalty and control input cost. The Koopman NMPC and NMPC controllers are initialized by solving each of the NLPs to convergence with the SQP algorithm before only a single SQP iteration is performed at each timestep in closed loop.

The traces resulting from each controller are presented in Figure 4.2 for one of the experiments. Furthermore, summary statistics over the 100 experiments of the realized cost (as measured by the total trajectory cost, normalized by the NMPC cost), and the computation time at each timestep of each of the controllers are reported in Table 4.3. Because closed loop operation can correct for model errors, the performance difference between the controllers is smaller than for the trajectory

	DMD (MPC)		EDMD (MPC)		bEDMD (K-NMPC)		Benchmark (NMPC)	
	mean	std	mean	std	mean	std	mean	std
Control effort	0.97	0.02	0.97	0.06	1.01	0.01	1.00	0.00
Terminal error	2.24	1.50	2.46	1.77	0.70	0.31	0.36	0.17
SQP iterations					21.88	11.57	9.33	9.63

Table 4.2: Summary statistics over 100 experiments of the MPC trajectory cost, error, and SQP iterations.

generation case. The controllers based on the DMD and EDMD models achieve a 4 and 2 percent higher cost than the NMPC, respectively. The K-NMPC again closely follows the behavior of the NMPC.

The linear and lifted linear MPCs require less computational effort than the SQP-based approaches with an average computation time of 2 and 7 ms, respectively. K-NMPC requires somewhat higher computational effort than NMPC for this system with an average of 13 ms compared to 7 ms. This is dominated by longer solution time of the QP because a higher number of variables and constraints as a result of the lifting. The relative computational effort between K-NMPC and NMPC will ultimately depend on the complexity of linearizing the nonlinear model for NMPC, which can be expensive for complicated models, as well as the lifting dimension of the Koopman bilinear model. Finally, I note that even with a relatively simple python implementation, the controllers are approaching real-time capability.

	DMD (MPC)		EDMD (MPC)		bEDMD (K-NMPC)		Benchmark (NMPC)	
	mean	std	mean	std	mean	std	mean	std
Realized cost	1.04	0.05	1.02	0.03	1.00	0.00	1.00	0.00
Comp time (ms)	1.50	0.34	7.40	1.78	13.14	8.35	6.99	1.11

Table 4.3: Summary statistics over 100 experiments of the MPC closed loop control cost and computation times.

4.6 Joint Learning of the Koopman Dictionary and Model

I now turn to the second learning method that encodes the bilinear structure in a neural network model that jointly learns the function dictionary and bilinear model from data. To formulate the learning problem, an analogous approach to the bEDMD method in Section 4.3 is followed. However, unlike bEDMD, the function dictionary is now parametrized by a neural network, and the function dictionary and bilinear model matrices are jointly learned. Let $\boldsymbol{\theta}$ parametrize the neural network representing the function dictionary and $\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta})$, and $F, G_1, \dots, G_m, C^{\mathbf{x}}$ parametrize the bilinear model matrices described in Section 4.3.

The key idea of the learning method is that both the function dictionary parametrization and the bilinear model structure can be encoded in a single neural network. Then, once training is complete, the learned function dictionary and model matrices can be extracted to maintain the benefits of Koopman-based learning methods outlined in Section 4.1, while improving the prediction performance and/or reducing the dimension of the function dictionary. To achieve this, I formulate a loss function, $\mathcal{L}$ to be minimized by empirical risk minimization over all the input-output pairs in the training data set, $\mathcal{D}$:

$$\mathcal{L}(\mathbf{x},\mathbf{u},\mathbf{x}') = \alpha\mathcal{L}_{\text{rec}}(\mathbf{x},\mathbf{x}') + \mathcal{L}_{\text{pred}}(\mathbf{x},\mathbf{u},\mathbf{x}') + \beta\mathcal{L}_{\text{kct}}(\mathbf{x},\mathbf{u},\mathbf{x}'),$$

$$\begin{aligned}
\mathcal{L}_{\text{rec}}(\mathbf{x},\mathbf{x}') &= ||\mathbf{x} - C^{\mathbf{x}}\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta})||^2, \\
\mathcal{L}_{\text{pred}}(\mathbf{x},\mathbf{u},\mathbf{x}') &= ||\mathbf{x}' - C^{\mathbf{x}}\Big(F\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta}) + \sum_{i=1,\dots,m} G_i\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta})\mathbf{u}_i\Big)||^2, \\
\mathcal{L}_{\text{kct}}(\mathbf{x},\mathbf{u},\mathbf{x}') &= ||\boldsymbol{\phi}(\mathbf{x}';\boldsymbol{\theta}) - \Big(F\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta}) + \sum_{i=1,\dots,m} G_i\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta})\mathbf{u}_i\Big)||^2,
\end{aligned} \tag{4.12}$$

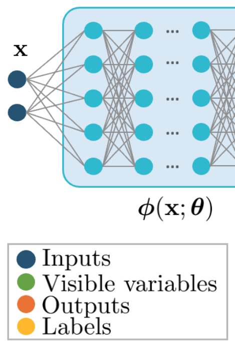
where

- $\mathcal{L}_{\text{rec}}$ is the mean squared error (MSE) of the reconstruction loss when lifting the system's state using the encoder $\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta})$ and projecting back down to the original system state with the projection matrix $C^{\mathbf{x}}$,
- $\mathcal{L}_{\text{pred}}$ is the MSE of the one-step prediction error of the model when projected down in the original state space,
- and $\mathcal{L}_{\text{kct}}$ is the MSE of the one-step prediction error in the lifted space.

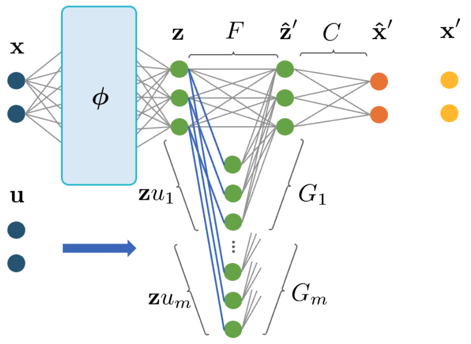
Tunable hyperparameters α and β determine the weight of the prediction and KCT losses, respectively, relative to the reconstruction loss.

The learned model should realize good prediction performance in the original state space: minimizing $\mathcal{L}_{\text{pred}}$ encourages this goal. Loss terms $\mathcal{L}_{\text{recon}}$ and $\mathcal{L}_{\text{kct}}$ respectively promote accurate projection of the function dictionary to the original state space and an approximately bilinear dynamical system model in the lifted space. Loss $\mathcal{L}_{\text{kct}}$ promotes good prediction accuracy over multiple time steps, as multi-step prediction is performed in the lifted space before the result is projected to the original state space only at relevant times of interest.

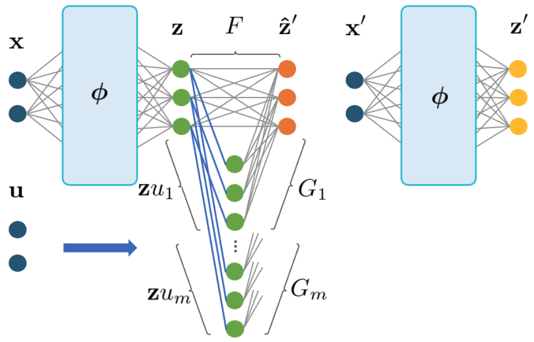
Figure 4.3 depicts the neural network architecture that implements loss (4.12). The autoencoder (Fig. 4.3a) passes the system state $\mathbf{x}$ through the encoder, $\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta})$, to obtain the lifted state $\mathbf{z}$. Subsequently, the lifted state is projected back to the original state space through the projection matrix $C^{\mathbf{x}}$, resulting in the reconstructed



(a) Autoencoder model



(b) State prediction model



(c) Lifted state prediction model

Figure 4.3: Koopman neural network model architecture.

state, $\hat{\mathbf{x}}$, which is compared to the original state, $\mathbf{x}$, in $\mathcal{L}_{\text{recon}}$. The state prediction model (Fig. 4.3b) passes the system state through the encoder to obtain the lifted state, $\mathbf{z}$, before evolving the lifted state one time-step with the bilinear model based on $\mathbf{z}$ and $\mathbf{u}$ to get the one-step-ahead lifted state, $\mathbf{z}'$. The lifted state prediction is projected to the original state space to get the one-step-ahead state prediction, $\hat{\mathbf{x}}'$, which is compared to the true one-step-ahead state, $\mathbf{x}'$, in $\mathcal{L}_{\text{pred}}$. Finally, the lifted state prediction model (Fig. 4.3c) follows the same forward pass through the same state prediction model to get the one-step-ahead lifted state prediction $\hat{\mathbf{z}}'$. This is compared to the true one-step-ahead lifted state $\mathbf{z}'$ in $\mathcal{L}_{\text{kct}}$, obtained by passing the true one-step-ahead state, $\mathbf{x}'$ through the encoder $\boldsymbol{\phi}$.

This approach can be implemented using modern neural network software packages with basis functions $\boldsymbol{\phi}$ modeled as a feedforward neural network having fully connected layers and $F, G_1, \dots, G_m, C^{\mathbf{x}}$ as single fully connected layers with no nonlinear activations. The entire model can be trained simultaneously using gradient-based learning algorithms applied to the loss function (4.12).

4.7 Experimental Low Altitude Trajectory Tracking with Quadrotor Drone

Experiments are conducted to demonstrate the performance of the proposed method on a quadrotor drone. To capture the nonlinearity in the quadrotor dynamics, 6-second long "Figure 8" tracking maneuvers are performed (see Fig. 4.5b) at very low altitudes above the ground so that aerodynamic "ground effect" corrupts the nominal model (Sanchez-Cuevas, Heredia, and Ollero, 2017). The $x - y$ coordinates of the "Figure 8" trajectory require high roll and pitch maneuvers and the low altitude flights highlight the effect of unmodeled dynamics, encapsulated in the $\mathbf{f}_\mathbf{v}$ and $\boldsymbol{\tau}_\omega$ terms in (4.13).

Modeling assumptions

The quadrotor dynamics are modeled using states global position $\mathbf{p} \in \mathbb{R}^3$, velocity $\mathbf{v} \in \mathbb{R}^3$, attitude rotation matrix $R \in \mathrm{SO}(3)$, and body angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$, and consider the following dynamics:

$$
\begin{aligned}
\mathbf{p} &= \mathbf{v}, & m\mathbf{v} &= m\mathbf{g} + R\mathbf{f} + \mathbf{f}_\mathbf{v}(\mathbf{p},\mathbf{v},R,\boldsymbol{\omega},\mathbf{f},\boldsymbol{\tau}), \\
\dot{R} &= RS(\boldsymbol{\omega}), & J\dot{\boldsymbol{\omega}} &= J\boldsymbol{\omega} \times \boldsymbol{\omega} + \boldsymbol{\tau} + \boldsymbol{\tau}_\omega(\mathbf{p},\mathbf{v},R,\boldsymbol{\omega},\mathbf{f},\boldsymbol{\tau}),
\end{aligned}
\tag{4.13}
$$

where m and J are the vehicle mass and inertia matrices, respectively, $S(\cdot)$ is a 3×3 skew symmetric mapping, and $\mathbf{g} = [0, 0, -g]^T$ is the gravitational force vector. The state and control-dependent functions $f_\mathbf{v}(\cdot)$ and $\tau_\omega(\cdot)$ capture unmodelled dynamic terms, such as the aerodynamic ground effect.

As is common, I abstract the system's control inputs to a total thrust in the body z-direction, $\mathbf{f} = [0, 0, T]^T$, and body torques $\boldsymbol{\tau} = [\tau_x, \tau_y, \tau_z]^T$, $\mathbf{u} = [T, \tau_x, \tau_y, \tau_z]^T$. The mapping of rotor speeds to the abstract controls is typically modeled by a linear combination of the squared rotor speeds:

$$
\mathbf{u} = \mathcal{T}\boldsymbol{\eta}, \quad \mathcal{T} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ 0 & c_T l & 0 & -c_T l \\ -c_T l & 0 & c_T l & 0 \\ -c_Q & c_Q & -c_Q & c_Q \end{bmatrix}, \quad \boldsymbol{\eta} = \begin{bmatrix} \eta_1^2 \\ \eta_2^2 \\ \eta_3^2 \\ \eta_4^2 \end{bmatrix},
\tag{4.14}
$$

where c_T and c_Q are the propeller thrust and torque coefficients, l is the distance from the vehicle's center of gravity to the propeller axle, and $[\eta_1, \ldots, \eta_4]^T$ are the propeller rotation rates. This mapping enables the abstracted controls to be translated into propeller rotational rates.

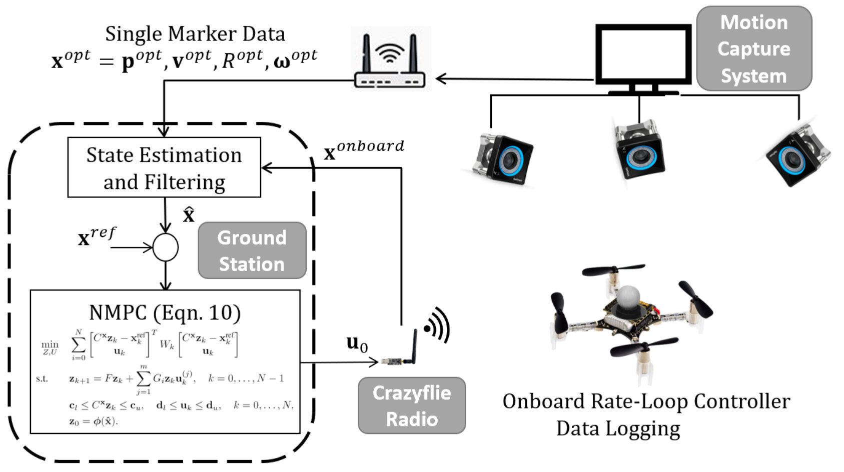


Figure 4.4: Experiment set-up.

Implementation and experimental details

The experiments are conducted with a commercially available *Crazyfile 2.1* quadrotor drone. Global position is measured via an *OptiTrack* motion capture system (tracking at 120 Hz) and fused with the onboard IMU data to get filtered state estimates. The all-up quadrotor weight is 33.6 g, with a thrust to weight ratio of 1.7. Control commands are computed on an offboard computer and communicated to the drone over radio, see Fig. 4.4. All communication with the drone is performed using the *Crazyswarm* Python API (Preiss et al., 2017).

To collect training data, waypoints were sampled uniformly at random within $x, y, z \in [-0.3, 0.3] \times [-0.3, 0.3] \times [0.1, 1.0]$ meters and tracked with a PID position controller for a total of 4 minutes. The estimated position, attitude, linear and angular velocites, motor pulse width modulation (PWM) signals, and battery voltage were collected at 500 Hz. The state estimates where downsampled and the

Table 4.4: Learning architecture and tuned hyperparameter values

# of encoder layers	2	Learning rate	1e-3
Layer width	100	KCT loss penalty, β	2e-1
Activation function	tanh	l_2-regularization strength	5e-4
Total lifting dimension	23	# of epochs	200

Table 4.5: Crazyflie system properties and NMPC parameters

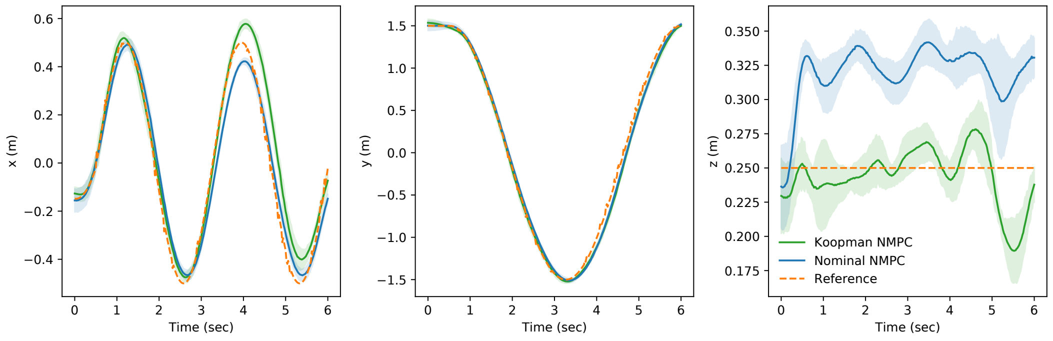
Mass	33.6 g	State error penalty, Q	[10, 10, 10]
Max thrust	57.0 g	Control penalty, R	[1, 1, 1, 1]
J_{xx} (Landry et al., 2016)	1.7e-5 kg $\cdot$ m^2	NMPC prediction	
J_{yy} (Landry et al., 2016)	1.7e-5 kg $\cdot$ m^2	horizon	0.5 s
J_{zz} (Landry et al., 2016)	2.9e-5 kg $\cdot$ m^2	Controller timestep	0.02 s

PWMs and voltage averaged over each 0.02 s interval to obtain a dataset at 50 Hz, the target update rate for the controller. This smoothing process better captures motor PWM inputs, as the raw data showed significant variability within each 0.02 s period. Finally, PWMs and voltage are mapped to motor thrust commands using the model identified in G. Shi, Hönig, et al., 2020 (see Tab. 1, G. Shi, Hönig, et al., 2020) and the thrust mixing in (4.14) is applied to yield total thrust and torques around each axis, resulting in the inputs used in the learning process.

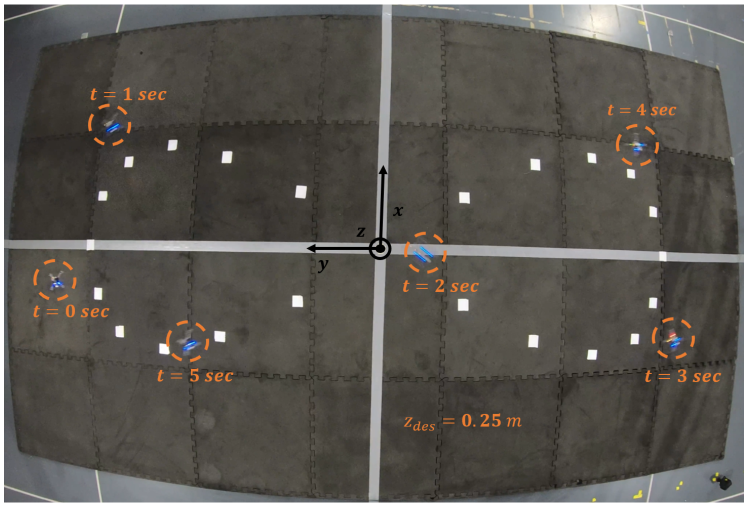
A discrete-time lifted-dimensional bilinear model is learned as described in Section 4.6. Thirty percent of the data set is held out for validation and testing and the hyperparameters are tuned to obtain good open loop prediction performance over 0.5 seconds, the same prediction horizon used in the model predictive controllers. The final parameters used are included in Tab. 4.4. Note that determining the lifting dimension is a part of the hyperparameter tuning process, and keeping the lifting dimension low is more important than keeping the width and number of layers low as the lifting dimension directly affects the NMPC solution time. The neural network is implemented using *PyTorch* (Paszke et al., 2019)[2]. To simplify the process of encoding the NMPC objective and constraints, the state itself is added to the lifted state, $\mathbf{z} = [\mathbf{x}^T, \boldsymbol{\phi}(\mathbf{x}; \boldsymbol{\theta})^T]^T$, making the projection matrix known, $C^{\mathbf{x}} = [I, 0]$. This makes the reconstruction loss, $\mathcal{L}_{\text{recon}}$ in (4.12) redundant and I therefore set $\alpha = 0$. In this experiment, setting the total lifting dimension (including the state itself) to 23 led to a good compromise between lifting dimension and prediction and generalization performance.

Two controllers are implemented for the task. First, a nominal NMPC using the dynamic model (4.13) with $\boldsymbol{\tau}_\omega = \mathbf{f}_\mathbf{v} = \mathbf{0}$ and the system parameters described in Tab. 4.5, are used as a performance baseline. Second, the Koopman NMPC described in Section 4.2, using the learned model, is implemented. Both controllers are coded in *Python* using the *OSQP* quadratic program solver (Stellato et al., 2018). The objective penalizes errors between the state and desired trajectory in x, y, z. The control inputs are constrained using the vehicle's mechanical limitations. The position error penalties, $Q = [q_x, q_y, q_z]$, and control effort penalties, $R = [r_T, r_{\tau_x}, r_{\tau_y}, r_{\tau_z}]$, and remaining control parameters are described in Tab. 4.5. The control thrusts and desired attitude are calculated by the NMPCs and sent to the

[2]Code available at `github.com/Cafolkes/koopman-learning-and-control`



(a) Mean position coordinates (solid lines) +/- 3 std (shaded area) of 10 consecutive experiment runs with both controllers at 0.25 m altitude.



(b) “Figure 8” tracking with position depicted at selected times.

	Nominal NMPC	Koopman NMPC
Avg. position tracking error (mse)	4.7e-3	4.8e-3
Avg. control effort (thrust norm)	10.7	10.5

Figure 4.5: Hardware configuration and experimental results from “Figure 8” trajectory tracking control experiment.

onboard drone controller, which decides the final motor control allocations. This architecture enables both controllers to cycle at 50 Hz while attitude tracking runs at a higher rate onboard the drone.

Results and discussion

Fig. 4.5 details the tracking performance of the nominal and Koopman NMPC closed-loop controllers tracking a 6-second long “Figure 8” trajectory over 10 consecutive experiment runs. The yaw and pitch performance is comparable, indicating

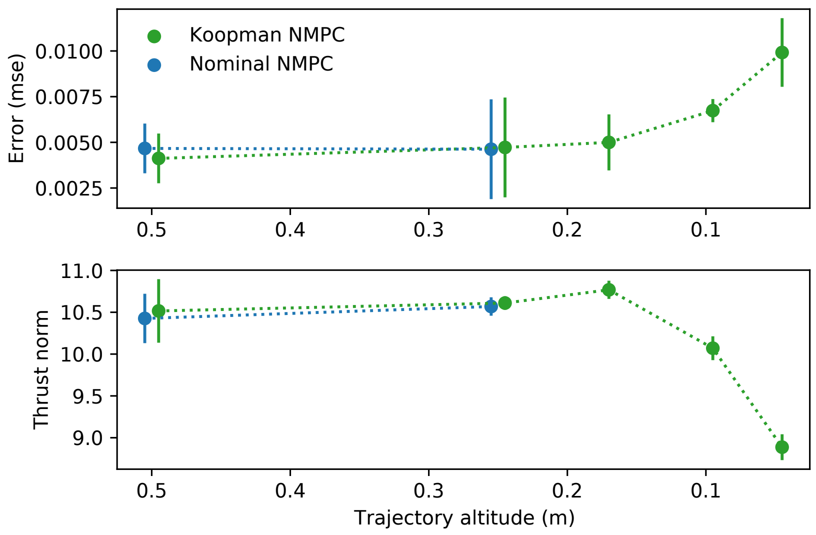


Figure 4.6: Average MSE and total thrust (dots) +/- 3 std (error bars) from 5 consecutive experiment runs with each of the controllers at decreasing altitudes. Stable flight not achieved by the nominal NMPC for altitudes below 0.25 m.

accurate identification of the roll-pitch-yaw dynamics. Furthermore, the large z-tracking error from the nominal NMPC induced by the ground effect is successfully captured by the Koopman learned dynamics. Note that the drone both starts and ends at zero velocity, causing increased tracking error at the beginning and end of the trajectory.

To further the impact of the ground effect on tracking performance, 5 “Figure 8” trajectories were executed at decreasing altitudes, as shown in Fig. 4.6. At altitudes 0.25 m and higher, nominal and Koopman NMPCs tracking performance is similar. However, at lower altitudes nominal NMPC stability fails, leading to catastrophic crashes. In contrast, the Koopman NMPC completed all five tests, albeit with increased tracking error as the altitude setpoint approaches 0.05 m above the ground. I also note that the Koopman NMPC exploits the buoyancy in resulting from the ground effect at lower altitudes to significantly reduce the thrust needed to complete the task. Furthermore, both controllers demonstrated good repeatability (when the nominal NMPC did not catastrophically fail), by being able to repeat the experiments many times with no signs of failure, and robustness to minor wind gusts in the semi-outdoor arena where the experiments were conducted.

4.8 Conclusion

This chapter has presented methods that couple Koopman-based bilinear models and NMPC in order to allow for real-time optimal control of robots that captures important nonlinearities, while allowing for critical state and control limits.

The first method showed how fixed function dictionaries can be designed to use EDMD-based learning to identify a lifted bilinear model from data. Through a simulated planar quadrotor example, I demonstrated the advantages of learning lifted bilinear models over lifted linear models to capture control-affine dynamics. I also showed that the resulting data-driven controller could achieve similar performance to the case of NMPC design with an exactly known dynamic model.

However, function dictionary design is a key challenge in Koopman-based modeling and control methods and fixed dictionaries typically result in high dimensional function dictionaries to achieve the desired prediction performance. The second learning method therefore jointly learns a lifted Koopman bilinear model and KCT function dictionary strictly from data, enabling more compact models and/or better prediction performance compared to predefined function dictionaries. More compact models, i.e. lower lifting dimension, are crucial for robotic applications where real-time control is needed. Data-driven models are valuable in cases where first-principles modeling may be difficult. The quadrotor drone experiments demonstrate good prediction performance with a lifting dimension of only 23. The associated Koopman NMPC can match the performance of a NMPC based on the nominal model (4.13) far away from the ground, and outperforms the nominal NMPC in near-ground regimes. Notably, the nominal controller is unable to maintain stable flight near the ground, whereas the Koopman NMPC maintains acceptable tracking performance down to 0.05 m altitude.

Chapter 5

SAFETY-CRITICAL CONTROL WITH DATA-DRIVEN CONTROL BARRIER FUNCTIONS

This chapter describes a data-driven method to both increase computational efficiency and improve the performance of safety-critical systems operating under safety filters based on control barrier functions. The method was first presented in the IEEE Control Systems Letters (Folkestad, Y. Chen, et al., 2021).

5.1 Introduction

The field of safety-critical control has received increasing attention since safety is a primary requirement for important autonomous systems, such as autonomous cars and robots. While control barrier functions (CBFs) (Ames, Xu, et al., 2017; Borrmann et al., 2015) can provide safety guarantees for autonomous systems, the feasibility of this approach in the presence of input bounds relies on control invariant sets, which may be difficult to compute (Raković et al., 2006; Blanchini, 1999; Y. Chen, Peng, et al., 2019). In (Gurriet, Mote, et al., 2019), the authors proposed a CBF approach that uses an implicitly defined control invariant set based on a *backup strategy*, computed by forward integrating the dynamics. The approach was extended to multi-agent systems in a fully decentralized manner in (Y. Chen, Singletary, and Ames, 2020). However, online evaluation of the CBF and its Lie derivative requires forward integration of the system dynamics and a sensitivity matrix, which can be a bottleneck for nonlinear and high dimensional systems. Moreover, when parts of the dynamics are unmodelled, the inaccurate integration can lead to safety violations.

I propose to learn an approximate Koopman operator for the closed-loop dynamics under the backup strategy. This replaces the expensive forward integration of the dynamics and sensitivity matrix with matrix multiplication. I also develop an error bound on the learned model that supports a robust version of the supervisory controller: it guarantees safety when the learned model forward propagates the backup trajectory. Additionally, a Koopman operator trained on real system data produces more accurate backup trajectories, especially in the presence of unmodeled dynamics, which improves system safety guarantees and, potentially, the system's mobility.

Koopman inspired modelling and identification techniques have received substantial attention (Rowley et al., 2009; Budišić, Mohr, and Mezić, 2012). In particular, the Dynamic Mode Decomposition (DMD) and extended DMD (EDMD) methods efficiently identify finite dimensional approximations of the Koopman operator (Schmid, 2010; M. O. Williams, Kevrekidis, and Rowley, 2015). However, as most prior work focused on learning unknown dynamics, utilizing Koopman-based learning to improve the computation efficiency and dynamic model uncertainty for safety-critical applications has not been previously explored.

Building upon (Gurriet, Mote, et al., 2019), I introduce a data-driven approach that combines Koopman-based learning and CBFs to achieve a safety-critical control system that

1. guarantees safety under limited actuation and errors in the learned Koopman model.
2. requires little online computation to implement.
3. can learn backup trajectories from data, improving the applicability of the approach in real-world scenarios where accurate models may be unavailable.

Our experiments and simulations show that the method can be incorporated in the decentralized framework of (Y. Chen, Singletary, and Ames, 2020), further expanding the impact of our efficiency improvements.

The rest of this chapter is organized as follows. Section 5.2 reviews CBFs, the backup approach to craft CBFs, and Koopman theory. Section 5.3 describes how to learn Koopman operators of closed-loop dynamics under a backup strategy. Section 5.4 introduces a CBF controller using the Koopman model. Section 5.5 presents experimental results.

5.2 Preliminaries

Control barrier functions

Using CBFs (Ames, Grizzle, and Tabuada, 2014; Ames, Xu, et al., 2017), a supervisory controller can be designed to maintain system safety with minimum intervention. Specifically, consider the control-affine dynamic system:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{a}(\mathbf{x}) + \mathbf{b}(\mathbf{x})\mathbf{u}, \tag{5.1}$$

where $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d, \mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^m$, $\mathbf{a} : \mathcal{X} \to \mathbb{R}^d, \mathbf{b} : \mathcal{X} \to \mathbb{R}^{d\times m}$. Suppose we can encode the safety criterion as a CBF, $h : \mathbb{R}^d \to \mathbb{R}$, that satisfies

$$\begin{aligned} &\forall\, \mathbf{x} \in \mathcal{X}_0, && h(\mathbf{x}) \geq 0 \\ &\forall\, \mathbf{x} \in \mathcal{X}_d, && h(\mathbf{x}) < 0 \\ &\forall\, \mathbf{x} \in \mathcal{X}, \quad \exists\, \mathbf{u} \in \mathcal{U} \text{ s.t. } && \dot{h} + \alpha\left(h\right) \geq 0, \end{aligned} \tag{5.2}$$

where $\mathcal{X}_0$ is the set of initial states and $\mathcal{X}_d$ are the states to avoid. $\alpha(\cdot)$ is an extended class-$\mathcal{K}$ function, i.e., $\alpha(\cdot)$ is strictly increasing and satisfies $\alpha(0) = 0$. For any legacy controller, the supervisory CBF controller constraints the state inside $\{\mathbf{x} \mid h(\mathbf{x}) \geq 0\}$ via the following quadratic program:

$$\begin{aligned} \mathbf{u}^\star = &\mathop{\arg\min}_{\mathbf{u}\in\mathcal{U}} \left\|\mathbf{u} - \mathbf{u}^0(\mathbf{x})\right\|^2 \\ &\text{s.t. } \nabla h \cdot \mathbf{f}\left(\mathbf{x}, \mathbf{u}\right) + \alpha\left(h(\mathbf{x})\right) \geq 0, \end{aligned} \tag{5.3}$$

where $\mathbf{u}^0(x)$ is the input of the legacy controller. To ensure that (5.3) is always feasible when $h(\mathbf{x}) \geq 0$, $\{\mathbf{x}|h(\mathbf{x}) \geq 0\}$ need to be a control invariant set, which is defined as follows.

Definition 5.1. A set $\mathcal{S} \subseteq \mathbb{R}^d$ is a control invariant set if there exists a control law $\boldsymbol{\pi} : \mathbb{R}^d \to \mathcal{U}$ such that for all initial conditions $\mathbf{x}(0) \in \mathcal{S}$, $\forall t \geq 0, \Phi^t_{\mathbf{f}_\pi}(\mathbf{x}(0)) \in \mathcal{S}$.

Here, $\mathbf{f}_\pi$ denotes the closed-loop dynamics under controller $\boldsymbol{\pi}$, and $\Phi^t_{\mathbf{f}_\pi}(\mathbf{x})$ denotes the system flow map: $\Phi^t_{\mathbf{f}_\pi}(\mathbf{x})$ is the state at t, following $\mathbf{f}_\pi$, starting at $\mathbf{x}$.

Control barrier function with backup controller

This section reviews a backup strategy for control invariant set generation (Gurriet, Mote, et al., 2019). Given system (5.1), suppose the following constraint must hold for all $t \geq 0$ to ensure system safety:

$$\mathbf{x}(t) \in \mathcal{C} \doteq \{\mathbf{x} \in \mathcal{X} | h^{\mathcal{C}}(\mathbf{x}) \geq 0\},$$

where the smooth function $h^{\mathcal{C}} : \mathbb{R}^d \to \mathbb{R}$ defines the safe set $\mathcal{C}$. If a control invariant set $\mathcal{S} \subseteq \mathcal{C}$ is known, a CBF can be constructed and the supervisory controller (5.3) guarantees that the state will remain in $\mathcal{S}$, and thus in $\mathcal{C}$, for any initial condition $\mathbf{x}(0) \in \mathcal{S}$. However, computing $\mathcal{S}$ is often difficult. To bypass this problem, the backup strategy approach was introduced in (Gurriet, Mote, et al., 2019) and (Singletary, Nilsson, et al., 2019). Suppose we find a small initial control invariant set, e.g. by linearizing the dynamics, $\mathcal{S}_0 \doteq \{\mathbf{x}|h^{\mathcal{S}}(\mathbf{x}) \geq 0\} \subseteq \mathcal{C}$, where $h^{\mathcal{S}}$ is smooth. Any equilibrium point of (5.1) inside C satisfies the requirement. Then, for a backup strategy $\boldsymbol{\pi} : \mathbb{R}^d \to \mathcal{U}$ and horizon T, define

$$\mathcal{S} \doteq \{\mathbf{x} \in \mathcal{X} | \Phi^T_{\mathbf{f}_\pi}(\mathbf{x}) \in \mathcal{S}_0 \wedge \forall t \in [0, T], \Phi^t_{\mathbf{f}_\pi}(\mathbf{x}) \in C\}, \tag{5.4}$$

which is the set of all initial conditions from which the backup strategy would bring the state to $\mathcal{S}_0$ at $t = T$ while satisfying the constraint $\mathbf{x}(t) \in C, \forall t \in [0, T]$.

Since $\mathcal{S}_0$ is a control invariant set, by Definition 5.1, there exists a control law $\boldsymbol{\pi}_0$ that keeps any state starting inside $\mathcal{S}_0$ within $\mathcal{S}_0$. Therefore, $\boldsymbol{\pi}|_{\mathcal{S}_0} = \boldsymbol{\pi}_0|_{\mathcal{S}_0}$ is fixed such that any state reaching $\mathcal{S}_0$ will remain within $\mathcal{S}_0$ under $\boldsymbol{\pi}$. As a result, $\mathcal{S}$ contains the initial condition that can reach $\mathcal{S}_0$ in $[0, T]$ (instead of exactly at T) while satisfying the state constraint. This result and how to construct a CBF from $\mathcal{S}$ are summarized in the two following lemmas.

Lemma 5.2 (Y. Chen, Singletary, and Ames, 2020)**.** *$\mathcal{S}$ is a control invariant set and $\mathcal{S}_0 \subseteq \mathcal{S} \subseteq C$.*

Lemma 5.3 (Y. Chen, Singletary, and Ames, 2020)**.** *$\mathcal{S}$ is the 0-level set of the following function*

$$h(\mathbf{x}) = \min\{\min_{t \in [0,T]} h^C(\Phi^t_{\mathbf{f}_\pi}(\mathbf{x})), h^S(\Phi^T_{\mathbf{f}_\pi}(\mathbf{x}))\}. \tag{5.5}$$

Using Lemma 5.3, a modified constraint set of the controller (5.3) enforces the system to satisfy both h^C and h^S:

$$\begin{aligned} \mathbf{u}^* = &\operatorname{argmin}_{\mathbf{u} \in \mathcal{U}} ||\mathbf{u} - \mathbf{u}^0||^2 \\ &\text{s.t.}\, \forall t \in [0, T], \frac{dh^C(\Phi^t_{\mathbf{f}_\pi}(\mathbf{x}))}{dt}(\mathbf{x}, \mathbf{u}) + \alpha(h^C(\Phi^t_{\mathbf{f}_\pi}(\mathbf{x}))) \geq 0 \\ &\frac{dh^S(\Phi^T_{\mathbf{f}_\pi}(\mathbf{x}))}{dt}(\mathbf{x}, \mathbf{u}) + \alpha(h^S(\Phi^T_{\mathbf{f}_\pi}(\mathbf{x}))) \geq 0, \end{aligned} \tag{5.6}$$

where for any $t \in [0, T]$, the total derivative of $h^C(\Phi^t_{\mathbf{f}_\pi}(\mathbf{x}))$ is computed as

$$\frac{dh^C(\Phi^t_{\mathbf{f}_\pi}(\mathbf{x}))}{dt} = \nabla h^C \frac{d\Phi^t_{\mathbf{f}_\pi}}{dt} = \nabla h^C (\nabla \Phi^t_{\mathbf{f}_\pi}(\mathbf{x}) f(\mathbf{x}, \mathbf{u}) - \frac{\partial \Phi^t_{\mathbf{f}_\pi}}{\partial t}),$$

where $\nabla \Phi^t_{\mathbf{f}_\pi}$ is the sensitivity matrix and $-\frac{\partial \Phi^t_{\mathbf{f}_\pi}}{\partial t}$ accounts for the nominal flow under the backup strategy. For fixed dynamics, $\mathbf{f}_\pi$, and initial condition x, denote the sensitivity matrix at time t as $Q_{\mathbf{x},\mathbf{f}_\pi}(t) = \nabla \Phi^t_{\mathbf{f}_\pi}$. Then, the evolution of $Q_{\mathbf{x},\mathbf{f}_\pi}(t)$ can be described by the following ordinary differential equation

$$Q_{\mathbf{x},\mathbf{f}_\pi}(0) = I, \quad \dot{Q}_{\mathbf{x},\mathbf{f}_\pi} = \nabla \mathbf{f}_\pi Q_{\mathbf{x},\mathbf{f}_\pi}(t). \tag{5.7}$$

Similar to the safety guarantees for the nominal supervisory CBF controller (5.3), guarantees can be given when using the backup controller derived supervisory controller (5.6). This is summarized in the following proposition.

Proposition 5.4 (Y. Chen, Singletary, and Ames, 2020)**.** *For all* $\mathbf{x} \in \{\mathbf{x} | h(\mathbf{x}) \geq 0\}$*, a solution to* (5.6) *is always feasible and Eq.* (5.6) *implies* $\dot{h}(\mathbf{x}, \mathbf{u}) + \alpha(h(\mathbf{x})) \geq 0$*. Moreover, h is a CBF that satisfies*

- $\forall \mathbf{x} \notin C, h(\mathbf{x}) < 0,$
- $\forall \mathbf{x} \in \{\mathbf{x} \mid h(\mathbf{x}) \geq 0\}, \exists \mathbf{u} \in \mathcal{U}, \text{s.t. } \dot{h}(\mathbf{x},\mathbf{u}) + \alpha(h(\mathbf{x})) \geq 0.$

When implementing the supervisory controller (5.6), the first constraint is not implementable because there are uncountably many t in $[0, T]$. Consequently, the continuous interval $[0, T]$ is replaced with a finite sequence, $0 = t_0 < t_1 < \cdots < t_K = T$, $t_{i+1} - t_i = \Delta t$, and the constraint enforced at these points instead. This discretization calls for additional robustness of the control strategy, as is discussed in (Singletary, Y. Chen, and Ames, 2020).

Assumption 5.5. *Δt is sufficiently small such that robustly satisfying the constraints of* (5.6) *at each t_k (by adding an error buffer) implies constraint satisfaction for all* $\Phi^t_{\mathbf{f}_\pi}(\mathbf{x})\big|_{t=t_k}^{t=t_k+\Delta t}$.

5.3 Koopman Operator for Backup Trajectories

Motivating the use of Koopman operators

For high dimensional systems and/or long time spans T, the forward integration of the sensitivity matrix in (5.6) may be prohibitively expensive. Besides, in the presence of unmodelled dynamics, the online integration can be inaccurate. Additionally, the trajectory under the backup strategy typically evolves for a finite time, which allows us to bound the prediction error under the Koopman operator, making it suitable for safety-critical applications. As the constraint of (5.6) must be evaluated at discrete points, I will approximate the flow of the system under the backup controller at these points using an approximated discrete-time Koopman operator. Specifically, a finite dimensional Koopman approximation is identified which enables estimating the sensitivity matrix in (5.6) as

$$\nabla \Phi^{t_k}_{\mathbf{f}_\pi}(\mathbf{x}_0) \approx \nabla\big(C^{\mathbf{x}} A^k \boldsymbol{\phi}(\mathbf{x}_0)\big) = C^{\mathbf{x}} A^k \nabla \boldsymbol{\phi}(\mathbf{x}_0), \tag{5.8}$$

which only requires matrix multiplication. Moreover, $C^{\mathbf{x}}A^k$ can be precomputed for $k = 0, \dots, K$, significantly reducing the real-time computational cost of the forward integration of the sensitivity matrix, improving the method's scalability.

Learning Koopman operators for backup trajectories

To learn an approximate Koopman operator associated with the closed-loop backup controller dynamics, M_t trajectories of length T_t are simulated from initial conditions $\mathbf{x}_0^j \in \Omega, j = 1, \dots, M_t$ under backup control. The set Ω is chosen to cover the operating region of interest. From each trajectory, $M_s = T/\Delta t$ state snapshots are sampled at a fixed time interval, Δt, resulting in the data set

$$\mathcal{D} = \left((\mathbf{x}_k^j)_{k=0}^{M_s} \right)_{j=1}^{M_t}. \tag{5.9}$$

Define a dictionary of n functions $\mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$, $\boldsymbol{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^n$. The choice of basis can be based on system knowledge (i.e. feature engineering) or they can be a generic basis of functions such as monomials of the state up to a certain degree. Selecting the type and number of dictionary functions is an open question in Koopman-based learning. Promising efforts in this direction use data-driven Koopman eigenfunctions (See Chapter 3 and Korda and Mezić, 2018a). Because I aim to learn approximate Koopman operators of the closed-loop dynamics, which is autonomous, the learning of data-driven Koopman eigenfunctions is simplified. In particular, the method presented in Chapter 3 can be applied with minor modifications.

The data set $\mathcal{D}$ is organized into matrices X, X' as described in (5.10). Then, the lifted state data matrices are constructed by applying $\boldsymbol{\phi}(\mathbf{x})$ to each column of X and X', denoted $Z = \boldsymbol{\phi}(X), Z' = \boldsymbol{\phi}(X')$ by slight abuse of notation. Utilizing the constructed data matrices, the best fit Koopman operator approximation is inferred by solving a linear least squares regression problem (5.10a). In addition, regularization can be added to the regression formulation.

$$\min_{A \in \mathbb{R}^{n \times n}} ||AZ - Z'||^2 \tag{5.10a}$$

$$\min_{C^{\mathbf{x}} \in \mathbb{R}^{d \times n}} ||C^{\mathbf{x}}Z - X||^2 \tag{5.10b}$$

$$X = [\mathbf{x}_0^1 \dots \mathbf{x}_{(M_s-1)}^1 \dots \mathbf{x}_0^{M_t} \dots \mathbf{x}_{(M_s-1)}^{M_t}], \qquad Z = \boldsymbol{\phi}(X)$$

$$X' = [\mathbf{x}_1^1 \dots \mathbf{x}_{M_s}^1 \dots \mathbf{x}_0^{M_t} \dots \mathbf{x}_{M_s}^{M_t}], \qquad Z' = \boldsymbol{\phi}(X')$$

The second regression problem (5.10b) is formulated to learn the matrix projecting the lifted state back to the original state. As discussed in Chapters 3 and 4, $C^{\mathbf{x}}$ may be computed analytically for certain choices of the function dictionary $\boldsymbol{\phi}(\mathbf{x})$, rendering the regression problem obsolete. For example, a monomial basis will typically include the state itself, making the computation trivial.

Quantifying the error of the Koopman approximation

For safety-critical application, bounding the prediction error of the Koopman operator is essential. The supervisory controller requires the system flow at a finite number of sample points $t_0, t_1, \dots, t_K$, where $t_k = k\Delta t$. Denote the true flow of the closed-loop dynamics under the backup controller at these sampling points as $\Phi^{t_k}_{\mathbf{f}_\pi}(\mathbf{x})$. Similarly, denote the estimate of the same flow using the learned Koopman operator as $\hat{\Phi}^{t_k}_{\mathbf{f}_\pi}(\mathbf{x}) = C^{\mathbf{x}} A^k \boldsymbol{\phi}(\mathbf{x})$. Then, the true system evolves as:

$$\Phi^{t_k}_{\mathbf{f}_\pi}(\mathbf{x}) = \hat{\Phi}^{t_k}_{\mathbf{f}_\pi}(\mathbf{x}) + \underbrace{\Phi^{t_k}_{\mathbf{f}_\pi}(\mathbf{x}) - \hat{\Phi}^{t_k}_{\mathbf{f}_\pi}(\mathbf{x})}_{\Xi_k(\mathbf{x})} \tag{5.11}$$

To guarantee safety under the approximation error of the Koopman operator, $\Xi_k(\mathbf{x})$, I first introduce definitions and assumptions that are later used in Prop. 5.8 to derive an error bound. A key concept to bound the prediction error is the incremental stability of discrete-time systems (Tran, Rüffer, and Kellett, 2016).

Definition 5.6. The dynamical system (5.1) is *incrementally stable* in $\mathcal{X} \subseteq \mathbb{R}^d$ if $\forall t \in \mathbb{N}$, $\forall\ \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ and control sequence $\mathbf{u}(\cdot) : \mathbb{N} \to \mathbb{R}^m$ such that the closed-loop evolution $\Phi^t_{\mathbf{f}_{\mathbf{u}(\cdot)}}(\mathbf{x}_1)$ and $\Phi^t_{\mathbf{f}_{\mathbf{u}(\cdot)}}(\mathbf{x}_2)$ remain in $\mathcal{X}$, the state evolution satisfies $\left\|\Phi^t_{\mathbf{f}_{\mathbf{u}(\cdot)}}(\mathbf{x}_1) - \Phi^t_{\mathbf{f}_{\mathbf{u}(\cdot)}}(\mathbf{x}_2)\right\| \leq \beta(\|\mathbf{x}_1 - \mathbf{x}_2\|, t)$, where $\beta : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ is nonincreasing in t and $\beta(\cdot, t)$ is a class-$\mathcal{K}$ function, i.e. β is a class-$\mathcal{KL}$ function.

Remark 5.7. *Since the backup strategy is designed before implementing the CBF, techniques exist to synthesize backup strategies that renders the closed-loop system incrementally stable, even for open-loop unstable systems, e.g., the LMI approach presented in Section III.B in (Singletary, Y. Chen, and Ames, 2020).*

Proposition 5.8. *For system* (5.1)*, assume that the closed-loop dynamics under the backup controller* $\dot{\mathbf{x}} = \mathbf{f}_\pi(\mathbf{x})$ *is incrementally stable, and that projection matrix* $C^{\mathbf{x}}$ *is exact, i.e.* $\mathbf{x} = C^{\mathbf{x}}\mathbf{z}, \mathbf{z} = \boldsymbol{\phi}(\mathbf{x})$*. Further assume that* $\boldsymbol{\phi}(\mathbf{x})$ *is Lipschitz continuous on* $\mathcal{X}$ *with Lipschitz constant L and that the distance between points in* $\mathcal{X}$ *and the closest point in* $\mathcal{D}$ *is bounded by* μ*, where* $\mu = \max_{\mathbf{x}\in\mathcal{X}} \min_{\hat{\mathbf{x}}\in\mathcal{D}} \|\mathbf{x} - \hat{\mathbf{x}}\| < \infty$*. Then,* *error $\Xi_k(\mathbf{x})$ defined in (5.11) can be upper bounded as*

$$||\Xi_k(\mathbf{x})|| \leq ||C^{\mathbf{x}} A^k|| L\mu + ||\varepsilon^{max}|| \sum_{j=1}^{k-1} ||C^{\mathbf{x}} A^j|| + \mu$$

where the norm of the maximum residual error is given by

$$||\varepsilon^{max}|| = \max_{\mathbf{x}_k^j \in \mathcal{D} \setminus \mathbf{x}_K^j} ||A\boldsymbol{\phi}(\mathbf{x}_k^j) - \boldsymbol{\phi}(\mathbf{x}_{k+1}^j)||.$$

Proof. Let $\mathbf{x} \in \mathcal{X}, \hat{\mathbf{x}} \in \mathcal{D}$, and add and subtract $\Phi_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}})$

$$||\Xi_k(\mathbf{x})|| = ||\Phi_{\mathbf{f}_\pi}^{t_k}(\mathbf{x}) - \hat{\Phi}_{\mathbf{f}_\pi}^{t_k}(\mathbf{x}) + \Phi_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}}) - \Phi_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}})||.$$

Define the global error of the Koopman approximation on the training data as $E_k = \boldsymbol{\phi}(\mathbf{x}_k) - A^k \boldsymbol{\phi}(\mathbf{x}_0)$. Then, $\Phi_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}}) = \hat{\Phi}_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}}) + C^{\mathbf{x}} E_k$ and separating each term of the norm yields

$$||\Xi_k(\mathbf{x})|| \leq ||\hat{\Phi}_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}}) - \hat{\Phi}_{\mathbf{f}_\pi}^{t_k}(\mathbf{x})|| + ||C^{\mathbf{x}} E_k|| + ||\Phi_{\mathbf{f}_\pi}^{t_k}(\mathbf{x}) - \Phi_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}})||.$$

The first term follows from the definition of $\hat{\Phi}_{\mathbf{f}_\pi}^{t_k}$, Lipschitz continuity of $\boldsymbol{\phi}(\mathbf{x})$, and a bound on the distance from $\mathcal{X}$ to $\mathcal{D}$:

$$||\hat{\Phi}_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}}) - \hat{\Phi}_{\mathbf{f}_\pi}^{t_k}(\mathbf{x})|| \leq ||C^{\mathbf{x}} A^k (\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\phi}(\hat{\mathbf{x}}))|| \leq ||C^{\mathbf{x}} A^k|| L\mu.$$

To bound the second term, consider that the error can be expressed as $E_k = \sum_{j=1}^{k-1} A^j \varepsilon_{j-1}$, where $\varepsilon_j = \boldsymbol{\phi}(\mathbf{x}_j) - A\boldsymbol{\phi}(\mathbf{x}_{j-1})$. The term can then be bounded as follows (Mamakoukas, Abraham, and Murphey, 2020)

$$||C^{\mathbf{x}} E_k|| \leq ||C^{\mathbf{x}} \sum_{j=1}^{k-1} A^j \varepsilon_{j-1}|| \leq ||\varepsilon^{max}|| \sum_{j=1}^{k-1} ||C^{\mathbf{x}} A^j||.$$

By incremental stability, the last term $||\Phi_{\mathbf{f}_\pi}^{t_k}(\mathbf{x}) - \Phi_{\mathbf{f}_\pi}^{t_k}(\hat{\mathbf{x}})|| \leq ||\mathbf{x} - \hat{\mathbf{x}}|| \leq \mu$. This results in the desired bound. □

Remark 5.9. *Following Section 5.3, the system state is usually included in the function dictionary, making the projection $C^{\mathbf{x}}$ exact. If this is not the case, the state can be added, $\bar{\boldsymbol{\phi}}(\mathbf{x}) = [\mathbf{x}^T \boldsymbol{\phi}(\mathbf{x})^T]^T$.*

Remark 5.10. *The maximum distance between a new data point $\mathbf{x}$ to the nearest point in data set $\mathcal{D}$ needs only to consider the non-cyclic states of the system. Furthermore, the closed-loop system under the backup controller is stable to the* *initial control invariant set $\mathcal{S}_0$, and thus the Koopman operator associated with the dynamics is often at least marginally stable. This can be enforced in the learning process if needed, see for example (Mamakoukas, Abraham, and Murphey, 2020). Consequently, the terms $||C^{\mathbf{x}}A^k||$ in the bound are non-increasing w.r.t. k.*

Remark 5.11. *If h^C, h^S depend on a subset of states, the bound in Prop. 5.8 can be tightened by replacing $C^{\mathbf{x}}$ by C_h, which projects the state subset. If computation permits, $L\mu$ can be replaced with $||C^{\mathbf{x}}A^k(\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\phi}(\hat{\mathbf{x}}))||$ and μ by the exact distance from the current state $\mathbf{x}$ to the closest point in $\mathcal{D}$.*

5.4 CBF with Koopman Predicted State Flow

Using the learned Koopman operators for the closed-loop dynamics under the backup controller, the supervisory controller (5.6) can be reformulated as an optimization problem:

$$\begin{aligned} \mathbf{u}^* = &\operatorname{argmin}_{\mathbf{u}\in\mathcal{U}} ||\mathbf{u} - \mathbf{u}^0||^2 \\ &\text{s.t. } \nabla h^C(C^{\mathbf{x}}A^k\nabla_{\mathbf{x}}\boldsymbol{\phi}(\mathbf{x})(\mathbf{f}(\mathbf{x},\mathbf{u}) - \mathbf{f}_\pi(\mathbf{x})) \\ &\qquad\quad + \alpha(h^C(C^{\mathbf{x}}A^k\boldsymbol{\phi}(\mathbf{x}))) \geq 0,\ k = 1, \dots, K \\ &\qquad \nabla h^S(C^{\mathbf{x}}A^K\nabla_{\mathbf{x}}\boldsymbol{\phi}(\mathbf{x})(\mathbf{f}(\mathbf{x},\mathbf{u}) - \mathbf{f}_\pi(\mathbf{x}))) \\ &\qquad\quad + \alpha(h^S(C^{\mathbf{x}}A^K\boldsymbol{\phi}(\mathbf{x}))) \geq 0, \end{aligned} \tag{5.12}$$

where the sensitivity matrix for every t_k is replaced as (5.8).

By using the generalization bound derived in Prop. 5.8, the supervisory controller can be made robust to the approximation error of the flow estimated by the Koopman operator.

Theorem 5.12. *Consider the control system (5.1) and an associated backup controller with backup trajectories approximated by a learned Koopman operator satisfying the assumptions of Prop. 5.8. If the constraints (5.12) are satisfied for $C^{\mathbf{x}}A^k\boldsymbol{\phi}(\mathbf{x}) + \boldsymbol{\Delta}_k(\mathbf{x})$, with $\boldsymbol{\Delta}_k(\mathbf{x}) = \{\boldsymbol{\delta} : ||\boldsymbol{\delta} - \mathbf{x}|| \leq ||\Xi_k(\mathbf{x})||\}$, then the true system flow satisfies*

$$\Phi^{t_k}_{\mathbf{f}_\pi}(\mathbf{x}) \in C^{\mathbf{x}}A^k\boldsymbol{\phi}(\mathbf{x}) + \boldsymbol{\Delta}_k(\mathbf{x}).$$

Under Assumption 5.5, $\Phi^t_{\mathbf{f}_\pi}(\mathbf{x}) \in \mathcal{C}\ \forall t \in [0, T]$, i.e., the true system evolution under $\boldsymbol{\pi}$ satisfies the safety constraint.

Proof. The error bound on the evolution of the true system follows directly from Proposition 5.8. The robustness of the controller follows from (Singletary, Y. Chen, and Ames, 2020), Theorem 1. □

Remark 5.13. *For an incrementally stable system, the constraints of (5.12) can be enforced for all $\Phi^{t_k}_{\mathbf{f}_\pi}(\mathbf{x}) \in C^{\mathbf{x}} A^k \boldsymbol{\phi}(\mathbf{x}) + \boldsymbol{\Delta}_k(\mathbf{x})$ by calculating the flow over the backup trajectory for the nominal value of* $\mathbf{x}$*, and evaluating the constraints over the set. This greatly simplifies the computation and enables the use of interval arithmetic libraries such as* `INTLAB` *(Rump, 1999) and* `libaffa` *(Gay, n.d.), see (Singletary, Y. Chen, and Ames, 2020) for details.*

5.5 Simulation and Experimental Results

Experiments are conducted to validate the method using two multi-agent systems. First, physical ground robots are used to demonstrate that the method can work on a physical system. Then, simulated quadrotors are used to highlight the computational benefits of the method for high-dimensional systems.

Ground robot obstacle avoidance

I first demonstrate the method performing single-agent obstacle avoidance. A legacy controller drives the robot in a straight line to a distant point, and then returns. The supervisory controller maintains safety by utilizing a backup strategy applying maximum brake and turn. The robot is modeled as a Dubin's car with dynamics:

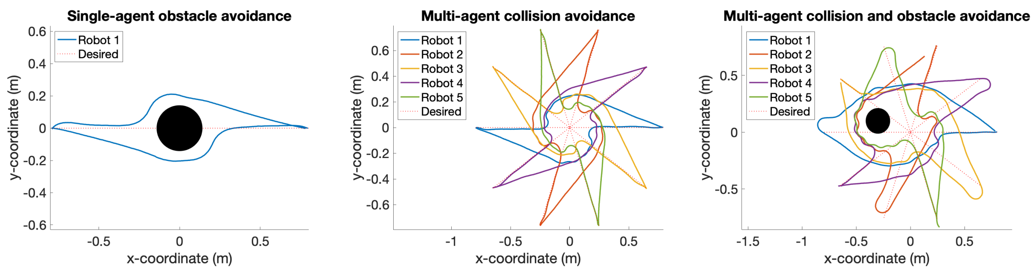
$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{v} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ a \\ r \end{bmatrix}, \tag{5.13}$$

where X, Y, v, θ denote its Cartesian coordinates, velocity, and heading angle. The acceleration a and yaw rate r inputs are bounded by $a^{\max}$ and $r^{\max}$. I conduct simulated and physical experiments using Georgia Tech's Robotarium (Pickem et al., 2017).

The Koopman operator for the closed-loop system is learned by collecting 200 data points sampled in the operating region of interest while the system operates under backup control[1,2]. Based on knowledge of the system, 21 dictionary functions are chosen, $\boldsymbol{\phi}(\mathbf{x}) = [1\, X\, Y\, v\, \theta\, v^2 \ldots, v^5\, cos(\theta)\, sin(\theta)\, vcos(\theta) \ldots v^5 sin(\theta)]^T$. The

[1]Code available at `https://github.com/Cafolkes/koopman-cbf`

[2]Video of all the experiments available at `https://youtu.be/IfBUbtKP53c`



(a) (b) (c)

Figure 5.1: Robotarium experiment traces, the Koopman CBF controller guarantees zero collisions. Each robot visits a point and returns to its initial position while (a) avoiding obstacle, (b) avoiding collision, and (c) avoiding collision and obstacle.

Lipschitz constant of the function dictionary used in Proposition 5.8 is calculated by maximizing the symbolic Jacobian over the operating region of interest. The robust supervisory controller that enforces the conditions of Theorem 5.12 is implemented with `INTLAB` (Rump, 1999). Fig. 5.1(a) shows the robot's path. Table 5.1 reports computation times for forward integration of the sensitivity matrix and dynamics using `ode45`, `CasADi`, and the learned Koopman operator. Our approach reduced computation time by $\sim 80\%$.

Multi-agent ground robot collision avoidance

The single-agent supervisory controller (5.12) can be extended to a safe decentralized multi-agent controller, following (Y. Chen, Singletary, and Ames, 2020). Each agent has a backup strategy that brings it to a stable equilibrium. All agents' backup strategies are known a priori to every agent as part of a centralized design. Then, each agent measures the adjacent agents' states, and ensures that if other agents execute a backup strategy, its own backup strategy avoids the danger set.

The initial positions of 5 robots are equally spaced on a circle. A greedy legacy controller drives each robot to the opposite point on the circle, and then back. The supervisory controller avoids collision between agents and with a fixed obstacle. Fig. 5.1(b) shows the robot's motion traces as they execute the task without a fixed obstacle. As the robots near the circle center, they circle around each other and/or the obstacle to avoid collision. The seemingly coordinated behavior is the result of the decentralized supervisory controllers. Similarly, Fig. 5.1(c) shows the result when a fixed obstacle is added. The results demonstrate that the CBF utilizing the learned Koopman operator can maintain system safety.

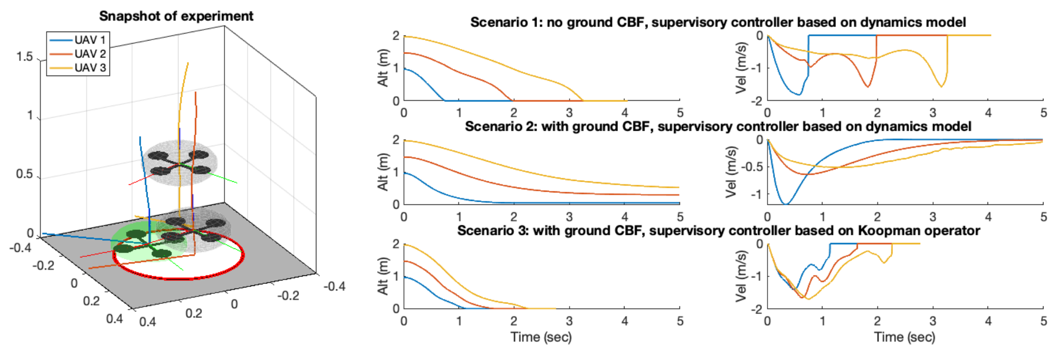


Figure 5.2: Simulated quadrotor landing experiment. (left) Snapshot of scenario 3 simulation and traces of full trajectory for each UAV. (right) Altitude and velocity of each agent. The Koopman CBF controller guarantees zero collisions while maintaining sufficient mobility to achieve landing.

Simulated multi-agent quadrotor collision avoidance and landing

To showcase the method's scalability, I consider 3 quadrotors trying to land on the same landing pad while avoiding collisions and hard ground impacts. The 16-dimensional quadrotor state is $\mathbf{x} = [\mathbf{p}, \mathbf{v}, \boldsymbol{\theta}, \boldsymbol{\omega}, \boldsymbol{\Omega}]^\top$ where $\mathbf{p}$ is the position, $\mathbf{v}$ is the velocity, $\boldsymbol{\theta}$ are the Euler angles, $\boldsymbol{\omega}$ is the angular velocity, and $\boldsymbol{\Omega}$ is the vector of propeller rotation rates. The motor drive voltages are the control inputs, $\mathbf{u} = [V_1, V_2, V_3, V_4]^\top$. The dynamics are derived from a force-balance in a rotating frame, and a first order motor model. This high dimensional system makes the sensitivity matrix expensive to forward integrate with previous methods, especially in the multi-agent case. To highlight the benefits of learning the backup-controlled dynamics from data, an external forcing that models the *ground effect*: a thrust amplification model (see Sec. 2.2. of Sanchez-Cuevas, Heredia, and Ollero, 2017) acts when the quadrotor nears the ground. This unmodeled effect is not captured by the nominal dynamics model, but is captured when learning the Koopman operator of the closed-loop dynamics under backup control.

	`ODE45` (ms)		`CasADi` (ms)		Koopman (ms)	
	mean	std	mean	std	mean	std
Dubin's car	21.5	12.4	1.1	0.5	0.2	0.1
Quadrotor	28.0	1.5	4.5	0.2	0.2	0.1

Table 5.1: State and sensitivity matrix forward integration computation times per agent.

The legacy PD controller is designed to drive each agent from its initial position to a setpoint $\mathbf{p}_d = [0, 0, 0]^T$ with velocity $\mathbf{v}_d = [0, 0, -2]^T$, chosen such that the legacy controller lands at a velocity that may cause damage. The backup policy is a PD controller that aims to quickly decelerate the quadrotor to hover. The corresponding backup set is a small ball around zero linear velocity, pitch, and roll angles.

The Koopman-based supervisory controller is compared with controllers using forward integration (see Section 5.2). I consider 3 scenarios to highlight some benefits of the method; (1) landing with the supervisory controller only enforcing collision avoidance, (2) landing with the supervisory controller enforcing collision avoidance and avoiding crashing into the ground, and, finally, (3) landing with the supervisory controller enforcing collision avoidance and avoiding crashing into the ground with a Koopman operator trained on data that captures the ground effect. The collision avoidance barrier function $h_c(\mathbf{p}_i, \mathbf{p}_j)$ is defined pairwise for all $i, j = 1, 2, 3, i \neq j$ as a ball of radius r around each agent, $h_c(\mathbf{p}_i, \mathbf{p}_j) = \mathbf{p}_i^T \mathbf{p}_j - r^2$. The ground avoidance barrier function $h_g(\mathbf{p}_i)$ seeks to keep each agent above a paraboloid placed at the center of the landing pad $h_g(\mathbf{p}_i) = z - \mathbf{p}_i^T \mathbf{p}_i$. Landing is defined as reaching an altitude less than 1 cm.

The data set is collected by simulating the system under the backup controller from 250 initial points sampled from the operating region of interest. The data set includes data points close to the ground, thereby capturing trajectories influenced by the ground effect. Based on knowledge of the system, 47 dictionary functions are chosen consisting of the state, products of the angular rates and trigonometric functions of the Euler angles, rotation matrix terms, angular and linear acceleration terms, and a simple ground effect nonlinearity, $\frac{\Omega_i^2}{1-(R/(4z)^2}$ for each $\Omega_i, i = 1, \dots, 4$, where R is the propeller radius. Then, the Koopman operator is learned and the controller for scenario 3 constructed.

A snapshot of scenario 3, and the altitude and velocities for all scenarios are shown in Fig. 5.2. When not enforcing the ground avoidance CBF (scenario 1), the quadrotors may hit the ground at speeds up to 1.2 m/s. Enforcing the ground avoidance CBF based on the dynamics model, but without ground effects, causes the supervisory controller to be too conservative, thereby prohibiting the quadrotor from reaching the ground. Finally, when the ground avoidance is enforced with a Koopman operator trained on data capturing the ground effect, smooth landing is achieved with impact speeds less than 0.3 m/s. Furthermore, the forward integration computation time is reduced by approximately 95 % when using the learned Koopman operator (see Table 5.1).

5.6 Conclusion

This chapter presented a method using learned Koopman operators to improve the computational efficiency of a control barrier function-based supervisory controller and showed how to maintain theoretical guarantees even though backup trajectories are learned from data. Through physical experiments on wheeled robots and simulated experiments on quadrotors, I show that the method can be incorporated in a decentralized supervisory controller design that can take full advantage of the computational benefits of Koopman-based forward integration. Furthermore, I exemplify how learning the backup trajectories from data can improve the mobility of the system, thereby improving overall control performance.

Chapter 6

CONCLUSION

Although Koopman-inspired methods have received considerable attention in certain domains, such as fluid dynamics, they are relatively less explored for robotic applications. As discussed in Chapter 1, robotic applications come with a set of challenges that make learning particularly challenging. First, since data collection is typically expensive, models need to achieve good performance with a limited amount of data. Furthermore, if the learned models are used for control design, evaluation of the model at run time must be fast enough such that high controller rates are possible. Second, as failure can lead to material damage and human injuries and fatalities, safety and performance must be understood either by having the ability to interpret the resulting models or through theoretical guarantees. Finally, it must be possible to design controllers with the learned models that can exploit the model's prediction performance while respecting state and actuation limitations that arise in virtually all real world robotic systems.

This thesis has focused on how ideas from Koopman operator theory can be used to improve the performance, data efficiency, and theoretical guarantees of systems that leverage data and learning to boost controller performance. The overarching message is that Koopman-based learning and control methods can be powerful tools for nonlinear control design of agile robotic systems if the model structure and observable functions are carefully designed. The results in Chapters 3-5 indicate that these methods can achieve good prediction performance with realistic data amounts and that the learned models can be used for model predictive control design, allowing optimization of relevant objectives while satisfying state and control constraints.

Chapter 3 presented a novel method to learn nonlinear dynamics using Koopman eigenfunctions constructed in a principled way combined with a method to learn a lifted linear model of the dynamics. I motivated theoretically and showed empirically that the principled design of eigenfunctions led to significantly better prediction performance and improved control performance when the learned models are used for model predictive control design. The presented method and associated papers were some of the early works making Koopman learning and control functional on agile robots in the laboratory, and the quadrotor experiments demonstrated that the structured approach to designing Koopman eigenfunctions also can be practically useful for control purposes. However, the method utilizes a linear model structure which prohibits nonlinear actuation effects to be appropriately captured. This is a significant limitation for robotic systems, which typically have control-affine dynamics with forces and torques entering the system dynamics through rotation matrices.

To alleviate this, Chapter 4 described two methods underpinned by a lifted bilinear model structure that theoretically can capture control-affine dynamics. In the first method, I showed that by carefully designing the function dictionary, EDMD-type learning can be used to identify a bilinear model from data. Then, using a simulated planar quadrotor with nonlinear control-affine dynamics, I demonstrated that bilinear models indeed are more suitable to capture nonlinear effects, as prediction error could be reduced by an order of magnitude compared to a lifted linear model using the same, fixed function dictionary. Furthermore, I showed that the resulting bilinear model can be utilized for model predictive control design and that the resulting data-driven controller achieved similar performance to the case of a NMPC designed with an exactly known dynamic model. However, for realistic systems, it is not clear how to choose the functions in the function dictionary to achieve good performance. Also, using fixed function dictionaries typically result in a high lifting dimension to obtain the desired prediction error, making the NMPC based on the learned model intractable for real time applications.

Instead, the second method encodes both the function dictionary and the Koopman bilinear model in a single neural network architecture. This allows joint learning of both the function dictionary and dynamic model matrices strictly from data, alleviating the need to manually design a function dictionary and enabling more compact models and/or better prediction performance compared to predefined function dictionaries. The compactness, i.e. lower lifting dimension, is crucial to make NMPC based on the resulting models computationally feasible for real time control. I show that the joint dictionary and model learning achieves sufficient prediction performance using only a lifting dimension of 23, and that the resulting controller can match or outperform a NMPC based on a nominal model although it is strictly data-driven. The resulting controller showed remarkable repeatability and indicated that the learned model was able to capture aerodynamic ground effect such that agile trajectories could be executed close to the ground with acceptable tracking error even when the controller based on the nominal model failed to achieve stable flight.

Whereas Chapter 3 and 4 focused on learning and control performance, Chapter 5 introduced a method to improve the computational efficiency of a control barrier function-based supervisory controller and showed how to maintain theoretical guarantees even though backup trajectories are learned from data. I showed how to derive theoretical error bounds on the learned Koopman model and how this bound can be incorporated in the safety filter to guarantee safety satisfaction under learning error. Furthermore, I showed that the safety filter computation time can be drastically reduced by exploiting the linearity of the learned model. To my knowledge, this is the only work that is primarily motivated by replacing a complicated dynamics model with a lifted linear model to make computation of a underlying controller feasible in real time.

Through the contributions of this thesis, various approaches for learning and control underpinned by Koopman operator theory have been explored. A lot of the excitement and interest in Koopman-based methods in both controls, robotics, and other domains is based on the fact that the theory indicates that nonlinear systems can be globally transformed to linear ones allowing the vast array of linear analysis and control design methods and theory to be applied. This feature is demonstrated in Chapters 3 and 5, where the linear models allowed both computationally efficient linear model predictive control and evaluation of the implicitly defined control barrier function safety filter, while incorporating nonlinear model information. Linear MPC using lifted linear models is a flexible and practical method that likely can work very well for many systems and there is probably many more methods that could become significantly more computationally efficient by replacing complicated nonlinear models with appropriate linear Koopman approximations, as in Chapter 5. However, as is demonstrated in Chapter 4, control-affine dynamics, needed to describe a large class of robotic systems, require bilinear models to appropriately capture the underlying phenomena. Although bilinear models have a lot of structure that potentially can be exploited for theoretical derivations and control design, they are significantly less studied than linear models. As a result, the theoretical and computational benefits of Koopman learning using bilinear models are less evident than their linear counterpart, but their prediction performance when combined with modern machine learning techniques and control performance when coupled with NMPC warrant further investigation for robotic applications.

6.1 Future Work

Develop a unified framework for learning and control: Chapter 4 presented a method for model learning and high performance control with nonlinear MPC given a sufficiently rich data set. At the same time, Chapter 5 presented a method to speed up and enhance the practical applicability of safety filtering utilizing a learned Koopman operator based on data collected under a backup strategy. By combining these two methods a framework for safe learning and high performance control can result. For example, a system designer can do initial system identification to design a backup policy, e.g. a proportional-derivative controller stabilizing a quadrotor drone to hover. Then, a Koopman CBF safety filter (Chapter 5) can be designed based on data collected from the system operating under the backup controller. Subsequently, data can be collected with an arbitrary controller filtered using the Koopman CBF safety filter to ensure safety of the system, and the Koopman NMPC framework (Chapter 4) can be used to learn a model and seek high control performance. This can be done either episodically or in a batch learning process to obtain a high performance NMPC leveraging the learned Koopman bilinear model. As a result, such a method would enable end-to-end data collection and learning with safety guarantees.

Develop theoretical model error bounds for the Koopman bilinear model and utilize them in NMPC control design: To enable the methods described in Chapter 4 to be fully utilized in safety-critical systems, model error bounds on the learned model are needed to be able to analyze the model accuracy. The error bound derived in Chapter 5 is a potential starting point to achieve such bounds. Theoretical bounds on the prediction error as a function of the lifting dimension and/or type of function dictionary is also desired to better understand and guide design choices when deploying Koopman-based learning methods. Once practical bounds are derived, results already exist that can be used as a basis for robust control design of the NMPC. For example, as mentioned in Chapter 1, the bilinear model structure can be used to simplify the construction of a control Lyapunov function enforced as a constraint in a nonlinear MPC method to obtain stability guarantees (Narasingam and

Kwon, 2020). Then, robustness based on the model error bounds can be considered using input-to-state stability reasoning, as introduced for NMPC by Bayer, Burger, and Allgower, 2013.

Expand breadth of experiments and compare with alternative learning and control methods: Multiple simulated and physical experiments are presented in this thesis but many more are needed to more fully understand the benefits and limitations of the proposed methods. Furthermore, in the reinforcement learning and learning and control communities, there are currently significant research activity and many new methods are presented every year but a clear, transparent way to compare the methods on equal grounds does not exist. As various methods become more mature, it is crucial that the methods can be compared on equal baselines to better understand what applications, systems, and tasks are suitable for the different methods and how do they compare in terms of performance, data requirements, computational efficiency, theoretical guarantees, etc. Comprehensive benchmarking suites are currently emerging (cf. Voloshin et al., 2021 and references therein), and it is now on the research community to start presenting results of how their proposed methods perform on these benchmarks.

BIBLIOGRAPHY

Abdolhosseini, Mahyar, Youmin M. Zhang, and Camille-Alain Rabbath (2013). "An Efficient Model Predictive Control Scheme for an Unmanned Quadrotor Helicopter". In: *J Intell Robot Syst* 70, pp. 27–38. DOI: 10.1007/s10846-012-9724-3.

Ames, Aaron D., Jessy W. Grizzle, and Paulo Tabuada (2014). "Control Barrier Function Based Quadratic Programs with Application to Adaptive Cruise Control". In: *Proceedings of the IEEE Conference on Decision and Control*. DOI: 10.1109/CDC.2014.7040372.

Ames, Aaron D. and Matthew Powell (2013). "Towards the Unification of Locomotion and Manipulation through Control Lyapunov Functions and Quadratic Programs". In: *Lecture Notes in Control and Information Sciences*. ISBN: 9783319011585. DOI: 10.1007/978-3-319-01159-2{_}12.

Ames, Aaron D., Xiangru Xu, et al. (2017). "Control Barrier Function Based Quadratic Programs for Safety Critical Systems". In: *IEEE Transactions on Automatic Control* 62.8, pp. 3861–3876. ISSN: 00189286. DOI: 10.1109/TAC.2016.2638961.

Amos, Brandon et al. (2018). "Differentiable MPC for End-to-End Planning and Control". In: *Advances in Neural Information Processing Systems* 2018-Decem.NeurIPS, pp. 8289–8300. ISSN: 10495258.

Aswani, Anil et al. (2013). "Provably Safe and Robust Learning-based Model Predictive Control". In: *Automatica* 49.5, pp. 1216–1226. ISSN: 00051098. DOI: 10.1016/j.automatica.2013.02.003. URL: http://dx.doi.org/10.1016/j.automatica.2013.02.003.

Balaram, Bob J. et al. (2018). "Mars Helicopter Technology Demonstrator". In: *AIAA Atmospheric Flight Mechanics Conference, 2018* 209999. DOI: 10.2514/6.2018-0023.

Bangura, Moses and Robert Mahoney (2012). "Nonlinear Dynamic Modeling for High Performance Control of a Quadrotor". In: *Proc. Australasian Conf. on Robotics and Automation*.

Bangura, Moses and Robert Mahony (Jan. 2014). "Real-time Model Predictive Control for Quadrotors". In: *IFAC Proceedings Volumes* 47.3, pp. 11773–11780. ISSN: 1474-6670. DOI: 10.3182/20140824-6-ZA-1003.00203.

Bayer, Florian, Mathias Burger, and Frank Allgower (2013). "Discrete-time Incremental ISS: A framework for Robust NMPC". In: *2013 European Control Conference, ECC 2013*, pp. 2068–2073. DOI: 10.23919/ECC.2013.6669322.

Beckers, Thomas, Dana Kulić, and Sandra Hirche (May 2019). “Stable Gaussian Process Based Tracking Control of Euler–Lagrange Systems”. In: *Automatica* 103, pp. 390–397. ISSN: 00051098. DOI: 10.1016/j.automatica.2019.01.023.

Berger, Erik et al. (May 2015). “Estimation of Perturbations in Robotic Behavior Using Dynamic Mode Decomposition”. In: *Advanced Robotics* 29.5, pp. 331–343. ISSN: 0169-1864. DOI: 10.1080/01691864.2014.981292. URL: https://doi.org/10.1080/01691864.2014.981292.

Berkenkamp, Felix et al. (2017). “Safe Model-based Reinforcement Learning with Stability Gurantees”. In: *31st Conference on Neural Information Processing Systems (NIPS 2017)* 47.12, pp. 737–742. ISSN: 1340-3451.

Blanchini, Franco (Nov. 1999). “Set Invariance in Control”. In: *Automatica* 35.11, pp. 1747–1767. ISSN: 00051098. DOI: 10.1016/S0005-1098(99)00113-2.

Bloemen, Hayco H. J., Mark Cannon, and Basil Kouvaritakis (2002). “Closed-loop Sta bilizing MPC for Discrete-time Bilinear Systems”. In: *European Journal of Control* 8.4, pp. 304–314. ISSN: 09473580. DOI: 10.3166/ejc.8.304-314. URL: http://dx.doi.org/10.3166/ejc.8.304-314.

Borrmann, Urs et al. (Jan. 2015). “Control Barrier Certificates for Safe Swarm Behavior”. In: *IFAC-PapersOnLine*. Vol. 48. 27. Elsevier B.V., pp. 68–73. DOI: 10.1016/j.ifacol.2015.11.154.

Bouffard, Patrick, Anil Aswani, and Claire Tomlin (2012). “Learning-based Model Predictive Control on a Quadrotor: Onboard Implementation and Experimental Results”. In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 279–284. DOI: 10.1109/ICRA.2012.6225035.

Bruder, Daniel, Xun Fu, and Ram Vasudevan (July 2021). “Advantages of Bilinear Koopman Realizations for the Modeling and Control of Systems with Unknown Dynamics”. In: *IEEE Robotics and Automation Letters* 6.3, pp. 4369–4376. DOI: 10.1109/LRA.2021.3068117.

Bruder, Daniel, Brent Gillespie, et al. (2019). “Modeling and Control of Soft Robots Using the Koopman Operator and Model Predictive Control”. In: DOI: 10.15607/rss.2019.xv.060.

Brunton, Bingni W. et al. (2016). “Extracting Spatial-temporal Coherent Patterns in Large-scale Neural Recordings Using Dynamic Mode Decomposition”. In: *Journal of Neuroscience Methods* 258, pp. 1–15. ISSN: 1872678X. DOI: 10.1016/j.jneumeth.2015.10.010. URL: http://dx.doi.org/10.1016/j.jneumeth.2015.10.010.

Brunton, Steven L. and J. Nathan Kutz (2019). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press.

Brunton, Steven L., Joshua L. Proctor, et al. (2016). "Discovering Governing Equations from Data by Sparse Identification of Nonlinear Dynamical Systems". In: *Proceedings of the National Academy of Sciences of the United States of America* 113.15, pp. 3932–3937. ISSN: 10916490. DOI: `10.1073/pnas.1517384113`. URL: `http://www.pnas.org/content/pnas/early/2016/03/23/1517384113.full.pdf`.

Budišić, Marko, Ryan Mohr, and Igor Mezić (2012). "Applied Koopmanism". In: *Chaos* 22.4. ISSN: 10541500. DOI: `10.1063/1.4772195`.

Castaneda, Fernando et al. (May 2021). "Gaussian Process-based Min-norm Stabilizing Controller for Control-Affine Systems with Uncertain Input Effects and Dynamics". In: *Proceedings of the American Control Conference* 2021-May, pp. 3683–3690. DOI: `10.23919/ACC50511.2021.9483420`.

Chang, Alexander H. et al. (2017). "Learning to Jump in Granular Media: Unifying Optimal Control Synthesis with Gaussian Process-based Regression". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2154–2160. ISSN: 10504729. DOI: `10.1109/ICRA.2017.7989248`.

Chen, Yuxiao, Huei Peng, et al. (Jan. 2019). "Data-Driven Computation of Minimal Robust Control Invariant Set". In: *Proceedings of the IEEE Conference on Decision and Control*. Vol. 2018-December. Institute of Electrical and Electronics Engineers Inc., pp. 4052–4058. ISBN: 9781538613955. DOI: `10.1109/CDC.2018.8619312`.

Chen, Yuxiao, Andrew Singletary, and Aaron D. Ames (2020). "Guaranteed Obstacle Avoidance for Multi-Robot Operations With Limited Actuation: A Control Barrier Function Approach". In: *IEEE Control Systems Letters* 5.1, pp. 127–132. DOI: `10.1109/lcsys.2020.3000748`.

Cheng, Richard et al. (July 2019). "End-to-End Safe Reinforcement Learning Through Barrier Functions for Safety-Critical Continuous Control Tasks". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01, pp. 3387–3395. ISSN: 2374-3468. DOI: `10.1609/AAAI.V33I01.33013387`. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/4213`.

Choi, Jason et al. (Apr. 2020). "Reinforcement Learning for Safety-Critical Control under Model Uncertainty, using Control Lyapunov Functions and Control Barrier Functions". In: URL: `http://arxiv.org/abs/2004.07584`.

Cohen, Max H. and Calin Belta (Dec. 2020). "Approximate Optimal Control for Safety-Critical Systems with Control Barrier Functions". In: *Proceedings of the IEEE Conference on Decision and Control* 2020-December, pp. 2062–2067. DOI: `10.1109/CDC42340.2020.9303896`.

Cruise (2018). *2018 SELF-DRIVING SAFETY REPORT*. Tech. rep.

Deisenroth, Marc Peter, Dieter Fox, and Carl Edward Rasmussen (2015). “Gaussian Processes for Data-efficient Learning in Robotics and Control”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ISSN: 01628828. DOI: 10.1109/TPAMI.2013.218.

Fisac, Jaime F., Anayo K. Akametalu, et al. (2019). “A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems”. In: *IEEE Transactions on Automatic Control* 64.7, pp. 2737–2752. ISSN: 15582523. DOI: 10.1109/TAC.2018.2876389.

Fisac, Jaime F., Andrea Bajcsy, et al. (May 2018). “Probabilistically Safe Robot Planning with Confidence-Based Human Predictions”. In: URL: https://arxiv.org/abs/1806.00109v1.

Folkestad, Carl and Joel W. Burdick (May 2021). “Koopman NMPC: Koopman-based Learning and Nonlinear Model Predictive Control of Control-affine Systems”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. URL: http://arxiv.org/abs/2105.08036.

Folkestad, Carl, Yuxiao Chen, et al. (Dec. 2021). “Data-Driven Safety-Critical Control: Synthesizing Control Barrier Functions with Koopman Operators”. In: *IEEE Control Systems Letters* 5.6, pp. 2012–2017. ISSN: 24751456. DOI: 10.1109/LCSYS.2020.3046159.

Folkestad, Carl, Daniel Pastor, and Joel W. Burdick (May 2020). “Episodic Koopman Learning of Nonlinear Robot Dynamics with Application to Fast Multirotor Landing”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., pp. 9216–9222. ISBN: 9781728173955. DOI: 10.1109/ICRA40945.2020.9197510.

Folkestad, Carl, Daniel Pastor, Igor Mezic, et al. (July 2020). “Extended Dynamic Mode Decomposition with Learned Koopman Eigenfunctions for Prediction and Control”. In: *Proceedings of the American Control Conference*. Vol. 2020-July. Institute of Electrical and Electronics Engineers Inc., pp. 3906–3913. ISBN: 9781538682661. DOI: 10.23919/ACC45564.2020.9147729.

Folkestad, Carl, Skylar X. Wei, and Joel W. Burdick (2022). “Quadrotor Trajectory Tracking with Learned Dynamics: Joint Koopman-based Learning of System Models and Function Dictionaries”. In: *2022 International Conference on Robotics and Automation (ICRA) (submitted)*.

Gay, Olivier (n.d.). *Libaffa - C++ Affine Arithmetic Library for GNU/Linux*.

Goswami, Debdipta and Derek A. Paley (2018). “Global Bilinearization and Controllability of Control-affine Nonlinear Systems: A Koopman Spectral Approach”. In: *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017* 2018-Janua.Cdc, pp. 6107–6112. DOI: 10.1109/CDC.2017.8264582.

Grandia, Ruben et al. (2020). “Nonlinear Model Predictive Control of Robotic Systems with Control Lyapunov Functions”. In: DOI: 10.15607/rss.2020.xvi.098. URL: http://arxiv.org/abs/2006.01229.

Gros, Sébastien et al. (2020). “From Linear to Nonlinear MPC: Bridging the Gap via the Real-time Iteration”. In: *International Journal of Control* 93.1, pp. 62–80. ISSN: 13665820. DOI: 10.1080/00207179.2016.1222553.

Gurriet, Thomas, Sylvain Finet, et al. (2018). “Towards Restoring Locomotion for Paraplegics: Realizing Dynamically Stable Walking on Exoskeletons”. In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2804–2811. ISSN: 10504729. DOI: 10.1109/ICRA.2018.8460647.

Gurriet, Thomas, Mark Mote, et al. (2019). “An Online Approach to Active Set Invariance”. In: *Proceedings of the IEEE Conference on Decision and Control*. ISBN: 9781538613955. DOI: 10.1109/CDC.2018.8619139.

Hoffmann, Gabe M. et al. (2007). “Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment”. In: *Proc. AAIA Guidance and Control Conf.*

Huang, Bowen, Xu Ma, and Umesh Vaidya (2019). “Feedback Stabilization Using Koopman Operator”. In: *Proceedings of the IEEE Conference on Decision and Control* 2018-Decem.1, pp. 6434–6439. ISSN: 07431546. DOI: 10.1109/CDC.2018.8619727.

Kaiser, Eurika, J. Nathan Kutz, and Steven L. Brunton (2018). “Sparse Identification of Nonlinear Dynamics for Model Predictive Control in the Low-data Limit”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474.2219. ISSN: 14712946. DOI: 10.1098/rspa.2018.0335. URL: https://docs.wixstatic.com/ugd/c50953_e5f09192e29340e3872a333a5556bed1.pdf.

– (June 2021). “Data-driven Discovery of Koopman Eigenfunctions for Control”. In: *Machine Learning: Science and Technology* 2.3, p. 035023. ISSN: 2632-2153. DOI: 10.1088/2632-2153/ABF0F5. URL: https://iopscience.iop.org/article/10.1088/2632-2153/abf0f5%20https://iopscience.iop.org/article/10.1088/2632-2153/abf0f5/meta.

Khalil, Hassan K. (2002). *Nonlinear Systems*. 3rd ed. Pearson.

Kober, Jens, J. Andrew Bagnell, and Jan Peters (2013). “Reinforcement Learning in Robotics: A Survey”. In: *International Journal of Robotics Research* 32.11, pp. 1238–1274. ISSN: 02783649. DOI: 10.1177/0278364913495721.

Koopman, Bernard O. and John V. Neumann (May 1932). “Dynamical Systems of Continuous Spectra.” In: *Proceedings of the National Academy of Sciences of the United States of America* 18.3, pp. 255–263. ISSN: 0027-8424. DOI: 10.1073/PNAS.18.3.255. URL: http://www.ncbi.nlm.nih.gov/pubmed/16587673%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC1076203.

Korda, Milan and Igor Mezić (2018a). “Learning Koopman Eigenfunctions for Prediction and Control: the Transient Case”. In: pp. 1–32. URL: http://arxiv.org/abs/1810.08733.

Korda, Milan and Igor Mezić (2018b). “Linear Predictors for Nonlinear Dynamical Systems: Koopman Operator Meets Model Predictive Control”. In: *Automatica* 93, pp. 149–160. ISSN: 00051098. DOI: `10.1016/j.automatica.2018.03.046`.

Kouzoupis, Dimitris et al. (2018). “Recent Advances in Quadratic Programming Algorithms for Nonlinear Model Predictive Control”. In: *Vietnam Journal of Mathematics* 46.4, pp. 863–882. ISSN: 23052228. DOI: `10.1007/s10013-018-0311-1`.

Lan, Yueheng and Igor Mezić (2013). “Linearization in the Large of Nonlinear Systems and Koopman Operator Spectrum”. In: *Physica D* 242.1, pp. 42–53. ISSN: 01672789. DOI: `10.1016/j.physd.2012.08.017`. URL: `http://dx.doi.org/10.1016/j.physd.2012.08.017%20www.elsevier.com/locate/physd`.

Landry, Benoit et al. (June 2016). “Aggressive Quadrotor Flight Through Cluttered Environments using Mixed Integer Programming”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2016-June, pp. 1469–1475. DOI: `10.1109/ICRA.2016.7487282`.

Lecun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep Learning”. In: *Nature* 521.7553, pp. 436–444. ISSN: 14764687. DOI: `10.1038/nature14539`.

Li, Qianxiao et al. (2017). “Extended Dynamic Mode Decomposition with Dictionary Learning: A Data-driven Adaptive Spectral Decomposition of the Koopman Operator”. In: *Chaos* 27.10. ISSN: 10541500. DOI: `10.1063/1.4993854`. URL: `http://dx.doi.org/10.1063/1.4993854`.

Liu, Cunjia, Hao Lu, and Wen Hua Chen (Sept. 2015). “An Explicit MPC for Quadrotor Trajectory Tracking”. In: *Chinese Control Conference, CCC* 2015-September, pp. 4055–4060. DOI: `10.1109/CHICC.2015.7260264`.

Ljung, Lennart (1987). *System Identification—Theory for the User*. ISBN: 0136566952.

Lupashin, Sergei et al. (2014). “A Platform for Aerial Robotics Research and Demonstration: The Flying Machine Arena”. In: *Mechatronics*. ISSN: 09574158. DOI: `10.1016/j.mechatronics.2013.11.006`.

Lusch, Bethany, J. Nathan Kutz, and Steven L. Brunton (2018). “Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics”. In: *Nature Communications* 9.1. ISSN: 20411723. DOI: `10.1038/s41467-018-07210-0`.

Mamakoukas, Giorgos, Ian Abraham, and Todd D. Murphey (2020). “Learning Data-Driven Stable Koopman Operators”. In: pp. 1–13. URL: `http://arxiv.org/abs/2005.04291`.

Manek, Gaurav and J. Zico Kolter (2020). “Learning Stable Deep Dynamics Models”. In: NeurIPS, pp. 1–9. URL: `http://arxiv.org/abs/2001.06116`.

Mauroy, Alexandre and Jorge Goncalves (2016). “Linear Identification of Nonlinear Systems: A Lifting Technique Based on the Koopman Operator”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 6500–6505.

Mauroy, Alexandre, Igor Mezic, and Yoshihiko Susuki (2020). *The Koopman Operator in Systems and Control*. 1st ed. Springer Nature.

Mauroy, Alexandre and Igor Mezić (2016). "Global Stability Analysis Using the Eigenfunctions of the Koopman Operator". In: *IEEE Transactions on Automatic Control* 61.11, pp. 3356–3369. ISSN: 00189286. DOI: `10.1109/TAC.2016.2518918`.

Mayne, David Q. et al. (June 2000). "Constrained Model Predictive Control: Stability and Optimality". In: *Automatica* 36.6, pp. 789–814. ISSN: 0005-1098. DOI: `10.1016/S0005-1098(99)00214-9`.

Mohr, Ryan and Igor Mezić (2014). "Construction of Eigenfunctions for Scalar-type Operators via Laplace Averages with Connections to the Koopman Operator". In: pp. 1–25. URL: `http://arxiv.org/abs/1403.6559`.

Morris, Benjamin J., Matthew J. Powell, and Aaron D. Ames (2015). "Continuity and Smoothness Properties of Nonlinear Optimization-based Feedback Controllers". In: *Proceedings of the IEEE Conference on Decision and Control* 54rd IEEE.Cdc, pp. 151–158. ISSN: 07431546. DOI: `10.1109/CDC.2015.7402101`.

Narasingam, Abhinav and Joseph Sang-Il Kwon (2020). "Data-driven Feedback Stabilization of Nonlinear Systems: Koopman-based Model Predictive Control". In: pp. 1–11. URL: `http://arxiv.org/abs/2005.09741`.

O'Brien, Ed (2018). *Drone Delivery Update : Moving Forward in the Skies*. Tech. rep., pp. 1–8.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems* 32.

Pedregosa, Fabian et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830. URL: `http://scikit-learn.sourceforge.net.`.

Phan, Minh Q. et al. (2012). "Discrete-time Bilinear Representation of Continuous-time Bilinear State-space Models". In: *Advances in the Astronautical Sciences*. ISBN: 9780877035817.

Pickem, Daniel et al. (2017). "The Robotarium: A Remotely Accessible Swarm Robotics Research Testbed". In: *Proceedings - IEEE International Conference on Robotics and Automation*. ISBN: 9781509046331. DOI: `10.1109/ICRA.2017.7989200`.

Preiss, James A. et al. (July 2017). "Crazyswarm: A Large Nano-quadcopter Swarm". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3299–3304. DOI: `10.1109/ICRA.2017.7989376`.

Proctor, Joshua L., Steven L. Brunton, and J. Nathan Kutz (2016). "Dynamic Mode Decomposition with Control". In: *SIAM Journal on Applied Dynamical Systems* 15.1, pp. 142–161. ISSN: 15360040. DOI: `10.1137/15M1013857`.

Proctor, Joshua L., Steven L. Brunton, and J. Nathan Kutz (Mar. 2018). "Generalizing Koopman Theory to Allow for Inputs and Control". In: *SIAM J. Appl. Dyn. Syst.* 17.1, pp. 909–930. DOI: 10.1137/16M1062296. URL: http://www.siam.org/journals/siads/17-1/M106229.html.

Raković, Saša V. et al. (2006). "Simple Robust Control Invariant Tubes for Some Classes of Nonlinear Discrete Time Systems". In: *Proceedings of the IEEE Conference on Decision and Control*. Institute of Electrical and Electronics Engineers Inc., pp. 6397–6402. ISBN: 1424401712. DOI: 10.1109/cdc.2006.377551.

Rasmussen, Carl Edward and Christopher K. I. Williams (2018). *Gaussian Processes for Machine Learning*. MIT Press. DOI: 10.7551/mitpress/3206.001.0001.

Rawlings, James B. and David Q. Mayne (2012). *Model Predictive Control Theory and Design*. Nob Hill Publishing. ISBN: 978-0-975-93770-9.

Recht, Benjamin (2019). "A Tour of Reinforcement Learning: The View from Continuous Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 2.1, pp. 253–279. ISSN: 2573-5144. DOI: 10.1146/annurev-control-053018-023825.

Richards, Spencer M., Felix Berkenkamp, and Andreas Krause (2018). "The Lyapunov Neural Network: Adaptive Stability Certification for Safe Learning of Dynamical Systems". In: *The 2nd Conference on Robotic Learning*. CoRL. URL: http://arxiv.org/abs/1808.00924.

Rosolia, Ugo, Xiaojing Zhang, and Francesco Borrelli (2018). "Data-Driven Predictive Control for Autonomous Systems". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1, pp. 259–286. ISSN: 2573-5144. DOI: 10.1146/annurev-control-060117-105215.

Rowley, Clarence W. et al. (2009). "Spectral Analysis of Nonlinear Flows". In: *Journal of Fluid Mechanics* 641.Rowley 2005, pp. 115–127. ISSN: 00221120. DOI: 10.1017/S0022112009992059.

Rump, Siegfried M. (1999). *INTLAB — INTerval LABoratory*. DOI: 10.1007/978-94-017-1247-7{_}7. URL: https://link.springer.com/chapter/10.1007/978-94-017-1247-7_7.

Sanchez-Cuevas, Pedro, Guillermo Heredia, and Anibal Ollero (2017). "Characterization of the Aerodynamic Ground Effect and its Influence in Multirotor control". In: *International Journal of Aerospace Engineering* 2017. ISSN: 16875974. DOI: 10.1155/2017/1823056.

Schmid, Peter J. (2010). "Dynamic Mode Decomposition of Numerical and Experimental Data". In: *Journal of Fluid Mechanics* 656, pp. 5–28. ISSN: 14697645. DOI: 10.1017/S0022112010001217.

Schmidhuber, Jürgen (2015). “Deep Learning in Neural Networks: An Overview”. In: *Neural Networks* 61, pp. 85–117. ISSN: 18792782. DOI: `10.1016/j.neunet.2014.09.003`. URL: `http://dx.doi.org/10.1016/j.neunet.2014.09.003`.

Shi, Guanya, Wolfgang Hönig, et al. (Dec. 2020). “Neural-Swarm2: Planning and Control of Heterogeneous Multirotor Swarms using Learned Interactions”. In: *IEEE Transactions on Robotics*, pp. 1–17. URL: `https://arxiv.org/abs/2012.05457v2`.

Shi, Guanya, Xichen Shi, et al. (2019). “Neural Lander: Stable Drone Landing Control using Learned Dynamics”. In: *International Conference on Robotics and Automation (ICRA)*, pp. 9784–9790. URL: `http://arxiv.org/abs/1811.08027`.

Singletary, Andrew, Yuxiao Chen, and Aaron D. Ames (Dec. 2020). “Control Barrier Functions for Sampled-Data Systems with Input Delays”. In: *Proceedings of the IEEE Conference on Decision and Control* 2020-December, pp. 804–809. DOI: `10.1109/CDC42340.2020.9304281`.

Singletary, Andrew, Petter Nilsson, et al. (2019). “Online Active Safety for Robotic Manipulators”. In: *IEEE International Conference on Intelligent Robots and Systems*. ISBN: 9781728140049. DOI: `10.1109/IROS40897.2019.8968231`.

Stellato, Bartolomeo et al. (2018). “OSQP: An Operator Splitting Solver for Quadratic Programs”. In: *2018 UKACC 12th International Conference on Control, CONTROL 2018* 1, p. 339. DOI: `10.1109/CONTROL.2018.8516834`.

Surana, Amit (2016). “Koopman Operator Based Observer Synthesis for Control-affine Nonlinear Systems”. In: *2016 IEEE 55th Conference on Decision and Control, CDC 2016* Cdc, pp. 6492–6499. DOI: `10.1109/CDC.2016.7799268`.

Surana, Amit et al. (2018). “Koopman Operator Framework for Constrained State Estimation”. In: *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*. ISBN: 9781509028733. DOI: `10.1109/CDC.2017.8263649`.

Taira, Kunihiko et al. (2017). “Modal Analysis of Fluid Flows: An Overview”. In: *AIAA Journal* 55.12, pp. 4013–4041. DOI: `10.2514/1.J056060`. URL: `www.aiaa.org/randp.`.

Taylor, Andrew J., Victor D. Dorobantu, Sarah Dean, et al. (Nov. 2020). “Towards Robust Data-Driven Control Synthesis for Nonlinear Systems with Actuation Uncertainty”. In: URL: `http://arxiv.org/abs/2011.10730`.

Taylor, Andrew J., Victor D. Dorobantu, Meera Krishnamoorthy, et al. (2019). “A Control Lyapunov Perspective on Episodic Learning via Projection to State Stability”. In: *Proceedings of the IEEE Conference on Decision and Control*. Vol. 2019-Decem. 0, pp. 1448–1455. ISBN: 9781728113982. DOI: `10.1109/CDC40024.2019.9029226`. URL: `http://arxiv.org/abs/1903.07214`.

Taylor, Andrew J., Victor D. Dorobantu, Hoang M. Le, et al. (2019). “Episodic Learning with Control Lyapunov Functions for Uncertain Robotic Systems”. In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 6878–6884. ISBN: 9781728140049. DOI: 10.1109/IROS40897.2019.8967820. URL: http://arxiv.org/abs/1903.01577.

Taylor, Andrew J., Andrew Singletary, et al. (July 2020). *Learning for Safety-Critical Control with Control Barrier Functions.* URL: https://proceedings.mlr.press/v120/taylor20a.html.

Tran, Duc N., Björn S. Rüffer, and Christopher M. Kellett (Dec. 2016). “Incremental Stability Properties for Discrete-time Systems”. In: *2016 IEEE 55th Conference on Decision and Control, CDC 2016.* Institute of Electrical and Electronics Engineers Inc., pp. 477–482. ISBN: 9781509018376. DOI: 10.1109/CDC.2016.7798314.

Tu, Jonathan H. et al. (2014). “On Dynamic Mode Decomposition: Theory and Applications”. In: *Journal of Computational Dynamics* 1.2, pp. 391–421. ISSN: 21582505. DOI: 10.3934/jcd.2014.1.391.

Turchetta, Matteo, Felix Berkenkamp, and Andreas Krause (2019). “Safe Exploration for Interactive Machine Learning”. In: *Advances in Neural Information Processing Systems.* URL: http://arxiv.org/abs/1910.13726.

Voloshin, Cameron et al. (2021). “The Caltech Off-Policy Policy Evaluation Benchmarking Suite”. In: *Conference on Neural Information Processing Systems.* URL: https://github.com/clvoloshin/COBS.

Wang, Li, Evangelos A. Theodorou, and Magnus Egerstedt (Sept. 2018). “Safe Learning of Quadrotor Dynamics Using Barrier Certificates”. In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2460–2465. DOI: 10.1109/ICRA.2018.8460471.

Wang, Rongyao, Yiqiang Han, and Umesh Vaidya (2021). “Deep Koopman Data-Driven Optimal Control Framework for Autonomous Racing Deep Koopman Data-Driven Optimal Control Framework for Autonomous Racing”. In: *Easy-Chair Preprint.*

Wenk, Philippe et al. (Apr. 2020). “ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04, pp. 6364–6371. ISSN: 2374-3468. DOI: 10.1609/AAAI.V34I04.6106. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6106.

Williams, Matthew O., Ioannis G. Kevrekidis, and Clarence W. Rowley (2015). “A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition”. In: *Journal of Nonlinear Science* 25.6, pp. 1307–1346. ISSN: 14321467. DOI: 10.1007/s00332-015-9258-5.

Zeilinger, Melanie N. (2011). “Real-time Model Predictive Control”. PhD thesis. ETH Zürich. DOI: 10.3929/ethz-a-6619878. URL: https://doi.org/10.3929/ethz-a-6619878.

Zheng, Lei et al. (Oct. 2020). “Learning-Based Safety-Stability-Driven Control for Safety-Critical Systems under Model Uncertainties”. In: *12th International Conference on Wireless Communications and Signal Processing, WCSP 2020*, pp. 1112–1118. DOI: 10.1109/WCSP49889.2020.9299833.