Towards Learning Robotic Dynamics: Application to Multirotor Takeoff and Landing

Thesis by
Daniel Pastor

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2021
Defended February 23th, 2021

Daniel Pastor
ORCID: 0000-0003-3452-0605

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Joel Burdick, to whom I am in great debt for my academic experience at Caltech. He gave me all the freedom I needed while guiding me on the best path forward.

Special thanks to the members of my committee: Dr. Soon-Jo Chung, Dr. Mory Gharib, Dr. Sergio Pellegrino, and Dr. Yisong Yue. In particular, Dr. Chung and Dr. Yue for insightful conversations during the frequent *Learning To Fly* meetings.

I would like to thank *La Caixa* for its Post-graduate Studies Fellowship. It opened me the door to study at Caltech and I would simply not be here without their support. Additional thanks to Raytheon for their financial support during the Learning To Fly project. I am honored to be a Keck Institute for Space Studies fellow, and thankful for all the interesting people I had the opportunity to meet. Thanks to Michele Judd and her relentless efforts.

Thank you to all the people I met at JPL. To Dr. Larry Matthies for welcoming me at the Computer Vision Group. To Dr. Ali Agha and all the members of the Subterranean Challenge. To Dr. Jacob Izraelevitz and Dr. Brett Kennedy for all the achievements SQUID got. Collaborating with JPL perfectly balanced the experience at Caltech.

I must acknowledge the CAST facility, where most of the experiments presented in this thesis were conducted. It is truly a unique facility that allows for fast paced innovation in aerial vehicles. Thank you to its director Dr. Mory Gharib and its manager Reza Nemovi.

Thank you to all the friends I made at Caltech. It really felt like a huge family, from countless barbecues at Sierra Bonita, lunches at Chandler, to late-nights at Gates-Thomas. There are too many individuals to list them here, but special thanks to Gasjod, and all the unnecessary suffering we had together. To the Sierra Bonita dwellers, including Fortran. To all the members of the Burdick group and the warm atmosphere at the office. To Richard, Ke, Joel and Florian. Working side by side with Carl and Amanda made writing papers a breeze; each of you deserve half of this thesis.

Despite the distance, my family has always been the strong foundation I could rely on when I needed it. I am extremely grateful for their tireless efforts and unconditional love. Everything I ever achieved is because of them.

Finally, to my wife Danielle, for being on my side during all these years. You supported me through all the good and bad days, always looking for the best way forward. I cannot think of a better partner to experience the adventures to come.

ABSTRACT

Multirotors have become widespread but their usage is still limited. Ensuring safety during take-off and landing is still an open problem. Towards this goal this thesis proposes two different solutions to address this problem. The two approaches complement each other and they are tested on hardware.

The first approach is to design a vehicle that is stable during take-off, despite hardware failures or unsteady take-off platforms. A solution is to use a ballistic launch to impose a deterministic path, preventing collisions with its environment. Following this approach led to the development of several SQUID (*Streamlined Quick Unfolding Investigation Drone*) vehicles. The main challenges are the ballistic initial flight, large accelerations during launch, and limited volume. A first prototype was developed, which is able to transition mid-flight from stable ballistic flight to a fully controllable multirotor. The system has been fabricated and field tested from a moving vehicle up to 50mph to successfully demonstrate the feasibility of the concept and experimentally validate the design's aerodynamic stability and deployment reliability. A second prototype expanded the first one's capabilities incorporating fully-autonomous vision-based navigation, while keeping the ballistic passive stability and stable transition abilities. The new design includes a more reliable plate-based structure and more effective folding fins.

The second approach focuses on designing controllers that are safe regardless of the platform. For that purpose, a Model Predictive Control (MPC) is used to ensure state and input constraints. Given the highly non-linear dynamics platforms and fast dynamics that require a quick controller evaluation, the work in this thesis is built using Koopman Operator theory, which allows tools from linear analysis to be applied to systems with inherently non-linear dynamics. One of the main contributions is a novel method to find Koopman Eigenfunctions directly from data. Another key contribution is an episodic approach to model non-linear actuation dynamics. The proposed method is first tested on simulation and it outperforms comparable approaches. The method is also demonstrated on-board a multirotor for a fast landing application, where the nonlinear ground effect is learned and used to improve landing speed and quality. An additional extension considers model uncertainty in the MPC architecture, where an Ensemble Kalman Sampler is used to learn the uncertainty distribution.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] A. Bouman, P. Nadan, M. Anderson, D. Pastor, J. Izraelevitz, J. Burdick, and B. Kennedy. Design and autonomous stabilization of a ballistically-launched multirotor. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8511–8517. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9197542. Contribution: vehicle design, experiments and document writing. Content is part of Chapter 3.

[2] C. Folkestad, D. Pastor, and J.W. Burdick. Episodic Koopman Learning of Nonlinear Robot Dynamics with Application to Fast Quadrotor Landing. 2020. doi: 10.1109/ICRA40945.2020.9197510. Contribution: algorithm development, experiment design, document writing. Content is part of Chapter 5.

[3] C. Folkestad, D. Pastor, I. Mezic, R. Mohr, M. Fonoberova, and J.W. Burdick. Extended dynamic mode decomposition with learned Koopman eigenfunctions for prediction and control. In *Proc. American Control Conf.* IEEE, Sep. 2020. doi: 10.23919/ACC45564.2020.9147729. Contribution: algorithm development, MPC code implementation, document writing. Content is part of Chapter 4.

[4] E. Heiden, D. Pastor, P. Vyshnav, and A. Agha-Mohammadi. Heterogeneous sensor fusion via confidence-rich 3d grid mapping: Application to physical robots. In *International Symposium on Experimental Robotics*, pages 725–736. Springer, 2018. doi: 10.1007/978-3-030-33950-0_62. Contribution: experiment design, data analysis, and document writing. Content not included in this thesis.

[5] D. Pastor, J. Izraelevitz, P. Nadan, A. Bouman, J. Burdick, and B. Kennedy. Design of a ballistically-launched foldable multirotor. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019. doi: 10.1109/IROS40897.2019.8968549. Contribution: vehicle design, experiments and document writing. Content is part of Chapter 3.

[6] D. Pastor, C. Folkestad, and J. W. Burdick. Ensemble model predictive control: Learning and efficient robust control of uncertain dynamical systems. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1254–1259, 2020. doi: 10.1109/CDC42340.2020.9304442. Contribution: algorithm development, code implementation, document writing. Content is part of Chapter 6.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

The focus of this thesis is to develop control algorithms and hardware prototypes for highly agile aerial robotic platforms. These kinds of robots are characterized by complex interactions with their environment, strongly actuated dynamics, and unstable autonomous dynamics. To cope with these challenges, this thesis presents results to this problem using two different approaches: the first approach is to create multi-rotor vehicles that are *inherently stable* by design, despite the high level of aerodynamic uncertainty. This is achieved by using traditional aerospace design methods while considering the limitations of small multi-rotors. In particular, Chapter 3 solves the problem of stabilizing a platform that is launched from a vehicle moving at high speed. It led to the development of several SQUID (*Streamlined Quick Unfolding Investigation Drone*) prototypes.

The second approach focuses on designing controllers that are safe regardless of the platform. For this purpose, one of the gaps that was identified is the extensive use of linear models for controllers. But because most systems have at least some non-linear dynamical behavior, the effective usage of controllers based on linear models requires cumbersome engineering. It is clear why linear models are so widely used. There are decades of rigorous analysis for linear systems and the real-time controller evaluation does not pose a problem for the strict time requirements of these systems. Despite their popularity, linear systems cannot accurately model most robots of interest, and careful linearization and gain scheduling is needed when significant nonlinearities are present. This process is slow, and limits the ability of the robot to adapt to changes in the environment.

To deal with these challenges, Chapters 4, 5, and 6 are based on the Koopman Operator, which allows tools from linear analysis to be applied to systems with inherently non-linear dynamics. The main identified issue in this field is the limitation from hand-crafted lifting functions that "lift" the nonlinear dynamics to a higher dimension linear representation. One of the contributions of this thesis is a systematic approach to generate lifting functions. Another gap in the literature is the lack of practical applicability of existing Koopman modeling approaches. For example, [41] requires a dense set of uniformly sampled initial conditions in order to com-

pute the lifting functions, which is not feasible in practice for robotic systems, and many engineering systems of interest. This thesis will presents solutions to these problems, and hardware experiments illustrate the applicability of the approach.

Although the methods developed in this thesis can be applied to any robotic platform, this work focuses on small aerial vehicles, as they are ideal for research and for many practical information gathering applications. On the one hand, their complexity is high enough that they allow multiple research directions to be explored, like path planning [82], non-linear control [47], or uncertainty modeling [44], among many others. Compared to simpler platforms, like the cart pendulum, multirotors offer a rich environment to test algorithms. Additionally, they can easily be constrained to one or two dimensions for initial testing as is shown in Chapter 5. On the other hand, their usage is far simpler than other complex platforms, like bipedal robotics [2] or autonomous cars, that are very expensive, require cumbersome maintenance, and need an extensive domain knowledge to get started. Besides research, multirotors' main usage is in cinematography, and they are expanding into sectors such as inspection, agriculture, mapping, and surveying.

The remainder of this chapter will give an overview of robotic control systems from a general perspective, using examples from multirotor control and providing a brief literature review. This discussion will help to frame each chapter into appropriate context. See the introduction of each chapter for more specific related work and motivation.

1.1 Controlling Robotic Systems

The control problem of a robotic platform consists of generating the next control command to be sent to the actuators. This command is computed taking into consideration the prescribed goal and the history of measurements. Examples of typical goals are reaching a destination, exploring an area, or following a trajectory.

The solution of this problem is a continuous vector of actuation commands. Given the nature of digital computers and the need to sample the measurements, the actuation commands are only updated at finite time increments. Additionally, evaluating the controller and testing goal satisfaction take a non-zero amount of time, limiting the effective update rate. For most practical applications, the control command is updated after evaluating each measurement, and the complexity is adjusted so that no measurements are discarded. Examples of common measurements are camera images, time of flight distance sensing, or acceleration measured by an accelerom-

eter.

The process of using measurements sequentially splits the control problem into two seemingly independent problems: first, measurements can be used to update the controller. This process will be refereed as *learning*, and next section will cover it in more detail. Second, after the learning update, the control command is computed directly from the current state.

In *model-based control*, the calculation of the control command includes an explicit parametrization and consideration of the dynamical model that describes the evolution of the state. Linear models in continuous time are the most popular choice, combined with a linear controller. In *model-free control*, the control inputs do not explicitly include the dynamical system model. Historically, model-free control has been relegated to the control of simple models for online feedback control. The simplest model, a constant term, is a key element of most control implementations. It is widely used to ensure zero stationary error for constant disturbances. Model-free control with high capacity models is widely used in the reinforcement learning approach for control. While traditional control has focused on finding stability and safety guarantees using relatively simple models, reinforcement learning addresses the same problem, but it incorporates the learning problem as part of the problem setup. Its roots come from the Machine Learning community and it has been highly successful in synthesized control policies directly from raw pixels [51].

Model Predictive Control (MPC) computes the control input by solving an optimization problem at each iteration. Originally relegated to problems with slow dynamics such as chemical plants [26] due to its expensive computational cost, it is now a fundamental control technique thanks to its ability to incorporate state and input constraints. Applications include aerospace [22], multirotors [52], and process control [75]. Despite all these advantages, MPC has several shortcomings besides the aforementioned computational cost. One of the problems is the need to control the receding horizon length to ensure stability and recursive feasibility. In [70], the author uses past trajectories of repetitive tasks as target goals for short horizon MPC, thus ensuring cost improvement over several repetitions. A survey of robust MPC can be seen in [5].

1.2 Learning Robotic Systems

Learning has been always an integral part of any scientific discipline, including robotics. In this context, learning is considered as the process of modifying the

control actions based on new data. Using this definition, learning is ubiquitous in a robotic system, from the initial design to the state estimation subsystem, including iterative model improvements.

A learning problem is defined using 3 elements: a dataset, a hypothesis set, and an identification method. The dataset includes all past measurements and rewards/costs.

The hypothesis set is the set of all considered controllers. It is normally specified using a model class, for example, polynomials of a given order or a neural network with a given structure. As mentioned above, a useful approach is to parametrize the state evolution equations, the model, as part of the controller design. Linear models are traditionally at the core of any robotic design, as any system can be approximated by a linear model close to an operating point. In addition, the combined response of a set of linear systems will be the combination of the responses of each individual component, allowing design and analysis of individual components independently. This led to the development of powerful methods to study and design linear models. However, any deviation from a linear model has to be captured as a disturbance, resulting in conservative or even not feasible controllers. One solution is to compute the approximated linear model at each state. Similarly, gain scheduling divides the operation regime in different areas, each with its own controller. This allows to optimize the controller for its operation regime and this approach only requires to ensure stability on finitely many transitions. Non-linear models allow more accurate representations, which reduces the necessary disturbance in the model to satisfy the data, and they can potentially be used to synthesize better controllers.

Another alternative to linear models is to use Koopman spectral analysis, and it is one of the main focuses of this thesis. It uses non-linear functions to lift the original states to a higher dimensional system where a linear model can more accurately predict the original system's evolution. This approach has gained a lot of attention in the field of fluid dynamics as it can generate parsimonious representations directly from data. For robotic applications this approach allows one to apply linear controller theory to systems whose linearized model would not produce accurate predictions. In addition, the computation of the lifting function and the linear controller can occur significantly faster than comparable non-linear models. As shown originally by Korda [41], Koopman-based modeling can be combined with Model Predictive Control to generate a controller that satisfies state and input constraints. In comparison, methods that directly model the right hand side of the dynamics require to solve a computationally expensive non-convex optimization

problem without guarantees. The main limitation of Koopman-based methods is that the controlled dynamics have to be formulated as linear in the lifted space, and Chapter 5 proposes a solution to this problem. Another limitation is the need of a equilibrium point in the autonomous dynamics. It limits to directly apply the method to systems with unstable dynamics or limit cycles, and it requires a known controller to stabilize the system.

The identification method is the procedure to select a subset of the hypothesis set using the dataset. For identification problems that can be formulated as a convex optimization problem, fast and reliable methods have been developed, for example, least squares methods. For general non-convex problems, such as fitting a neural network, gradient methods are usually employed. In this thesis, both approaches are used, depending on the situation. For example, the process of learning Koopman Eigenfunctions in Chapter 4 uses Extended Dynamic Decomposition [86] to learn the linear dynamics in the lifted system, and gradient descent to learn the non-linear diffeomorphism.

The main classification of a learning algorithm is *first-principles* vs *end-to-end learning*: in first-principles learning, the robotic system is split into smaller subsystems, and each of those subsystems is identified individually. This facilitates validation, allows sharing of models between systems, and it is more aligned with the models used for system design. Each model can be then executed at the required frequency on the required computer, allowing great flexibility. This approach is also referred as physics-based approach in the literature. For example, the airplane wings model, engine model, and landing gear model are learned independently, and then combined to generate a model of the full aircraft. Ultimately, each subsystem is decomposed into simple components, like materials and fluids, for which their models are already known through basic experimentation. This approach can use information about the design of the platform as data to fit the model. For example, a standard multirotor model considers the force for each individual motor to be parallel, not as a result of any curve fitting, but as data coming from the design process. Chapter 3 is a clear example of this approach.

Alternatively to the first-principles approach, one can identify the model for the whole system without looking into its components. This approach has been fueled by advances in the machine learning community, and it is also referred to as end-to-end learning. The main advantage of this type of learning is that it does not require any additional data not available during operations. This reduces the need

for expert knowledge for each of the subsystems, and it could allow the robot to adapt better to unexpected changes in the environment. Moreover, it allows for end-to-end learning using complex models whose usage would be infeasible using subsystem methods. A clear success story can be seen when applied to high dimensional measurements, such as raw images. The main disadvantage is the increased amount of operational data required. Gathering all that data could be very expensive or even not practical, for example, to satisfy the strict safety requirements of 10^{-9} failures/hour in the aerospace industry. Another disadvantage is that the model has to be evaluated entirely at every timestep, imposing strong computational requirements on the platform.

In practice, as it is done in the experiments in this thesis, it is better to use a combination of both approaches, where subsystem data helps to create a sparse controller structure, and operational data is used to fit the selected function. Chapters 4, 5, and 6 will focus on the data-driven learning, keeping the first-principles learning in the layers before the model, state estimation, and in the layers after the model, i.e. low level control.

1.3 Thesis Contribution and Organization

Chapters 2 describes basic concepts for multirotor control and Model Predictive Control (MPC). These concepts will serve as background material for the rest of the chapters.

Chapter 3 presents the development of SQUID, the aerial platform for ballistic launch and quick deployment. The contributions include a requirements list for multirotor ballistic launch, an analysis of passive stability for ballistic launches, a design that satisfies those previous requirements, and extensive testing to validate the design. Two distinct prototypes will be explained: a 3" in diameter proof of concept with streamlined body and the capability to release the arms mid-flight. This prototype was tested from a moving truck and it relies on human pilot control. The contributions of the second prototype are the development of a vision-based autonomy pipeline. It required a new design that was validated on a wind tunnel for cross-wind launches.

Chapters 4, 5, and 6 are related to the Koopman Operator. Chapter 4 proposes a new method to compute the Koopman Eigenfunction. The first contribution is how to build Koopman eigenfunctions from the principal eigenfunctions directly from data. An analytical example is shown to illustrate the method. The Koopman

Eigenfunction are then used as lifting functions to learn the approximated linear dynamics. It is shown in simulation that it has better prediction capabilities than user-defined lifting functions. In addition, it also shows an improvement in closed-loop performance when it is used with a MPC controller.

Chapter 5 builds upon the previous chapter to learn models with non-linear control actions. These kinds of dynamics are important for practical robotic applications. The proposed method sidesteps the linear input dynamics limitations by exciting the non-linear control effects and then learn then as part of the non-linear Koopman autonomous dynamics. This process creates a series of episodes where each subsequent controller learns the residual dynamics using the methods in the previous chapter. The importance of these contributions is shown in hardware experiments of multirotor landing. The proposed controller can quickly generate at each timestep optimal input commands, respecting state and input bounds.

Chapter 6 describes a new type of robust MPC based on Ensemble Kalman Sampling. It extends the controller used in Chapter 4 to include model uncertainty. This chapter also shows how this method can be applied to Koopman-based MPC.

Finally, Chapter 7 concludes the thesis and presents future work directions.

Chapter 2

PRELIMINARIES

This chapter describes some of the fundamental concepts used in later chapters of this thesis. Section 2.1 describes basic concepts for multirotor control, used later in Chapter 3 to discuss the development of SQUID, and in Chapter 5 to support the discussion of the Episodic KEEDMD experiments. Section 2.3 introduces Model Predictive Control (MPC) as the main control algorithm that will be used in Chapters 4, 5, and 6.

2.1 Multirotor Basics

A multirotor is an aerial vehicle with more than one lift-generating motors, as opposed to a helicopter with only one motor to generate lift. Most commonly, they have 4 motor or more, so the control of the vehicle can be achieved by changing the speed of each motor. This design avoids the complex and delicate swashplate mechanism that helicopters use for control. Combined with advances in batteries, this simple design helped multirotors surge in popularity for small vehicles. For convenience, this section will focus on the most common type with 4 parallel motors, but the control techniques can be generalized to higher number of motors. Figure 2.1 shows a picture of a multirotor with its main reference frames. The vehicle state is defined using the position in world coordinates of the center of mass $p = [x, y, z] \in \mathbb{R}^3$, the velocity of the center of mass $v = [v_x, v_y, v_z] \in \mathbb{R}^3$, the rotation $R \in SO(3)$ from the body reference frame to the world reference frame, and the instantaneous rotation vector $\omega \in \mathbb{R}^3$ in world coordinates. This section will also use the 321 Euler Angles, that is, a rotation ψ around the z axis, a rotation θ around the y axis, and a rotation ϕ around the x axis, in this order. For more details on transformations for robotics, see [36].

2.2 Control

To control a multirotor, a speed command has to be generated for each motor. The command is normally referred to as *PWM*, Pulse Width Modulation, a communication protocol used to transmit the desired speed from the microcontroller to the electronic speed controllers (ESC). To generate these commands, a cascaded controller is used, that is, the control problem is divided into multiple sequential

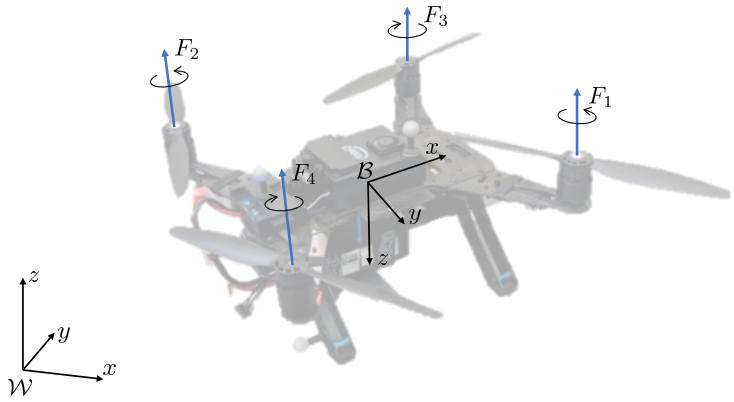


Figure 2.1: Coordinate reference frames used to describe the multirotor dynamics. Body frame $\mathcal{B}$ is rigid to the vehicle, while world frame $\mathcal{W}$ is considered an inertial frame.

problems and the output of one subcontroller is the command for the next one.

For autonomous flight, the following controllers are used, in order from the goal to the motors: a trajectory planner, a higher level attitude controller, a lower level torque controller, and a motor mixer. Most of these controllers are already implemented by the autopilot and the research in this thesis only replaces the trajectory planner and the attitude controller, leaving the low level controller and motor mixer from the default stock autopilot implementation.

The trajectory planner computes the sequence of positions in order to reach a final goal position. The common approach, presented in [57], is to use $q = [x, y, z, \psi]$ as flat outputs [17], and minimize its 4th derivative. The states and the inputs can be written as a combination of the 4th derivative of previously chosen flat outputs. This allows to write state and input constraints directly in the optimization problem.

The higher level controller transforms desired positions into desired attitude. Newton's equations is used

$$m\dot{\boldsymbol{v}} = -mg\boldsymbol{z}_W + F\boldsymbol{z}_B \tag{2.1}$$

where m is the vehicle's mass, g is the gravity's constant, $\boldsymbol{z}_W = [0, 0, 1]^{\mathrm{T}}$ is the unity vector along the z axis in the world coordinate frame, and $\boldsymbol{z}_B = R\boldsymbol{z}_W$ in body coordinate frame. As the motors can only generate force parallel to $\boldsymbol{z}_B$, the desired attitude should have the vertical axis parallel to $\boldsymbol{z}_B$.

$$\boldsymbol{F}_d = mg\mathbf{z}_W + m\ddot{\boldsymbol{p}}_{\boldsymbol{d}} \tag{2.2}$$

Moreover, a proportional derivative control action is added to balance the equation as

$$\boldsymbol{F} = \boldsymbol{F_d} - K_p \boldsymbol{e}_p - K_v \boldsymbol{e}_v \tag{2.3}$$

where e_p is the error in position and e_v is the error in velocity with respect to the desired trajectory. The remainder degree of freedom is commonly chosen so that the vehicle points forward during the trajectory. This defines the desired rotation R_d.

Using the desired attitude, the lower level controller computes the desired torques $\tau = [\tau_\phi, \tau_\theta, \tau_\psi]^\mathsf{T}$. Similarly to Equation (2.3), a proportional derivate controller is used to satisfy the desired value as

$$\tau = K_\tau e_R \tag{2.4}$$

where e_R is the error in orientation using the vee map $\vee : \mathfrak{so}(3) \to \mathbb{R}^3$

$$\mathbf{e}_R = \frac{1}{2}\left(R_d^\mathsf{T} R - R^\mathsf{T} R_d\right)^\vee. \tag{2.5}$$

The motor mixer simply converts the commanded torque and total force to motor commands using the geometry of the vehicle and motor characteristics. On first approximation, dimensional analysis can be used to model motor force and torque proportional to the squared speed, $F_i = c_f \omega_i^2 = c_f u_i$. The total force is the sum of all forces $F = \sum_i F_i$. Each motor contributes with a torque $\tau_\psi = s_i c_t$, where s_i is $+1$ or -1 depending on if it rotates clockwise or counterclockwise, and a term $F_i l$ from the force F_i with arm l. For the case with 4 motorsm the solution is unique and it is given by

$$\begin{pmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} = \begin{pmatrix} c_f & c_f & c_f & c_f \\ 0 & -\ell c_f & 0 & \ell c_f \\ \ell c_f & 0 & \ell c_f & 0 \\ -c_t & c_t & -c_t & c_t \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \mathcal{M} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} \tag{2.6}$$

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \mathcal{M}^{-1} \begin{pmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix}. \tag{2.7}$$

Besides autonomous flight, there are several *manual* modes that are used to control the vehicle by a human pilot. This mode of operation is used in Chapte 3 for the

SQUID 3" prototype, and during initial testing in Chapter 5. The signal is transmitted using a dedicated radio-link at 2.4GHz optimized for low latency. There are two principal modes, depending of the input. In *angle mode* (termed manual in PX4), 3 of the joystick inputs are interpreted as desired angle, and the autopilot follows this command using the same low level controller described above. The remaining joystick axis is read as total thrust command and it is fed directly in the mixer. This mode makes it easier to control the vehicle near hover conditions. In *rate mode*, or acro mode, 3 of the inputs are desired angular rates. The autopilot compares them directly to the gyroscope read outs and it generates a torque command using a PID controller on the angular rate error. As in angle mode, the fourth input is total thrust, and it is combined in the mixer with the torque commands to generate the PWM commands to the motors.

Estimation

To estimate its position, a multirotor normally uses several sensors. An inertial measurement unit (IMU) provides acceleration and angular velocity in body coordinates. A standard IMU has a sampling frequency of 1kHz. The acceleration can only be integrated for very short periods of time before it is corrupted by noise. To solve this issue, an additional sensor is required. For experiments inside CAST (Center for Autonomous Systems and Technologies) pose measurements are directly available. These measurements are very precise but they limit the usage to purpose built rooms. More generally, on-board cameras are widely used due to its small size, power requirements and rich information.

2.3 Model Predictive Control (MPC)

In Model Predictive Control (MPC), the control input is obtained by minimizing the future cost, given the initials conditions $x(t_0) = x_0$, and subject to the model dynamics $x_p = f(x_{p-1}, u_{p-1})$ to propagate the state given the previous state $x \in \mathbb{R}^{N_s}$ and control input $u \in \mathbb{R}^{N_u}$, state constraints $x \in \mathcal{X}$ and control input constraints $u \in \mathcal{U}$. Figure 2.2 shows a diagram with the main elements of an MPC problem. It can be written as the following optimization problem

$$
\begin{aligned}
\min_{\substack{\boldsymbol{u}\in\mathbb{R}^{N_u\times N_p}\\ \boldsymbol{x}\in\mathbb{R}^{N_s\times N_p}}} \quad & \textstyle\sum_{p=1}^{N_p-1} l(x_p, u_p, \tau_p) + l_f(x_{N_p}) \\
\text{s.t.} \quad & x_p = f(x_{p-1}, u_{p-1}) \\
& x_p \in \mathcal{X} \qquad\qquad p = 1, \ldots, N_p \\
& u_p \in \mathcal{U} \\
& x_0 = x_k
\end{aligned}
\tag{2.8}
$$

where N_p is the prediction horizon of the controller, $l : \mathbb{R}^{N_s} \times \mathbb{R}^{N_u} \times \mathbb{R}^{N_s} \to \mathbb{R}$ is the *stage cost* that encodes the objective of the control task at each stage, $l_f : \mathbb{R}^{N_s} \to \mathbb{R}$ is the terminal cost function, $\boldsymbol{\tau} \in \mathbb{R}^{N_s \times N_p}$ is the reference trajectory, $\mathcal{X} \subseteq \mathbb{R}^{N_s}$ is the set of allowable states, $\mathcal{U} \subseteq \mathbb{R}^{N_u}$ is the set of allowable control actions, and x_k is the state at time step k.

The solution of (2.8) is a sequence of control actions $\boldsymbol{u} \in \mathbb{R}^{N_u \times N_p}$. The predicted trajectory at time k_1 evaluated using information up to timestep k_2 will be referred as $x_{k_1|k_2}$. To allow the controller to mitigate model errors, state feedback is introduced by performing receding horizon control, *i.e.* , only the first controller command, $u_{k|k}$, is deployed on the system at timestep k before the optimization problem is resolved at the next timestep [9].

Note that the cost, the constraints, and the reference trajectory are considered known, and they are normally imposed by the user. However, the dynamic evolution and the initial state are often approximated from data, and they are obtained through system identification before the timestep k. If the data is obtained from past experiments, the identification process is often called *offline learning*. If the identification occurs simultaneously with control execution, the identification process is often called *online learning*.

This thesis will mostly consider problems where the stage cost is a quadratic function. This is the most common cost function and it simplifies calculations because efficient MPC solvers exist in this case. Quadratic costs arise naturally for gaussian disturbances. In addition, for notation simplicity, constraint will apply to one state at a time. Under these conditions, the problem can be written as:

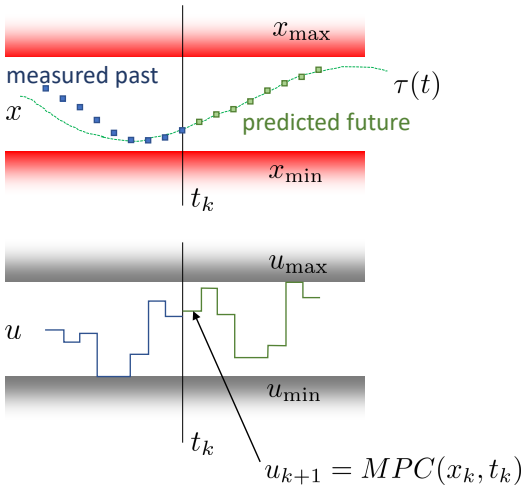


Figure 2.2: Model Predictive Control (MPC) diagram. At each time step t_k, an optimization problem is solved to compute the best control action u using the desired trajectory $\tau(t)$. State constraints $x_{\min}, x_{\max}$ and input constraints $u_{\min}, u_{\max}$ can be enforced as part of a constrained optimization problem. Once a solution sequence is found, the first value is executed for the remaining time step.

$$
\begin{aligned}
\min_{\substack{\boldsymbol{u}\in\mathbb{R}^{N_u\times N_p}\\ \boldsymbol{x}\in\mathbb{R}^{N_s\times N_p}}} \quad & \textstyle\sum_{p=1}^{N_p-1}[(x_p-\tau_p)^T Q(x_p-\tau_p)+u_p^T R u_p]+x_N^T Q_N x_N) \\
\text{s.t.} \quad & x_p = f(x_{p-1}, u_{p-1}) \\
& x_{\min} \le z_p \le x_{\max} \qquad\qquad p = 1,\dots,N_p \\
& u_{\min} \le u_p \le u_{\max} \\
& x_0 = x_k
\end{aligned}
\tag{2.9}
$$

where

Solving problem (2.9) in real-time is normally not feasible for nonlinear dynamics. A common approach is to linearize the dynamics around an initial solution, solve the generated quadratic program, and iterate with the new solution until it converges.

Dense Form MPC

To reduce the size of the optimization problem, we remove the dynamic constrain by explicitly solving them. Starting at $z_0 = z_k$, the z_1 results from the first iteration of the dynamic model: $z_1 = Az_0 + Bu_0$. The next state, z_2, and subsequent states

result from iteration, starting at z_1:

$$z_1 = Az_0 + Bu_0 \tag{2.10}$$
$$z_2 = Az_1 + Bu_1 \tag{2.11}$$
$$= A(Az_0 + Bu_0) + Bu_2 \tag{2.12}$$
$$= A^2 z_0 + ABu_0 + Bu_1 \tag{2.13}$$
$$\dots \tag{2.14}$$
$$z_N = A^N z_0 + A^{N-1}Bu_0 + \cdots + ABu_{N-2} + Bu_{N-1} \tag{2.15}$$
$$= A^N z_0 + \sum_{i=0}^{N-1} A^{N-i} B u_i. \tag{2.16}$$

Writing each variable vector with its bold equivalent, *i.e.* $\mathbf{z} = [z_0, \dots, z_{N+1}]$, the linear relationship can be expressed as:

$$\mathbf{z} = az_0 + b\mathbf{u} \tag{2.17}$$

where

$$a = \begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}, b = \begin{bmatrix} B & & & & \\ AB & B & & & \\ A^2B & AB & B & & \\ \vdots & \vdots & \ddots & & \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \cdots & B \end{bmatrix},$$

then substituting these terms in the objective function yields

$$J(z,u) = (z_p - \tau_p)^T Q (z_p - \tau_p) + u_p^T R u_p \tag{2.18}$$
$$J(u) = ((az_0 + b\mathbf{u}) - \tau_p)^T Q ((az_0 + b\mathbf{u}) - \tau_p) + u_p^T R u_p \tag{2.19}$$
$$= \boldsymbol{u}^T \left[R_{bd} + b^T Q_{bd} b\right] \boldsymbol{u} + u^T b^T Q_{bd}(ax_0 - \tau_r) \tag{2.20}$$

This result can be compared to a quadratic program in standard form

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2} x^T P x + q^T x \\ &\text{subject to} && l \le Ax \le u \end{aligned} \tag{2.21}$$

where

$$P = R_{bd} + b^T Q_{bd} b \quad q = b^T Q_{bd}(ax_0 - \tau_r) \quad A = [b; I_m]$$
$$l = [x_{min} - ax_0, u_{min}] \quad u = [x_{max} - ax_0, u_{max}]$$

Chapter 3

SQUID

This chapter describes the design of several air vehicles for safe takeoff from a moving platform. The contents of this chapter have been previously published in two collaboration papers. The first paper was presented at the 2019 International Conference on Intelligent Robots and Systems (IROS) [63]. It introduced the basic concepts and results from the first prototype, SQUID 3". The second paper was presented at the 2020 IEEE International Conference on Robotics and Automation (ICRA) [1], where it won the *Unmanned Aerial Vehicles* Best Paper Award. It extended the basic principles with a bigger design and it focused on vision-based autonomous position stabilization. This new design was led by Amanda Bouman.

3.1 Introduction

Unmanned fixed-wing and multirotor aircraft are usually launched manually by an attentive human operator. Aerial systems that can instead be launched ballistically without operator intervention can play an important role in emergency response and space exploration applications where the situational awareness provided by a camera-equipped aerial vehicle is often required, but the ability to conventionally launch aircraft to gather this information is not available.

For example, firefighters responding to massive and fast-moving fires could benefit from the ability to quickly launch drones through the forest canopy from a moving vehicle. This eye-in-the-sky could provide valuable information on the status of burning structures, fire fronts, and safe paths for rapid retreat.

Takeoff is one of the most dangerous portions of a multirotor's flight, as it involves hazards to not only the multirotor, but also other assets on the ground, such as nearby humans and their equipment or infrastructure. A ballistic launch addresses this problem by creating a pre-determined path for the multirotor away from higher-value assets, even in the case of aircraft failure. A typical scenario would involve deployment from a windy roof, the bed of a truck, or a ship operating in waves. In these scenarios, the vehicle is stored for long periods of time and must quickly provide air support in the case of an unexpected event. Current drone designs are slow to deploy, require user intervention prior to takeoff, and cannot be deployed

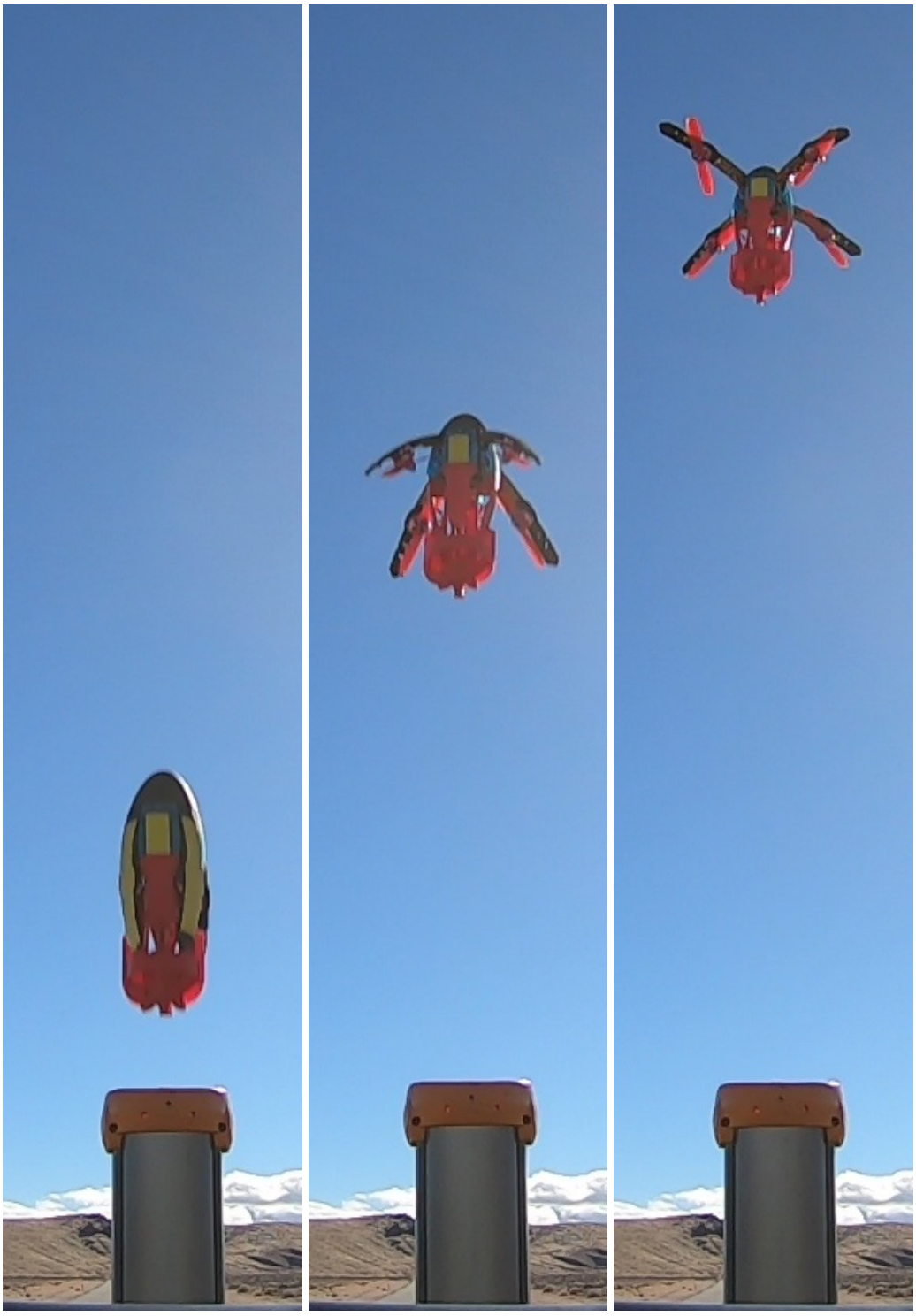


Figure 3.1: The *SQUID 3″* prototype in both ballistic (left), deploying its arms (center) and multirotor (right) configurations during flight. Selected frames during a launch from a moving vehicle at 50mph (22 m/s). After deployment the drone hovered around the area.

from a moving vehicle. Furthermore, traditional foldable designs require the user to unfold the arms, slowing the process and putting the user at risk. In the case of deployment from a moving vehicle, the drone also needs to be aerodynamically stable to avoid tumbling when exposed to sudden crosswinds. The design, development and testing of such a vehicle is the main contribution of this chapter.

While mature tube-launched fixed-wing aircraft are already in active use [49, 68, 84], tube-launched rotorcraft (both co-axial and multirotor) are much rarer and primarily still in development. Several consumer drones (e.g., the DJI *Mavic* series [19] and Parrot *Anafi* [62]) can be folded to occupy a small volume, but these designs cannot fit smoothly inside a launch system, and the unfolding is manual. Other manually unfolding rotorcraft can achieve a cylindrical form factor like *SQUID*: the *Power Egg* from Power Vision folds into an egg shape [66], the *LeveTop* drone folds into a small cylinder [50], and the coaxially designed *Sprite* from Ascent Aerosystems packs into a cylinder shape [1]. Automatic in-flight unfolding mechanisms for quadrotors, using both active [25] and passive [15] actuation, have been developed for the traversal of narrow spaces. However, to enable the ability to ballistically launch like *SQUID*, these existing foldable platforms must be redesigned to withstand launch loads and maintain passive aerodynamic stability post-launch. Ballistically-launched aerial systems that combine an aerodynamically stable structure and a foldable airfoil system have been developed in coaxial rotorcraft [31] and multirotor [32] formats, but both designs are theoretical designs , and have yet to demonstrate a transition from ballistic to stabilized flight.

While the SQUID prototype, as outlined in this chapter, has been designed for operation on Earth, the same concept is potentially adaptable to other planetary bodies, in particular Mars and Titan. *Ingenuity*, the future Mars helicopter, planned to deploy from the Perseverance rover, will provide a proof-of-concept for powered rotorcraft flight on the planet, despite the thin atmosphere [61]. A rotorcraft greatly expands the data collection range of a rover, and allows access to sites that a rover would find impassible. However, the current deployment method for the Mars Helicopter from the underbelly of the rover reduces ground clearance, resulting in stricter terrain constraints. Additionally, the rover must move a significant distance away from the helicopter drop site before the helicopter can safely take off. The addition of a ballistic, deterministic launch system for future rovers or entry vehicles would isolate small rotorcraft from the primary mission asset, as well as enable deployment at longer distances or over steep terrain features. Titan is another major

candidate for rotorcraft flight. The Dragonfly mission proposal to the New Frontiers Program illustrates how rotorcraft can take advantage of the thick atmosphere and low gravity of Titan to fly to many different sites with the same vehicle [80]. A SQUID-type launch applied to Titan could be used for deployment of small daughter rotorcraft from landers, airships, or lake buoys, expanding the option space for Titan mission design.

3.2 Scaling Arguments

The SQUID concept can be implemented at many different sizes, depending upon the needs for specific missions or applications. This section considers the principles of scaling the SQUID concept to different sizes.

When designing a ballistic launch for a different-sized SQUID (larger tube diameter, etc.), the following non-dimensionalized argument can be used to predict the aerodynamic performance. The idea is that a specific design that has been successful in testing at one given size can be scaled to other applications, with a high confidence of success. Dimensional analysis has been widely used in aerospace applications [30] to design experiments. This scaling analysis broadens the scope of our field testing conclusions, which can then be applied to other aircraft given the appropriate scaling.

The launch trajectory of the multirotor must be a function of an input variable set; namely the launch velocity (U), vehicle velocity (U_{vehicle}), air properties (density and viscosity ρ and μ), gravity (g), time (t), and the geometry of the aircraft (mass m, diameter d, length L, inertia I). Given that these input variables can be expressed using three independent physical units (mass, time, and length), we can describe the same equations using three fewer non-dimensional variables than input variables using the Buckingham π Theorem [46]. To compute the non-dimensional variables, we build the dimensional matrix using the dimension power of each variable (length, time, and mass) and we solve for its nullspace. The following non-dimensional variables accordingly span the input space:

$$\tilde{t} = \frac{tU}{L}, \quad \textit{Fr} = \frac{U}{\sqrt{gL}}, \quad \textit{Re} = \frac{\rho U L}{\mu}, \tag{3.1}$$

$$\tilde{U}_{\text{vehicle}} = \frac{U_{\text{vehicle}}}{U}, \quad \tilde{m} = \frac{m}{\rho L^3}, \quad \tilde{d} = \frac{d}{L}, \quad \tilde{I} = \frac{I}{\rho L^5} \tag{3.2}$$

where *Fr* is the Froude number and *Re* is the Reynolds number. Further nondimensional groups can represent the fin area ratio A_{fin}/L^2 and other geometry details,

but are generally held constant for exact scale models. Reynolds number *Re* effects are expected to be minimal and can be neglected for models scaled by a single order-of-magnitude, as drag coefficients are only weakly dependent on *Re* given the fully transitioned flow and only partial streamlining of the model [34].

Finally, the trajectory during launch (position $x(t)$, $y(t)$, $z(t)$ and rotation $R(t)$) once non-dimensionalized can only be a function of these input groups. For example for $x(t)$:

$$\tilde{x}(\tilde{t}) = \frac{x(\tilde{t})}{L} = f_x(\tilde{t}, \mathit{Fr}, \tilde{U}_{\text{vehicle}}, \tilde{m}, \tilde{d}, \tilde{I}), \tag{3.3}$$

Accordingly, the trajectory of the 3" SQUID prototype that was launched at 35mph from a 50mph vehicle ($\mathit{Fr} = 9.4$, $\tilde{d} = 0.27$, $\tilde{U}_{\text{vehicle}} = 1.4$) can be used to predict trajectories for scaled prototypes. For example, a 2x scale model (i.e. 8 times the weight, 32 times the inertia, etc.) launched at 50mph from a 70mph vehicle will match these same non-dimensional inputs. Such a model would therefore follow the same trajectory scaled by 2x the distance and take $\sqrt{2}$ times the amount of time to do so.

3.3 SQUID 3": Proof of Concept

The first design targeted a launcher with a 3 inches tube diameter. The smaller size allowed for fast design iterations and a low mass vehicle that is relatively safe to operate. The main limitations of this first small design are its limited payload and reduced flight time due to its small onboard batter.

This section will describe the design process for the first prototype as a requirements-driven process. These requirements are:

(a) it will be launched from an approximately 3 inch tube (70-85mm),

(b) it should fly ballistically to reach an altitude of 10m,

(c) it should be able to stabilize its flight after launch. In addition,

(d) it should be a multirotor,

(e) it should be able to carry a payload of 200g.

From this set, of requirements we derive functional requirements that help the design process: the first requirement sets a form factor and, combined with requirement

(d), requires that the vehicle be able to deploy a set of arms that hold the motors. Requirement (a) also implies high vertical acceleration loads during launch, up to 50g's from direct measurements, which will drive the structural design. Requirement (e) does not constrain the design space, as the vehicle is more volume limited than thrust limited.

Operations

The operation of SQUID is composed of six different phases from loading to controlled flight. See Figure 3.2 for an illustrative diagram.

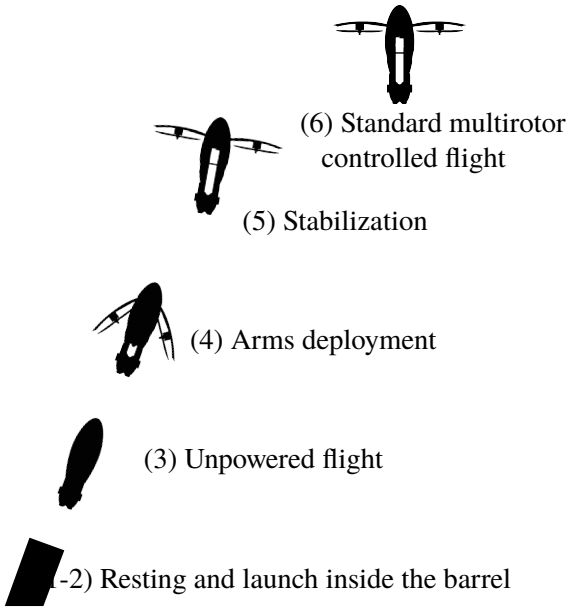


Figure 3.2: SQUID 3" deployment sequence

1. **Resting inside the launching device:** The vehicle is static and ready to be launched. Before this phase, the vehicle's onboard electronics and control systems are turned on and armed. In order to maintain compatibility with the nature of the onboard PX4-based control and navigation system, the vehicle's command state is set to the *'kill'* mode, which causes the onboard controller to neglect all input commands.

2. **Launch acceleration inside the barrel:** After launch is triggered, the compressed air accelerates the vehicle through a barrel (76 cm in length for the 3" SQUID) with high accelerations and high volumetric g forces. This acceleration can be used by the on board autopilot to detect the launch. Figure 3.10

shows a typical acceleration profile throughout all phases of flight operation. The first acceleration spike corresponds to the launch acceleration. All SQUId 3" test used a pneumatic ZS740 baseball pitching machine from Zooka (see Figure 3.3), which can realize 15m/s (35mph) muzzle velocity for the described SQUID prototype.

3. **Unpowered (or ballistic) flight:** After launch, SQUID travels at high initial speeds and follows a parabolic trajectory. In the case of a moving vehicle launch, SQUID's relative velocity is the composition of the launch speed and the moving vehicle speed.

4. **Arms deployment:** The folded arms are initially retained by the monofilament line, as described above. The arms deploy when a relay actuates the nichrome burn wire. Without the restraining forces of the monofilament line, the torsion spring deploys the arms. While the arm deployment angle is not controlled, the arms fully deploy in 70ms, but they recoil by up to 30° before the motion is damped.

5. **Stabilization:** The pilot sends the command to *'unkill'* the drone and it automatically orients itself to the hovering attitude. For convenience, in the tested SQUID prototype, the pilot must compensate for altitude and lateral motion, but the vehicle includes a GPS for waypoint navigation. Autonomous stabilization using vision-based methods (which requires a larger volume to house a computer vision camera, 1D lidar and a bigger onboard computer) is described below in the context of the second generation SQUID prototype. In [24], the authors implement an algorithm to recover drone stability in midair using onboard sensors. SQUID requires a similar approach, but the vehicle speeds are much greater.

6. **Controlled Flight:** After SQUID stabilizes, it operates as a normal multirotor aircraft. The current design was not optimized for long battery life, but future prototypes might be able to carry different batteries depending on the mission length. While this SQUID prototype does not have dedicated landing legs, it can safely land if the bottom touches the ground first at a low speed. It naturally falls to one side without damaging any component. Another landing method is to grab the bottom part of SQUID. The second SQUID prototype, described below, does have integrated landing legs.

Table 3.1: SQUID 3" System Properties

Property	Value
Mass	530 g
Inertia about yaw axis, folded	$0.4\,10^{-3}$kg m^2
" " " " , unfolded	$2.3\,10^{-3}$kg m^2
Inertia about pitch axis, folded	$2.0\,10^{-3}$kg m^2
" " " " , unfolded	$1.6\,10^{-3}$kg m^2
Length	270mm
Folded Diameter	83mm (≈3in)
Maximum amperage	38 A
Thrust at hovering	28%
Launch speed	15m/s

Vehicle Design

This section will focus on the new challenges for SQUID design, as compared to the design of a standard multirotor: first, the limited volume reduces the number of possible choices for most of the components. Second, the arms are not rigidly attached to the body. This will induce vibrations that affect the structure and control. Lastly, the strong vertical acceleration during launch imparts a large axial load on the multirotor. The main consequence of this high acceleration is the need to reinforce the structure, as well as ensure all components are properly secured and electrical connectors are tightly locked. Table 3.1 provides a summary of the main design figures and Table 3.2 contains a list of key SQUID components.

Figure 3.3: The pneumatic baseball pitching machine used to launch the SQUID 3" prototype.

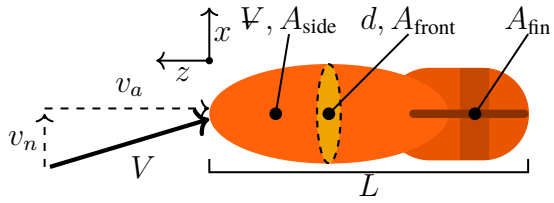


Figure 3.4: Aerodynamic Nomenclature

Vehicle Sizing and Aerodynamic Design: Due to the launcher diameter constraint, we design the outer shell in a compromise of internal volume, air drag, and stability (see Figure 3.5 for the selected shape). No detailed numerical simulations were performed, but we followed the insights from classic projectile design [34, 37] with aerodynamic forces and moments estimated as:

$$M_{\text{munk}} = \rho v_a v_n \mathcal{V}(1 - d/L) \tag{3.4}$$

$$F_{\text{base},n} = \rho v_a v_n A_{\text{front}} C_{d,\text{front}} \tag{3.5}$$

$$F_{\text{lift},n} = \frac{1}{2}\rho v_a v_n A_{\text{fin}} C_{l\alpha,\text{fin}} \tag{3.6}$$

$$F_{\text{side},n} = \frac{1}{2}\rho v_n v_n A_{\text{side}} C_{d,\text{side}} \tag{3.7}$$

where $F_{\text{base},n}$, $F_{\text{lift},n}$, and $F_{\text{side},n}$ are the components of the base drag, fin lift, and side drag taken normal to the primary axis of the body, and M_{munk} is the Munk moment, the moment of a streamlined body in an inviscid flow. Symbols ρ, v_a, v_n, L, d, and $\mathcal{V}$ are the air density, axial and normal velocities, length, diameter, and volume, respectively. Equations 3.4-3.7 are applicable for the designed SQUID model (a mildly streamlined body operating beyond turbulent transition) [34], but are not expected to apply to substantially smaller, slower, or smoother aircraft whose operation may be more sensitive to the Reynolds number. The aerodynamic center, which should be placed after the center of mass for passive stability, is given by:

$$z_{\text{AC}} = \frac{-M_{\text{munk}} + F_{\text{base},n}L + F_{\text{fin},n}L + F_{\text{side},n}L/2}{F_{\text{base},n} + F_{\text{lift},n} + F_{\text{side},n}} \tag{3.8}$$

The Munk moment is unstable and grows with the object's volume, while both the drag and fin lift are generally stabilizing. Accordingly, both standard fins and a ring-fin are required to lower the aerodynamic center (and increase fin structural integrity to handle launching forces) in order to compensate for the low-drag high-volume design. The estimated aerodynamic center location of the final design resides at

roughly 65% of the folded SQUID length (as measured from the front of the body), leading to stable damped pitch oscillations of a 0.6s period and stability margin of 5cm.

The "arms" which support the thrusting motors are folding when SQUID is placed in its launch tube. As described below, the arms are released, using a programmable fuse, partway through the ballistic launch phase. The arm deployment has three effects related to aerodynamic stability: the center of mass moves 3cm towards the nose after complete arm deployment (increasing stability), both the axial and normal drag are increased (increasing damping, but also shifting the aerodynamic center 3cm towards the nose due to the arm location), and it increases the yaw inertia by a factor of 5 (thereby decreasing yaw rate due to conservation of angular momentum). The effects act in concert to maintain stability during the transition from the ballistic launch phase to the geometry of the controlled flight phase. Deliberate spin-stabilization during launch was rejected for ease of piloting and to simplify the transition dynamics between launch and flight. The design was experimentally validated as is shown in Section 3.3.

Propeller and motor selection: The selection of the motors, propellers, and their associated power electronics will naturally affect the controlled flight performance. However, the mass and inertia have secondary impacts, and the propellers must fit within the body structure when the vehicle is in its folded state. The propeller size can be derived for ideal disc loading at hover [48]:

$$\frac{mg}{4\pi r_{\text{prop}}^2} = \frac{1}{2}\rho v_{\text{tip}}^2 (\sigma_{\text{prop}} C_{d0,\text{prop}}/k_{\text{prop}})^{2/3} \tag{3.9}$$

where $\sigma_{\text{prop}} \approx 0.1$, $C_{d0,\text{prop}} \approx 0.02$, $k_{\text{prop}} \approx 1.25$ are rough estimates of the propeller solidity, nominal drag coefficient, and induced loss factors. Assuming a referred maximum tip speed of $v_{\text{tip}} = 100$ m/s at hover (Mach 0.3), the ideal propeller size for hover with payload is around 6 or 7 inches. However, given the strong volume constraints for a passively stable aeroshell that folds within the launch tube, we can only choose the biggest propeller accommodated in the full system design, in this case 5 inches in diameter. This still gives us a large margin of excess thrust for operations using racing motors designed for smaller propellers. Knowing the propeller size, we select the motor Air40 from TMotors as it can drive this propeller and it has a good compromise of responsiveness and efficiency. Note that, despite the fact that maximum expected flight time is not a requirement for this vehicle (and therefore the design is not optimized for it), the battery was selected as the biggest

battery that can be accommodated in the given nose cone space, in this case a Tattu 850mAh.

Component Placement: The heaviest component, the battery, is placed as close to the nose as possible to increase the center of mass vertical location. This will increase aerodynamic stability during the ballistic launch [34]. The rest of the electronic components are placed directly below the battery: autopilot, BEC, and radio receiver. In addition, the ESC are placed on each arm to avoid the limited space in the airframe core and the radio antennas are extended to the bottom core piece for improved radio reception. Similarly, the GPS module is situated on top of the battery for better reception.

Structure Design: The main structural load experienced by the SQUID airframe occurs during the vertical acceleration from launch. From experiments with early SQUID prototypes, we measured a vertical acceleration of 50G's (490 m/s^2) to meet the launch height requirement with a sub-meter acceleration distance. This acceleration will appear as a volumetric force to all components. In particular, we designed the main structure to connect the inertial load from the battery, situated at the top and the heaviest component, to the launcher at the bottom. The 3D printed parts were printed using high impact resistance materials, using a Markforge printer with Onyx and carbon fiber. Another important load arises during the process of arm unfolding. Limited space in the overall system design prevents us from adding additional material to make the arms more rigid, and the curved surface (needed for aerodynamic streamlining in the folded state) limits the use of traditional CNC fabrication methods. Another benefit of 3D printed carbon fiber is the added rigidity, which is needed in our design in order to provide a tight fit when the arms are folded.

Hinge Design: The hinges between the folding arms and the central airframe body allow the arms to rotate freely after release and limit their movement so that the propellers are horizontal during normal flight. The unfolding limit is set by a mechanical stop. The hinges each hold a torsion spring that push the arms to open after their release. During normal flight, the springs are strong enough to maintain the arms in a fully open position, and provide resistance against vertical disturbances. An overly stiff spring creates large shock loads during arm unfolding. During launch, the arms fold to slightly beyond 90° from their open posture so that the propellers are tilted inside the body to allow more space at the top for the electronics.

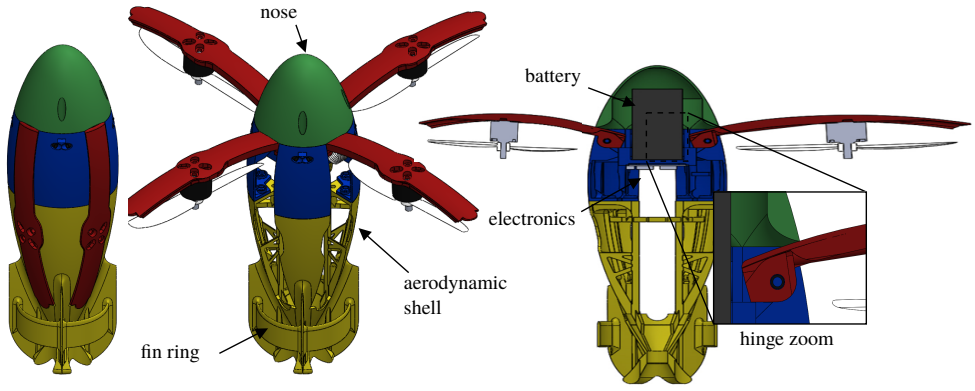


Figure 3.5: *SQUID 3"* CAD model. From left to right: ballistic configuration view, multirotor configuration view and section view with a hinge closer look.

Arm Release Mechanism: A critical function in SQUID deployment is the deployment or "release" of the arms during the ballistic flight phase. While several potential release mechanisms were considered, including designs employing electromagnets and servo motors, we selected a nichrome burn-wire trigger due to its reliability, efficient use of space, low susceptibility to G-forces, and low mass. The small downside of this design is the need to manually replace the burn-wire between launches.

Electrical current passing through the nichrome wire causes it to heat up and cut through a restraining loop of nylon monofilament line. This technique has been previously used on CubeSats, proving effective in both Earth atmosphere and vacuum [81]. The greatest downside of a nichrome release mechanism is the inconvenience of manually replacing the monofilament line after every launch, so the mechanism was designed for ease of access. A shallow groove runs around the circumference of the SQUID in its ballistic configuration to hold a loop of monofilament line in place (see Figure 3.6 for details). The tension in the arms causes them to push outwards against the line, but the chosen line is strong enough to withstand both the spring and launch forces without snapping. Mounted on one of the arms is a length of nichrome wire, held under tension by screw terminals that have been heat-set into the arm. The nichrome wire presses against the line, so that when heated, it severs the line and releases the spring-loaded arms. As described above, once the arms are released, the spring-loaded arm hinges quickly rotate under the spring forces to their final configuration.

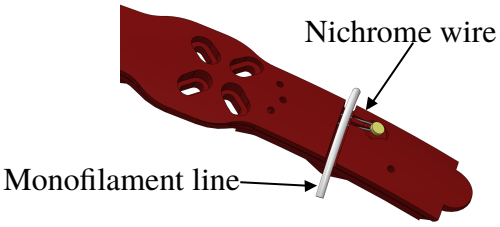


Figure 3.6: Release Mechanism Detail.

Table 3.2: Key SQUID 3" components

Component	Name	Weight (g)	Quantity
Autopilot	Pixracer running PX4	14	1
Motor	T-Motors Air40	24	4
ESC	T-Motors F30A	7	4
Propeller	DAL 5050	4	4
Receiver	FrSky R-RXR	1.2	1
Battery	Tattu 850mAh	104	1
Power board	ACSP7	15	1
Frame	Custom	181	1
Arms	Custom	16	4

Field Testing

We designed a set of tests to verify SQUID's capabilities. There were three main phases during the development of the field testing results.

1. **Aerodynamic test:** We used a mass model in order to evaluate aerodynamic effects in the vehicle prior to integrating electrical components, slowly increasing the fin size within volume constraints until enough stability margin was achieved for the test conditions. The selected shape includes a ring-fin for added stability and structural integrity.

2. **Delayed deployment test:** This test demonstrates deployment from a static launcher, see Figure 3.8 for a picture during midair flight. It contains all the phases described in Section 3.3.

3. **Moving vehicle test:** For this test, we launched SQUID from a car moving at 22m/s (50mph); see Figure 3.1 and 3.7 for keyframes from the video and Figures 3.10 and 3.9 for key data during flight. It demonstrates that SQUID can

be deployed at high speeds from a moving vehicle, and succesfully transition through all operational phases, concluding with a stable controlled flight.

Figure 3.7: Snapshots taken from a video of the process of launching *Squid 3* from a moving vehicle. The time from launch is noted on the upper left of each video frame. From left to right: (a) 20ms after deployment the arms are still closed. It is moving straight up in the cannon direction. (b) The arms have been deployed around 70ms after launch. The vehicle is still moving up. (c) The body is passively aerodynamically stable as it predictably orients itself relative to the apparant wind velocity. By 200ms, it is oriented upwind. (d) In this snapshot the vehicle is already stable and hovering.

3.4 SQUID 6": A Vision-Based Stabilization Prototype

This section refines and advances the capabilities of the SQUID concept that were described above and in [63]. In particular, this section presents the design, development and testing of a full-scale *SQUID* prototype. Capable of carrying a significant sensor payload, *SQUID* 6" transitions from a folded, 6 inch-diameter (152.4 mm) launch configuration to an autonomous, fully-controllable hexacopter after launch (Fig. 3.11). The entire process from launch to stabilization requires no user input and demonstrates the viability of using ballistically-launched multirotors for useful and varied missions.

Figure 3.8: Picture of the field testing setup deployed on a Caltech sports field, with a net to protect the *SQUID 3"* prototype from crashes

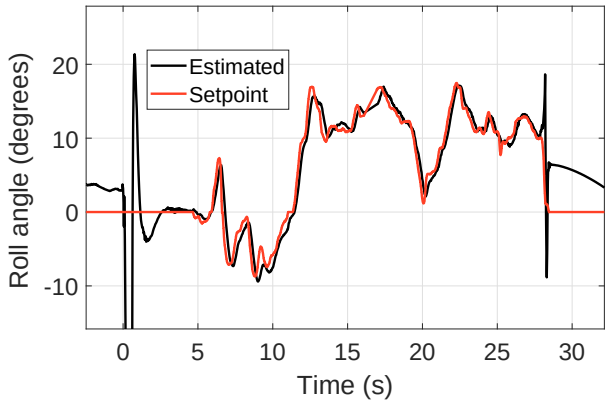


Figure 3.9: *SQUID 3"* roll angle profile during the moving vehicle test. The vehicle takes around one second to stabilize to the commanded roll angle.

Mechanical Design

The mechanical design of the second *SQUID* prototype (hereafter termed *SQUID 6"*, while *SQUID 3"* will refer to the earlier 3-inches *SQUID* prototype) is dictated by three broad functional requirements. The multirotor must: (i) launch from a tube (6-inch diameter for this prototype), (ii) travel ballistically to a predetermined height, and (iii) *autonomously* transition into stable, multirotor flight. To satisfy these non-traditional flight requirements, *SQUID* blends design elements from both ballistic and multirotor platforms. The multirotor's central rigid body houses a battery and the perception and control systems, and interfaces with six fold-out arms with rotors

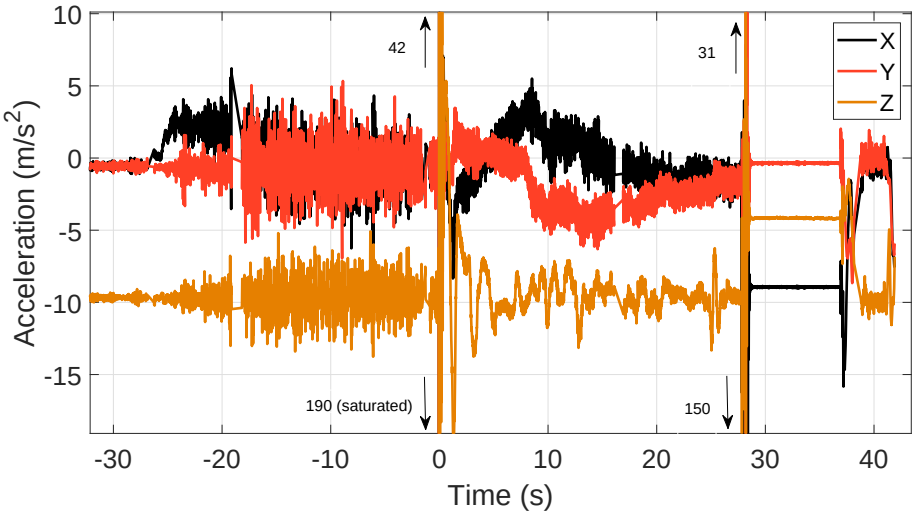


Figure 3.10: The three Cartesian accelerations measured during *SQUID 3"* moving vehicle test, where the x axis is pointing forward, the z axis is pointing up, and time starts when SQUID is launched. At -25s before launch, the vehicle accelerates to 80km/h (50mph) which can be seen as a constant acceleration on the x axes. After that, the acceleration is very noisy due to the bumpy road. The launch is indicated by a large acceleration spike in the z axis. There is another spike 29s later when the vehicle lands and tilts sideways onto its arms. During the flight, the z acceleration is close to negative one-g (9.8 m/s^2) indicating level flight, and the x and y acceleration commanded by the pilot compensate for the initial 50mph vehicle speed.

and three fold-out fins which passively stabilize the multirotor during ballistic motion (and act as landing gear). The layout of key *SQUID* components is shown in Fig. 3.12 and the configuration in folded and deployed states are shown in Fig. 3.13. Table 3.3 and Table 3.4 provide a list of key *SQUID* components and main design attributes. Some of the features that were demonstrated in the SQUID 3" prototype, such as a controllable arm release mechanism and a streamlined ballistic launch shape, were deemphasized in this prototype, as they were less important to demonstrating the key goal of autonomous stabilization in the ballistic-to-controlled-flight transition.

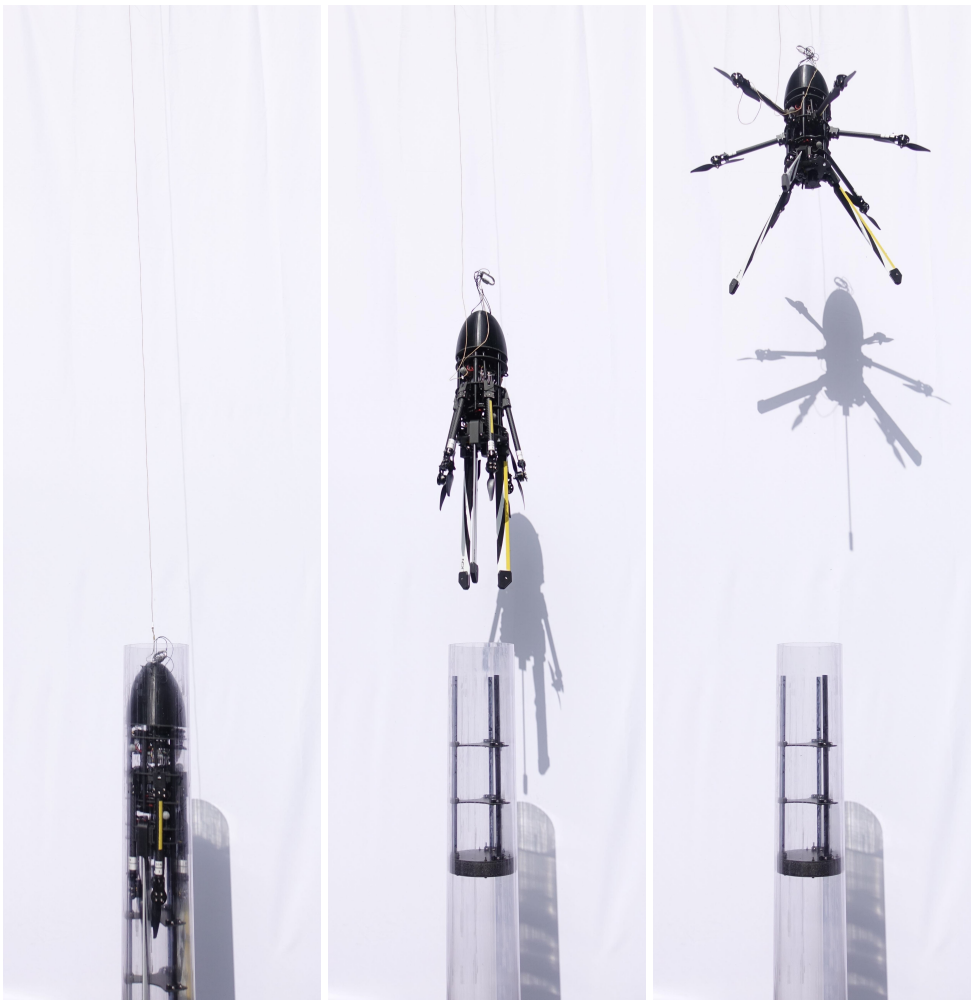


Figure 3.11: Launching *SQUID 6″*: inside the launcher tube (left), deploying the arms and fins (center), and fully-deployed configuration (right). Note the slack in the development safety tether and how the carriage assembly remains in the tube throughout launch. Each picture is 82 ms apart.

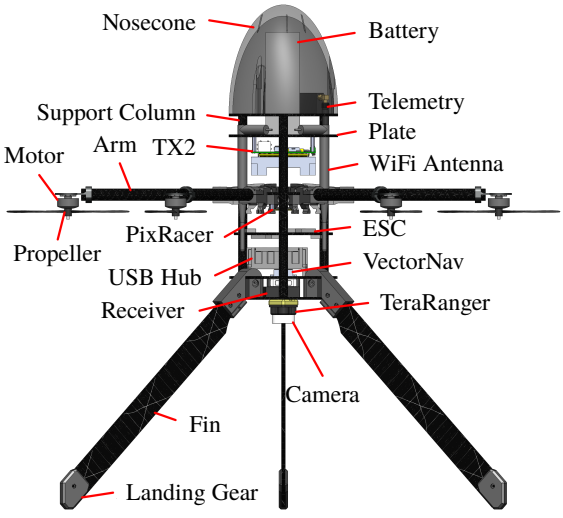


Figure 3.12: An annotated view of *SQUID 6″*.

Table 3.3: *SQUID 6″* System Properties

Property	Value	Units
Mass	3.3	kg
Length	79	cm
Folded Diameter	15	cm
Unfolded Diameter (propeller tip-to-tip)	58	cm
Thrust at Hover	56	%
Launch Speed	12	m/s

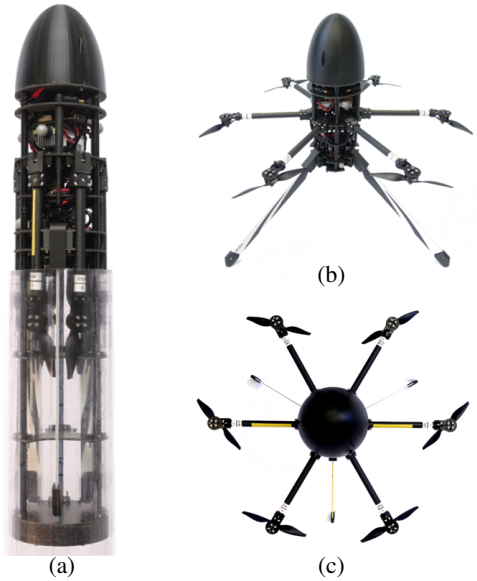


Figure 3.13: *SQUID 6″* partially inside the launcher tube and interfacing with the carriage (a), and with its arms and fins fully deployed from a side (b) and top perspective (c).

Central Rigid Body

In contrast to conventional multirotors, *SQUID*'s central body must sustain high transient forces during ballistic launch. Unlike prior *SQUID 3″*, which was manually stabilized by a pilot, *SQUID 6″* also requires a perception system comprising a camera (FLIR Chameleon3), rangefinder (TeraRanger Evo 60m), IMU/barometer (VectorNav VN-100), and onboard computer (NVIDIA Jetson TX2) to achieve full autonomous stabilization. Due to these added components, the original 3D-printed aeroshell structure was abandoned in favor of a hollow carbon fiber frame in order to maximize payload volume, increase strength, and allow easy access to the perception and control system components.

The frame consists of six thick carbon fiber plates separated by support columns

(made of aluminum standoff pins surrounded by carbon fiber tubes) that transmit and support the launch loads generated during the launch phase. A 3D printed nosecone reduces drag by approximately 50% compared to a bluff body nose. The placement of the heavy LiPo battery in the nosecone shifts the center of mass (COM) upward. This placement ensures that *SQUID*'s aerodynamic center (AC) trails behind the COM, which improves the passive ballistic stabilization. Passive stabilization is further addressed in Section 3.4.

Rotor Arms

The six rotors are mounted on carbon fiber tubes which attach to the central body with passive, spring-loaded hinges to allow 90° of rotation. The arms can exist in two states: constrained by the launch tube to be parallel to the body axis (closed), or extending radially outward perpendicular to the central axis (open). For *SQUID 3"*, the timing of the transition was controlled by an arm release mechanism. For *SQUID 6"* however, the transition from the closed state to the open, or unfolded, state occurs immediately after the multirotor leaves the launch tube, reducing mechanical complexity.

A torsional spring inside the hinge generates 1.04 N·m of torque when the arm is closed, and half that amount when the arm is open. Vibration in the motor arms during flight dictates the addition of a spring-loaded latch to keep the arms rigidly open after deployment.

Fins

SQUID 6"'s fins provide significant aerodynamic stabilization during ballistic flight to ensure that the vehicle maintains the launch direction before active stabilization is engaged. Aerodynamic forces on the fins shift the multirotor's AC downward behind the COM, enabling *SQUID 6"* to passively weathercock and align with the direction of flight. That is, when *SQUID 6"* is launched vertically from a stationary tube, the fins maintain a vertical trajectory during the ballistic phase. When the vehicle is launched from a moving vehicle, *SQUID 6"* will orient into the apparent wind direction, which has a component in the direction of the moving base's direction. Folding fins, rather than fixed fins, are a major design change from *SQUID 3"* [63] and were driven by a compromise between competing requirements of aerodynamic stability, low drag, constrained tube volume, and design simplicity. This design change was guided by the use of literature-derived expressions [33, 34] and scale model testing.

Fixed fins have a number of disadvantages. Any fin requires clean, unseparated flow to operate as designed. Therefore, fins that remain fixed within the tube area must also be paired with a streamlined tailbox in order to have access to said flow. This tailbox streamlining however reduces the wake drag and hence also reduces the stabilizing force it provides. Additionally, small fins which fit within the tube can only be partially effective as they have a limited wingspan. Expanding the fins along the tube only further lowers their aspect ratio (and therefore lift coefficient), reducing their capacity to move the AC. Deploying fins radially is therefore a much more effective means of enhancing stability, improving *SQUID*'s ability to predictably rotate upwind.

SQUID 6"'s tubular cross section (vs. the *SQUID 3"* teardrop shape) and foldout fins increase stability relative to *SQUID 3"* and simplify launch packaging issues with a simple cylindrical geometry, but do so at the cost of more ballistic drag. For most *SQUID* applications however, ballistic efficiency can often be sacrificed for these gains. Foldout fins can be tailored to provide a desired stability margin between the COM and AC, and provides margin for swappable payloads that may shift the COM. Given our selected 30 cm fins, the AC is located 38 cm from the nose, with a margin of 14 cm from the COM. Uncertainties in aerodynamic coefficients, drag on the arms, and the dynamics of the unfolding components can lead to substantial deviations from this calculated margin however. Accordingly, we validated our aerodynamic stability with a 3:1 scale model (50 mm diameter, 150 grams) using an open air wind tunnel (see Section 3.2) prior to full-scale tests.

While the hinges connecting the fins to the body are similar to the arm hinges, the fins do not use a latching mechanism because vertical vibrations have little impact on their functionality. "Feet" attached to the ends of the fins protect the tips and enable them to double as landing gear when the fins are in their extended configuration.

Ballistic Launch Process and the Autonomous Transition to Stabilized Flight

SQUID 6"'s mechanical design and onboard active controls manage the deployment sequence (Fig. 3.14). The deployment pipeline comprises two primary phases: passive stabilization and active stabilization. In the first phase, the multirotor's aerodynamic design ensures attitude stability as it travels along a ballistic trajectory after launch. Active stabilization begins once the arms are fully deployed and occurs before the trajectory's apogee. The following sections provide details on the launch stabilization process and our experimental validation of these concepts.

Table 3.4: Key *SQUID* components

Component	Description	Mass (g)
Flight Electronics		
Motors	T-Motor F80 Pro, 1900kv	36 (x6)
ESCs	T-Motor F30A 2-4S	6 (x6)
Propellers	7" diameter x 4" pitch	8 (x6)
Flight Controller	mRo PixRacer (PX4 Flight Stack)	11
Receiver	X8R 8-Channel	17
Telemetry	HolyBro 100 mW, 915 MHz	28
Battery	4S LiPo, 6000 mAh, 50C	580
Perception System		
Onboard Computer	NVIDIA TX2	144
Carrier Board	Orbitty Carrier Board	41
Rangefinder	TeraRanger Evo 60mm	9
IMU/Barometer	VectorNav VN-100	4
Camera	FLIR Chameleon3 w/ 3.5 mm Lens	128

Ballistic Launch Process

SQUID is ballistically launched to a minimum height that depends on both the safety requirements of the assets near the launch site and the altitude required for the targeted investigation. All the energy needed to loft the multirotor to the desired height, as well as to overcome the drag of the passive stabilization process, must be generated over the launching tube's very short length. Consequently, the airframe experiences very large acceleration forces while being launched.

The core of the launch mechanism is a re-purposed T-shirt cannon [83]. Pressure is supplied by a liquid CO_2 canister that is regulated between 5.5 bar (indoor, to stay within ceiling clearance) and 6.9 bar (outdoor, maximum safe pressure of the pressure regulation system) chamber pressure in gas phase. An aluminum stand holds the launch tube in place and allows adjustment of the launch angle. Accordingly, both the launch height and angle can be adjusted to avoid local hazards.

Prior to launch, *SQUID 6"* rests in a folded state inside the launch tube, which is generally pointed upwards. A 300 gram carriage assembly sits between *SQUID 6"* and the tube base, transmitting launch loads generated by the compressed gas directly to the frame's support columns. A 25 mm-thick polyethylene foam disk at the base of the carriage creates a low-friction seal which maximizes the transfer of energy from the compressed gas into kinetic energy and also prevents the carriage from leaving the tube during launch.

This launching mechanism meets the physical requirements to launch *SQUID 6″*, but has a number of inefficiencies. After launch is triggered, the compressed gas accelerates *SQUID 6″* through the tube at approximately 21 g's (estimated from video as the IMU saturates at 16 g's), but short of the unchoked valve throughput prediction of ≈350 g's. The maximum height achieved with this system is also 32 m (or 1 kJ potential energy), less than a third of the imparted energy as calculated from the ideal adiabatic expansion of the CO_2 chamber. Discrepancies between the predicted and estimated values are thought to arise from friction within the tube, losses in the pressure valve throughput, and air drag during the launch and ballistic phases.

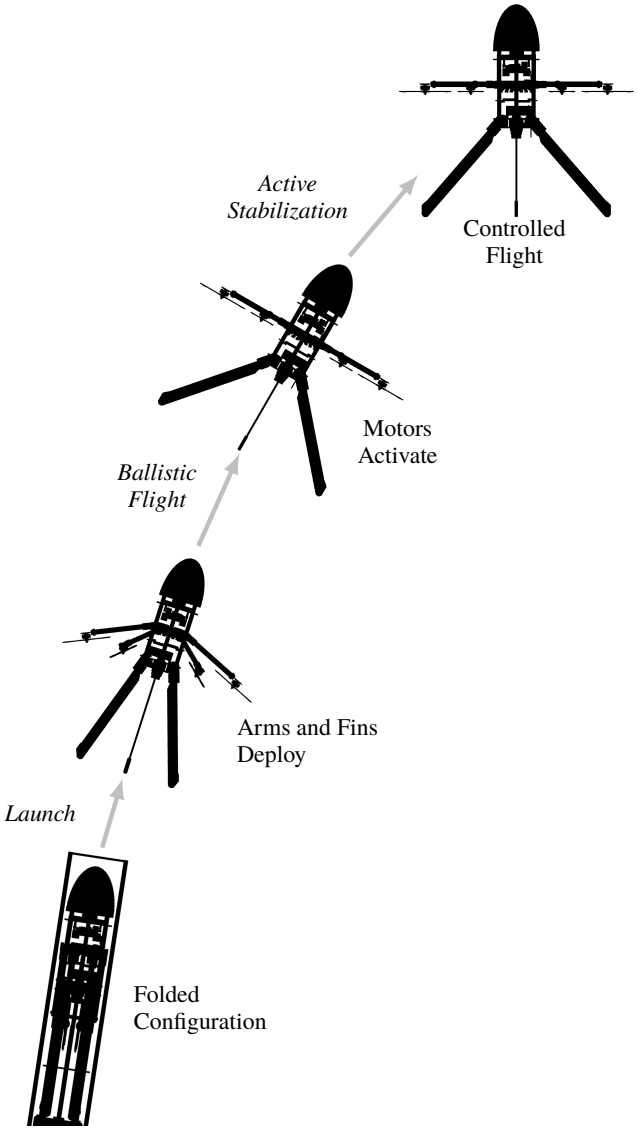


Figure 3.14: *SQUID 6″* deployment sequence.

Passive Stabilization - Launch without Wind

After exiting the launch tube, SQUID's arms and fins deploy immediately due to the spring-loaded hinges. This deployment has four effects on the aerodynamic stability: the COM is shifted towards the nose, the AC is shifted rearward due to the fin lift, the fins increase aerodynamic damping in yaw, and mass moves outwards which increases yaw inertia, thereby reducing yaw rate.

As described in Section 3.4, the lower AC helps *SQUID* maintain orientation and follow the intended flight path until active stabilization begins. The large displacement between the COM and AC, coupled with the launch momentum, causes *SQUID* to orient robustly into the apparent wind. When the launch tube is stationary and roughly vertical, this effect helps *SQUID* to passively maintain orientation during the ballistic phase, which simplifies the transition to active stabilization.

Passive Stabilization - Launch in Crosswind

During launch from a moving vehicle, *SQUID* experiences a strong crosswind, and will weathercock its nose in the direction of the launch platform's motion. Accordingly, *SQUID*'s passive stabilization design ensures that the multirotor travels smoothly during the ballistic phase and that its orientation at the beginning of the active stabilization phase is quite predictable from the knowledge of SQUID's launch speed and the vehicle speed at launch.

To validate *SQUID*'s expected passive aerodynamic behavior before field testing, sub-scale wind tunnel tests were performed at the Center for Autonomous Systems and Technologies (CAST) at Caltech. These tests were intended to prove that the new folding fin architecture could provide a sufficient stabilizing effect in the presence of a crosswind.

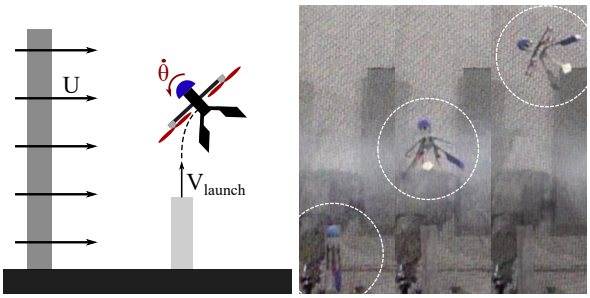


Figure 3.15: *SQUID 2″* Wind Tunnel Testing. Left – Definition of experiment parameters. Right – Snapshot sequence showing stable upwind pitching of the *SQUID 2″* model.

The sub-scale wind tunnel tests were performed using a 1/3 scale model of *SQUID*. Scaling for ballistically-launched drones near apogee, as seen in Section 3.2 of this chapter, primarily depends upon the Froude number ($U/\sqrt{gL}$), launch- to wind-velocity ratio, geometric parameters, and launch angle. Since *SQUID*'s tailbox can be modeled as a bluff-body disc, separation at the base is virtually guaranteed, meaning Reynolds effects can be neglected [34]. To correct the sub-scale results to be representative of the full-scale model, the trajectories and velocities were scaled by a factor of 3 and $\sqrt{3}$, respectively [63].

Accordingly, the performance of a vertical launch of 4.5 m/s in 10 m/s crosswinds (Fig. 3.15) can be extrapolated to the behavior of a full-sized drone launched at 7.8 m/s in a 17 m/s crosswind. The aerodynamically stable behavior, as indicated by the upwind turn, illustrates that the multirotor with deployed fins and motor arms produces a sufficient righting moment to predictably orient the multirotor upwind on launch. While not perfectly analogous (full-scale tests were performed at 12 m/s and a slightly different geometry), these sub-scale trajectories had a similar one-third scaled stability margin (5cm) and provided confidence that the full-sized *SQUID* would have a predictable trajectory if launched from a moving vehicle.

Transition from Passive to Active Stabilization

SQUID 6" commences the autonomy pipeline once the laser-based distance sensor indicates that the vehicle has cleared the launch tube. The passive-to-active transition occurs after the vehicle has exited the tube and the arms are fully deployed, allowing the motors to spin. Starting the motors early in the ballistic phase of launch is important, as the motors need to be fully spooled up and stabilizing the multirotor before apogee. At apogee, the airspeed may not be sufficient to provide enough aerodynamic stabilization, risking the multirotor entering a tumbling state from which it may not recover.

Active Stabilization

Our active stabilization solution is based upon previous research into autonomously recovering a monocular vision-based quadrotor after its state-estimator fails due to a loss of visual tracking [11, 24]. For our visual inertial odometry pipeline, we utilize the open-source Robust Visual Inertial Odometry (ROVIO), an extended Kalman Filter that tracks both 3D landmarks and image patch features [8]. Since it tightly integrates image intensity information with inertial data to produce odometry

estimates, ROVIO is capable of operating in stark, low-texture environments such as over pavement, water, and the surface of other planets.

The first stage of the active stabilization process controls the attitude to a nominal zero-roll/pitch orientation using the IMU-based attitude estimate. As the air pressure around the multirotor spikes on launch, the barometric altitude estimates become unreliable and the altitude must be maintained open-loop, biased upwards for safety. The barometric readings stabilize within three seconds of launch, and at this point, *SQUID* begins actively controlling its altitude and attempts to reduce the vertical velocity to zero. As no horizontal position or velocity information is available, active control of the lateral position is not possible and *SQUID* continues to drift in plane until the VIO can be initialized.

Several conditions need to be met before the VIO can be successfully initialized. Firstly, the pitch and roll rates need to be near-zero to ensure that the camera captures frames with low motion blur. Secondly, the vertical velocity needs to be near-zero so the distance between the multirotor and the ground remains constant and the initial feature depth can be well established using measurements from the onboard laser-based distance sensor. Finally, the lateral velocity must be small (once again to minimize motion blur), so the multirotor is allowed to drift for 10 s post spool up to enable aerodynamic drag to bleed off excess speed. Future iterations of the autonomy pipeline will sense when to initialize VIO directly from the detected motion blur, enabling the vehicle to enter position stabilization sooner after launch.

The VIO is considered initialized when the cumulative variance of the VIO's x- and y-position estimates drop below a preset threshold. The pose estimates are then fed into the flight controller state estimator filter to be fused with the IMU. At this point, *SQUID 6″* has full onboard state estimation and can now control both altitude and lateral position.

Experimental Validation

To demonstrate the proposed passive-to-active stabilization pipeline, we launched *SQUID 6″* in a 42 foot-tall flying arena at CAST (Fig. 3.17). The arena has two tiers of Optitrack motion capture cameras allowing *SQUID 6″*'s position and orientation to be tracked throughout the duration of a flight for offline analysis. During initial development, a tether system was constructed inside the arena to prevent the multirotor from damaging the facility in the event of a launch failure. A small weight was used to passively eliminate any slack in the tether. As SQUID

accelerates significantly faster than the 1 g of the counterweight (note the slack in the tether in Fig. 3.11), it is unlikely that the tether interfered with the critical passive-to-active attitude stabilization phase.

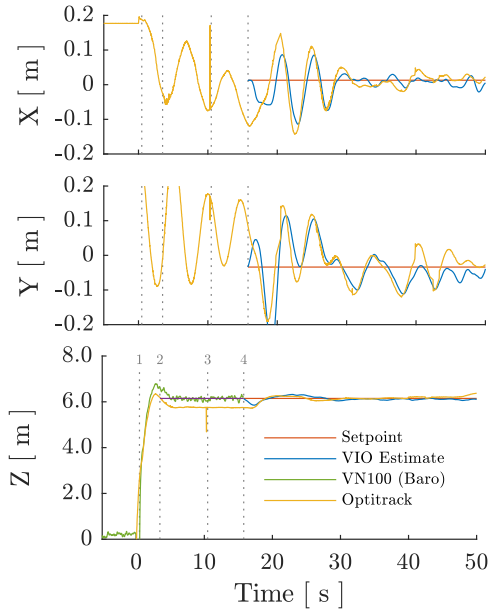


Figure 3.16: Onboard state estimates and ground truth during launch. 1: Motors on, 2: Closed-Loop altitude control, 3: VIO initialization, 4: Position control.

Fig. 3.16 plots the position of a point on the vehicle during the period from launch to active position stabilization. At launch (t=0), altitude is quickly gained as the multirotor accelerates from the forces of the expanding propellant gases. The motors turn on at Point 1 and begin actively stabilizing the attitude. By Point 2, the barometer has recovered from the launch and closed-loop altitude control commences. Ten seconds after the motors are turned on (Point 3), VIO initialization begins. At Point 4, the VIO is initialized and starts to feed pose estimates to the flight controller, which then actively controls the position of the multirotor, completing the pipeline. The pipeline was successfully demonstrated across several days, lighting conditions, and launch pressures. Footage of the launches can be found at `https://youtu.be/mkotvIK8Dmo`.

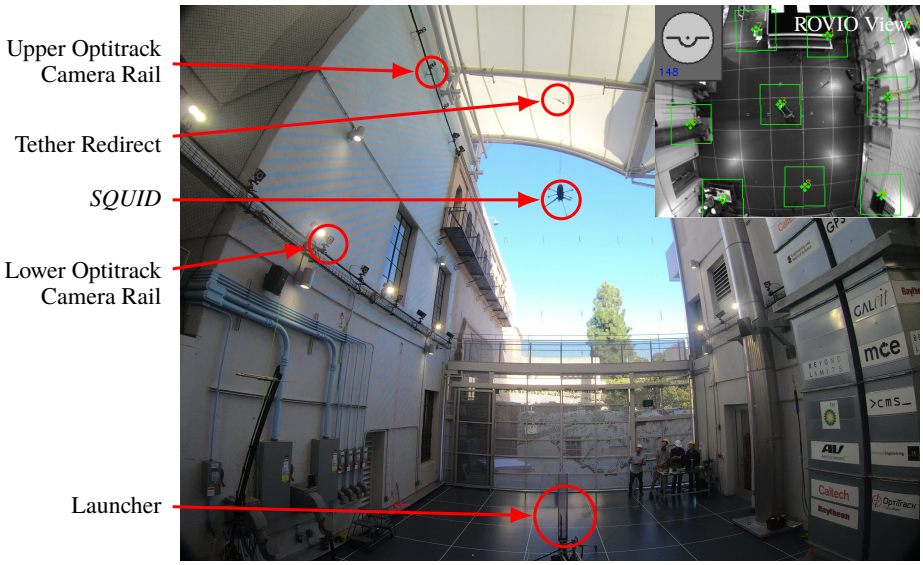


Figure 3.17: Launching *SQUID* inside CAST.

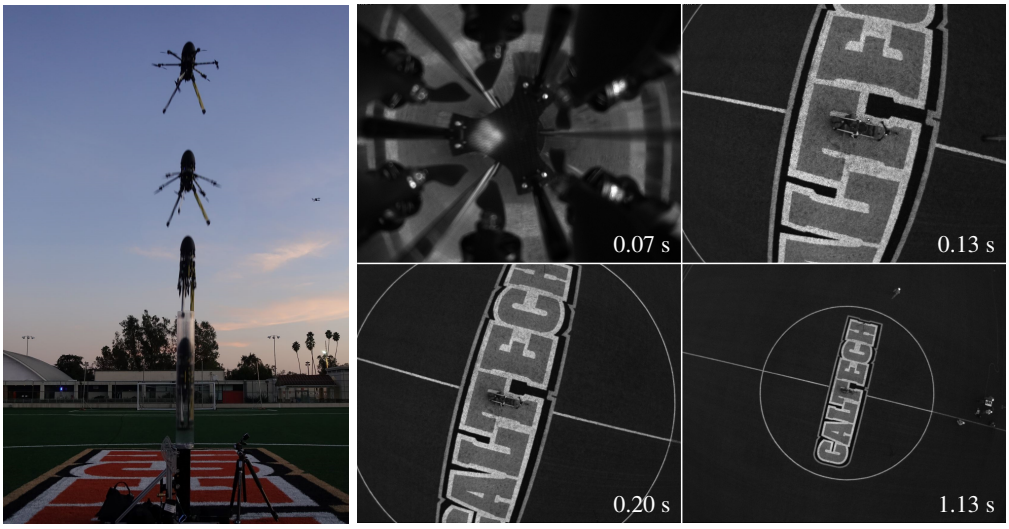


Figure 3.18: Preliminary outdoor free-flight *SQUID* testing.

3.5 Conclusion

Together, *SQUID 3″* and *SQUID 6″* successfully demonstrated the ability to ballistically launch and transition into autonomous stable flight, using only onboard sensors and control. In particular, the work presented in this chapter demonstrated:

1. A 450g quadcopter with aerodynamic body and passive aerodynamic stability
2. A smooth launch and stabilization from a moving vehicle (at vehicle speeds of 50 mph).
3. A 3.3 kg hexacopter with a payload of an advanced sensor package and mission computer, plate-based airframe strong enough to carry and transmit launch loads without damaging onboard components, and aerodynamic stability generated by folding fins
4. Wind tunnel testing that validates the proposed multirotor design in crosswind launches.
5. An autonomy pipeline that carries the platform from launch detection to full 6-degree of freedom stabilization using only onboard sensing (IMU, barometer, rangefinder, and camera) and without the need for GPS.

This proof-of-concept system validates the viability of a ballistically-launched multirotor that deploys without human involvement, opening up new applications in fields such as disaster response and space exploration.

Chapter 4

LEARNING KOOPMAN EIGENFUNCTIONS

This chapter describes a new method to learn Koopman Eigenfunctions using a non-linear transformation of the linearized principal eigenfunctions. The results of this chapter have been presented at the 2020 American Control Conference [27] in collaboration with Carl Folkestad.

4.1 Introduction

A key step in developing a high performance robotic application is the modeling of the robot's mechanics. Standard cyberphysical system modeling and identification methods require extensive knowledge of the system and laborious system identification procedures [55]. Moreover, although methods to show stability and safety of nonlinear systems exist [3, 40], the design of control systems that incorporate state and control limitations remains a challenging endeavor.

Learning can capture the salient aspects of a robot's complex mechanics and environmental interactions. Gaussian process dynamical systems models [85] can identify nonlinear affine control models in a non-parametric way. Alternatively, spectrally normalized neural networks [76] can fit dynamics models with stability guarantees. Yet, the effective design of nonlinear controllers that incorporate state and actuator constraints, after identifying the model, can be challenging. Deep neural networks for control Lyapunov function augmentation [79] can be used for control design with different types of constraints, but learns a task-specific augmentation that cannot be used for other objectives. Similarly, model-free reinforcement learning (MFRL) [20] learns feedback policies that implicitly incorporate the robot's dynamics. However, sample efficiency is very low. Moreover, while safety during model free reinforcement learning is now possible [18, 29], one cannot yet guarantee that learned policies will satisfy performance requirements or state and actuator limits.

The work presented in this chapter contributes to Koopman inspired modeling and identification techniques, which have received substantial recent attention [16, 72]. 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 [74, 86]. The methods are easy to implement, mainly relying on least squares regression, and computationally and mathematically flexible, enabling numerous extensions and applications [13].

For example, DMD-based methods have been successfully used in the field of fluid mechanics to capture low-dimensional structure in complex flows [78], in robotics for external perturbation force detection [6], and in neuroscience to identify dynamically relevant features in ECOG data [12]. More recently, Koopman-style modeling has been extended to *controlled* nonlinear systems [38, 67]. 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) [42] can be utilized to design nonlinear controllers.

Typically, EDMD-methods 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, we 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 [39], but rely on defining an appropriate candidate function library. Other previous methods (e.g., [41]) depend upon assumptions that are problematic for robotic systems: the ID data is gathered while the robot operates under open loop controls, which can lead to catastrophic system damage for naturally unstable systems.

4.2 Preliminaries on Koopman Operator Theory

Consider the autonomous dynamical system:

$$\dot{\mathbf{x}} = f(\mathbf{x}) = A\mathbf{x} + v(\mathbf{x}) \tag{4.1}$$

with state $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$ and $f(\cdot)$ Lipschitz continuous on $\mathcal{X}$. We assume that system (4.1) has a fixed point at the origin: $f(0) = 0$. For a system with a single attractor in $\mathcal{X}$, this assumption can always be achieved without loss of generality by a change of

coordinates. The flow of this dynamical system is denoted by $S_t(\mathbf{x})$ and is defined as

$$\frac{d}{dt}S_t(\mathbf{x}) = f(S_t(\mathbf{x})) \tag{4.2}$$

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

$$U_t\gamma = \gamma \circ S_t \tag{4.3}$$

for all $\gamma \in \mathcal{C}(\mathcal{X})$, where $\circ$ denotes function composition. Each element of the Koopman operator maps continuous functions to continuous functions, $U_t : \mathcal{C}(\mathcal{X}) \to \mathcal{C}(\mathcal{X})$. Crucially, each U_t is a *linear* operator. An *eigenfunction* of the Koopman operator associated to an eigenvalue $e^\lambda \in \mathbb{C}$ is any function $\phi \in \mathcal{C}(\mathcal{X})$ that defines a coordinate evolving linearly along the flow of (4.1) satisfying

$$(U_t\phi)(\mathbf{x}) = \phi(S_t(\mathbf{x})) = e^{\lambda t}\phi(\mathbf{x}). \tag{4.4}$$

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 [58]. To see this, consider asymptotically stable dynamics of the form (4.1). The linearization of the dynamics around the origin is

$$\dot{\mathbf{y}} = \mathbf{D}f(0)\mathbf{y} = \hat{A}\mathbf{y},\ \mathbf{y} \in \mathcal{Y}. \tag{4.5}$$

The following proposition describes how to construct eigenfunction-eigenvalue pairs for the linearized system (4.5).

Proposition 1. Let $\hat{A}_1$ denote the linearization (4.5) of the nonlinear system (4.1) 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 $\langle \mathbf{v}_j, \mathbf{w}_k \rangle = \delta_{jk}$ and $\mathbf{w}_j$ is an eigenvector of $\hat{A}_1^*$ at eigenvalue $\bar{\lambda}_j$. Then, the linear functional

$$\psi_j(\mathbf{y}) = \langle \mathbf{y}, \mathbf{w}_j \rangle \tag{4.6}$$

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

$$\left(\prod_{j=1}^{d} e^{m_j\lambda_j}, \prod_{j=1}^{d} \psi_j^{m_j} \right) \tag{4.7}$$

is an eigenpair of the Koopman operator $U_{\hat{A}_1}$.

Proof. A less formal description of the results in the proposition and associated proofs are described in [58], Example 4.6. By utilizing inner-product properties, ψ_j is an eigenfunction of $U_{\hat{A}}$ as described in (4.4) since

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

By scaling the state-space such that $\mathcal{Y}_1 \subset \mathcal{Q}_1$, the linear eigenfunctions (4.6) form a vector space on $\mathcal{Y}_1$ that is closed under point-wise products. The construction of arbitrarily many eigenpairs (4.7) therefore follows from the semi-group property of eigenfunctions (see [16], Prop. 5). □

In the following, linear functionals (4.6) are denoted as *principal eigenfunctions*. The eigenfunctions for the Koopman operator associated with the linearized dynamics can be used to construct eigenfunctions associated with the Koopman operator of the nonlinear dynamics through the use of a *conjugacy map*, as described in the following proposition.

Proposition 2. Assume that the nonlinear system (4.1) is topologically conjugate to the linearized system (4.5) 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 $U_{\hat{A}_1}$ at e^{λ}, then $\psi \circ g \circ h$ is an eigenfunction for U_f at eigenvalue e^{λ}, where U_f is the Koopman operator associated with the nonlinear dynamics (4.1).

Proof. See [16], Proposition 7. □

The following extension of the Hartman-Grobman theorem guarantees the existence of the diffeomorphism, h described in Proposition 2, between the linearized and nonlinear systems in the entire basin of attraction of a fixed point, for sufficiently smooth dynamics.

Theorem 3. (Theorem 2.3 in [45]) Consider the system (4.1) with $v(\mathbf{x}) \in \mathcal{C}^2(\mathcal{X})$. 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 $\exists h(\mathbf{x}) \in \mathcal{C}^1(\Omega) : \Omega \to \mathbb{R}^d$, such that

$$\mathbf{y} = c(\mathbf{x}) = \mathbf{x} + h(\mathbf{x}) \tag{4.8}$$

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

Koopman Theory for Controlled Systems

There are several ways to extend the Koopman operator to actuated systems such that systems with external forcing can be analyzed through the spectral properties of its associated Koopman operator [42, 67]. These observations underpin the adaption of EDMD methods to controlled systems to construct finite-dimensional approximations to the Koopman operator. In particular, given a dictionary of D dictionary functions $\phi(x)$ and N data snapshots of the states, X, control inputs, U, and state derivatives, Y, from a n-dimensional system with m control inputs, a linear regression problem can be formulated as

$$\min_{A\in\mathbb{R}^{(D\times D)},B\in(D\times m)} ||A\phi(X) + BU - Y||. \tag{4.9}$$

The solution to this regression problem results in a linear model of the dynamics of the form $\dot{\mathbf{z}} = A\mathbf{z} + B\mathbf{u}$ where the outputs of interest are predicted by $y = C\mathbf{z}$ where C can be approximated by another regression problem aiming to minimize $||CZ - Y||$ [42].

4.3 Motivating Analytic Example

Certain systems have a structure that leads to a closed Koopman subspace if a correct set of observables is chosen. This section demonstrates how the theory presented in Section 4.2 can be used to construct eigenfunctions when the system dynamics are known and we can analytically construct the diffeomorphism described in Theorem 3. We consider the system

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} \mu x_1 \\ \lambda(x_2 - x_1^2) \end{bmatrix} \tag{4.10}$$

which has a finite dimensional Koopman operator. These equations are used in Section 4.3 to construct three eigenfunctions that can completely describe the evolution of the system by utilizing the Koopman modes associated with each eigenfunction [16]. Then, we demonstrate how to arrive at the same eigenfunctions through the use of the diffeomorphism in Section 4.3. This underpins the data-driven approach described in Section 4.4, 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 $y = [x_1, x_2, x_1^2]^T$, Equation (4.10) can be rewritten as an equivalent linear system

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{bmatrix} = \underbrace{\begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix}}_{K} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{4.11}$$

where K is the Koopman operator of the system. From this result we can construct three Koopman eigenfunctions of (4.10). Let $\{\mathbf{v}_i\}_{i=1}^3$ be the eigenvectors of 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

$$\begin{aligned} \psi_1(\mathbf{y}) &= \langle \mathbf{y}, \mathbf{w}_1 \rangle = y_1 = x_1 \\ \psi_2(\mathbf{y}) &= \langle \mathbf{y}, \mathbf{w}_2 \rangle = y_3 = x_1^2 \\ \psi_3(\mathbf{y}) &= \langle \mathbf{y}, \mathbf{w}_3 \rangle = y_2 + \frac{\lambda}{\lambda - 2\mu} y_3 = x_2 + \frac{\lambda}{\lambda - 2\mu} x_1^2 \end{aligned} \tag{4.12}$$

Calculating Eigenfunctions Based on the Diffeomorphism

We now show how the calculated eigenfunctions can be obtained through the diffeomorphism between the linearized and nonlinear dynamics. The linearization of the dynamics (4.10) around the origin is

$$\begin{bmatrix} \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} \mu & 0 \\ 0 & \lambda \end{bmatrix}}_{A} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} \tag{4.13}$$

and we can construct principal eigenfunctions for the linearized system, $\hat{\psi}_1(\mathbf{x}) = \langle \hat{\mathbf{w}}_1, \mathbf{x} \rangle = x_1$, $\hat{\psi}_2(\mathbf{x}) = \langle \hat{\mathbf{w}}_2, \mathbf{x} \rangle = x_2$, where $\hat{\mathbf{w}}_1, \hat{\mathbf{w}}_2$ are the eigenvectors of the adjoint of A. As described in Proposition 1, we can construct arbitrarily many eigenfunctions for the linearized system 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

$$c(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\lambda}{\lambda - 2\mu} x_1^2 \end{bmatrix} \tag{4.14}$$

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

the nonlinear dynamics

$$\begin{aligned}\phi_1(\mathbf{x}) &= \hat{\psi}_1(c(\mathbf{x})) = x_1 \\ \phi_2(\mathbf{x}) &= \hat{\psi}_2(c(\mathbf{x})) = x_2 + \frac{\lambda}{\lambda - 2\mu} x_1^2 \\ \phi_i(\mathbf{x}) &= \hat{\psi}_i(c(\mathbf{x})) = x_1^{m_i^{(1)}} \left(x_2 + \frac{\lambda}{\lambda - 2\mu} x_1^2 \right)^{m_i^{(2)}}, \; i = 3, \dots\end{aligned} \tag{4.15}$$

and it can be seen that with $m_3 = (2, 0)$, the analytic eigenfunctions of Equation (4.12) are recovered.

4.4 Data-driven Koopman Eigenfunctions for Unknown Nonlinear Dynamics

This section presents a data-driven approach to learn the diffeomorphism $h(x)$ described in Proposition 2 and Equation 4.8, resulting in a methodology for constructing Koopman eigenfunctions from data.

Modeling Assumptions

We consider the dynamical system

$$\dot{\mathbf{x}} = a(\mathbf{x}) + B\mathbf{u} \tag{4.16}$$

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

$$\dot{\mathbf{x}} = A_{nom}\mathbf{x} + B_{nom}\mathbf{u} \tag{4.17}$$

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 (4.16) to the origin in a region of attraction Ω around the origin. The nominal model (4.17) 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.

Constructing Eigenfunctions from Data

Algorithm 1 constructs Koopman eigenfunctions from data, based on the foundations introduced in Section 4.2. 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

$$\mathcal{D} = \left(\left(\mathbf{x}_k^j, \mathbf{u}_k^j \right)_{k=0}^{M_s} \right)_{j=1}^{M_t} \tag{4.18}$$

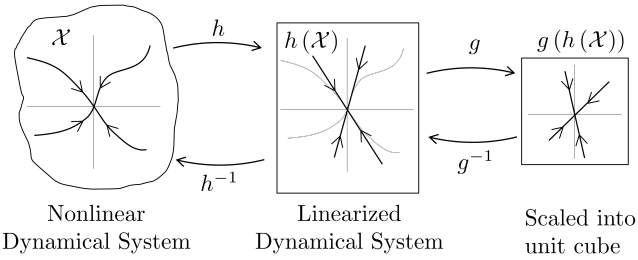


Figure 4.1: Chain of topological conjugacies used to construct eigenfunctions, adapted from [58].

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 (4.17) can be constructed as in Proposition 1 using the eigenvectors and eigenvalues of the closed loop dynamics matrix $A_{cl} = A_{nom} + B_{nom}K_{nom}$. I.e. let $\mathcal{Q}_r$ be a hypercube of radius r such that $\mathcal{X} \subset \mathcal{Q}_r$. A scaling function $g : \mathcal{Q}_r \to \mathcal{Q}_1$ 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 $A_{cl,1}$ 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 $U_{A_{cl,1}}$ with eigenvalue e_j^λ and we can construct an arbitrary number of eigenpairs using the product rule (4.7).

The eigenfunction construction for the linearized system only relies on the nominal model. To construct Koopman eigenfunctions for the true nonlinear dynamical system, we aim to learn the diffeomorphism (4.8) between the linearized model (4.17) and the true dynamics (4.16), see Figure 4.1. This diffeomorphism is guaranteed to exist in the entire basin of attraction Ω by Theorem 3. Let $\mathcal{H}_h$ be a class of continuous nonlinear function mapping $\mathbb{R}^d$ to $\mathbb{R}^d$ such that $h(\mathbf{x}) \in \mathcal{H}_h$. The diffeomorphism is found by solving the following optimization problem:

$$\begin{aligned} \min_{h \in \mathcal{H}_h} \sum_{k=1}^{M_t} \sum_{j=1}^{M_s} (\dot{\mathbf{x}}_k^j + \dot{h}(\mathbf{x}_k^j) - A_{cl}(\mathbf{x}_k^j + h(\mathbf{x}_k^j)))^2 \\ \text{s.t.} \qquad \mathbf{D}h(\mathbf{0}) = \mathbf{0} \end{aligned} \tag{4.19}$$

which is a direct transformation of Theorem 3 into the setting with unknown nonlinear dynamics. The form of problem (4.19) 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 adding the constraint $\mathbf{D}c(\mathbf{0}) = I$ results in the formulated optimization problem (4.19).

We next formulate (4.19) 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), and aggregated to a data matrix. 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 that $h(\mathbf{x}) \in \mathcal{H}_h$, we define the loss function

$$\begin{aligned} &\mathcal{L}_h(\mathbf{x}, \dot{\mathbf{x}}, A_{cl}\mathbf{x} - \dot{\mathbf{x}}) = \\ &\quad ||\dot{h}(\mathbf{x}) - A_{cl}h(\mathbf{x}) - (A_{cl}\mathbf{x} - \dot{\mathbf{x}})||^2 + \alpha||\mathbf{D}h(\mathbf{0})||^2 \\ &\quad = ||\mathbf{D}h(\mathbf{x})\dot{\mathbf{x}} - A_{cl}h(\mathbf{x}) - (A_{cl}\mathbf{x} - \dot{\mathbf{x}})||^2 + \alpha||\mathbf{D}h(\mathbf{0})||^2 \end{aligned} \tag{4.20}$$

where parameter α penalizes the violation of constraint (4.19). The supervised learning goal is to select a function in $\mathcal{H}_h$ through empirical risk minimization (ERM):

$$\min_{h \in \mathcal{H}_h} \frac{1}{M_s \cdot M_t} \sum_{k=1}^{M_s \cdot M_t} \mathcal{L}_h(\mathbf{x}_k, \dot{\mathbf{x}}_k, A_{cl}\mathbf{x}_k - \dot{\mathbf{x}}_k) \, . \tag{4.21}$$

Finally, with function h identified from ERM (4.21), Proposition 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:

$$\phi_j(\mathbf{x}) = \tilde{\psi}_j(g(h(\mathbf{x}))) \tag{4.22}$$

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 (4.7).

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 user-generated perturbation of the control action deployed on the system and is akin to persistence of excitation in adaptive control [53]. To understand these effects, state dependent model error bounds are needed, but they are left open for future work.

4.5 Koopman Eigenfunction Extended Dynamic Mode Decomposition

To use the constructed Koopman eigenfunctions for prediction and control, we develop an EDMD-based method to build a linear model in a lifted space. Since this method exploits the structure of the Koopman eigenfunctions, it is dubbed *Koopman Eigenfunction Extended Dynamic Mode Decomposition* (KEEDMD). We construct N eigenfunctions $\{\phi_j\}_{j=1}^N$ with associated eigenvalues $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_N)$ as outlined in Section 4.4, and define the lifted state as

$$\mathbf{z} = [\mathbf{x}, \boldsymbol{\phi}(\mathbf{x})]^T \tag{4.23}$$

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

$$\dot{\mathbf{z}} = A\mathbf{z} + B\mathbf{u} \tag{4.24}$$

where matrices $A \in \mathbb{R}^{(N+d)\times(N+d)}, B \in \mathbb{R}^{(N+d)\times m}$ are unknown, and are to be inferred from the collected data.

We 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

Algorithm 1 Data-driven Koopman Eigenpair Construction

Require: Data set $\mathcal{D} = \left((\mathbf{x}_k^j, \mathbf{u}_k^j)_{k=0}^{M_s}\right)_{j=1}^{M_t}$, nominal model matrices A_{nom}, B_{nom}, nominal control gains K_{nom}, number of lifting functions N, N power combinations $(m_1^{(i)}, \ldots, m_d^{(i)}) \in \mathbb{N}_0^d, i = 1, \ldots, N$

1: Construct principal eigenpairs for the linearized dynamics:
$(\lambda_j, \psi_j(\mathbf{y})) \leftarrow (\lambda_j, \langle \mathbf{y}, \mathbf{w}_j \rangle), \qquad j = 1, \ldots, n$
2: Construct N eigenpairs from the principal eigenpairs:
$(\tilde{\lambda}_i, \tilde{\psi}_i) \leftarrow \left(\sum_{j=1}^d \lambda_j^{m_j^{(i)}}, \prod_{j=1}^d \psi_j^{m_j^{(i)}}\right), \quad i = 1, \ldots, N$
3: Fit diffeomorphism estimator: $h(\mathbf{y}) \leftarrow \text{ERM}(\mathcal{H}_h, \mathcal{L}_h, \mathcal{D})$
4: Construct scaling function: $g(\mathbf{y}) \leftarrow g : \mathcal{Q}_r \to \mathcal{Q}_1$
5: Construct N eigenpairs for the nonlinear dynamics:
$(\tilde{\lambda}_i, \phi_i) \leftarrow (\tilde{\lambda}_i, \tilde{\psi}_i(g(c(\mathbf{x})))), \qquad i = 1, \ldots, N$

Output: $\Lambda = \text{diag}(\tilde{\lambda}_1, \ldots, \tilde{\lambda}_N), \qquad \boldsymbol{\phi} = [\phi_1, \ldots, \phi_N]^T$

Position dynamics:

$$\min_{B_{\mathbf{p}} \in \mathbb{R}^{(d/2)\times m}} \quad ||\mathbf{y_p} - X_{\mathbf{p}} B_{\mathbf{p}}^T||_2^2, \qquad X_{\mathbf{p}} = [U], \qquad \mathbf{y_p} = [\dot{P} - IV] \tag{4.26a}$$

Velocity dynamics:

$$\min_{\substack{A_{\mathbf{v}} \in \mathbb{R}^{(d/2)\times(n+N)} \\ B_{\mathbf{v}} \in \mathbb{R}^{(d/2)\times m}}} \quad ||\mathbf{y_v} - X_{\mathbf{v}}[A_{\mathbf{v}}\ B_{\mathbf{v}}]^T||_2^2, \quad X_{\mathbf{v}} = [P\ V\ \Phi\ U], \quad \mathbf{y_v} = [\dot{V}] \tag{4.26b}$$

Eigenfunction dynamics:

$$\min_{B_{\boldsymbol{\phi}} \in \mathbb{R}^{N\times m}} \quad ||\mathbf{y}_{\boldsymbol{\phi}} - X_{\boldsymbol{\phi}} B_{\boldsymbol{\phi}}^T||_2^2, \qquad X_{\boldsymbol{\phi}} = [U - U_{nom}], \quad \mathbf{y}_{\boldsymbol{\phi}} = [\dot{\Phi} - \Lambda\Phi] \tag{4.26c}$$

known. As a result, the lifted state space model has the following structure:

$$\begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\mathbf{v}} \\ \dot{\boldsymbol{\phi}}\left(\begin{bmatrix}\mathbf{p}\\ \mathbf{v}\end{bmatrix}\right) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I & 0 \\ A_{\mathbf{vp}} & A_{\mathbf{vv}} & A_{\mathbf{v}\boldsymbol{\phi}} \\ -B_{\boldsymbol{\phi}} K_{nom} & \Lambda \end{bmatrix}}_{A} \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \boldsymbol{\phi}\left(\begin{bmatrix}\mathbf{p}\\ \mathbf{v}\end{bmatrix}\right) \end{bmatrix} + \underbrace{\begin{bmatrix} B_{\mathbf{p}} \\ B_{\mathbf{v}} \\ B_{\boldsymbol{\phi}} \end{bmatrix}}_{B} \mathbf{u} \tag{4.25}$$

where $0, I, \Lambda, K_{nom}$ are fixed matrices and $A_{\mathbf{vp}}, A_{\mathbf{vv}}, A_{\mathbf{v}\boldsymbol{\phi}}$. The matrices $B_{\mathbf{p}}, B_{\mathbf{v}}, B_{\boldsymbol{\phi}}$ 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 (4.25), we construct the data matrices and formulate the loss function for three separate ordinary least squares regression problems defined in Equation (4.26). The data matrices are aggregations of the data samples concatenating all observations, e.g. $P = [\mathbf{p}_1^1, \ldots, \mathbf{p}_{M_s}^1, \ldots, \mathbf{p}_1^{M_t}, \ldots, \mathbf{p}_{M_s}^{M_t}]^T$. Furthermore, the variables P, V, Φ, U, U_{nom} are derived from measurements p, v, ϕ, u, and u_{nom}. $\dot{P}, \dot{V}, \dot{\Phi}$ are found by numerically differentiating P, V, Φ, respectively. U and U_{nom} are related by $U = U_{nom} + U_{pert}$, where U_{nom} is the nominal linear feedback control action and U_{pert} is the added random perturbation to induce exploratory behavior as discussed in Section 4.4. The KEEDMD exploits the control perturbation to learn the effect of actuation on the Koopman eigenfunctions.

To reduce overfitting, regularization can be added to the objectives of the regression formulations. In particular, LASSO-regularization promoting sparsity in the learned matrices has been shown to perform well for dynamical systems [14] when used in normal EDMD. This has also been the case in our numerical simulation, where

LASSO-regularization seems to improve the prediction performance and the stability of the results.

When the lifted state space model is identified, state estimates can be obtained as $\mathbf{x} = C\mathbf{z}$, where $C = [I \quad 0]$. C 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. We now discuss 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 this 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 4.2. Then, the evolution of the principal eigenfunctions becomes

$$\begin{aligned}\dot{\psi}_j(\mathbf{y}) = \langle \dot{\mathbf{w}_j}, \mathbf{y} \rangle &= \mathbf{w}_j^T \dot{\mathbf{y}} \\ &= \langle \mathbf{w}_j, A_{cl}\mathbf{y} - B_{nom}K_{nom}\boldsymbol{\tau}(t) \rangle \\ &= \lambda_j \langle \mathbf{w}_j, \mathbf{y} \rangle - \langle \mathbf{w}_j, B_{nom}K_{nom}\boldsymbol{\tau}(t) \rangle \\ &= \lambda_j \psi(\mathbf{y}) - \mathbf{w}_j^T B_{nom}K_{nom}\boldsymbol{\tau}(t)\end{aligned} \tag{4.27}$$

where λ_j and $\mathbf{w}_j$ are the j-th eigenvalue and adjoint eigenvector of A_{cl}, respectively. Notably, the principal eigenfunctions evolve as described in Section 4.4, but with an additional forcing term, $-\mathbf{w}_j^T B_{nom}K_{nom}\boldsymbol{\tau}(t)$.

Utilizing the fact that the dynamical equations considered in this section have linear (constant) control vector fields (see Eq. 4.16), we show that the evolution of the eigenfunctions of the Koopman operator associated with the full dynamics is affine in the input signal.

Proposition 4. Assume that B_{nom} in the linearized model of the dynamics (4.17) is equal to the actuation matrix of the true dynamics (4.16) and that the dynamics are controlled by a linear trajectory-tracking feedback controller of the form $u = K_{nom}(\mathbf{x} - \boldsymbol{\tau}(t)$. Then, the time derivatives of the eigenfunctions of the Koopman operator associated with the dynamics (4.16) constructed as described in Proposition 1-2 are affine in the external forcing signal $\boldsymbol{\tau}(t)$.

Proof. We first show that the diffeomorphism between the linearized and nonlinear dynamics is linear in the forcing signal. Consider the diffeomorphism described in Theorem 3 with an additional forcing term. Derived from the linearized dynamics, we seek to find $h(\mathbf{x})$ such that

$$\dot{\mathbf{y}} = A_{cl}\mathbf{y} - B_{nom}K_{nom}\boldsymbol{\tau}(t), \qquad \mathbf{y} = \mathbf{x} + h(\mathbf{x}). \tag{4.28}$$

By algebraic manipulations, we get that

$$\begin{aligned} \dot{\mathbf{y}} = \dot{\mathbf{x}} + \dot{h}(\mathbf{x}) &= A_{cl}(\mathbf{x} + h(\mathbf{x}) - B_{nom}K_{nom}\boldsymbol{\tau}(t) \\ &\Rightarrow a(x) + BK_{nom}(\mathbf{x} - \boldsymbol{\tau}(t)) + \dot{h}(\mathbf{x}) \\ &= (A_{nom} + B_{nom}K_{nom})(\mathbf{x} + h(\mathbf{x})) - B_{nom}K_{nom}\boldsymbol{\tau}(t) \\ &\Rightarrow \dot{h}(\mathbf{x}) - A_{cl}h(\mathbf{x}) = A_{nom}\mathbf{x} - a(\mathbf{x}) \end{aligned} \tag{4.29}$$

Hence, $h(\mathbf{x})$ 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 (4.16) evolve affinely in the forcing signal. □

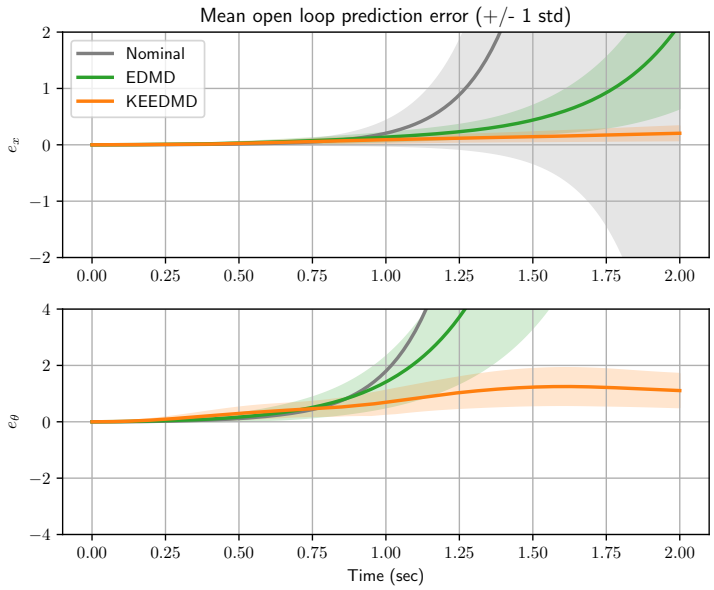
Because the eigenfunctions evolve linearly in the forcing signal, the KEEDMD-framework can readily learn the effect of external forcing on the eigenfunctions by minor modifications. First, the loss term of the diffeomorphism empirical risk minimization (4.19) must be modifed to account for the forcing term following the construction of Equation (4.28). The new loss function becomes

$$\begin{aligned} &\mathcal{L}_h(\mathbf{x}, \dot{\mathbf{x}}, A_{cl}\mathbf{x} - \dot{\mathbf{x}}, \boldsymbol{\tau}(t)) = \\ &\qquad ||\dot{h}(\mathbf{x}) - A_{cl}h(\mathbf{x}) - (A_{cl}\mathbf{x} - \dot{\mathbf{x}}) + B_{nom}K_{nom}\boldsymbol{\tau}||^2 \\ &\qquad + \alpha||\mathbf{D}h(\mathbf{0})||^2 \end{aligned} \tag{4.30}$$

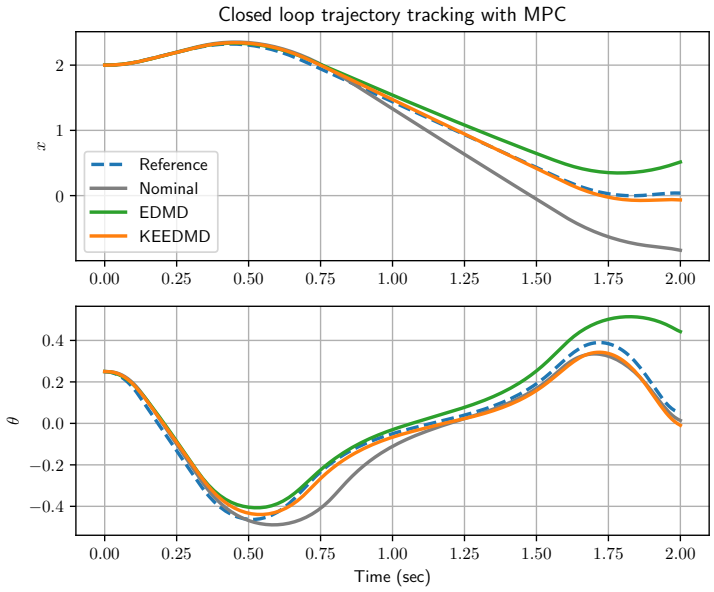
where $\boldsymbol{\tau}$ is the vector of desired states corresponding to the time that $\mathbf{x}, \dot{\mathbf{x}}$ were sampled. Second, the data matrix $X_{\boldsymbol{\phi}}$ in the regression formulation (4.26) must be modified so the effect of the forcing on the eigenfunction evolution can be learned. This is achieved by setting $X_{\boldsymbol{\phi}} = [U - K_{nom}[P \quad V]]$.

4.6 Model Predictive Control Design

Inspired by [42], the Koopman operator is used to transform the original nonlinear optimization problem into an efficient quadratic program (QP) that is solved at each



(a) Open loop prediction



(b) Closed loop trajectory tracking

Figure 4.2: Performance comparison of the nominal model, EDMD, and KEEDMD for (a) prediction and (b) closed loop.

time step. The QP formulation requires to discretize the previously learned linear continuous dynamics. The algorithm assumes a known objective function that is solely a function of states and controls. For simplicity, it uses a quadratic objective function, but other objective functions are possible by adding them to the lifting functions. It assumes known control bounds $u_{\min}, u_{\max} \in \mathbb{R}^m$ and state bounds

$x_{\min}, x_{\max} \in \mathbb{R}^n$. These assumptions define the following optimization problem:

$$\begin{aligned} \min_{\substack{u\in\mathbb{R}^{m\times N_p} \\ z\in\mathbb{R}^{N\times N_p}}} \quad & \textstyle\sum_{p=1}^{N_p} \left[(Cz_p - \tau_p)^T Q (Cz_p - \tau_p) + u_p^T R u_p \right] \\ \text{s.t.} \quad & z_p = A_d z_{p-1} + B_d u_p \\ & x_{\min} \leq Cz_p \leq x_{\max} \qquad p = 1, \ldots, N_p \\ & u_{\min} \leq u_p \leq u_{\max} \\ & z_0 = \phi(x_k) \end{aligned} \tag{4.31}$$

where $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ are positive semidefinite cost matrices, $\tau \in \mathbb{R}^{n\times N_p}$ 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 (4.24), $C \in \mathbb{R}^{n\times N}$ is the projection matrix, and $\phi \in \mathbb{R}^N$ are the eigenfunctions. If the projection matrix C is learned from data, a margin should be added to account for reprojection errors. This margin can be obtained from the residual statistics and a desired satisfaction probability δ.

To remove the dependency on the lifting dimension N in Eq. (4.31), the state is eliminated via an explicit relation with the control input. This formulation is referred as the *dense* form MPC. This step greatly reduces the number of optimization variables, which is beneficial as we must solve the MPC problem 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 ϕ and linear matrices A_d, B_d, and C do not directly appear in the formulation.

4.7 Simulation Results

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

$$\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{M+m}\left(ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2 + F\right) \\ \frac{1}{l}\left(g\sin\theta + \ddot{x}\cos\theta\right) \end{bmatrix} \tag{4.32}$$

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

[1]The code for learning and control Koopman Eigenfunctions is publicly available on `https://github.com/Cafolkes/keedmd`

trajectory designed based on the nominal model to move to the origin from an initial condition two meters away. We will collect data with a nominal controller, learn the lifted state space model, and use this model 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 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 will not work. The PD controller is perturbed with white noise of variance 0.5 to aid the model fitting as described in Section 4.4, and state and control action values 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 (4.26).

To benchmark our results, we compare our prediction and control results against (1) the nominal model, and (2) an 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* [64]. The EDMD and KEEDMD regressions are implemented with *Scikit-learn* [65].

First, we compare the open loop prediction performance 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 with 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 model, EDMD-model, and the KEEDMD-model. The mean error between the predicted evolution and the true system evolution over all the trajectories is depicted in Figure 4.2a. Both the nominal and EDMD models are able to predict the evolution for the first second, but then diverge. In contrast, KEEDMD is able to maintain good prediction performance over the entire duration of the trajectories, with relatively low and constant standard deviation.

To evaluate the closed loop performance, we compare the behavior of the three different models on the task of moving from the initial point $(x_0, \theta_0, \dot{x}_0, \dot{\theta}_0) =$

Table 4.1: Improvement in MPC cost with learned models

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

$(2, 0.25, 0, 0)$ in two seconds. The nominal model is used to generate a trajectory from the initial point to the origin. Then, a dense form MPC using the learned lifted state space model is implemented in Python using the QP solver *OSQP* [77]. The MPC costs on the trajectory tracking task are significantly improved when the lifted state space models are used, see Figure 4.2b. 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.

4.8 Conclusion

This chapter presented a novel method based on the Koopman Operator to learn the dynamics of controlled robotic systems. Koopman eigenfunctions are used to learn the system's nonlinear dynamics and to learn a near optimal control strategy (MPC) for given tasks. By using a Koopman approach, we are able to implement a real-time MPC framework for optimal system control during the learning process.

Chapter 5

EPISODIC KOOPMAN EIGENFUNCTIONS

This chapter extends the work in Chapter 4 to make it practically useful for robotics. The main results in this chapter were presented at the 2020 IEEE International Conference on Robotics and Automation (ICRA) in collaboration with Carl Folkestad. First, the method presented below gathers data while the system operates under any *nonlinear* stabilizing controller. This enables nonlinearities in the input vector fields to be captured during the learning process, unlike prior Koopman-based model ID approaches. Second, this chapter introduces an episodic learning procedure, by considering the closed-loop dynamics obtained with a non-linear 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. Third, it should be noted that data collected from robots while they execute trajectories may formally violate the i.i.d. assumption underlying the performance guarantees of most learning paradigms. In practice, this fact can lead to error cascades and poor performance guarantees. Episodic learning mitigates this problem [71]. Finally, our method integrates Model Predictive Control (MPC) [56] into its structure, thereby allowing control and state constraints to be satisfied during the learning process.

5.1 Problem Setup and Dynamics Modeling

Assume that we have selected a fixed trajectory $\boldsymbol{\tau}(t)$ 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 the KEEDMD process described in the last chapter. Finally, the system's governing dynamics are assumed to be *unknown*

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

where $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n, u \in \mathcal{U} \subset \mathbb{R}^m$, and $f(\mathbf{x}, \mathbf{u})$ is assumed to be Lipschitz continuous on $\mathcal{X} \times \mathcal{U}$.

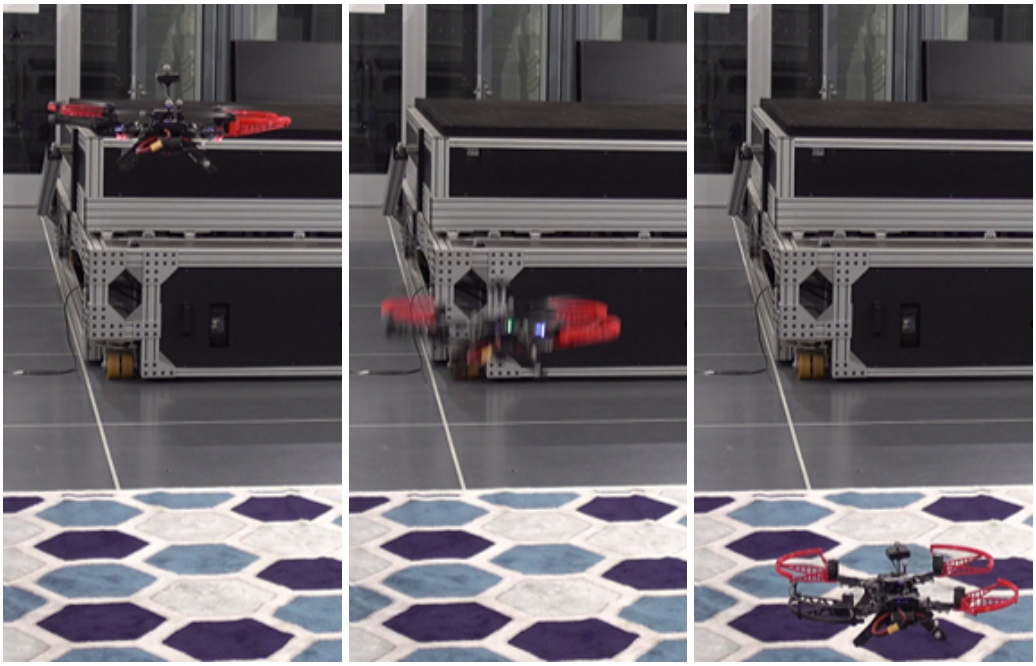


Figure 5.1: From left to right: hovering before the sequence start, high speed descent with learned dynamics, and soft landing.

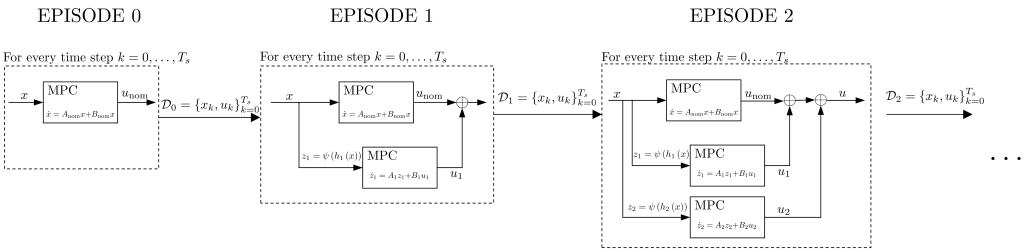


Figure 5.2: Flow chart showing the different elements for each episode.

Learning with Arbitrary Stabilizing Control Laws

KEEDMD (Section 4.4) requires batch training data to be collected from a system that operates under a nominal linear control law: $\mathbf{u}_{nom}(\mathbf{x}) = K_{nom}(\mathbf{x} - \boldsymbol{\tau}(t))$. A main contribution of this chapter is to iteratively learn an improving sequence of eigenfunctions and nonlinear controllers. Specifically, we will iteratively use the lifted state-space model to design an MPC-controller to track learning trajectories (see Figure 5.2).

If a candidate nonlinear controller $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$ can stabilize system (5.1) to a given trajectory $\boldsymbol{\tau}$, the controlled system can be described by the autonomous dynamics

$$\dot{\mathbf{x}} = f(\mathbf{x}, \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)) \triangleq F_{\hat{\mathbf{u}}, \boldsymbol{\tau}}(\mathbf{x}, t). \tag{5.2}$$

where $F_{\hat{\mathbf{u}}, \boldsymbol{\tau}}(\mathbf{x}, t)$ denotes an autonomous dynamical system, under control law $\hat{\mathbf{u}}$

stabilizing the system to $\boldsymbol{\tau}$. Importantly, for the autonomous dynamics (5.2), there exists an associated Koopman operator $U_{F_{\hat{\mathbf{u}},\boldsymbol{\tau}}}$ that depends on control law $\hat{\mathbf{u}}$ and trajectory $\boldsymbol{\tau}$. Therefore, approximate eigenpairs for $U_{F_{\hat{\mathbf{u}},\boldsymbol{\tau}}}$ can be constructed (see Section 4.2) from the gathered state and control samples. A lifted state-space model can be constructed from these eigenpairs.

However, unlike the framework reviewed in Section 4.4, we 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. 5.2), and the B-matrix captures the effect of control variations around the nominal controller:

$$\dot{\hat{\mathbf{z}}} = A\hat{\mathbf{z}} + B(\mathbf{u}(\mathbf{x}, \boldsymbol{\tau}, t) - \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)). \tag{5.3}$$

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.

Capturing Nonlinear Control and Dynamics Effects

Recall that the last chapter defined a diffeomorphism, $h(\cdot)$, that was crucial to mapping between the eigenfunctions of a linearized dynamical system and the Koopman eigenfunctions of the nonlinear system. To enable the proposed learning framework to capture nonlinear effects caused by the nonlinear controller and actuated dynamics, a minor modification to the function approximator that represents 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 (4.20). In the case considered in the previous chapter, however, the actuated dynamics and controller are assumed to be linear. This linearity causes the effect of the forcing signal $\boldsymbol{\tau}(t)$ to cancel out such that the diffeomorphism is independent of the trajectory τ. 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 $h : \mathcal{X} \times \mathcal{X} \to \mathcal{Y}$ (see 4 for details).

5.2 Episodic Eigenfunction Construction and KEEDMD Inference

This section describes the main contribution of this paper, a substantial extension of the KEEDMD framework to allow iterative learning and improvement of the lifted state-space model and its associated controller.

Overview of the Episodic Learning Algorithm

Algorithm 2 summarizes the 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 was designed at the end of the previous episode, $\mathbf{u}_{e-1}(\mathbf{x}, \boldsymbol{\tau}, t)$. The state $\mathbf{x}$, control actions $\mathbf{u}_{e-1}$, control difference $\tilde{\mathbf{u}}$, 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:

$$\mathcal{D}_x^{(e)} = \left\{ \left(\mathbf{x}_i^{(e)}, \mathbf{u}_i^{(e)}, \tilde{\mathbf{u}}_i^{(e)}, \boldsymbol{\tau}_i \right), \dot{\mathbf{x}}_i^{(e)} \right\}_{i=1}^{T_s} \tag{5.4}$$

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 $\mathcal{D}_x^{(e)}$, we estimate the diffeomorphism h and construct the eigenfunctions $\boldsymbol{\phi}^{(e)}(\mathbf{x})$ with associated eigenvalues $\Lambda^{(e)}$, via Algorithm1. Since changes in the control law between episodes are expected to be small, we warm start the learning algorithm with model coefficients from the previous episode.

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

$$\mathcal{D}_z^{(e)} = \left\{ \left(\mathbf{z}_i^{(e)}, \mathbf{u}_i^{(e)}, \tilde{\mathbf{u}}_i^{(e)}, \boldsymbol{\tau}_i \right), \dot{\mathbf{z}}_i^{(e)} \right\}_{i=1}^{T_s} \tag{5.5}$$

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

Algorithm 2 Episodic KEEDMD

Require: 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 $\mathcal{L}_z$

$\mathcal{D}_z = \emptyset, \quad \mathbf{u}_0(\mathbf{x}, \boldsymbol{\tau}, t) = \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$

for $e = 1, \ldots, N_{ep}$ **do**

Sample initial condition: $\mathbf{x_0} \leftarrow \text{sample}(X_0)$

Execute experiment: $\mathcal{D}_x^{(e)} \leftarrow \text{run}(\mathbf{x}_0, \mathbf{u}^{(e-1)}(\mathbf{x}, \boldsymbol{\tau}, t))$

Fit diffeomorphism estimator: $h(\mathbf{x}) \leftarrow \text{ERM}(\mathcal{H}_h, \mathcal{L}_h, \mathcal{D}_x^{(e)})$

Construct eigenpairs: $(\boldsymbol{\phi}^{(e)}(\mathbf{x}), \Lambda^{(e)}) \leftarrow h(g(\boldsymbol{\psi}(\mathbf{x})))$

Construct and aggregate lifted data set: $\mathcal{D}_z \leftarrow \mathcal{D}_z \cup \mathcal{D}_z^{(e)}$

Fit KEEDMD model: $\dot{\mathbf{z}}^{(e)}(\mathbf{z}) \leftarrow \text{ERM}((\boldsymbol{\phi}^{(e)}, \Lambda^{(e)}), \mathcal{L}_z, \mathcal{D}_z)$

Update controller: $\mathbf{u}^{(e)} \leftarrow \mathbf{u}^{(e-1)} + w^{(e)}\text{MPC}(\dot{\mathbf{z}}^{(e)}, \mathbf{u}^{(e-1)})$

end for

Output: Final control law $\mathbf{u}^{(N_{ep})}$

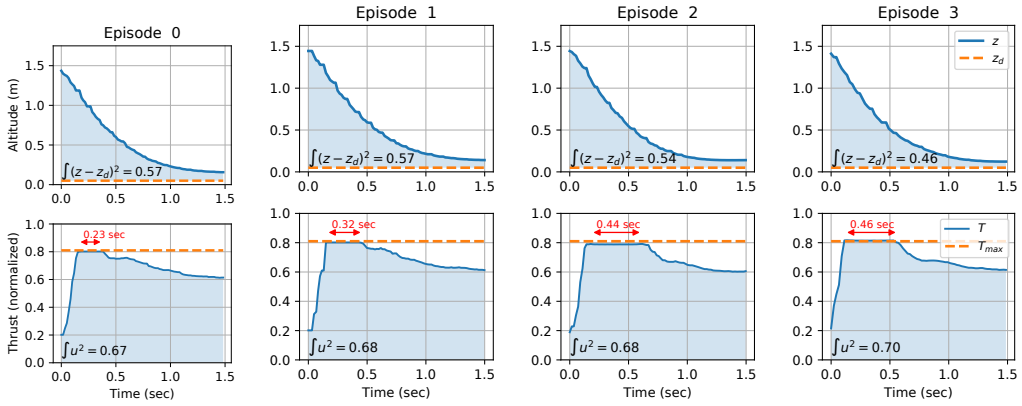


Figure 5.3: Evolution of drone altitude p_z with accumulated error and control effort after each episode. Episode 0: baseline controller, Episode 1-3: performance after each episode of learning. Red arrows: duration the thrust constraint is active.

constructed from this data using the framework of Section 5.1. This results in a model of the form (5.3).

In the third and final step, an augmenting MPC is designed (see Section 5.2) 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.

Efficient Model Predictive Controller Implementation

Inspired by [42], we transform the original non-linear optimization problem into an efficient quadratic program (QP). The QP formulation requires us to discretize the previously learned linear continuous dynamics. We assume a known objective function of states and controls only. For simplicity, we use a quadratic objective function with respect to the state error and control action, but other objective functions can be used by simply adding it to the lifting functions. We assume known control bounds $u_{\min}, u_{\max} \in \mathbb{R}^m$ and state bounds $x_{\min}, x_{\max} \in \mathbb{R}^n$. Because the control input for each MPC problem refers to the change from the previous controller, we have to correct for this change in the control bounds. All these assumptions define the following optimization problem that is solved at each time step:

$$
\begin{aligned}
\min_{\substack{u\in\mathbb{R}^{m\times N_p}\\ z\in\mathbb{R}^{N\times N_p}}} \quad & \sum_{p=1}^{N_p}\left[(C_j z_p-\tau_p)^T Q\,(C_j z_p-\tau_p)+u_p^T R u_p\right]\\
\text{s.t.} \quad & z_p = A_j z_{p-1}+B_j u_p\\
& x_{\min}\le C_j z_p\le x_{\max} \qquad p=1,\dots,N_p\\
& u_{\min}\le u_p-\textstyle\sum_{i=1}^{j-1} w^{(i)}u_p^{(i)}\le u_{\max}\\
& z_0=\phi_j(x_k)
\end{aligned}
\tag{5.6}
$$

where $Q\in\mathbb{R}^{n\times n}$ and $R\in\mathbb{R}^{m\times m}$ are positive semidefinite cost matrices, $\tau\in\mathbb{R}^{n\times N_p}$ is the reference trajectory, $A_j\in\mathbb{R}^{N\times N}$ and $B_j\in\mathbb{R}^{N\times m}$ are the discrete time versions of (5.3) for controller j, $C_j\in\mathbb{R}^{n\times N}$ is the j^{th} controller's projection matrix, and $\phi_j\in\mathbb{R}^N$ are the j^{th} controller's eigenfunctions. See Figure 5.2 to see how each controller is used as more episodes are being executed. In addition, we add a smoothing regularizer to avoid chatter that may arise from optimization-based controllers [60] of the form $\sum_{p=1}^{N_p}\alpha_R(u_p-u_{p-1})^2$ where u_0 is the deployed control action at the previous timestep.

5.3 Improving Fast Multirotor Descent and Landing by Learning the Ground Effect

To validate our methodology, we apply it to fast descent and landing of a multirotor[1]. As the vehicle approaches the landing plane, a ground effect from the interaction of the prop downwash and the landing surface becomes prominent. This effect induces added upward thrust on the drone, which can lead to poor tracking performance for control designs that rely on models which omit these fluid flow interactions.

Modeling and Problem Statement

To simplify the discussion, we 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, T, as input:

$$
\begin{bmatrix}\dot{p}_z\\ \ddot{p}_z\end{bmatrix}=\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}p_z\\ \dot{p}_z\end{bmatrix}+\begin{bmatrix}0\\ 1/m\end{bmatrix}T. \tag{5.7}
$$

Using this model, we design a nominal MPC as described in Section 5.2 with the goal of reaching a fixed point of $0.05\,\mathrm{m}$ above ground at zero velocity.

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

[1]The code for learning and control is publicly available on `github.com/Cafolkes/keedmd`

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.

Implementation and Experimental Details

Our experiments use the *Intel Aero RTF* Drone. The Drone's 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 diffeomorphism, h, is parameterized by a neural network and implemented with *PyTorch* [64], and the KEEDMD regression is implemented with elastic net regularization in *Scikit-learn* [65]. A dense form MPC-controller is implemented in Python using the QP solver *OSQP* [77], 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 5.1.

We execute Algorithm 2 as discussed in Section 5.2 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 to evaluate the performance of the current episode controller.

Results and Discussion

Figure 5.3 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

Table 5.1: 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, $x_{\min}$	$0.05\,\mathrm{m}$	Hover thrust, u_{hover}	0.66

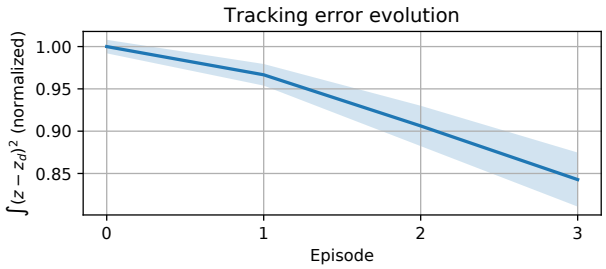


Figure 5.4: Mean $\pm$ 1 standard deviation of tracking performance after each episode over 5 independent campaigns.

is rigorously satisfied, and this constraint is active for longer duration. As the system learns more accurate dynamic models, it takes on more of an 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. 5.4. The tracking performance improves in every episode. Furthermore, the methodology has low variance between campaigns.

Chapter 6

ENSEMBLE MPC

This chapter focuses on learning parametric uncertainty of an unknown dynamical system, and formulates a computationally efficient Model Predictive Controller (MPC) that is robust to the considered uncertainty under certain conditions. The results of this Chapter have been presented at the 2020 Conference on Decision and Control [6].

6.1 Introduction

To motivate this work, consider a multi-rotor drone that must land quickly. Naturally, the drone should not hit the ground, even with limited actuator authority. Intuitively, the minimum time landing trajectory should incorporate aspects of bang-bang control: the drone should accelerate towards the ground as quickly as possible, and then brake as fast as possible. However, any error in the modeling of the actuation dynamics or the vehicle aerodynamics near the ground might not be recovered during closed loop control, as the actuation is under saturation. Practically, some actuation margin can be added to handle the modeling errors, but this solution leads to conservatism and does not provide any quantitative information about the performance-safety trade-offs. Next, consider the case of emergency braking in autonomous vehicles. The autonomous dynamics are easy to model, but the actuation dynamics present a harder challenge since in emergency situations, the control behavior depends on road conditions or hard-to-model variables, e.g. brake and tire temperatures. In these scenarios, actuation dynamics could be learned from operational data, and then used to rigorously impose safety during the planning and control process.

Several authors have considered uncertainty for MPC. One of the early works [59], considered the worst case scenario using impulse responses of the system, leading to exponential explosion of cases. The subsequent LTF-approach [43] avoided the combinatorial complexity of [59] at the expense of considering a more conservative linear controller in the analysis. The minimax approach [54] posed the problem as a second-order cone, and considered an extension to uncertainties in the actuation matrix ([54], chapter 10). We deviate from this work as we only consider formulations leading to quadratic programs that can be solved in real-time for practical

applications at the cost that it imposes some restrictions on the types of systems and uncertainties that can be considered. One of the restrictions is linear time invariant dynamics. As seen in previous chapters, we will use Koopman Operator theory to extend this resctriction to non-linear systems.

We employ Bayesian methods to update the parametric uncertainty. Specifically, we use an Ensemble Kalman method, as it models uncertainty as a polytope of parameters and directly propagates the polytope's vertices. But other methods like set membership identification could be used. These features are used synergistically when formulating the robust controller. Other sample-based methods, like Unscented or Quadrature methods have been proven useful for state estimation [4] and could be used in the parameter-estimation inversion setting. In comparison, ensemble methods allow the user to control the number of vertices, and the resampling step is avoided. Particle filters are similar to Ensemble methods, but the particles interact weakly only by re-sampling, avoiding the Gaussian assumption at the cost of very expensive computation for high-dimensional systems.

The Ensemble Kalman Filter was first introduced by Evensen [23] and then used in [10, 21] for state uncertainty estimation of large scale problems. Subsequently, it was introduced to solve inverse problems [35]. Different parametrizations were studied in [17], and convergence properties were analyzed in [73]. In [28], they proposed the Ensemble Kalman Sampler approach that we use in this paper, to solve the ensemble collapse of the Ensemble Kalman Inversion. In their preliminary results. the solution compares to the more expensive Random Walk Metropolis Hastings Monte Carlo Markov Chain [69].

6.2 Preliminaries on the Ensemble Kalman Sampler

The Ensemble Kalman Sampler (EKS) formulates the estimation problem as an inverse problem of the vector of parameters $\theta \in \mathbb{R}^{N_\theta}$ given noisy data as

$$y = G(\theta) + \eta, \quad \eta \overset{\text{i.i.d}}{\sim} \mathcal{N}(0, \Gamma) \tag{6.1}$$

where $y \in \mathbb{R}^{N_y}$ are the measurements, $G(\theta) : \mathbb{R}^{N_\theta} \to \mathbb{R}^{N_y}$ is the forward model, and η is the measurement noise, usually assumed known from sensor calibration. The objective of the inverse problem is to compute the unknown model parameters θ given the observation y, the known forward model G, and noise characteristics η of the process. For the Ensemble Kalman Sampler, the problem is formulated as a Bayesian regression where it maximizes the posterior probability represented by an ensemble, or set of particles. Starting with an initial estimate for each particle,

$u_n^{(0)}, n = 1, \ldots, N_u$, the new estimate for each particle is computed by performing a Kalman Measurement Update using the empirical covariances C_{GG} and $C_{\theta G}$, where the empirical estimates C_{xy} is computed as $C_{xy} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x}) \otimes (y_n - \bar{y})$ for $\{x, y\}$ equal to $G(u)$ or u. The new estimate for each particle is

$$K = C_{uG}\left(C_{GG} + \Gamma\right)^{-1} \tag{6.2}$$

$$u_n^{(k+1)} = u_n^{(k)} + K\left(y_n - G\left(u_n^{(k)}\right)\right). \tag{6.3}$$

The method as presented above would converge to the posterior mean, that is, the particle distribution will collapse to the mean of the true distribution. This result is desired in the optimization setting, but our interest is to recover the original distribution of the data. To do so, we follow the extension named Ensemble Kalman Sampler and we add two corrections: (1) we add a prior regularization, (2) we add a random variable sampled from the current empirical covariance.

$$u_n^{+(k+1)} = (I + C_{uu}\Gamma_0^{-1})u_n^{(k+1)} \tag{6.4}$$

$$u_n^{++(k+1)} = u_n^{+(k+1)} + \sqrt{2C_{uu}}\xi, \quad \xi \sim \mathcal{N}(0, I) \tag{6.5}$$

Algorithm 3 shows the pseudo-code version of the algorithm. Note that the state covariance is not updated but the spread of the ensemble particles determines the empirical covariance. Garbuno-Inigo *et al.* [28] showed that the posterior density is a global attractor for all initial densities of finite energy which are not a Dirac measure.

6.3 Ensemble Model Predictive Control (EnMPC)

We first describe an overview of the algorithm and how to incorporate parametric uncertainties in the MPC formulation. Then, we describe how to learn these uncertainties from data using ensemble Bayesian methods. Two particular cases will be explored: first, we consider linear dynamics and quadratic objective functions to formulate the problem as a quadratic program, second, we consider nonlinear autonomous dynamics and model them based on Koopman theory.

Overview of the EnMPC-algorithm

Consider a system with discrete-time control-affine dynamics

$$x_{k+1} = a(x_k) + b(x_k)u_k + w_k \tag{6.6}$$

where $a : \mathbb{R}^{N_s} \to \mathbb{R}^{N_s}, b : \mathbb{R}^{N_s} \to \mathbb{R}^{N_s} \times \mathbb{R}^{N_u}$, the allowed set for the states is $\mathcal{X}$, the control inputs must lie in the set $\mathcal{U}$, and the disturbance w is compactly

Algorithm 3 Ensemble Kalman Sampler

Input: initial ensemble members $u \in \mathbb{R}^{N \times N_e}$, number of iterations J, data $Y \in \mathbb{R}^{N_y}$, forward function $G : \mathbb{R}^N \longrightarrow \mathbb{R}^{N_y}$, measurement Noise covariance $\Gamma \in \mathbb{R}^{N_y \times N_y}$, prior covariance Γ_0

procedure EKS(u, J, Y, $G(\cdot)$, Γ, Γ_0)
  for $j = i \dots J$ **do**
    $\bar{u} = \frac{1}{N} \sum_{n=1}^{N} u_n$
    $\bar{G} = \frac{1}{N} \sum_{n=1}^{N} G(u_n)$
    $C_{uG} = \frac{1}{N} \sum_{n=1}^{N} (u_n - \bar{u}) \otimes \left(G(u_n) - \bar{G}\right)$
    $C_{uu} = \frac{1}{N} \sum_{n=1}^{N} (u_n - \bar{u}) \otimes (u_n - \bar{u})$
    $C_{GG} = \frac{1}{N} \sum_{n=1}^{N} \left(G(u_n) - \bar{G}\right) \otimes \left(G(u_n) - \bar{G}\right)$
    for $n = i \dots N$ **do**
      $y_n = Y + \eta$ ▷ $\eta \sim \mathcal{N}(0, \Gamma)$, i.i.d.
      $u_n = u_n + \Delta t C_{uG} \left(C_{GG} + \Gamma\right)^{-1} \left(y_n - G(u_n)\right)$
      $u_n = (I + \Delta t C_{uu} \Gamma^{-1}) u_n$
      $u_n = u_n + \sqrt{2 \Delta t C_{uu}} \xi$ ▷ $\xi \sim \mathcal{N}(0, I)$, i.i.d.
  return $u \in \mathbb{R}^{N_u \times N_e}$

supported by $\mathcal{W}$. We consider the setting where the dynamics of the true system (6.6) are unknown but that we know a model of the dynamics parametrized by $\theta = [\theta_a, \theta_b, \theta_w]$

$$x_{k+1} = a(x_k, \theta_a) + b(x_k; \theta_b) u_k + \theta_w \tag{6.7}$$

such that there exists a θ^* that exactly describes the true dynamics. We do not assume knowledge of θ^* but instead assume that we know a polytope $\mathcal{P} = \mathcal{A} \otimes \mathcal{B} \otimes \mathcal{W}$ defined by the convex full of the ensemble of θ such that $\theta^* \in \mathcal{P}$, $\theta_a^* \in \mathcal{A}$, $\theta_b^* \in \mathcal{B}$, $\theta_w^* \in \mathcal{W}$.

EnMPC consists of two main steps: first, we collect a dataset of state and input pairs from the system's trajectory. The collected data is used to fit a model of the dynamics as an ensemble of parameters θ, i.e. inferring the vertices of the polytope $\mathcal{P}$. Then, we generate a controller that is robust to the possible system realizations captured in the learned polytope. Deploying that controller allows us to generate more data and repeat the process. As the knowledge of $\mathcal{P}$ improve, the volume of $\mathcal{P}$ can be reduced, and the performance of the controller improved. Our proposed algorithm is summarized in Algorithm 4.

By incorporating the uncertainty of the system model, as described by the polytope $\mathcal{P}$, in a model predictive controller, robust forward invariance of the set $\mathcal{X}$ can be certified under specific assumptions on $a(x_{p-1})$, $b(x_{p-1})$, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{W}$. We first present the general MPC formulation and then discuss needed assumptions and

analyze the robustness of the controller in Sections 6.3 through 6.3.

We extend the standard MPC (2.8) with an additional set of constraints for each of the N_v vertices of the polytope $\mathcal{P}$. We denote

$$x_k^{(v)} \in \mathcal{X}, \forall k = 1, \cdots, N_t, v = 1, \cdots, N_v \tag{6.8}$$

where $x_k^{(v)}$ denotes the predicted state at time k as predicted by the dynamics model with θ equal to the v-th vertex of $\mathcal{P}$. This increases the complexity of the MPC formulation by adding $(N_v - 1)N_pN_s$ new constraints to the sparse formulation (2.8). This results in the robust MPC formulation

$$\begin{aligned} \min_{\substack{\boldsymbol{u}\in\mathbb{R}^{N_u\times N_p} \\ \boldsymbol{x}\in\mathbb{R}^{N_s\times N_p\times N_e}}} \quad & \textstyle\sum_{p=1}^{N_p-1} l(\bar{x}_p, u_p, \tau_p) + l_f(\bar{x}_{N_p}) \\ \text{s.t.} \quad & x_p^{(v)} = a(x_{p-1}^{(v)}; \theta_a^{(v)}) + b(x_{p-1}^{(v)}; \theta_b^{(v)})u_p + \theta_w^{(v)} \\ & x_p^{(v)} \in \mathcal{X} \qquad\qquad p = 1, \ldots, N_p \\ & u_p \in \mathcal{U} \qquad\qquad v = 1, \ldots, N_v \\ & x_0 = x_k \end{aligned} \tag{6.9}$$

where $\theta^{(v)} = [\theta_a^{(v)}, \theta_b^{(v)}, \theta_w^{(v)}]$ denotes the v-th vertex of $\mathcal{P}$.

In this work, we are particularly interested in applications where the uncertainty of the control vector fields is large, such $\max_{x\in\mathcal{X},u\in\mathcal{U},\theta_b\in\mathcal{B}} ||b(x,\theta_b)u - b(x,\theta_b^*)u|| >> \max_{x\in\mathcal{X},\theta_a\in\mathcal{A}} ||a(x,\theta_a) - a(x,\theta_a^*)||$, either because the uncertainty in the actuation matrix is big, or the elements of the matrix themselves are big. Considering these systems allows us to formulate (6.9) as a computationally efficient quadratic program under certain assumptions.

Learning Unknown Dynamics with Ensemble Kalman Sampler

In this section we discuss how to learn the unknown system dynamics model (6.7). For simplicity, we assume that we have access to noisy data snapshots gathered from executing trajectories with the system (6.6) under an arbitrary control law, such that we have a data set of state and control action snapshots $\mathcal{D} = \left\{ \{\tilde{x}_k, u_k\}_{k=1}^{N_t} \right\}_{d=1}^{N_{\text{traj}}}$, where N_t is the number of timesteps, and N_{traj} the number of trajectories. It can be extended to any other function of the state $h(x)$.

We define the forward function $G(\theta) \in \mathbb{R}^{N_g}, N_g = (N_t - n_k) \times N_s \times N_{\text{traj}}$ as the difference between the predicted state for two different time steps, usually known as multi-step prediction. For clarity, we omit the trajectory subindex in the following.

At timestep k,

$$G_k(\theta) = \hat{x}_{k+n_k} - \hat{x}_k \tag{6.10}$$

$$= \prod_{i=k}^{k+n_k} \left(a(\hat{x}_i, \theta_a) + b(\hat{x}_i, \theta_b)u_i + \theta_w \right) - \hat{x}_k, \tag{6.11}$$

where n_k represents the number of timesteps between state comparisons. Note that an alternate approach is to directly use the right hand side of the state evolution as the forward function, also called single-step prediction. The advantages of multi-step prediction are twofold: it increases sensitivity by integrating the dynamics over long periods of time, and avoids computation of numerical derivatives, but the measurements become loosely correlated.

From the raw dataset, we construct the measurement vector y by subtracting each measurement from a copy of the measurement shifted n_k timesteps, $y_k = \tilde{x}_{k+n_k} - \tilde{x}_k$. Other multi-step approaches are possible, but it is outside the scope of this chapter. Finally, the covariance of the measurement noise, η, is defined as the difference of two identically distributed i.i.d. Gaussian variables. Let $\tilde{x}_k = x_k + \eta_x$, $\eta_x \sim \mathcal{N}(0, R_x)$, where $R_x \in \mathbb{R}^{N_s \times N_s}$ is assumed to be known as part of the sensor calibration process. The EKS noise covariance matrix $\Gamma \in \mathbb{R}^{N_g \times N_g}$ is then the block diagonal matrix N_g times of $2R_x$.

Algorithm 4 Ensemble Dynamical Learning with Robust EnMPC Algorithm

procedure EN-DL-MPC($\theta_0, N_{\text{traj}}, N_t$)
  Input: Initial Prior Dynamics θ_0, Number of trajectories N_{traj}, number of timesteps N_t
  $Y = [\,]$
  for $j = i \ldots N_{\text{traj}}$ **do**
    Initialize the system x_0
    for $k = 1 \ldots N_t$ **do**
      Collect New State x_k
      $u_k = \text{EnMPC}\,(x_k; \theta_k)$
      Command u_k
    $Y_j = [Y_j, \{\boldsymbol{x}, \boldsymbol{u}\}]$ ▷ Aggregate Measurements
    $\theta_j = \text{EKS}(\theta_{j-1}, Y)$ ▷ Compute new ensemble
  return Learned Dynamics $\theta_{N_{\text{traj}}}$

Linear EnMPC

Consider a linear dynamical system of the form (6.6) with $a(x) = Ax$ and $b(x) = B$

$$x_{k+1} = Ax_k + Bu_k + E \tag{6.12}$$

where $A \in \mathbb{R}^{N_s \times N_s}, B \subset \mathbb{R}^{N_s \times N_u}$, and $E \in \mathbb{R}^{N_s}$. Note that E adds a bias to w, shifting its compact support. In addition, we assume a quadratic cost $l(x_p, u_p, \tau_p) = (x_p - \tau_p)^T Q(x_p - \tau_p) + u_T R u_p$, and $l_f(x_{N_p}) = x_{N_p}^T Q_f x_{N_p}$, where $Q, Q_f \in \mathbb{R}^{N_s \times N_s}$ and $R \in \mathbb{R}^{N_u \times N_u}$ are positive semidefinite cost matrices. The system parametrization has the structure $\theta = \{\text{vec}(A), \text{vec}(B), E\}$. For convenience, we explicitly write the state variables as a function of the control input $\boldsymbol{x}^{(v)} = \boldsymbol{A}^{(v)} x_k + \boldsymbol{B}^{(v)} \boldsymbol{u} + \boldsymbol{w}^{(v)}$, where $\boldsymbol{w}^{(v)} = [1, \ldots, N_p]^T w^{(v)}$ propagates $w^{(v)}$. This defines the following MPC problem

$$\begin{aligned} \min_{u \in \mathbb{R}^{m \times N_p}} \quad & (\bar{\boldsymbol{x}} - \boldsymbol{\tau})^T \boldsymbol{Q} (\bar{\boldsymbol{x}} - \boldsymbol{\tau}) + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u} \\ \text{s.t.} \quad & \boldsymbol{A}^{(v)} x_0 + \boldsymbol{B}^{(v)} \boldsymbol{u} + \boldsymbol{w}^{(v)} \in \mathcal{X} \qquad b = 1, \ldots, N_v \\ & \boldsymbol{u} \in \mathcal{U} \\ & x_0 = x_k \end{aligned} \tag{6.13}$$

where bar elements denote the mean of all possible elements in $\mathcal{P}$. Importantly, the number of constraints added is *linear* in the number of vertices in $\mathcal{P}$. This is the key computational advantage of only considering uncertainty in the actuated dynamics and completely avoids costly combinatorial searches present in other robust MPC approaches [54], while still handling uncertainty in the dynamics.

Proposition 5. For the system (6.12), if the actuation matrix B associated with the true dynamics lies in the polytope $\mathcal{B}$ with probability $(1 - \delta)$, there is an initial feasible solution, then, with probability $(1 - \delta)$, the system under the designed control law (6.13) renders the constraint set $\mathcal{X}$ forward invariant in the presence of the actuation constraints $u \in \mathcal{U}$.

Proof. First, the cost function is convex, and propagating the states with linear dynamics generate a convex set [7], so the extreme of the propagated points must be generated with the vertices of the polytope $\mathcal{P}$. Next, we show recursive feasibility using the initially feasible open-loop input sequence $\boldsymbol{u}_0$. At $k = 1$, the initial feasible solution satisfies the constraint set, therefore, $x_{1|1}$ is inside the convex hull defined by $(x_{1|0}^v,\ v = 1, \ldots, N_v)$ with probability $1 - \delta$. If we applied the rest of

the open-loop input sequence at $x_{1|1}$, it would generate states $x_{p|0},\ p = 2, \ldots, N_p$, where each state is shifted to the previous sequence $x_{p|0}\, p = 1, \ldots, N_p$ by linearity of the dynamics, and bounded worst case of w, but starting from $x_{1|1}$. This sequence is shifted to the inside of the constraint set so it will also be inside the constraint set. Thus, the open loop sequence $u_{p|0},\ p = 1, \ldots, N_p$ is a valid feasible solution to the optimization problem at $k = 1$, but there might be other solutions with lower cost. In particular, the optimal solution to the mean cost is unique and is guaranteed to found [77]. A similar argument can be used for $k = 2$ using the open-loop controller from $k = 1$, showing recursive feasibility. □

Extension to Nonlinear Autonomous Dynamics using Koopman Operator Theory

We now consider affine system dynamics with a nonlinear drift term and constant linear input vector fields, i.e. as (6.6) with $b(x) = B$

$$x_{k+1} = a(x_k) + Bu_k + w_k. \tag{6.14}$$

We use Koopman-based learning to estimate (6.14) and thus aim to learn a lifted-dimensional linear model of the form

$$z_{k+1} = A_z z_k + B_z u_k + E_z \tag{6.15}$$

$A_z \in \mathbb{R}^{N_z \times N_z}$, $B_z \in \mathbb{R}^{N_z \times N_u}$, and $E_z \in \mathbb{R}^{N_z}$.

The state itself and learned Koopman eigenfunctions are used as the lifted state $z = [x, \phi(x)]^T$. These eigenfunctions, $\phi(x)$ are learned and constructed using the approach in Section 4.5. I.e. collected data from executed trajectories of the system (6.14) is used to construct eigenfunctions $\phi(x) = [\phi_1(x), \ldots, \phi_{N_z - N_s}(x)]^T$ that define nonlinear transformations of the state variables. Similarly to EDMD as described by (4.9), we parametrize (6.15) by $\theta = \{\text{vec}(A_z), \text{vec}(B_z), E_z\}$ and then use EKS to infer the best fit ensemble of parameters θ. This is done by applying EKS to the lifted dataset $\mathcal{D}_z = \left\{ \{z_k, u_k\}_{k=1}^{N_t} \right\}_{d=1}^{N_{\text{traj}}}$ which is the same data as $\mathcal{D}$, but with the state replaced with the lifted state. With z defined in this way, $x = Cz, C = [I \quad 0]$ and we obtain the following MPC problem

$$\begin{aligned} \min_{u \in \mathbb{R}^{m \times N_p}} \quad & (C\bar{\boldsymbol{z}} - \boldsymbol{\tau})^T \boldsymbol{Q} (C\bar{\boldsymbol{z}} - \boldsymbol{\tau}) + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u} \\ \text{s.t.} \quad & \boldsymbol{A}_z^{(v)} z_0 + \boldsymbol{B}_z^{(v)} \mathbf{u} + \mathbf{w}^{(v)} \in \mathcal{X} \qquad v = 1, \ldots, N_v \\ & \mathbf{u} \in \mathcal{U} \\ & z_0 = [x_0,\ \phi(x_0)]^T \end{aligned} \tag{6.16}$$

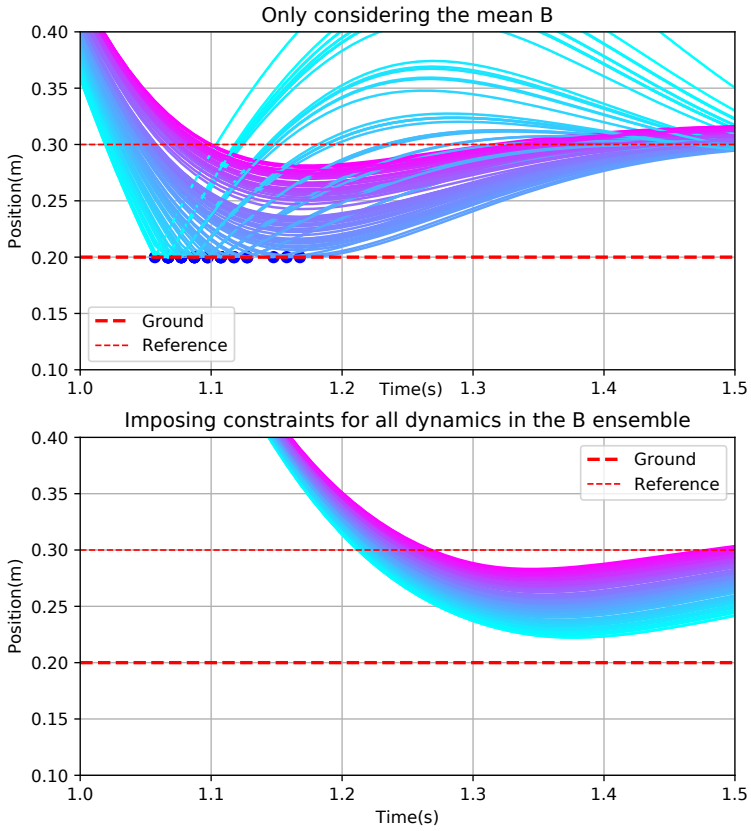


Figure 6.1: Comparison of standard MPC with robust EnMPC for different values of actuation matrices B.

Note that if the lifted dynamics model (6.15) satisfies the assumptions of Proposition 1, the controller (6.16) certifies $\mathcal{X}$ to be forward invariant for the system dynamics (6.14).

6.4 Simulation Results

To obtain an initial evaluation of the performance of the proposed framework, we study one dimensional multirotor dynamics and aim to design a controller that can land from an initial position as quickly as possible while being robust to model uncertainty such that state constraints are not violated. The code for running the experiments is publicly available at `https://github.com/PastorD/ensemblempc`. Consider the dynamics:

$$\ddot{z} = \frac{1}{m}\left(- mg - \frac{1}{2}C_d \rho A \dot{z}^2 + \frac{1}{1-(R_a/4z)^2}T \right) \tag{6.17}$$

where z is the altitude of the quadrotor, m, C_d, A, R_a is the drone's mass, drag coefficient, surface area facing the xy-plane, and rotor radius, respectively. Finally, ρ is the air density. The first term of the dynamics accounts for the gravity effect, the second term captures aerodynamic drag, and the last term is the effect of the total thrust of the rotors, T, including ground effects. For the following simulations we use the same cost function, $Q = \text{diag}(10^5, 1), R = 1$, and terminal state $x = [0, 0]^T$.

First, we show that without the robustness constraint, a traditional MPC algorithm might violate the state constraints in closed loop. For this purpose, we consider a simplified version of (6.17) using only the gravity and direct control input, *i.e.* $\ddot{z} = g - T$, control bounds $T \in [0, 1]$, and state constraints $z < 0 \, \forall t$. We sample the control matrix from the distribution $B = [0, b]^T, b \sim \mathcal{N}(1, 0.4)$, to simulate different conditions for each experiment. Figure 6.1 shows results for 200 trajectories. The traditional MPC algorithm, on the top, crashes for any large values of b. On the bottom, the EnMPC only hits the ground for high values of b outside the initial ensemble spread. Note that adding extra ground clearance can also avoid the impacts, but it is not known a priori how much ground clearance to add given a

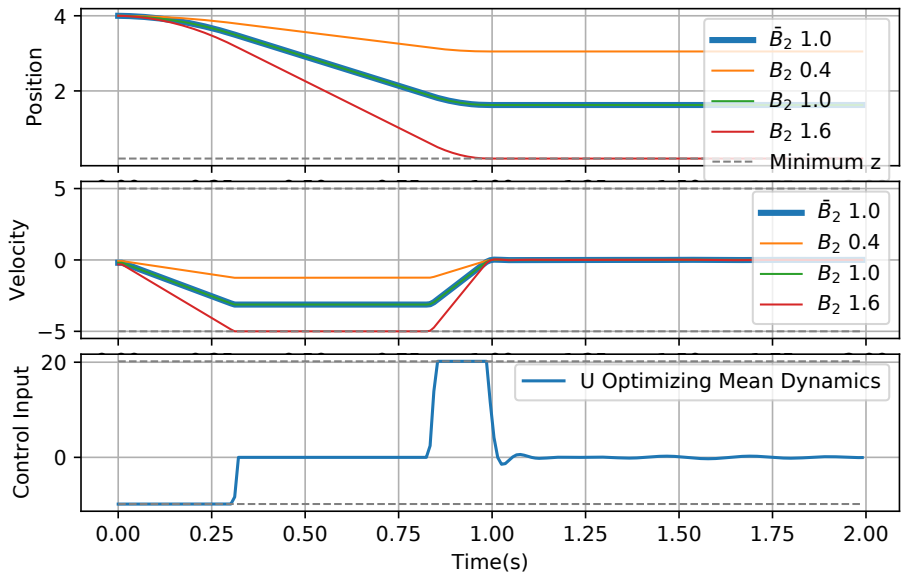


Figure 6.2: Example of initial prediction trajectories for different ensemble dynamics. The solution optimizes the mean dynamics, while keeping the position and velocity constraints for all dynamics in the ensemble.

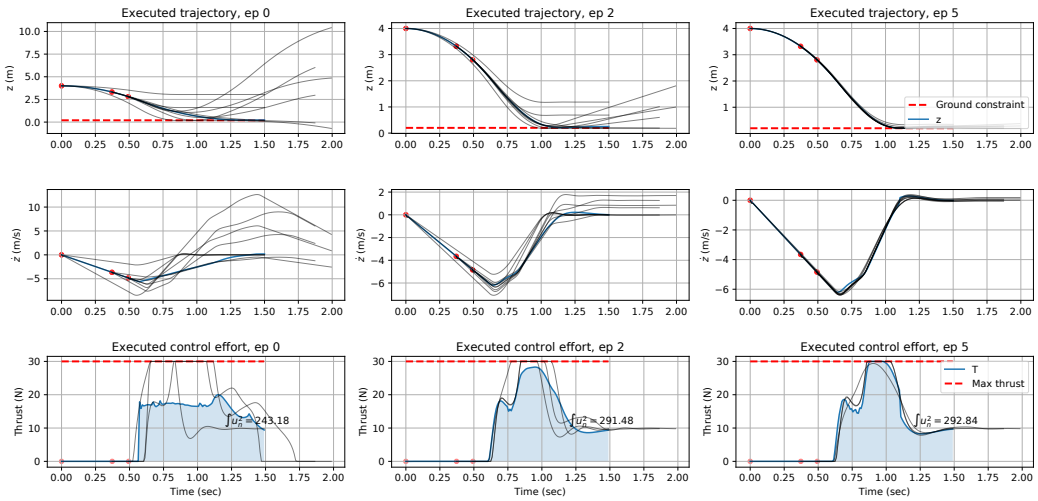


Figure 6.3: Closed-loop trajectories using EnMPC at selected episodes. For smaller uncertainties set, the MPC cost is not being reduced significantly, but the predicted trajectories (in gray starting from the red dots) become closer to the closed-loop trajectory. Similarly, the predicted thrust does not coincide with the closed-loop thrust when the uncertainty is high.

known uncertainty error. To illustrate the EnMPC solution, Figure 6.2 shows the predicted trajectories at the beginning of each simulation taking into consideration the dynamics in the ensemble.

Next, we show how a smaller ensemble provides lower prediction uncertainty, the controller can be more aggressive, and therefore improve performance while satisfying state constraints. Figure 6.3 shows the value for the second element of B for each episode, the evolution of the position, velocity, and control input for selected episodes. The predicted open-loop trajectory costs are displayed in gray for several timesteps. As the uncertainty in the actuation matrix decreases, the prediction becomes tighter and the performance improves.

6.5 Conclusion

This chapter presented a model predictive controller for uncertain systems. Parametric uncertainty was modelled as a polytopic set and updated using ensemble methods. This allows to formulate the optimization problem as a quadratic program, which is fast to solve for agile applications. Using past trajectories, the estimation of the parametric uncertainty is improved to reduce the uncertainty of the prediction, resulting in a less conservative controller with better performance while satisfying state and input constraints. The analysis of the proposed methods focused on linear systems and it was extended to certain nonlinear systems using

Koopman-based learning. The initial experiments show promising results, and future work will further develop the theoretical aspects of the algorithm and apply it to more challenging problems.

Chapter 7

CONCLUSION

This thesis has presented multiple results to ensure stability and safety, as it applied to multirotor flight. It is divided in two main approaches: one focusing on the mechanical design and the other one on the controller design. They are explained in Chapters 3-6 as follows.

Chapter 3 described several designs to quickly deploy an aerial asset using a pressurized tube. The first design, SQUID 3", was a proof of concept and it introduced several design criteria for successful deployment. It contained a 3D printed body with off-the-shelf electronics. It demonstrated several successful launches from a vehicle moving at 50 mph. The contributions include a novel design for multirotor take off and extensive testing under harsh conditions. The second prototype, SQUID 6", demonstrated autonomous flight using on-board cameras. To accommodate the bigger payload, the size was increased to six inches. The unibody design was abandoned for a open design consisting of carbon fiber plates, adding structural integrity and ease of usage. Additionally, the fins were replaced with a similar folding mechanism as the arms. This change allowed the fins to engage the free stream and it was validated using a 2" scaled model on a wind tunnel.

The next 3 chapters used the Koopman Operator Framework to model and control non-linear dynamical systems. The main application is multirotor control but simple examples are provided to illustrate each method. Chapter 4 proposed a novel technique to find the Koopman eigenfunctions directly from data. First, it computes the principal eigenfunctions analytically of the linearized system. Next, it learns the non-linear mapping using function approximation. The computed eigenfunctions are then used as lifting functions to learn a finite aproximation of the Koopman Operator. Simulations show that this approach predicts better the non-linear evolution than several standard choices of lifting functions. It also shows that the learned model can be used for Model Predictive Control, improving the closed-loop performance. One of the contributions is a dense approach to MPC, removing the dependency of the lifting dimensions by explicitly writing the state evolution as a function of the control input. This allows to quickly compute the optimal control action, while satisfying state and actuator constraints.

Chapter 5 demonstrated the usage of Koopman Operator Framework for a practical demanding robotic application. The method developed in the previous chapter allowed fast and safe control, however, it is limited to a constant actuation matrix. For many robotic applications this imposes a strong limitation and most commonly used dynamical models include an control-affine function. To solve this limitation, this chapter introduced an extension to the previous method using an episodic approach. At the end of each episode, the controller learns a Koopman Operator of the closed-loop dynamics. Wrapping the non-linear controller in the autonomous dynamics effectively allows to model the non-linear actuation effects using the methods seen in the previous chapter. To demonstrate the method, it applies the algorithm to the case of fast multirotor landing. During each trajectory, the controller satisfies a set speed, position, and control actuation limits.

Lastly, Chapter 6 incorporates model uncertainty into the Koopman-based Model Predictive Controller framework. It considers a model represented as a polytope in parameter space. The linear formulation of the Koopman Operator allows to keep the optimization problem as a quadratic program, enabling fast computation of the robust controller while satisfying state and control input constraints. Modeling uncertainty is particularly important for systems following optimal control actions with saturated inputs, as any disturbance in the direction of the constraint might lead to the violation of those constraints. This chapter showed how to learn model uncertainty from data using ensemble methods, and how to incorporate it to a MPC formulation. It showed in simulation the improved performance as more data is gathered and the uncertainty bounds decrease.

7.1 Further Work

SQUID

There are several directions being investigated at the publication of this thesis. A new design with four inches of diameter was being tested but, due to the pandemic restrictions, the development stopped. It shares the plate design of SQUID 6", but it aims to be fully autonomous by just using a GPS device instead of a camera. This allows to reduce the overall size and weight, making it easier and safer to operate. In addition, the application for aerial deployment for future Mars missions is an active project at JPL. An extension is being investigated where the Mars Helicopter is being released from mid-flight, instead of from under the belly after landing in the current design. This would allow to visit a big area of Mars that is currently inaccessible, as current rover landing technology only allows to land on low elevation terrain.

The original SQUID project was conceived as a way to put an aerial asset using a pressurized tube launcher. However, there are other methods to put a multirotor in the air without a pressurized tube launcher. In particular, it can simply take off using its own propellers like a standard multirotor. They can reach high speeds quickly, as a good rule of thumb is to design the maximum thrust at least twice the weight, but racing drones normally surpass a ratio of ten. This design would simplify and strengthen the structure avoiding moving arms. One of the main problems would be the high speed apparent wind when it is launched from a moving vehicle. Detailed analysis and testing would be needed to ensure it can take off without losing stability. From the control perspective, this vehicle introduces interesting challenges given the impulsive nature of the take off. To simplify the takeoff, guide rods could be deployed so the trajectory is constrained while it is in contact with the rods. Once it leaves the guide rods, it would have some speed and its flight would be similar to a traditional multirotor flying forward. Additionally, the platform could be oriented forward so the launching angle is similar to the pitch angle flying at that speed. This flat SQUID would occupy a larger footprint for a similar propeller size, but the vertical clearance would be significantly smaller. While several of the original tube launchers can be stacked laterally, several flat SQUID could be stacked vertically. This thin package could retrofitted into vehicles without a track bed, for example, a police car could attach one of these vehicles on top of the car roof to quickly deploy a multirotor at the press of a button. Another advantage is the lack of the pressurized components, which require especial usage. In summary, the flat version of SQUID would resemble a traditional multirotor, launched from a moving vehicle with the help of guide rods and an inclined platform. It would introduce new control challenges and thanks to its simplicity it would be closer to be used in real applications.

Koopman Operator

The Koopman Operator was originally intended for autonomous systems. Although there have been several extensions for controlled systems, they do not seem to fit within the formulation and they do not work well with robotic applications. Chapter 5 proposed one of those solutions, but not without drawbacks. A new paradigm is needed to directly model controlled systems, not as an extension to a Koopman model. A direction that could be exploited is to lift the control signal to a higher dimensional space, in a similar way as the state is lifted in the original formulation. An alternative direction could be to use a different model class to

model the control input. Bilinear models have been proposed in recent work and they are able to model generic control-affine dynamics.

Future work will test the Koopman Operator for more demanding applications. Most of the published work focus on simulations with simple dynamics. This thesis presented work on a physical platform, but only in 2 dimensions. A more challenging robotic application, such a bipedal robot, would test if it is feasible to implement a Koopman-based control for high dimensions. Similarly, Chapter 5 considered full-state feedback. Measurements in high dimensional spaces, such as raw pixels, are particularly challenging for traditional robotics, but they are commonly used in machine learning applications. Koopman Operator theory could be used to blend both approaches by providing rigorous analysis using high capacity models as part of the lifting functions.

BIBLIOGRAPHY

[1] Ascent AeroSystems. Ascent aerosystems, (accessed on January 2019). URL http://www.ascentaerosystems.com/.

[2] Aaron D. Ames. Human-inspired control of Bipedal walking robots. *IEEE Transactions on Automatic Control*, 59(5):1115–1130, 2014. ISSN 00189286. doi: 10.1109/TAC.2014.2299342.

[3] Aaron D. Ames, Xiangru Xu, Jessy W. Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2017. ISSN 00189286. doi: 10.1109/TAC.2016.2638961.

[4] Ienkaran Arasaratnam and Simon Haykin. Square-root quadrature Kalman filtering. *IEEE Transactions on Signal Processing*, 56(6):2589–2593, 6 2008. ISSN 1053587X. doi: 10.1109/TSP.2007.914964.

[5] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. In *Robustness in Identification and Control*, pages 207–226, London, 1999. Springer London. ISBN 978-1-84628-538-7.

[6] Erik Berger, Mark Sastuba, David Vogt, Bernhard Jung, and Heni Ben Amor. Estimation of perturbations in robotic behavior using dynamic mode decomposition. *Advanced Robotics*, 29(5):331–343, 3 2015. ISSN 0169-1864. doi: 10.1080/01691864.2014.981292. URL https://doi.org/10.1080/01691864.2014.981292.

[7] Dimitri P Bertsekas. Nonlinear Programming. *Journal of the Operational Research Society*, 48(3):334–334, 3 1997. ISSN 0160-5682. doi: 10.1057/palgrave.jors.2600425. URL https://www.tandfonline.com/doi/full/10.1057/palgrave.jors.2600425.

[8] Michael Bloesch, M Burri, S Omari, M Hutter, and R Siegwart. Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research*, 36(10):1053–1072, 2017.

[9] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017. doi: 10.1017/9781139061759.

[10] Jean-Michel Brankart, Emmanuel Cosme, Charles-Emmanuel Testut, Pierre Brasseur, and Jacques Verron. Efficient adaptive error parameterizations for square root or ensemble kalman filters: application to the control of ocean mesoscale signals. *Monthly weather review*, 138(3):932–950, 2010.

[11] Roland Brockers, M Humenberger, D Weiss, and L Matthies. Towards autonomous navigation of miniature UAV. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2014.

[12] Bingni W. Brunton, Lise A. Johnson, Jeffrey G. Ojemann, and J. Nathan Kutz. Extracting spatial-temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. *Journal of Neuroscience Methods*, 258: 1–15, 2016. ISSN 1872678X. doi: 10.1016/j.jneumeth.2015.10.010. URL http://dx.doi.org/10.1016/j.jneumeth.2015.10.010.

[13] Steven Brunton and Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[14] Steven L. Brunton, Joshua L. Proctor, J. Nathan Kutz, and William Bialek. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America*, 113(15):3932–3937, 2016. ISSN 10916490. doi: 10.1073/pnas.1517384113. URL http://www.pnas.org/content/pnas/early/2016/03/23/1517384113.full.pdf.

[15] Nathan Bucki and M Mueller. Design and control of a passively morphing quadcopter. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[16] Marko Budišić, Ryan Mohr, and Igor Mezić. Applied Koopmanism. *Chaos*, 22(4), 2012. ISSN 10541500. doi: 10.1063/1.4772195.

[17] Neil K. Chada, Marco A. Iglesias, Lassi Roininen, and Andrew M. Stuart. Parameterizations for ensemble Kalman inversion. *Inverse Problems*, 2018. ISSN 13616420. doi: 10.1088/1361-6420/aab6d9.

[18] Richard Cheng, G Orosz, R M Murray, and J W Burdick. End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks. In *Proceedings AAAI*, January 2019.

[19] DJI. Mavic 2-DJI store, (Accessed on 01/2019). URL https://store.dji.com/product/mavic-2.

[20] Yan Duan, Xi Chen, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *arXiv*, 2016. ISSN 10636919. doi: 10.1109/CVPR.2014.180.

[21] Ammon N. Eaton, Logan D.R. Beal, Sam D. Thorpe, Ethan H. Janis, Casey Hubbell, John D. Hedengren, Roar Nybø, Manuel Aghito, Knut Bjørkevoll, Rachid El Boubsi, Jelmer Braaksma, and Geertjan Van Og. Ensemble model Predictive control for robust automated managed pressure drilling. In *Proceedings - SPE Annual Technical Conference and Exhibition*, volume 2015-Janua,

pages 3335–3350. Society of Petroleum Engineers (SPE), 9 2015. ISBN 9781510813229. doi: 10.2118/174969-ms.

[22] Utku Eren, Anna Prach, Basaran Bahadir Koçer, Saša V. Rakovic, Erdal Kayacan, and Behçet Açikmese. Model predictive control in aerospace systems: Current state and opportunities. *Journal of Guidance, Control, and Dynamics*, 40(7):1541–1566, 2017. ISSN 15333884. doi: 10.2514/1.G002507.

[23] Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research*, 99(C5), 1994. ISSN 01480227. doi: 10.1029/94jc00572.

[24] Matthias Faessler, F Fontana, C Forster, and D Scaramuzza. Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[25] Davide Falanga, K Kleber, S Mintchev, D Floreano, and D Scaramuzza. The foldable drone: A morphing quadrotor that can squeeze and fly. In *IEEE Robotics and Automation Letters*. IEEE, 2019.

[26] Yizhou Fang and Antonios Armaou. Nonlinear Model Predictive Control using a bilinear Carleman linearization-based formulation for chemical processes. *Proceedings of the American Control Conference*, 2015-July:5629–5634, 2015. ISSN 07431619. doi: 10.1109/ACC.2015.7172221.

[27] C. Folkestad, D. Pastor, I. Mezic, R. Mohr, M. Fonoberova, and J.W. Burdick. Extended Dynamic Mode Decomposition with Learned Koopman Eigenfunctions for Prediction and Control. In *Proc. American Control Conf.*, September (submitted) 2020.

[28] Alfredo Garbuno-Inigo, Franca Hoffmann, Wuchen Li, and Andrew M. Stuart. Interacting Langevin diffusions: Gradient structure and ensemble Kalman sampler. *SIAM Journal on Applied Dynamical Systems*, 19(1):412–441, 2020. ISSN 15360040. doi: 10.1137/19M1251655.

[29] Javier García and Fernando Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 2015. ISSN 15337928. doi: 10.1109/TNNLS.2017.2654539.

[30] John Cecil Gibbings. *Dimensional analysis*. Springer Science & Business Media, 2011.

[31] Patrick Gnemmi, S Changey, K Meder, E Roussel, C Rey, C Steinbach, and C Berner. Conception and manufacturing of a projectile-drone hybrid system. *IEEE/ASME Transactions on Mechatronics*, 22(2):940–951, 2017.

[32] Luke Henderson, Twain Glaser, and Falko Kuester. Towards bio-inspired structural design of a 3D printable, ballistically deployable, multi-rotor UAV. In *Aerospace Conference, 2017 IEEE*, pages 1–7. IEEE, 2017.

[33] SH Hoerner. Fluid-dynamic lift. *Hoerner Fluid Dynamics*, 1985.

[34] Sighard F Hoerner. *Fluid-dynamic Drag: practical information on aerodynamic drag and hydrodynamic resistance*. Hoerner Fluid Dynamics, 1958.

[35] Marco A Iglesias, Kody JH Law, Andrew M Stuart, Kody J H Law, and Andrew M Stuart. Ensemble Kalman methods for inverse problems. *Inverse Problems*, 29:45001, 2013. doi: 10.1088/0266-5611/29/4/045001. URL http://iopscience.iop.org/0266-5611/29/4/045001.

[36] Joan Sola, Jérémie Deray, and Dinesh Atchuthan. A micro Lie theory for state estimation in robotics. *CoRR*, abs/1812.0, 12 2018. URL https://arxiv.org/abs/1812.01537.

[37] Leland H Jorgensen. *Prediction of static aerodynamic characteristics for space-shuttle-like and other bodies at angles of attack from 0 deg to 180 deg*. NASA, 1973.

[38] Erika Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2219), 2018. ISSN 14712946. doi: 10.1098/rspa.2018.0335. URL https://docs.wixstatic.com/ugd/c50953_e5f09192e29340e3872a333a5556bed1.pdf.

[39] Eurika Kaiser, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of Koopman eigenfunctions for control. *arXiv preprint arXiv:1707.01146*, 2017. URL http://arxiv.org/abs/1707.01146.

[40] Hassan K Khalil. Nonlinear systems. *Upper Saddle River*, 2002.

[41] Milan Korda and Igor Mezić. Learning Koopman eigenfunctions for prediction and control: the transient case. *arXiv preprint arXiv:1810.08733*, pages 1–32, 2018. URL http://arxiv.org/abs/1810.08733.

[42] Milan Korda and Igor Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93: 149–160, 2018. ISSN 00051098. doi: 10.1016/j.automatica.2018.03.046.

[43] Mayuresh V. Kothare, Venkataramanan Balakrishnan, and Manfred Morari. Robust constrained model predictive control using linear matrix inequalities. *Automatica*, 32(10):1361–1379, 10 1996. ISSN 00051098. doi: 10.1016/0005-1098(96)00063-5.

[44] Moussa Labbadi and Mohamed Cherkaoui. Robust adaptive backstepping fast terminal sliding mode controller for uncertain quadrotor UAV. *Aerospace Science and Technology*, 93:105306, 10 2019. ISSN 12709638. doi: 10.1016/j.ast.2019.105306. URL www.elsevier.com/locate/aescte.

[45] Yueheng Lan and Igor Mezić. Linearization in the large of nonlinear systems and Koopman operator spectrum. *Physica D*, 242(1):42–53, 2013. ISSN 01672789. doi: 10.1016/j.physd.2012.08.017. URL http://dx.doi.org/10.1016/j.physd.2012.08.017www.elsevier.com/locate/physd.

[46] Henry L Langhaar. *Dimensional analysis and theory of models*. Robert E. Krieger publishing company, 1980.

[47] Taeyoung Lee, Melvin Leok, and N Harris Mcclamroch. Geometric tracking control of a quadrotor UAV for extreme maneuverability. *IFAC Proceedings Volumes*, {44}, 2011. doi: 10.3182/20110828-6-IT-1002.03599.

[48] Gordon J Leishman. *Principles of helicopter aerodynamics with CD extra*. Cambridge University Press, 2006.

[49] Leonardo. Horus-detail-leonardo, (Accessed on July 2019). URL https://www.leonardocompany.com/en/allproducts.

[50] LeveTop. The foldable & portable drone. URL https://www.levetop.com/.

[51] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 9 2015. URL http://arxiv.org/abs/1509.02971.

[52] Cunjia Liu, Hao Lu, and Wen-Hua Chen. An explicit mpc for quadrotor trajectory tracking. In *2015 34th Chinese Control Conference (CCC)*, pages 4055–4060. IEEE, 2015.

[53] Lennart Ljung. *System identification—Theory for the user*. Prentice Hall, 1987. ISBN 0136566952.

[54] Johan Löofberg. *Minimax approaches to robust model predictive control*, volume 812. Linköping University Electronic Press, 2014. ISBN 9173736228.

[55] Sergei Lupashin, Markus Hehn, Mark W Mueller, Angela P Schoellig, Michael Sherback, and Raffaello D'Andrea. A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, 24(1):41–54, 2014.

[56] David Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000. ISSN 00051098. doi: 10.1016/S0005-1098(99)00214-9.

[57] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011. ISSN 10504729. doi: 10.1109/ICRA.2011.5980409.

[58] Ryan Mohr and Igor Mezić. Construction of eigenfunctions for scalar-type operators via Laplace averages with connections to the Koopman operator. *arXiv preprint arXiv:1403.6559*, pages 1–25, 2014. URL http://arxiv.org/abs/1403.6559.

[59] Peter Morari and Manfred Campo. Robust model predictive control. In *American Control Conference*, pages 217–. IEEE, 1987. doi: 10.1007/978-0-85729-398-58.

[60] Benjamin J. Morris, Matthew J. Powell, and Aaron D. Ames. Continuity and smoothness properties of nonlinear optimization-based feedback controllers. *Proceedings of the IEEE Conference on Decision and Control*, 54rd IEEE (Cdc):151–158, 2015. ISSN 07431546. doi: 10.1109/CDC.2015.7402101.

[61] NASA. Mars helicopter to fly on NASA's next red planet rover mission, May 2018 (accessed on January 2019). URL www.nasa.gov/press-release/mars-helicopter-to-fly-on-nasa-s-next-red-planet-rover-mission.

[62] Parrot. Drone camera 4k HDR ANAFI, (Accessed on 01/2019). URL https://www.parrot.com/us/drones/anafi.

[63] Daniel Pastor, Jacob Izraelevitz, Paul Nadan, Amanda Bouman, Joel Burdick, and Brett Kennedy. Design of a ballistically-launched foldable multirotor. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[64] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[65] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL http://scikit-learn.sourceforge.net.

[66] Powervision. Poweregg camera drone, fly to the future, (accessed on January 2019). URL https://www.powervision.me/en/product/poweregg.

[67] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Generalizing koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930, 2018.

[68] Raytheon. Coyote UAS, (Accessed July 2019). URL `https://www.raytheon.com/capabilities/products/coyote`.

[69] Gareth Roberts, A. Gelman, and W. R. Gilks. Weak convergence and optimal scaling of random walk Metropolis algorithms. *Annals of Applied Probability*, 7(1):110–120, 1997. ISSN 10505164. doi: 10.1214/aoap/1034625254.

[70] Ugo Rosolia and Francesco Borrelli. Learning model predictive control for iterative tasks. A data-driven fontrol framework. *IEEE Transactions on Automatic Control*, 63(7), 2017. doi: 10.1109/TAC.2017.2753460.

[71] Stéphane Ross, Geoffrey J Gordon, J Andrew Bagnell, and Machine Learning. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[72] Clarence W. Rowley, Igor Mezi, Shervin Bagheri, Philipp Schlatter, and Dan S. Henningson. Spectral analysis of nonlinear flows. *Journal of Fluid Mechanics*, 641(Rowley 2005):115–127, 2009. ISSN 00221120. doi: 10.1017/S0022112009992059.

[73] Claudia Schillings and Andrew M Stuart. Convergence Analysis of Ensemble Kalman Inversion: The Linear, Noisy Case. In *Applicable Analysis*, 2018.

[74] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010. ISSN 14697645. doi: 10.1017/S0022112010001217.

[75] E. Dale Seborg and A. Michael Henson. Nonlinear Process Control, 1997.

[76] Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural Lander: Stable Drone Landing Control using Learned Dynamics. *International Conference on Robotics and Automation (ICRA)*, pages 9784–9790, 2019. URL `http://arxiv.org/abs/1811.08027`.

[77] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: An Operator Splitting Solver for Quadratic Programs. *2018 UKACC 12th International Conference on Control, CONTROL 2018*, page 339, 2018. doi: 10.1109/CONTROL.2018.8516834.

[78] Kunihiko Taira, Steven L Brunton, Scott TM Dawson, Clarence W Rowley, Tim Colonius, Beverley J McKeon, Oliver T Schmidt, Stanislav Gordeyev, Vassilios Theofilis, and Lawrence S Ukeiley. Modal Analysis of Fluid Flows: An Overview. *Aiaa Journal*, 55(12):4013–4041, 2017. doi: 10.2514/1.J056060.

[79] Andrew J Taylor, Victor D Dorobantu, Hoang M Le, Yisong Yue, and Aaron D Ames. Episodic learning with control lyapunov functions for uncertain robotic systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6878–6884. IEEE, 2019.

[80] The Johns Hopkins University Applied Physics Laboratory. Mars helicopter to fly on NASA's next red planet rover mission, 2018 (accessed on January 2019). URL `dragonfly.jhuapl.edu`.

[81] Adam Thurn, S. Huynh, S. Koss, P. Oppenheimer, S. Butcher, J. Schlater, and P. Hagan. A nichrome burn wire release mechanism for cubesats. *Proceedings of the st Aerospace Mechanisms Symposium*, 41:16–18, May 2012.

[82] Yi Ju Tsai, Chia Sung Lee, Chun Liang Lin, and Ching Huei Huang. Development of flight path planning for multirotor aerial vehicles. *Aerospace*, 2(2): 171–188, 2015. ISSN 22264310. doi: 10.3390/aerospace2020171.

[83] tshirtguns. Bleacher reacher mega t-shirt launcher. URL `http://tshirtgun.com/bleacher\_reacher\_mega\_2014.pdf`.

[84] UVision. Hero UAV, (Accessed on July 2019). URL `https://uvisionuav.com/main-products/`.

[85] Jack Wang, D J Fleet, and A Hertzmann. Gaussian process dynamical systems. In *Proc. Neural Information Processing Systems*, December 2006.

[86] Matthew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015. ISSN 14321467. doi: 10.1007/s00332-015-9258-5.