POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Koopman-based Modeling for Rocket Landing

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Filippo Maria Cataldo**

Student ID: 217822
Advisor: Prof. Francesco Topputo
Co-advisor: Dr. Marco Sagliano
Academic Year: 2024-25

Abstract

The rocket landing problem has been one of the main interests of the space sector in the last decades and, more recently, the need for a fast and efficient way to solve precisely the guidance problem has emerged as crucial for reusability purposes. In this context, Koopman Operator Theory is promising because of its ability to globally linearize an uncontrolled autonomous system, by lifting the state onto a set of observables. In general, the conditions to include the control in a linear way are quite restrictive, but a bilinear model can be obtained with almost no further approximations. Well-established methods employ a fixed dictionary of observables, but more recent studies show that it is beneficial, in terms of accuracy, to learn a dictionary through neural network optimization. In this work, the atmospheric, fuel-optimal, rocket landing problem is addressed with the aim of assessing if a Koopman model can be used to successfully solve the Optimal Control Problem and if it is computationally advantageous. The main contributions of this thesis are the extension of dictionary learning techniques to include control and the formulation of a new framework to estimate a bilinear control model, after obtaining the equivalent linear modeling of the free dynamics. It is shown that dictionary learning methods provide more accurate linear control models than standard methods. The resulting dynamical model is used to build a Linear Program that can be solved efficiently; on the other hand, when a Koopman bilinear model is used, a solution can be found, but with poor computational efficiency.

Keywords: rocket landing, optimal control, Koopman theory, deep learning

Sommario

Il problema dell'atterraggio dei razzi è stato uno degli interessi principali del settore spaziale negli ultimi decenni e, più recentemente, è emersa come cruciale la necessità di risolvere in modo preciso, veloce ed efficiente il problema di guida, soprattutto in vista della riutilizzabilità degli stessi. In questo contesto, la Teoria degli Operatori di Koopman si rivela promettente per la sua capacità di linearizzare globalmente un sistema autonomo non controllato, utilizzando un insieme di osservabili dello stato. In generale, le ipotesi per includere il controllo in modo lineare sono piuttosto restrittive, ma un modello bilineare può essere costruito senza particolari approssimazioni. Metodi ben consolidati utilizzano un dizionario di osservabili statico, ma studi più recenti mostrano quanto sia utile, in termini di accuratezza, apprendere un dizionario attraverso l'ottimizzazione di una rete neurale. In questo lavoro viene affrontato il problema di atterraggio atmosferico con utilizzo ottimale di combustibile, con l'obiettivo di stabilire se un modello di Koopman possa essere usato per risolvere con successo il Problema di Controllo Ottimo e se possa essere vantaggioso dal punto di vista computazionale. I contributi principali di questa tesi sono l'estensione dei metodi con apprendimento del dizionario a modelli che includano il controllo e la formulazione di una nuova procedura per stimare un modello bilineare con controllo, dopo aver ottenuto l'equivalente modello lineare della dinamica libera. Si dimostra che i metodi con apprendimento del dizionario sono in grado di fornire modelli lineari che includano il controllo con maggiore accuratezza dei metodi tradizionali. Il modello dinamico che ne risulta viene usato per costruire un Programma Lineare che può essere risolto in modo efficiente; d'altra parte, quando viene utilizzato un modello di Koopman bilineare, una soluzione può essere ottenuta, ma con scarsa efficienza computazionale.

Parole chiave: atterraggio dei razzi, controllo ottimo, teoria di Koopman, apprendimento profondo

Contents

Abstract **i**

Sommario **iii**

Contents **v**

Introduction **1**

1 Literature Survey **5**
1.1 Optimal Control . . . 5
1.1.1 Pontryagin's Minimum Principle . . . 6
1.1.2 Two-Point Boundary-Value Problem . . . 6
1.1.3 Direct and Indirect Methods . . . 6
1.2 Optimization . . . 8
1.2.1 Karush-Kuhn-Tucker Conditions . . . 8
1.2.2 Interior Point Methods . . . 9
1.3 Rocket Landing . . . 10
1.3.1 1D Example . . . 13

2 Koopman Theory **17**
2.1 Preliminaries . . . 17
2.1.1 Uncontrolled Systems . . . 17
2.1.2 Controlled Systems . . . 19
2.2 Approaches . . . 22
2.2.1 Galerkin Method . . . 22
2.2.2 Extended Dynamic Mode Decomposition . . . 23
2.2.3 Dictionary Learning . . . 26
2.2.4 Comparison . . . 29
2.2.5 Truncated Singular Value Decomposition . . . 29

2.2.6 Other Methods . . . 31
2.3 Motivating Examples . . . 32
2.3.1 Closed System . . . 33
2.3.2 Asymptotically Stable Duffing Oscillator . . . 37
2.3.3 Stable Duffing Oscillator . . . 39
2.3.4 Non-polynomial System . . . 40

3 Rocket Landing **47**
3.1 Environment Modeling . . . 47
3.2 Equations of Motion . . . 51
3.2.1 1D Model with Aerodynamics . . . 52
3.2.2 2D Model with Aerodynamics . . . 53
3.2.3 3D Model without Aerodynamics . . . 54
3.3 Benchmark Rocket Landing Formulation . . . 54
3.4 Koopman-based Rocket Landing Formulation . . . 56
3.4.1 Koopman-based Linear Program . . . 58

4 Numerical Simulations **61**
4.1 Koopman Model . . . 61
4.1.1 1D Model with Aerodynamics . . . 62
4.1.2 2D Model with Aerodynamics . . . 69
4.1.3 3D Model without Aerodynamics . . . 78
4.2 Optimal Control . . . 82
4.2.1 1D Model with Aerodynamics . . . 83
4.2.2 2D Model with Aerodynamics . . . 86
4.2.3 3D Model without Aerodynamics . . . 89

5 Conclusions **93**
5.1 Lessons Learned . . . 93
5.2 Future Directions . . . 95

Bibliography **97**

A Neural Network Jacobian Matrix **103**

B Expression of the Aerodynamic Angles **105**

List of Figures **107**

List of Tables **109**

List of Symbols **111**

Acknowledgements **115**

Alla mia famiglia.
Al Prof. Moretti.

Introduction

Che fai tu, luna, in ciel? dimmi, che fai,
Silenziosa luna?

Giacomo Leopardi

Extensive research is undergoing in the space sector and, among all, rocket landing is one of the most popular branches due to the interest in reusable rockets, which are cost-saving and can allow many more missions to be carried out. The most famous example is probably SpaceX's Falcon 9 which has completed 445 landings up to this day.
The presence of nonlinearities in the dynamics makes it very difficult to solve the guidance problem in a fast and efficient way, and some simplifications are usually made to treat the problem. On the other hand, a high-fidelity environment is needed to have a robust guidance, which takes the uncertainties into account. For example, the study of endoatmospheric flights poses new challenges due to the nonlinearities introduced by the aerodynamic forces, which are however needed for a proper environment modeling. Numerical methods to solve the guidance problem onboard are sought and, to this end, new methodologies should be explored in order to reformulate the problem in a simpler way without approximations.
Koopman Operator Theory (KOT) [1] has gained strong interest in recent years due to its ability to globally linearize an uncontrolled autonomous system by lifting the state onto a set of observables [2]. Although the resulting system is, in general, infinite-dimensional, numerous works have proven the efficacy of finite approximations. Servadio et Al. [3] have shown the possibility to use Galerkin method with Legendre polynomials to analytically obtain a Koopman model, when the original system has only polynomial nonlinearities. Williams et Al. [4] introduced Extended Dynamic Mode Decomposition (EDMD), a data-driven method to get the best approximation of the Koopman operator, given a fixed dictionary of functions. Both methods have been successfully applied to simple problems such as the Duffing oscillator. Hofmann et Al. [5] extended the study to Keplerian motion, with J2 perturbation and atmospheric drag.
A fixed dictionary of functions might result in spurious eigenfunctions and eigenvalues

pairs for the system. More recent studies show that it is convenient to learn a dictionary of functions using neural networks, thereby obtaining an accurate spectrum. Li et Al. [6] modified EDMD by introducing a dictionary optimization step: the resulting system behaves better, with fewer basis functions.
When KOT is applied to controlled systems, the resulting model has a nonlinear dependence with respect to the control. However, if the original system is control-affine, a Koopman bilinear model can approximate the dynamics with a proper set of functions [7]. As shown in [8] the conditions for a linear control model to be obtainable are quite restrictive. Although many works only managed to use Koopman theory in a Model Predictive Control framework (for example [9–11]), Hofmann et Al. [12] applied it to a space flight, low-thrust Optimal Control Problem (OCP) with endpoint constraints, using Galerkin method.
The main contribution of this work is the application of KOT to OCPs in the context of rocket landing. The endoatmospheric, fuel-optimal, rocket landing problem is addressed. A dictionary learning method is explored and extended to include control. Moreover, a slight modification to EDMD-based methods is proposed to obtain a bilinear control model, after computing the free-dynamics part. The first objective is to prove that KOT can be used to achieve a solution to the OCP; secondly, it is assessed whether this approach can yield a performance improvement with respect to standard methods.
The thesis is organized as follows: chapter 1 is a summary about optimal control theory, optimization techniques and rocket landing; in chapter 2 KOT is explained and the approaches that are used in the remainder of the work are explored; in chapter 3 the mission scenario is developed and the OCP is stated; chapter 4 shows the numerical results; chapter 5 is a summary of the main results obtained and of some future directions that can be followed.

Notation

A part of the mathematical notation used throughout the text is specified hereafter. In general, vectors are denoted with lowercase bold symbols, matrices with uppercase plain symbols, scalars with generic plain symbols. Note that scalar quantities can be indicated with either lowercase or uppercase symbols, but they can be distinguished from matrices based on the contexts and the definitions. Matrix elements are denoted with subscripts that indicate the index in the different dimensions; for example, A_{ij} is the element of matrix A in row i and column j. Vector components can be specified in a similar way, dropping the boldface notation. When used in matricial operations, vectors are intended as column matrices.

The i-th basis vector of reference frame j is denoted as $\hat{\mathbf{j}}_i$; its k-th component is indicated as $\hat{j}_{ik}$. Vectors that depend on a reference frame have a subscript that indicates it. For example, the i-th component of vector $\mathbf{a}$ in reference frame j is indicated as $a_{i,j}$.
The gradient of a function must be intended as its partial derivatives with respect to all the variables on which it depends, unless differently specified, with a subscript after ∇. In the case of vector-valued functions, the Jacobian matrix is formed, instead. The Jacobian matrix of a generic vector function $\mathbf{a}(\mathbf{x}) \in \mathbb{R}^N$, that depends on the state $\mathbf{x} \in \mathbb{R}^n$, is defined as follows:

$$\nabla \mathbf{a}(\mathbf{x}) = \begin{bmatrix} \frac{\partial a_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial a_1(\mathbf{x})}{\partial x_i} & \cdots & \frac{\partial a_1(\mathbf{x})}{\partial x_n} \\ \vdots & & & & \vdots \\ \frac{\partial a_j(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial a_j(\mathbf{x})}{\partial x_i} & \cdots & \frac{\partial a_j(\mathbf{x})}{\partial x_n} \\ \vdots & & & & \vdots \\ \frac{\partial a_N(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial a_N(\mathbf{x})}{\partial x_i} & \cdots & \frac{\partial a_N(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

The gradient of scalar-valued functions can be obtained as a particular case of the Jacobian matrix.

1 | Literature Survey

This chapter is intended as a summary about the state of the art in optimal control theory, optimization techniques and rocket landing approaches.

1.1. Optimal Control

A OCP is a problem that maps functions (paths) into a scalar to be minimized [13]. This kind of problem was first introduced by Johann Bernoulli in 1696, when posing the brachistochrone problem [14]. It involves a point mass moving on a plane, subject to a uniform gravity field, along a frictionless track; the objective of the problem is to find the shape of the track that minimizes the time of arrival of the particle to the desired point. Although it could seem counterintuitive, a straight line track is not the fastest solution: a new approach was sought. This led to the development of the Calculus of Variations, and it was found out that the solution of the problem obeyed to the Euler-Lagrange (EL) differential equations [13]. They can be derived by considering one parameter that represents a small deviation from the optimal solution; the equations are then obtained by imposing that the solution of the problem is a minimum when the variation is zero.
The EL equations, however, can solve a problem which only involves a path: to take into account variations in a control input $\mathbf{u}$ and in the final time (which might not be fixed), the EL theorem is needed, which states the necessary conditions of the solution of the problem. The theorem introduces a costate vector $\boldsymbol{\lambda}$ which varies with time, and another multiplier vector $\boldsymbol{\mu}^{\boldsymbol{f}}$ associated with the endpoint constraints[1]; the optimal solution involves some conditions on the dynamics of the costate, as well as on their value at final time, and, most importantly, that $\partial H/\partial \mathbf{u} = \mathbf{0}$, where the Hamiltonian H is a scalar function that depends on the variables. Additionally, when the final time is free, a transversality condition holds. The costate vector can also be interpreted as a sensitivity of the cost function to a change in the associated state variable; in this way, it can be investigated how a small change in the state from the optimal solution affects the cost value [13]. The EL theorem takes only problems with unbounded and continuous control into account, therefore, it is not

[1] when other path constraints are present, other multipliers are needed.

general, as very often a practical application might involve a limited amount of control or different classes of functions. For instance, piecewise-constant functions are common in aerospace applications where the engine power is either at the minimum or maximum level (bang-bang control).

1.1.1. Pontryagin's Minimum Principle

The necessity to generalize the EL theorem led to the formulation of Pontryagin's Minimum Principle (PMP) [15], which is valid for a generic control space. Despite the proof of the principle being very long and complex, it can be summarized with the following statement: "The Hamiltonian must be minimized over the set of all admissible controls" [16]. It is emphasized that the optimal control is a minimum for the Hamiltonian at each time instant, but only when it is evaluated on the optimal path of the state, i.e. it is not a minimum for an arbitrary path. The main difference with respect to the EL theorem is that the minimum of the Hamiltonian is no longer evaluated with its partial derivative with respect to the control, since it is now bounded and, therefore, the minimum could be at the edges of the admissible set.

1.1.2. Two-Point Boundary-Value Problem

Once that the necessary conditions are developed, a Two-Point Boundary-Value Problem (TPBVP) is obtained, where the dynamics of the state and costate is known and subject to the control, which is also known to minimize the Hamiltonian. However, not all the initial conditions are specified. In particular both the initial and final state might not be completely constrained: the costate variable associated with a state not constrained at initial time has to be zero at initial conditions; the costate associated with a state not constrained at final time has to be either zero or dependent on the terminal constraints. For this reason, the optimal solution cannot be simply attained by propagating the initial conditions, but requires more care to ensure that all the conditions at the boundaries are satisfied. This is not a straightforward task and very rarely a solution can be expressed analytically; most of the time numerical methods must be employed.

1.1.3. Direct and Indirect Methods

After having outlined the theory behind OCPs, the various approaches to solve them can be discussed. They can be classified as direct or indirect methods. A direct method aims at directly finding the minimum of the problem, by evaluating the cost function at different values of the variables; a search direction must be established such that new

iterates get closer to the solution. An indirect methods aims at solving PMP's necessary conditions, by finding the root of the set of equations [17].
There are many reasons why using a direct method is often a better choice then using an indirect one. Direct methods do not require the explicit formulation of the necessary conditions, which may be more or less difficult, based on the cost function, the dynamics and the constraints involved in the problem; for this reason, an indirect method cannot be a priori formulated, but it needs experience of the user, while a direct method can be generalized to any OCP and used as a blackbox function. Moreover, the presence of inequality constraints introduces the need to estimate when they are active or not, increasing the number of variables [18]. Lastly, an indirect method needs an initial guess not only of the physical variables (i.e. the state and the control), but also of the costate and the multipliers, which usually carry no physical meaning, thereby increasing the difficulty in providing a good initial guess; however, even with a reasonable starting point, the solution is often very sensitive to small changes in the guess. In the context of rocket landing, examples of direct methods can be found in [19–22], whereas [23] is an example of indirect method.
Independently from the method, the continuous-time functions of the original problem must be converted into a finite number of parameters, i.e. the OCP is *transcribed* into a finite-dimensional problem [17]. There are some techniques that can be applied to formulate the new problem. The Simple Shooting method assesses whether the constraints are satisfied or not, after propagating the state initial conditions, and changes the variables (the initial state and the algebraic variables such as the control and the final time) accordingly [17]. For instance, it can be used to solve a TPBVP where the initial conditions must be modified such that the final constraints are satisfied. A small change in the variables can, however, have a big impact on the constraints and, for this reason, this approach can only be used in simple problems, but the idea can be extended by dividing the path into more segments: the initial condition of each segment is propagated until the beginning of the next one. In this method, called Multiple Shooting, the variables include the state at each discretization point, and new constraints are imposed to ensure continuity between the segments (defect constraints) [17]. Even if this method increases the dimension and complexity of the problem (more optimization variables and more constraints), it reduces its numerical complexity. For example, the Jacobian matrix associated with the defect constraints is very sparse, since a perturbation in the state at one point, only affects the neighboring variables and constraints. Another problem arises when analytic propagation cannot be performed; this requires numerical integration and collocation methods can be applied: the solution is approximated by a function, e.g. a polynomial or a spline, and the integral is obtained with a quadrature formula.

1.2. Optimization

After following a direct approach and transcribing the problem into a finite-dimensional one, a parameter optimization problem (program) is obtained; in general, the equations are nonlinear, such that it is a Nonlinear Program (NLP). In the next paragraphs, a theoretical formulation of the problem is given and, subsequently, Interior Point methods are presented as tools to solve it.
The NLP is characterized by the optimization variables, the cost function, the equality and inequality constraints. The constraints are taken into account by adding them to the cost function to create the Lagrangian, after multiplication by a new set of variables, called Lagrange multipliers or dual variables; the ones associated with inequality constraints are greater or equal to zero. Consequently, the dual function can be defined as the minimum value of the Lagrangian over any feasible set of optimization variables (so it depends only on the dual variables); a very important property of the dual function is that it provides a lower bound on the optimal value of the cost function [24]. For this reason, one would like to know which is the *best* lower bound, i.e. the value of the dual function which is closest to the solution of the optimal problem. This leads to the introduction of the dual problem: the maximization of the dual function, subject to the constraint that the dual variables associated with inequalities are greater or equal to zero. Likewise, the original problem can be called primal problem. For a generic problem, weak duality holds, that is, the optimal duality gap (the difference between the optimal cost functions of the primal and dual problem) is not zero; in some cases, strong duality holds and the optimal duality gap is zero. There are many qualifications that can be used to assess strong duality and one of the most common is represented by Slater's condition [24]. The dual problem is also important because a dual feasible point gives raise to a certificate of suboptimality: after denoting with ε_{opt} the difference between a primal feasible point and a dual feasible point, the difference between the same primal point and the optimal primal point is less than ε_{opt}, such that the point can be called ε_{opt}-suboptimal. This certificate can be used in optimization algorithms as a stopping criteria.

1.2.1. Karush-Kuhn-Tucker Conditions

Given a problem with differentiable cost function and constraints equations, and for which strong duality holds, the Karush-Kuhn-Tucker (KKT) necessary conditions can be defined for the optimal primal and dual points. They comprise the feasibility of the point (i.e. the constraints must be satisfied), the complementary slackness (either a primal inequality or the associated dual variable is equal to zero) and the gradient of the Lagrangian function

being equal to zero. In the case of a convex problem, the conditions are also sufficient, and they become very appealing to be used in optimization algorithms. A problem is defined as convex if the equality constraints are linear (including the dynamics) and the inequalities outline a convex feasibility region. Convexity is a more general property than linearity, allowing efficient optimization to be performed for a broader variety of problems [24]. A very popular example of convex problems, especially in rocket landing applications, is represented by problems with Second Order Cone inequalities; these kinds of problem are referred to as Second Order Cone Programs (SOCPs). See some examples in [25–27].

1.2.2. Interior Point Methods

Many algorithms aim at solving the KKT equations (or a modified version) in the set of variables that includes the primal and the dual ones. Among them, Interior Point methods, and in particular Primal-Dual methods, are very popular [28]. The first idea behind them is to apply Newton's method, modifying the search step such that the inequality constraints are strictly satisfied (from which, interior point). This is done to avoid infeasible points during the search: for example, starting from a point on the boundary, the new Newton step is likely to *push* outside the feasibility region. The second idea of the method is to keep iterations from being close to the boundary. Practically, the algorithm is built by introducing a bias term τ_b in the complementary slackness condition. This leads to the definition of central path, i.e. the set of all strictly feasible points for different values of τ_b. The bias term is modified through the process such that successive iterations converge to the solution for $\tau_b = 0$ (almost) through the central path. The selection of the best bias parameter is still an active research area, and the algorithms often rely on some merit function.

It must be pointed out that, for a generic nonlinear problem, the method has no guarantee to converge to the global optimum and, depending on the provided initial guess, it may converge to different local optimal solutions. This is due to the fact that, as explained before, KKT conditions are only necessary. However, there is a big interest in reformulating the problem as a convex one since in that case there are many well-established algorithms with guarantee of convergence to a global optimum, within a certain amount of time depending on the size of the problem. These features make convex OCPs very appealing for autonomous guidance and onboard control.

1.3. Rocket Landing

In this section, a survey about the evolution of spacecraft and rocket landing is given. Furthermore, the simple case of 1D landing is considered and the necessary conditions of the associated OCP are developed analytically; despite its simplicity, it gives some results which are useful for comparison with the results of this work.
The first-generation space missions were typically of exploratory nature and ensuring precise landing was not a critical requirement; however, a minimum-fuel approach was needed because of the limited resources available, and the problem was faced with different approximations. Early strategies employed in landing applications represent the so-called Apollo guidance [29]. It was first developed for landing on the Moon's surface and it is characterized by three distinct phases: a fuel-optimal phase before preparing for landing, a transition phase, the final translation and touchdown. The solution can be obtained by using polynomial basis functions [30]. However, the last phase does not optimize anything, and it just ensures that the spacecraft correctly lands in the desired position; moreover, it only considers vacuum flight, which is a good assumption for Moon landing, but not extensible to applications in which aerodynamic forces must be considered to reduce the uncertainties or points of failure; this is not a strategy adequate to current standards or needs. In a similar fashion, the gravity-turn guidance has been addressed, also taking the aerodynamic drag force into account [31]. A gravity-turn trajectory is defined as a path in which the only force not aligned with the velocity (and therefore the only force which keeps the spacecraft from following a straight line) is the gravity force; this means that the thrust force, for example, is always collinear with the velocity; the drag force, instead, acts by definition oppositely to the velocity. It has been shown, from simulations, that, in some way, the fuel consumption is reduced with respect to the Apollo guidance, since the thrust always contrasts the velocity [32]. On the other hand, the solution often has a very large terminal horizontal velocity (in the order of 500 m/s), thereby undermining the feasibility of the solution; a workaround to this issue is to start the maneuver earlier, but this is not always a possibility and it shows that the method is not applicable to every initial condition. The Viking mission used for its guidance two reference trajectories from gravity-turn optimization: one with initial high velocity and one with low velocity. The algorithm interpolated between the two references depending on the initial conditions of the spacecraft [33].
In subsequent missions, the need for precise landing arose as a fundamental requirement. In fact, landing safety due to uncertainties in the control may preclude interesting parts of the planets and impede new scientific discoveries [34]. In this context, the problem of pinpoint landing was introduced as the landing within 100 m from the desired target

[35]; it requires more robust algorithms and the modeling of a high-fidelity environment (for example aerodynamic effects for planets with an atmosphere). As explained in the previous sections, the OCP is transformed into a NLP through transcription and collocation. A popular strategy is the application of pseudo-spectral methods, in which the collocation points are usually the roots of a Legendre polynomial or a linear combination of a Legendre polynomial and its derivative [36], to have quasi-spectral convergence properties. They were used in [32] to design the transition phase with a high-low-high thrust profile: initially the throttle is at maximum level to increase the horizontal velocity and move the spacecraft above the desired position; then the throttle is kept low to allow the spacecraft to move; finally, the high thrust is used to make the horizontal velocity null. However, the terminal descent phase was not designed. In [37] a class of efficient pseudo-spectral approximations was presented, extending the method to non-smooth solutions (not always suitable for these techniques) by introducing hard and soft collocation points. Other works involved the development of suboptimal solutions, using polynomial basis functions [38]. Another approach is to obtain an analytic closed-form solution without imposing the minimum-fuel cost function [39] or using a cost function related to it [40]. In any case, either the problem is not fuel-optimal or it is formulated as a NLP for which there is no guarantee of optimality and, more importantly, the number of iterations of the algorithms cannot be a priori bounded.

Recent years have been characterized by a great interest towards rocket reusability and fuel saving, which would allow many more missions to be carried out and costs to be cut: it becomes more evident that a precise optimal solution must be found in an efficient way, for example through convex optimization. Among the common sources of non-convexity in rocket landing applications, there are the non-zero bounds on the thrust magnitude; in fact, a lower limit different from zero results in a non-convex feasibility region. This is a common constraint since once the engine has been turned on, for safety reasons, it cannot be completely switched off until touchdown. In this context, a big shift was introduced by Açıkmeşe et Ploen [25] that proposed a lossless convexification of the 3D minimum-fuel problem; they considered the descent phase towards a surface-fixed target, after the parachute phase (used to slow down the rocket), with uniform gravity and no atmospheric effects (due to the low speed during this phase). The coupling between the translational and attitude guidance was not considered since it makes the problem much more complex. In general, with non-convex constraints, there are two possibilities: 1) restrict the set of feasible solutions to a convex subset of the original set or 2) relax the set of feasible solutions to a convex set containing the original set [41]. In the first case, a feasible solution is produced, but the cost might be higher than the original one; in the second case, a lower cost may be produced, but the solution could be infeasible with

respect to the orignal set. The second approach was followed, reformulating the problem in a SOCP, through the introduction of a slack variable. However, it was proven that the solution to the new problem is not suboptimal with respect to the original one, but it is exactly the same, hence, *lossless*. This is an exciting result and it represented a great step towards real-time guidance: using ad hoc algorithms for SOCP, after discretization of the problem and collocation with piecewise-constant functions, a solution can be obtained in an efficient way. Other constraints can be imposed with this method: for example, a mission could require a minimum altitude angle or a maximum speed. In particular, in [41] the authors imposed thrust pointing constraints, since the rocket might require a certain attitude for navigation/sensors purposes; these constraints are non-convex for a certain range of angles, but through relaxation of the problem, they can be convexified in a lossless way. In [42], this same approach was extended to infeasible trajectories, by minimizing the landing position error when the initial conditions do not allow the rocket to land in the desired position.

In [26] convex optimization was used in conjunction with pseudo-spectral transcription to make the resulting finite-dimensional problem smaller, with the same accuracy, extending pseudo-spectral methods outside of generic NLP contexts in which they are usually employed. To avoid dealing with free final time problems, that are non-convex, in [43] the altitude was used instead of the time as the independent variable of the problem; this also simplifies the expression of constraints and bounds that are altitude-dependent, such as glide-slope or thrust direction constraints.

In [25] the aerodynamic forces were neglected, but they must be considered in order to have a trajectory more robust with respect to uncertainties; this introduces nonlinearities (thus non-convexities) in the dynamics. When relaxation techniques are not enough to make the problem convex, some approximations must be made. For example, the problem may be successively linearized and Quadratic Programs (QPs) can be solved sequentially; however this approach may yield only a local minimum and can be characterized by slow convergence [44]. In their work, Liu et Al. circumvented this issue by considering the highly nonlinear entry problem with aerodynamic and propulsive forces and by reformulating the control such that the problem is relaxed in a lossless way; afterwards, Successive Convex Programming (SCP) can converge very fast to a solution. In [45], the problem was rewritten such to minimize the presence of non-convexities, and only specific terms were linearized.

1.3.1. 1D Example

Consider the terminal descent phase in the 1D vacuum flight. The aim is to minimize the fuel consumption while reaching the target with zero speed. The equations of motion are:

$$\begin{cases} \dfrac{d}{dt}r(t) = v(t) \\ \dfrac{d}{dt}v(t) = -g + \dfrac{\Gamma(t)}{m(t)} \\ \dfrac{d}{dt}m(t) = -\dfrac{\Gamma(t)}{V_{ex}} \end{cases}$$

where r, v and m are, respectively, the position and velocity with respect to the target and the mass of the rocket; Γ denotes the thrust which has an upper limit Γ_{max}, g the gravity acceleration, V_{ex} the exhaust velocity associated with fuel consumption. The mass variation during the flight is strictly monotonic with the time of flight [30]:

$$m_0 - m_f = m_0 \left(1 - e^{(v_0 - g t_f)/V_{ex}}\right)$$

For this reason the problem can be reformulated as a time of flight optimization one:

$$\text{find} \quad \min_{\Gamma(t), t_f} t_f \text{ s.t.} \begin{cases} \dfrac{d}{dt}r(t) = v(t) \\ \dfrac{d}{dt}v(t) = -g + \dfrac{\Gamma(t)}{m(t)} \\ \dfrac{d}{dt}m(t) = -\dfrac{\Gamma(t)}{V_{ex}} \\ r(0) = r_0 \\ v(0) = v_0 \\ m(0) = m_0 \\ r(t_f) = r_f = 0 \\ v(t_f) = v_f = 0 \\ 0 \leq \Gamma(t) \leq \Gamma_{max} \end{cases}$$

The Hamiltonian and endpoint functions can be built:

$$H = \lambda_r v + \lambda_v \left(-g + \frac{\Gamma}{m}\right) - \lambda_m \frac{\Gamma}{V_{ex}}$$
$$\Upsilon = \lambda_0 t_f + \mu_r^f r_f + \mu_v^f v_f$$

Consequently, the necessary conditions are derived:

$$\begin{aligned}
\frac{d}{dt}\lambda_r(t) &= 0 && \text{with} \quad \lambda_{r,f} &= \mu_r^f \\
\frac{d}{dt}\lambda_v(t) &= -\lambda_r(t) && \text{with} \quad \lambda_{v,f} &= \mu_v^f \\
\frac{d}{dt}\lambda_m(t) &= \lambda_v(t)\frac{\Gamma(t)}{m(t)^2} && \text{with} \quad \lambda_{m,f} &= 0 \\
\frac{d}{dt}H(t) &= 0 && \text{with} \quad H_f &= -\lambda_0
\end{aligned}$$

It follows that both λ_r and H are constant, while λ_v varies linearly in time, at most. By applying PMP to the problem, considering the bounds on the thrust, the following result is obtained:

$$\Gamma = \begin{cases} 0 & \text{if} \quad \lambda_v - m\lambda_m/V_{ex} > 0 \\ \Gamma_{max} & \text{if} \quad \lambda_v - m\lambda_m/V_{ex} < 0 \\ \text{singular} & \text{if} \quad \lambda_v - m\lambda_m/V_{ex} = 0 \end{cases}$$

This means that the switching function $S_w = \lambda_v - m\lambda_m/V_{ex}$ determines whether the thrust is at the maximum or minimum level, depending on its sign. This is typical of problems in which the Hamiltonian depends linearly on the control; it is also true that, in these cases, singular arcs might be present, when the switching function is zero on a set of positive measure (i.e. when it is zero at an uncountable number of time instants). Suppose that this is the case: the switching function is continuous and, starting from a point in time $\bar{t}$ on one of the sets of positive measure, its expression can be obtained by integration of its time derivative:

$$S_w(t) = S_w(\bar{t}) + \int_{\bar{t}}^{t} -\lambda_r - S_w(\tau)\frac{\Gamma(\tau)}{m(\tau)V_{ex}}d\tau$$

It can be proven that, with the already mentioned necessary conditions, all the zeros of the switching function, if present, are isolated [30]; therefore, no singular arcs are present and the control is either at the maximum or minimum level (bang-bang).
Now suppose that t_1 and t_2 are two consecutive zeros of the switching function. From continuity and nonsingularity, it follows that S_w is different from zero with constant sign in (t_1, t_2). Two cases are distinguished, in this interval [30].

1. If $S_w < 0$, then $\Gamma = 0$. Consequently, $dS_w(t)/dt = -\lambda_r(t)$, which means that S_w is monotonic in the interval. This is impossible because otherwise S could not be zero

both in t_1 and t_2.

2. If $S_w > 0$, then $\Gamma = \Gamma_{max}$. Hence, $dS_w(t)/dt = -\lambda_r(t) + |S_w(t)|\Gamma_{max}/(V_{ex}m(t))$. Since the second term is strictly positive, for $dS_w(t)/dt$ not to be monotonic, $\lambda_r < 0$. However, in a sufficiently small neighborhood of t_2, where $S_w = 0$, $dS_w(t)/dt < 0$. This is impossible since the switching function is negative and decreasing near t_2, where it should be zero.

This implies that there is at most one control switch in the optimal solution. Moreover, the final thrust level must be maximum for optimality reasons [46]: it follows that, during flight, the thrust is always maximum, or starts from zero and switches once before touchdown.

2 | Koopman Theory

In this chapter, the Koopman operator theory (KOT) is described: section 2.1 sets out the theoretical background; in section 2.2 various practical methods to apply the theory are explored; in section 2.3 some simple examples show the advantages and disadvantages of the theory and the different approaches.

2.1. Preliminaries

In 1931 B.O. Koopman developed an operator theory that can transform a generic non-linear system of differential equations into a linear one, by lifting the state onto a set of observables [1]. The original theory, applied to uncontrolled autonomous systems, is explained hereafter, together with its extension to controlled systems.

2.1.1. Uncontrolled Systems

Consider the following system with time variable $t \in \mathbb{R}$:

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t))$$

where $\mathbf{x} \in \mathbb{R}^n$ is the n-dimensional state[1], and $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ represents the continuous-time dynamics. Its discrete counterpart, for a given time step Δt, reads as:

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k)$$

where $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ is the discrete-time dynamics. The Koopman discrete operator $\mathcal{K}^d$, or Koopman operator, advances observables $\xi : \mathbb{R}^n \to \mathbb{R}$ belonging to a Hilbert space [47]:

$$\mathcal{K}^d \xi = \xi \circ \mathbf{F}$$

[1] The theory is generalizable to other n-dimensional state spaces without much effort.

If the continuous-time dynamics is smooth, the Koopman infinitesimal generator $\mathcal{K}$, or simply Koopman generator, can also be defined [8]:

$$\frac{d}{dt}\xi(\mathbf{x}(t)) = \mathcal{K}\xi(\mathbf{x}(t)) = \nabla\xi(\mathbf{x}(t)) \cdot \mathbf{f}(\mathbf{x}(t))$$

The Koopman operator and generator are related through an exponential relation: $\mathcal{K}^d = e^{\mathcal{K}\Delta t}$ [8]. A vector subspace Ψ is called Koopman-invariant if $\mathcal{K}\Psi \in \Psi$. Although a Koopman-invariant subspace would allow a linear dynamical system to be obtained, it must be stressed that such a subspace is, in general, infinite-dimensional; nevertheless, a finite Koopman-invariant subspace can be spanned by any set of eigenfunctions $\{\varphi_1, \varphi_2, \ldots, \varphi_q\}$ [47], where $\varphi_i(\mathbf{x}) : \mathbb{R}^n \to \mathbb{C}$ is an eigenfunction of the Koopman operator, defined as follows:

$$\frac{d}{dt}\varphi_i(\mathbf{x}(t)) = \Lambda_i\varphi_i(\mathbf{x}(t))$$

where Λ_i is a continuous-time eigenvalue. Discovering a set of eigenfunctions is one of the main challenges for practical applications of the Koopman operator. Furthermore, linear observables must often be included in the subspace to apply physical constraints or to optimize real variables, but, except for special cases, it is impossible to find a subspace spanned by a finite set of observables, including the state itself, that is Koopman-invariant. However, very often a finite approximation can be obtained and, with a proper choice of basis functions, the error decreases as the dimension increases [2]; on the other hand, a bad choice of the set of observables can impact the dimensionality of the Koopman system or even raise closure issues, as shown in section 2.3. By defining $\boldsymbol{\psi}(\mathbf{x}) = [\psi_1(\mathbf{x}), \ldots, \psi_q(\mathbf{x})]^T$ as the q-dimensional vector of basis functions that span a finite Koopman-invariant subspace, the Koopman operator can be expressed as a matrix $K \in \mathbb{R}^{q\times q}$.

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = K\boldsymbol{\psi}(\mathbf{x}(t)) \tag{2.1}$$

Exploiting the exponential relation between the Koopman operator and generator, the discrete and continuous matrices can be obtained interchangeably, with the matrix logarithm and exponential functions [48]:

$$K = \frac{\log K^d}{\Delta t} \tag{2.2}$$

$$K^d = e^{K\Delta t} \tag{2.3}$$

Eigendecomposition If K is diagonalizable, one could take advantage of the eigendecomposition of the system in eq. (2.1). In fact, defining the vector of eigenfunctions as $\boldsymbol{\varphi}(\mathbf{x}(t)) = V^{-1}\boldsymbol{\psi}(\mathbf{x}(t))$, where V is the matrix that contains the right eigenvectors of K in its columns, the evolution of the system is as follows:

$$\begin{cases} \boldsymbol{\varphi}(\mathbf{x}(t)) = e^{\Lambda}\boldsymbol{\varphi}(\mathbf{x}(t_0)) \\ \boldsymbol{\varphi}(\mathbf{x}(t_0)) = V^{-1}\boldsymbol{\psi}(\mathbf{x}(t_0)) \\ \boldsymbol{\psi}(\mathbf{x}(t)) = V\boldsymbol{\varphi}(\mathbf{x}(t)) \end{cases} \tag{2.4}$$

with Λ denoting the diagonal matrix that contains the eigenvalues of the system; V^{-1} is also the matrix of the left eigenvectors of K.

Reconstruction of the State It is often useful to obtain the evolution of a generic function of the state $\xi(\mathbf{x})$. This can be done by projecting the function onto the chosen subspace, through the projection matrix P.

$$\xi(\mathbf{x}) = P\boldsymbol{\psi}(\mathbf{x})$$

If $\xi(\mathbf{x})$ belongs to the subspace, it can be exactly reconstructed. In general P can be obtained differently based on the used approach (section 2.2); furthermore, it is often necessary to recover the original state and, if it is included in the basis functions, P can be obtained trivially.

2.1.2. Controlled Systems

Consider the following system:

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{2.5}$$

where $\mathbf{u} \in \mathbb{R}^m$ represents the control. In this case, the Koopman discrete operator and infinitesimal generator can be parametrized by the control, therefore, a rather general expression can be obtained [8].

$$\mathcal{K}^d_{\mathbf{u}}\psi = \psi \circ \mathbf{F}(\cdot, \mathbf{u})$$
$$\mathcal{K}_{\mathbf{u}}\psi = \nabla\psi \cdot \mathbf{f}(\cdot, \mathbf{u})$$

In this form, no advantage is gained from the use of KOT, as the resulting system is gen-

erally nonlinear with respect to the control. However, the expression could be simplified depending on the structure of the original dynamics.

Bilinear Form In the case of a control-affine dynamical system, the Koopman generator is also control-affine [8].

$$\mathbf{f}(\mathbf{x},\mathbf{u}) = \mathbf{f}_0(\mathbf{x}) + \sum_{i=1}^{N} \xi_i(\mathbf{u})\mathbf{f}_i(\mathbf{x}) \quad \Rightarrow \quad \mathcal{K}_{\mathbf{u}} = \mathcal{K}_0 + \sum_{i=1}^{N} \xi_i(\mathbf{u})\mathcal{K}_i \tag{2.6}$$

where ξ_i is a scalar-valued function of the control and $\mathcal{K}_i$ is the Koopman generator associated with the control-affine vector field $\mathbf{f}_i$. Consequently, for a finite Koopman-invariant subspace associated with the free-dynamics part, one could write:

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = K_0\boldsymbol{\psi}(\mathbf{x}(t)) + \sum_{i=1}^{N} \xi_i(\mathbf{u}(t))(\mathcal{K}_i\boldsymbol{\psi})(\mathbf{x}(t)) \tag{2.7}$$

If also $\mathcal{K}_i\boldsymbol{\psi}$ lies in the span of $\{\psi_1,\dots,\psi_q\}$, the system becomes as in eq. (2.8) [49]. This hypothesis might be difficult to satisfy and, if it is not, eq. (2.7) might contain nonlinearities even more complicated than the ones of the original system; nonetheless, projecting $\mathcal{K}_i\boldsymbol{\psi}$ on the chosen subspace could still result in a good approximation.

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = K_0\boldsymbol{\psi}(\mathbf{x}(t)) + \sum_{i=1}^{N} \xi_i(\mathbf{u}(t))K_i\boldsymbol{\psi}(\mathbf{x}(t)) \tag{2.8}$$

Furthermore, if $\xi_i \equiv u_i$, the system becomes bilinear:

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = K_0\boldsymbol{\psi}(\mathbf{x}(t)) + \sum_{i=1}^{m} u_i(t)K_i\boldsymbol{\psi}(\mathbf{x}(t)) \tag{2.9}$$

This hypothesis is easier to satisfy, as every dynamical system (eq. (2.5)) can be generalized to a control-affine system as in eq. (2.10) by inflating the control [50]. In this case, $\xi_i \equiv \bar{u}_i$ with $\bar{u}_i$ being the rate of change of the original control u_i.

$$\frac{d}{dt}\bar{\mathbf{x}}(t) = \bar{\mathbf{f}}(\bar{\mathbf{x}}(t)) + \begin{bmatrix} 0_{n\times m} \\ I_{m\times m} \end{bmatrix} \bar{\mathbf{u}}(t) \tag{2.10}$$

with

$$\bar{\mathbf{f}}(\bar{\mathbf{x}}) := \begin{bmatrix} \mathbf{f}(\bar{\mathbf{x}}) \\ 0_{m\times m} \end{bmatrix}, \quad \bar{\mathbf{x}} := \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix}$$

It must also be noted that, in general, it is possible to obtain a bilinear model only for the Koopman generator and not for the Koopman operator; in fact, the exponential relationship between the two makes the dependence on the control unclear in discrete-time, even if the model is bilinear in continuous-time.
Moreover, when using the bilinear form, there may be no advantage in performing the eigendecomposition, since the dynamics associated with the control might still be dense and the state cannot be evaluated with a simple relationship with time as in eq. (2.4).

Linear Form The most interesting possibility is to obtain a Linear Time-Invariant (LTI) system as in eq. (2.11), due to the possibility to apply optimization algorithms for Linear Programs (LPs).

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = K_0\boldsymbol{\psi}(\mathbf{x}(t)) + K_c\mathbf{u}(t) \tag{2.11}$$

The conditions for the resulting system to be LTI are that the components of $\boldsymbol{\psi}$ span an invariant subspace of the Koopman generator and that the original system is control-affine with [8]

$$\begin{cases} \dfrac{d}{dt}\mathbf{x}(t) = \mathbf{f}_0(\mathbf{x}(t)) + \displaystyle\sum_{i=1}^{m} u_i(t)\mathbf{f}_i(\mathbf{x}(t)) & \text{(2.12a)} \\ \nabla\psi_i \cdot \mathbf{f}_k(\cdot, \mathbf{u}) = const \quad \forall i \in \{1, \dots, n\}, \ \forall k \in \{1, \dots, q\} & \text{(2.12b)} \end{cases}$$

However, eq. (2.12b) is a quite restrictive condition and cannot be generalized to any dynamical system and any Koopman-invariant subspace; adding the necessity to reobtain the original state, either by including it in the subspace from the beginning or by projection, makes it very difficult to obtain a LTI system. Despite that, it is still worth trying to explore some practical approximations that can give accurate enough results.
The relation between continuous and discrete matrices can be obtained similarly as in the uncontrolled case [48], when the control is constant over a time step [48]:

$$\begin{bmatrix} K_0 & K_c \\ 0_{m\times q} & 0_{m\times m} \end{bmatrix} \Delta t = \log \begin{bmatrix} K_0^d & K_c^d \\ 0_{m\times q} & I_{m\times m} \end{bmatrix} \tag{2.13}$$

2.2. Approaches

In recent years, many different methods to apply KOT emerged, both analytic and data-driven. In this section, some of them are explored, showing their differences and strengths; some other methods are only mentioned, since they are not useful in the context of this work.

2.2.1. Galerkin Method

Galerkin method (GAL), widely used also in other branches of mathematics, consists in projecting the Koopman generator onto a subspace defined by orthonormal basis functions. The projection is obtained through the inner product operator $\langle \cdot, \cdot \rangle$ [3].

$$\langle \xi_1(\mathbf{x}), \xi_2(\mathbf{x}) \rangle = \int_\Omega \xi_1(\mathbf{x}) \xi_2(\mathbf{x}) w_\Omega(\mathbf{x}) d\mathbf{x}$$

where $w_\Omega(\mathbf{x})$ and Ω are, respectively, the weighting function and domain of the subspace. One common choice of basis functions are Legendre polynomials, with $\Omega = [-1, 1]^n$ and $w_\Omega(\mathbf{x}) = 1$ (when the polynomials are normalized), due to the possibility to compute the integrals in closed-form and iteratively if the original dynamics is polynomial [3]. It must be noted that the number q of multivariable Legendre polynomials of maximum degree deg, for a state of dimension n, is as follows, considering the combinations of the single-variable polynomials [2]:

$$q = \binom{deg + n}{n} = \frac{(deg + n)!}{deg! n!}$$

Therefore, for high-dimensional systems, the number of basis functions might be prohibitive. For a control-affine system as in eq. (2.6), the approximation of the Koopman matrices of eq. (2.8) can be computed, considering the dictionary functions $\boldsymbol{\psi}(\mathbf{x})$ [12].

$$\begin{cases} K_{0,jk} = \langle \nabla \psi_j(\mathbf{x}) \cdot \mathbf{f}_0(\mathbf{x}), \psi_k(\mathbf{x}) \rangle = \displaystyle\int_\Omega (\nabla \psi_j(\mathbf{x}) \cdot \mathbf{f}_0(\mathbf{x})) \psi_k(\mathbf{x}) w(\mathbf{x}) d\mathbf{x} \\ K_{i,jk} = \langle \nabla \psi_j(\mathbf{x}) \cdot \mathbf{f}_i(\mathbf{x}), \psi_k(\mathbf{x}) \rangle \end{cases} \tag{2.14}$$

The matrix P to reconstruct an observable vector $\boldsymbol{\xi}(\mathbf{x})$ can be built similarly [3].

$$P_{i,j} = \langle \xi_i(\mathbf{x}), \psi_j(\mathbf{x}) \rangle = \int_\Omega \xi_i(\mathbf{x}) \psi_j(\mathbf{x}) w_\Omega(\mathbf{x}) d\mathbf{x}$$

2.2.2. Extended Dynamic Mode Decomposition

In case there is no way to iteratively compute the integrals of eq. (2.14), or if there is no closed-form solution, data-driven methods must be used. This is the case, for example, when Legendre polynomials are used, but the original dynamics is not polynomial and it is not straightforward to compute the Koopman matrices through integration.
Extended Dynamic Mode Decomposition (EDMD) aims at obtaining the Koopman operator through minimization of the single-step prediction, in a convex fashion [4]. It requires:

- a data set in the form of snapshot pairs from trajectories $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{S}$;
- a dictionary of functions, represented by the vector-valued function $\boldsymbol{\psi}(\mathbf{x})$.

In particular, $\mathbf{y}_i := \mathbf{F}(\mathbf{x}_i)$, that is, $\mathbf{y}_i$ is the state after a single-step propagation in time. The points $\mathbf{x}_i$ are randomly sampled in the domain or can be obtained by propagating random initial conditions. It is also useful to define the matrices $\psi_\mathbf{x}$ and $\psi_\mathbf{y}$ that group all the data:

$$\psi_\mathbf{x} = \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}_1) & \boldsymbol{\psi}(\mathbf{x}_2) & \cdots & \boldsymbol{\psi}(\mathbf{x}_S) \end{bmatrix}, \qquad \psi_\mathbf{y} = \begin{bmatrix} \boldsymbol{\psi}(\mathbf{y}_1) & \boldsymbol{\psi}(\mathbf{y}_2) & \cdots & \boldsymbol{\psi}(\mathbf{y}_S) \end{bmatrix}$$

It must be noted that the method aims at approximating the Koopman operator, not the generator, and that the resulting matrix is valid for the time step Δt used to obtain the snapshot pairs. In particular, the objective function that the method minimizes is:

$$J = \frac{1}{2} \sum_{i=1}^{S} \left|\left| \boldsymbol{\psi}(\mathbf{y}_i) - K^d \boldsymbol{\psi}(\mathbf{x}_i) \right|\right|^2 \tag{2.15}$$

This is a least squares problem with the following solution:

$$\begin{cases} K^d = \left[A^\dagger B\right]^T \\ A := \dfrac{1}{S} \psi_\mathbf{x} \psi_\mathbf{x}^T \\ B := \dfrac{1}{S} \psi_\mathbf{x} \psi_\mathbf{y}^T \end{cases} \tag{2.16}$$

where $\dagger$ denotes the pseudo-inverse. The Koopman generator can then be computed according to eq. (2.2).
It has been proven that EDMD converges to GAL as the number of snapshot pairs tends to infinity, if they are uniformly sampled in the domain, and, therefore, the solution obtained

with EDMD is equivalent to the one obtained by approximating Galerkin's integrals with Monte Carlo integration [4]. Other methods can be used to approximate the integrals, in order to reduce the amount of data needed; for example, quadrature rules can be used for multivariate integrals: in this case, $\mathbf{x}_i$ are not randomly sampled, but they are precise points in the domain, based on the selected quadrature rule [5].
Matrix P to reconstruct the observables $\boldsymbol{\xi}(\mathbf{x})$ can also be obtained by minimizing the following cost function [51]:

$$J = \frac{1}{2}\sum_{i=1}^{S} ||\boldsymbol{\xi}(\mathbf{x}_i) - P\boldsymbol{\psi}(\mathbf{x}_i)||^2$$

After building $\xi_{\mathbf{x}} = [\boldsymbol{\xi}(\mathbf{x}_1), \boldsymbol{\xi}(\mathbf{x}_2), \dots, \boldsymbol{\xi}(\mathbf{x}_S)]$, the solution can be obtained similarly as before:

$$\begin{cases} P = \left[A^{\dagger}B\right]^T \\ A := \dfrac{1}{S}\psi_{\mathbf{x}}\psi_{\mathbf{x}}^T \\ B := \dfrac{1}{S}\psi_{\mathbf{x}}\xi_{\mathbf{x}}^T \end{cases}$$

In general, data does not have to be the same used to approximate the Koopman operator.

Extension to Include Control EDMD can be generalized in order to obtain models that include the control. The data snapshots can be redefined as $\{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{u}_i)\}_{i=1}^{S}$ with $\mathbf{y}_i := \mathbf{F}(\mathbf{x}_i, \mathbf{u}_i)$. An additional set of functions is defined:

$$\boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}), \mathbf{u}) = [\xi_1(\boldsymbol{\psi}(\mathbf{x}), \mathbf{u}), \dots, \xi_N(\boldsymbol{\psi}(\mathbf{x}), \mathbf{u})]^T$$

The most common choice would obviously be $\xi_i \equiv u_i$, in the prospective of a linear model. The aim is to obtain the discrete-time system in eq. (2.17). However, the conditions such that an exact model exists must be verified, even for a generic $\boldsymbol{\psi}$.

$$\boldsymbol{\psi}(\mathbf{x}_{k+1}) = K_0^d\boldsymbol{\psi}(\mathbf{x}_k) + K_{\xi}^d\boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_k), \mathbf{u}_k) = \begin{bmatrix} K_0^d & K_{\xi}^d \end{bmatrix} \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}_k) \\ \boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_k), \mathbf{u}_k) \end{bmatrix} = C \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}_k) \\ \boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_k), \mathbf{u}_k) \end{bmatrix} \tag{2.17}$$

where matrices K_0^d, K_ξ^d have been grouped in C.
After building the data matrix $\xi_{\mathbf{x},\mathbf{u}} = [\boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_1), \mathbf{u}_1), \dots, \boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_S), \mathbf{u}_S)]$, it is possible to minimize a modified version of the cost function in eq. (2.15) [8]:

$$J = \frac{1}{2} \sum_{i=1}^{S} \left|\left| \boldsymbol{\psi}(\mathbf{y}_i) - C \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}_k) \\ \boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_k), \mathbf{u}_k) \end{bmatrix} \right|\right|^2 \tag{2.18}$$

which can be minimized similarly as in eq. (2.16).

$$\begin{cases} C = \left[A^\dagger B\right]^T \\ A := \dfrac{1}{S} \begin{bmatrix} \psi_{\mathbf{x}} \\ \xi_{\mathbf{x},\mathbf{u}} \end{bmatrix} \begin{bmatrix} \psi_{\mathbf{x}} \\ \xi_{\mathbf{x},\mathbf{u}} \end{bmatrix}^T \\ B := \dfrac{1}{S} \begin{bmatrix} \psi_{\mathbf{x}} \\ \xi_{\mathbf{x},\mathbf{u}} \end{bmatrix} \psi_{\mathbf{y}}^T \end{cases} \tag{2.19}$$

Subsequently, K_0^d and K_ξ^d can be extracted from C according to eq. (2.17). There is no general way to transform the discrete-time system into a continuous one, unless $\xi_i \equiv u_i$. In that case the system becomes

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = K_0 \boldsymbol{\psi}(\mathbf{x}(t)) + K_c \mathbf{u}(t)$$

where the continuous matrices are obtained through eq. (2.13).

About the Bilinear Form One may be tempted to apply the method explained in the last paragraph using the functions $\boldsymbol{\psi}(\mathbf{x})u_i$, in order to obtain a bilinear model similar to eq. (2.9). However, as explained before, that model is valid only for a continuous-time system, therefore, some changes must be applied to the last method.
The Koopman control matrices $\{K_i\}_{i=1}^m$ can be found after having computed the free-dynamics part K_0, with a selected subspace. The cost function is modified such that residual is the one of the continuous-time system:

$$\begin{aligned} J &= \frac{1}{2} \sum_{i=1}^{S} \left|\left| \left[\dot{\boldsymbol{\psi}}(\mathbf{x}_i, \mathbf{u}_i) - K_0 \boldsymbol{\psi}(\mathbf{x}_i)\right] - \sum_{j=1}^{m} u_{j,i} K_j \boldsymbol{\psi}(\mathbf{x}_i) \right|\right|^2 = \\ &= \frac{1}{2} \sum_{i=1}^{S} \left|\left| \left[\dot{\boldsymbol{\psi}}(\mathbf{x}_i, \mathbf{u}_i) - K_0 \boldsymbol{\psi}(\mathbf{x}_i)\right] - C\boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_i), \mathbf{u}_i) \right|\right|^2 \end{aligned} \tag{2.20}$$

with

$$\dot{\boldsymbol{\psi}}(\mathbf{x}_i, \mathbf{u}_i) := \frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t))\Big|_{\mathbf{x}=\mathbf{x}_i, \mathbf{u}=\mathbf{u}_i}$$
$$C := [K_1^T, \dots, K_m^T]^T$$
$$\boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}), \mathbf{u}) := [\boldsymbol{\psi}^T(\mathbf{x})u_1, \dots, \boldsymbol{\psi}^T(\mathbf{x})u_m]^T$$

After building $\dot{\psi}_{\mathbf{x}} = [\dot{\boldsymbol{\psi}}(\mathbf{x}_1), \dots, \dot{\boldsymbol{\psi}}(\mathbf{x}_S)]$ and $\xi_{\mathbf{x},\mathbf{u}} = [\boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_1), \mathbf{u}_1), \dots, \boldsymbol{\xi}(\boldsymbol{\psi}(\mathbf{x}_S), \mathbf{u}_S)]$, the matrix C that minimizes eq. (2.20) can be computed.

$$\begin{cases} C = \left[A^\dagger B\right]^T \\ A := \dfrac{1}{S}\xi_{\mathbf{x},\mathbf{u}}\xi_{\mathbf{x},\mathbf{u}}^T \\ B := \dfrac{1}{S}\xi_{\mathbf{x},\mathbf{u}}\left[\dot{\psi}_{\mathbf{x}} - K_0\psi_{\mathbf{x}}\right]^T \end{cases} \tag{2.21}$$

It is pointed out that, in this case, the time derivative of the basis functions is needed: it can be either approximated by finite-difference methods or computed analytically when the expression of the functions is known. In the latter case, the time derivative requires the knowledge of the basis functions' Jacobian matrix:

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = \nabla\boldsymbol{\psi}(\mathbf{x}(t)) \cdot \mathbf{f}(\mathbf{x}(t))$$

2.2.3. Dictionary Learning

Although EDMD is able to approximate the Koopman generator, the choice of a proper dictionary of functions remains the main bottleneck of the method. The idea behind EDMD with Dictionary Learning (dlEDMD) is to define the dictionary functions as the output of a neural network and to learn them through training of the network itself [6]. The cost function to be minimized is similar to the EDMD case, but with the addition of Tikhonov regularization, since the data matrices may not be well-posed now that the dictionary functions are arbitrary. The central difference between the two methods is that the problem now depends on both the Koopman matrix K^d and the network's parameters $\boldsymbol{\omega}$, such that the problem to be solved becomes:

$$(K^d, \boldsymbol{\omega}) = \underset{\bar{K}^d, \bar{\boldsymbol{\omega}}}{\arg\min}\, J(\bar{K}^d, \bar{\boldsymbol{\omega}}) \tag{2.22}$$

with

$$J = \frac{1}{2}\sum_{i=1}^{S} \left|\left|\boldsymbol{\psi}(\mathbf{y}_i, \boldsymbol{\omega}) - K^d \boldsymbol{\psi}(\mathbf{x}_i, \boldsymbol{\omega})\right|\right|^2 + r\left|\left|K^d\right|\right|_F^2 \tag{2.23}$$

where r is the regularization parameter. This problem is no longer convex, but gradient descent methods, widely used in Deep Learning contexts, can be implemented. In particular, following the work of Li et Al. [6], the optimization can be divided into two steps, repeated until convergence:

1. fix $\boldsymbol{\omega}$ and update K^d through eq. (2.16), taking care of including the regularization, such that $K^d = [(A + rI_{q\times q})^\dagger B]^T$;
2. fix K^d and train $\boldsymbol{\omega}$ with gradient descent, minimizing the cost function of eq. (2.23).

Hereafter, a practical algorithm, similar to the one in the work mentioned above, is reported.

Algorithm 2.1 dlEDMD optimization algorithm.

Initialize K^d, $\boldsymbol{\omega}$ from a random distribution.
Initialize the iterations counter $i = 0$ and the epochs counter $j = 0$.
Set learning rate $l_r > 0$, tolerance $\varepsilon > 0$, maximum number of iterations $i_{max} > 0$, number of epochs for iteration $j_{max} > 0$, regularizer $0 < r \ll 1$.
while $J(K^d, \boldsymbol{\omega}) > \varepsilon$ and $i < i_{max}$ **do**
    $K^d \leftarrow \left[(A + rI_{q\times q})^\dagger B\right]^T$
    for $j = 1 : j_{max}$ **do**
        $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - l_r \nabla_{\boldsymbol{\omega}} J(K^d, \boldsymbol{\omega})$
    end for
    $i \leftarrow i + 1$
end while
$K^d \leftarrow \left[(A + rI_{q\times q})^\dagger B\right]^T$

In this work, a fully connected neural network was implemented, with variable hidden layers number n_h and hidden layers width w_h, and with hyperbolic tangent as activation function. If no constraints are imposed to the network output, the optimization process would lead to the trivial solution $\boldsymbol{\psi}(\mathbf{x}) \equiv \mathbf{0}$, for which $J \equiv 0$ independently from K^d; for this reason, a function that maps the input to a constant function and to the state itself can be appended to the output by including a non-trainable layer.

The neural network scheme is reported in fig. 2.1. A complete investigation of other architectures, to find an optimal one, is a subject of future research.

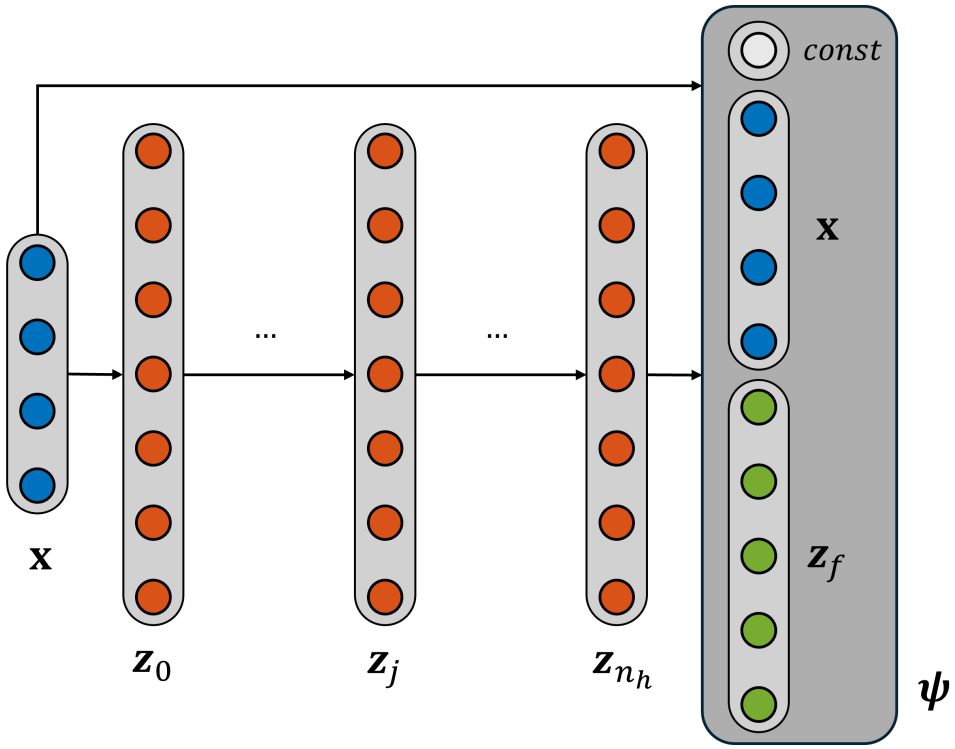


Figure 2.1: dlEDMD neural network scheme.

More explicitly, each layer is a function which receives the output of the previous layer and forwards its output to the next layer. The first input is the state $\mathbf{x}$ and the last output is the vector of dictionary functions $\boldsymbol{\psi}(\mathbf{x})$. The analytic expression is as follows:

$$\begin{cases} \mathbf{z}_0(\mathbf{x}) = \mathbf{l}_0(\mathbf{x}) = W_0\mathbf{x} + \mathbf{p}_0 & \text{(2.24a)} \\ \mathbf{z}_j(\mathbf{x}) = \mathbf{h}_j(\mathbf{z}_{j-1}(\mathbf{x})) = \mathbf{z}_{j-1}(\mathbf{x}) + \tanh\left(W_j\mathbf{z}_{j-1}(\mathbf{x}) + \mathbf{p}_j\right) \quad \forall j \in \{1, \dots, n_h\} & \text{(2.24b)} \\ \mathbf{z}_f(\mathbf{x}) = \mathbf{l}_f(\mathbf{z}_{n_h}(\mathbf{x})) = W_f\mathbf{z}_{n_h}(\mathbf{x}) + \mathbf{p}_f & \text{(2.24c)} \\ \boldsymbol{\psi}(\mathbf{x}) = \begin{bmatrix} const & \mathbf{x}^T & \mathbf{z}_f^T(\mathbf{x}) \end{bmatrix}^T & \text{(2.24d)} \end{cases}$$

where $W_0 \in \mathbb{R}^{w_h \times n}$, $\mathbf{p}_0 \in \mathbb{R}^{w_h}$, $W_j \in \mathbb{R}^{w_h \times w_h}$, $\mathbf{p}_j \in \mathbb{R}^{w_h}$, $W_f \in \mathbb{R}^{q \times w_h}$, $\mathbf{p}_f \in \mathbb{R}^q$ are the weight matrices and bias vectors and they all depend on $\boldsymbol{\omega}$; the hyperbolic tangent is the activation function that introduces the nonlinearity in the network and it is applied element-wise; the constant value can be selected based on the order of magnitude of the

data: for example, when it is normalized, $const = 1$ might be a good choice.
In order to include the control, the same approaches used for EDMD can be followed:

- modify the cost function in eq. (2.23) by including functions of the control (see eq. (2.18));
- apply dlEDMD without control and consequently obtain the bilinear model by minimizing eq. (2.20).

In this context, the first option would be a viable solution to obtain a linear control model, given the fact that the basis functions are specifically optimized towards the desired model. In this work, the dlEDMD algorithm has been modified to implement this functionality. The Jacobian matrix of the neural network function with respect to the state, $\nabla\boldsymbol{\psi}(\mathbf{x})$, is derived in appendix A, as it is useful to compute the time derivative of the basis functions and minimize eq. (2.20).

2.2.4. Comparison

Given the possibility to learn the optimal dictionary of functions, dlEDMD can achieve higher accuracy than GAL and EDMD. Table 2.1 summarizes the advantages and disadvantages of the three methods. In general, it may be useful to explore GAL and EDMD, to check if they are accurate enough on the dynamics under study, before spending time in the setup and training of a neural network.

2.2.5. Truncated Singular Value Decomposition

The previously discussed methods, even if they manage to approximate the original dynamics with good accuracy, can result in very high-dimensional systems which might not be useful for practical applications. Moreover, in the case of dlEDMD, some observables might be redundant if no additional constraints are imposed; not only this increases the model dimension in vain, but can also result in the ill-posedness of the matrices. In this context, it is useful to apply the theory behind Singular Value Decomposition (SVD).
The SVD is a unique matrix decomposition and it exists for every matrix $A \in \mathbb{R}^{n\times S}$ [52]:

$$A = U_\Sigma \Sigma V_\Sigma^T$$

In particular, $U_\Sigma \in \mathbb{C}^{n\times n}$ and $V_\Sigma \in \mathbb{C}^{S\times m}$ contain in the columns the left and right singular vectors, respectively; $\Sigma \in \mathbb{R}^{n\times S}$ is a rectangular diagonal matrix with entries called singular values, ordered from the largest one to the smallest one. The method is

Method	Pros	Cons
GAL	• Analytic computation possible...	• ...only for specific dynamics and basis functions
EDMD	• Always applicable	• Fixed basis functions • Accuracy dependent on amount of data
dlEDMD	• No need to choose basis functions	• Accuracy dependent on amount of data • Might need long training • Accuracy can be influenced by network architecture

Table 2.1: Comparison of Koopman approaches.

a generalization of eigendecomposition to rectangular matrices and it can have a precise physical meaning, depending on the case of application. For example, consider the data matrix:

$$A = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_S \end{bmatrix}$$

which is the collection of S measurements of the n-dimensional state $\mathbf{x}$. At best, n singular values can be found, in the likely case that $S \gg n$; therefore, the state can be expressed in a new basis defined by the singular values and the left singular vectors, while its S-th value is recovered through the S-th right singular vector.
The rank of A is defined by the number of non-zero singular values and the smaller is a singular value, the lower is its influence on the system: this means that the dimensionality of A can be reduced by eliminating the null singular values (without further approximation) or the less important ones (introducing more or less approximation). For the second option, an energy approach can be followed [52]: the total energy E and the cumulative energy associated with the N-th singular value E_N are defined as in eqs. (2.25) and (2.26); the matrix is truncated at the first singular value whose cumulative energy is the same as

the total energy, within a selected tolerance.

$$E = \sum_{i=1}^{n} \Sigma_i^2 \tag{2.25}$$

$$E_N = \sum_{i=1}^{N} \Sigma_i^2 \tag{2.26}$$

Considering the data matrix $\psi_{\mathbf{x}}$, this method can be applied to EDMD and derived methods: before computing the uncontrolled matrix [53], before computing the control matrix or after computing all the matrices. In the latter case, the matrices are transformed according to eq. (2.27), where the barred Koopman matrices are the truncated ones and $U_{\Sigma,N}$ is the matrix of the left singular vectors truncated at the N-th value; a practical algorithm is reported in algorithm 2.2.

$$\begin{cases} \bar{K}_0 = U_{\Sigma,N}^T K_0 U_{\Sigma,N} \\ \bar{K}_i = U_{\Sigma,N}^T K_i U_{\Sigma,N} \\ \bar{K}_c = U_{\Sigma,N}^T K_c U_{\Sigma,N} \end{cases} \tag{2.27}$$

Algorithm 2.2 Truncated SVD algorithm.

1: Load the data matrix $\psi_{\mathbf{x}}$ (for example from uncontrolled trajectories).
2: Set the tolerance value $0 \leq \varepsilon \leq 1$.
3: $[U\Sigma, \Sigma, V\Sigma] = \text{svd}(\psi_{\mathbf{x}})$
4: $E = \sum_{i=1}^{n} \Sigma_i^2$
5: Initialize the cumulative energy $E_N = 0$ and the counter $i = 1$
6: **while** $1 - \dfrac{E_N}{E} > \varepsilon$ **do**
7: $\quad E_N \leftarrow E_N + \Sigma_i^2$
8: $\quad i \leftarrow i + 1$
9: **end while**
10: $N \leftarrow i - 1$
11: $U_{\Sigma,N} \leftarrow U_{\Sigma}(1 : N)$
12: Compute the Koopman matrices according to eq. (2.27)

2.2.6. Other Methods

The Koopman literature consists of many other methods that can be very useful, depending on the context. Some of them, while not used in this work, are still worth of mention;

for a comprehensive list of Koopman approaches for vehicular applications, divided for scientific field, refer to the work of Manzoor et Al. [54].

EDMD with Quadrature Hofmann et Al. [5] suggested using EDMD with a quadrature rule (qEDMD) for multivariate integration. In this way, the number of snapshot pairs needed to achieve a certain accuracy can be reduced. In fact, using Monte Carlo integration, with uniformly distributed random points, would result in a slower rate of convergence, the higher the dimension of the original problem.

EDMD for Koopman Generator EDMD can be modified to directly approximate the Koopman generator (gEDMD) by including the time derivative of the basis functions in the cost function [55]. Equation (2.20) derives from this idea.

Sparse Identification of Nonlinear Dynamical Systems Brunton et Al. [56] aimed at identifying a dynamical system from data by imposing sparsity: gEDMD was modified by adding a L^1 regularization term to the regression. There are many works in which this algorithm was used to obtain the Koopman generator. It can be useful in some contexts, when a sparse Koopman model is expected.

Optimal Construction of Koopman Eigenfunctions Korda et Mezić [50] showed that it is possible to build a Koopman eigenfunction on an arbitrary eigenvalue, provided that some conditions are met. Their algorithm is able to find a set of eigenvalues that is accurate enough on a set of trajectories data. Eigenfunctions can be extended to all the domain through interpolation. It must be pointed out that:

- the hypothesis for its application might not always be satisfied;
- the optimization problem is non-convex;
- the interpolation can be challenging to perform, for high dimensional systems.

2.3. Motivating Examples

The aforementioned methods are applied to some examples to show in which cases they perform well and in which ones they fail to give a good model. The first example is a 2 Degrees of Freedom (DoFs) system with a quadratic nonlinearity for which a Koopman-invariant subspace that includes the original state exists; the second and third examples are two different variations of the Duffing oscillator, which is a 2 DoFs system with polynomial nonlinearities; the last example is a non-polynomial system and, therefore,

analytic GAL with Legendre polynomials cannot be applied straightforwardly. All the dynamical systems in this section are considered with a generic time unit TU and state unit SU. Moreover, the neural network architecture is fixed in this section, with $n_h = 3$, $w_h = 100$ and $const = 1$.
The error ε_T associated with a trajectory of time duration T is obtained in the following way (see also eq. (2.28)):

1. the initial condition is propagated with the original model to obtain $\mathbf{x}(t)$;
2. the propagation is performed also with the Koopman model to obtain $\boldsymbol{\psi}(t)$ and consequently $\bar{\mathbf{x}}(t) = P\boldsymbol{\psi}(t)$;
3. the norm of the absolute error vector is computed;
4. its maximum value along the time duration is the state reconstruction error of the trajectory.

$$\varepsilon_T = \arg\max_{t\in[0,T]} \sqrt{\sum_{j=1}^{n} \left(\bar{x}_j(t) - x_j(t)\right)^2} \tag{2.28}$$

Furthermore, a Monte Carlo campaign can be performed on a set of trajectories, with uniform random initial conditions in $[-1, 1]^2$ SU, to approximate the mean reconstruction error and its standard deviation.

2.3.1. Closed System

Consider the dynamical system in eq. (2.29) and the vector of basis functions $\boldsymbol{\psi}(\mathbf{x}) = [x_1, x_2, x_1^2, x_1x_2, x_1^3]^T$.

$$\frac{d}{dt}\mathbf{x}(t) = \frac{d}{dt}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} -\frac{1}{10}x_1(t) \\ x_1^2(t) - x_2(t) \end{bmatrix} \tag{2.29}$$

It can be rewritten by augmenting the state and making use of $\boldsymbol{\psi}(\mathbf{x})$ [47]:

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = \begin{bmatrix} -0.1 & 0 & 0 & 0 & 0 \\ 0 & -1.0 & 1.0 & 0 & 0 \\ 0 & 0 & -0.2 & 0 & 0 \\ 0 & 0 & 0 & -1.1 & 1.0 \\ 0 & 0 & 0 & 0 & -0.3 \end{bmatrix} \boldsymbol{\psi}(\mathbf{x}(t)) = K\boldsymbol{\psi}(\mathbf{x}(t))$$

It is an exact Koopman system with the Koopman generator matrix K, since the subspace spanned by the observables in $\boldsymbol{\psi}$ is Koopman-invariant. If, however, other polynomials are added to the basis functions, the system gets polluted by a bias term. For example, using $\boldsymbol{\psi}(\mathbf{x}) = [x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3]^T$:

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = A\boldsymbol{\psi}(\mathbf{x}(t)) + \mathbf{a}(\mathbf{x}(t))$$

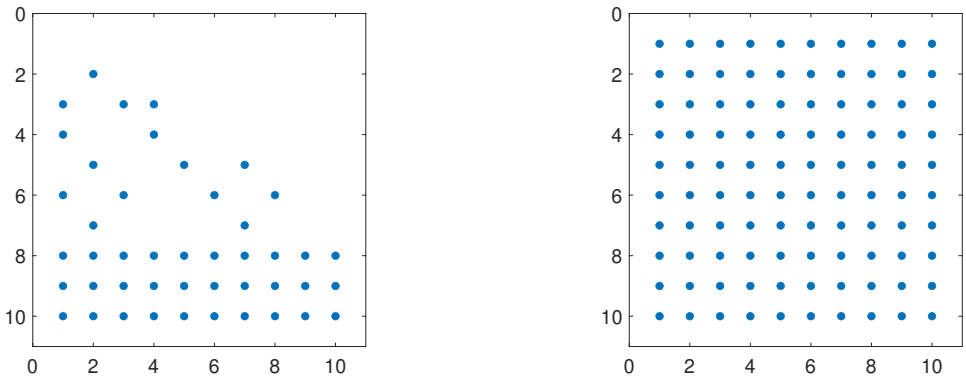
with

$$A = \begin{bmatrix} -0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1.1 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2.0 & 0 & 2.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3.0 \end{bmatrix}, \quad \mathbf{a}(\mathbf{x}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ x_1^4 \\ 2x_1^3x_2 \\ 3x_1^2x_2^2 \end{bmatrix}$$

In this form, it is obviously not a Koopman system, due to the presence of the bias vector $\mathbf{a}(\mathbf{x})$. In order to obtain the Koopman matrix, the higher-order polynomials must be projected onto $\boldsymbol{\psi}(\mathbf{x})$; this generates spurious pairs of eigenfunctions and eigenvalues, that is, the eigenfunctions evolution is not well-approximated by the eigenvalues. However, the dynamics associated with the original state is not influenced, since it belongs to an exact Koopman-invariant subspace, and the reconstruction error remains zero.
This behavior can be observed when using the GAL and EDMD methods with Legendre multivariate polynomials up to order 3. In order to obtain data for EDMD, uniformly random initial conditions were generated within the range $[-1, 1]^2$ SU and propagated for 1 TU. Data was sampled with time step $\Delta t = 0.1$ TU, resulting in a total number of snapshot pairs $S = \#$ initial conditions $\cdot\, 10$. When GAL is used, the Koopman generator is approximated by the following matrix:

$$K_{GAL} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.58 & 0 & -1.00 & 0.52 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.22 & 0 & 0 & -0.20 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.04 & 0 & 0 & -1.10 & 0 & 0.45 & 0 & 0 & 0 \\ -2.24 & 0 & 1.29 & 0 & 0 & -2.00 & 0 & 1.15 & 0 & 0 \\ 0 & -0.46 & 0 & 0 & 0 & 0 & -0.30 & 0 & 0 & 0 \\ 0.52 & 0 & -0.22 & 0.91 & 0 & 0 & 0 & -1.20 & 0 & 0 \\ 0 & -2.24 & 0 & 0 & 2.33 & 0 & 0 & 0 & -2.10 & 0 \\ 0.88 & 0 & -4.58 & 0.79 & 0 & 1.97 & 0 & 0 & 0 & -3.00 \end{bmatrix}$$

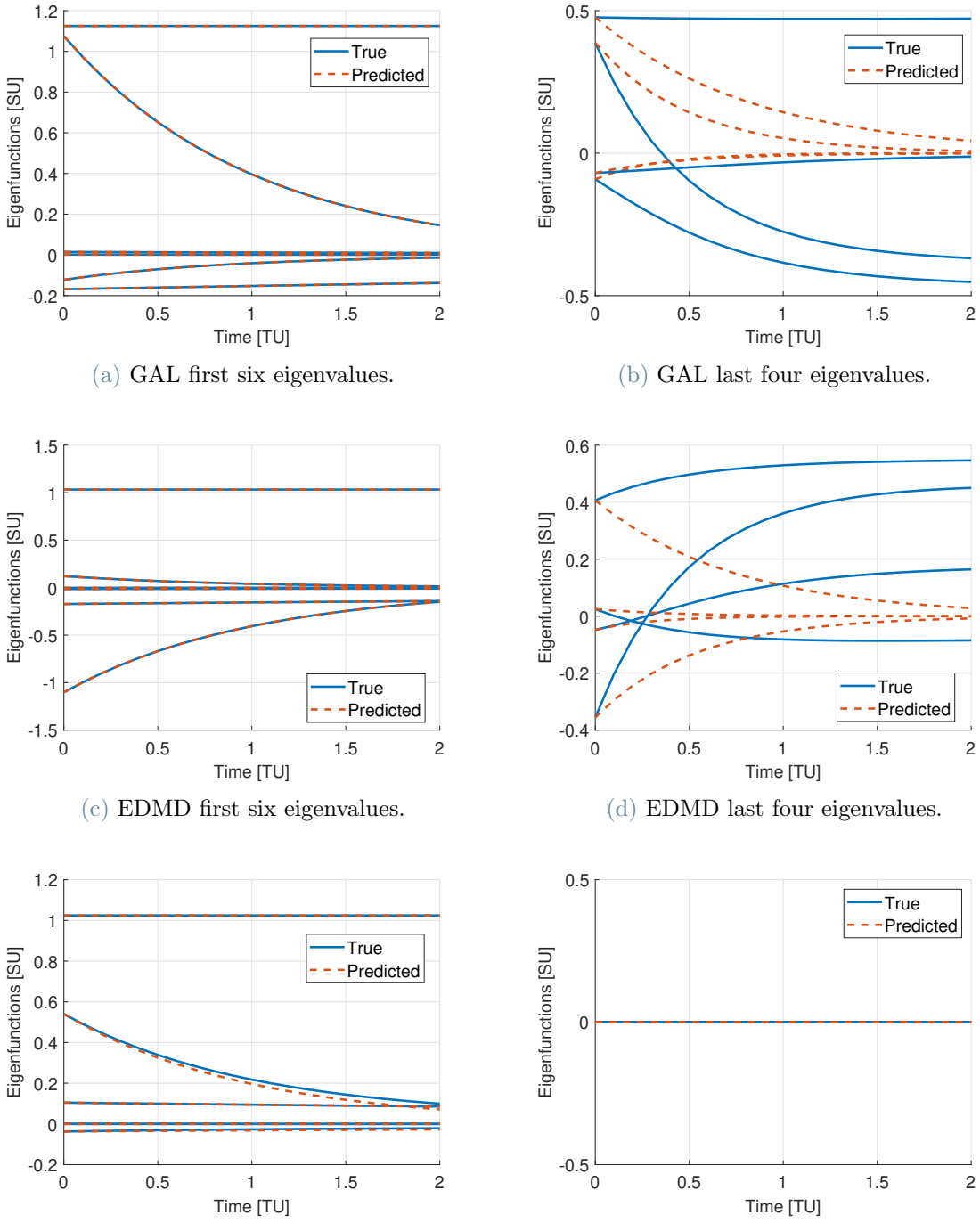
When EDMD and dlEDMD are used, with the same number of basis functions and with enough data, the resulting Koopman matrices have the sparsity pattern represented in fig. 2.2. It can be seen that EDMD is very similar to GAL for the first 7 observables, while dlEDMD does not have any non-zero elements, as the basis functions are different and obtained numerically through neural network optimization.



(a) EDMD. (b) dlEDMD.

Figure 2.2: Closed system Koopman sparsity pattern.

In fig. 2.3 an example evolution of the eigenfunctions coming from GAL and dlEDMD is shown. The first 6 eigenfunctions are exact, as expected, for both methods; the others are not well-approximated by GAL, whereas they are approximated as constant and equal to zero by dlEDMD. In fact, dlEDMD has no incentive in finding other eigenfunctions, since the zero function can be perfectly propagated by any Koopman matrix; it only optimizes the eigenfunctions needed to propagate the original state.



(a) GAL first six eigenvalues.

(b) GAL last four eigenvalues.

(c) EDMD first six eigenvalues.

(d) EDMD last four eigenvalues.

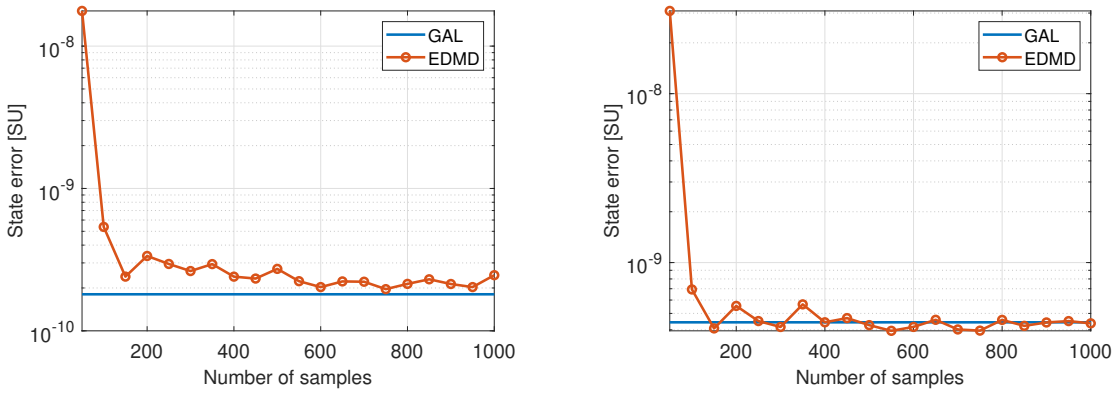
(e) dlEDMD first six eigenvalues.

(f) dlEDMD last four eigenvalues.

Figure 2.3: Closed system Koopman eigenfunctions evolution example.

Figure 2.4 shows the results of the Monte Carlo campaign for GAL and EDMD, for $T = 2$ TU and $T = 10$ TU: the EDMD error converges to the GAL one as S increases and, at convergence, the two methods have a numerically negligible error on the state;

furthermore, the error does not depend on the propagation time.



(a) Propagation time: 2 TU. (b) Propagation time: 10 TU.

Figure 2.4: Closed system mean error over 100 trajectories.

2.3.2. Asymptotically Stable Duffing Oscillator

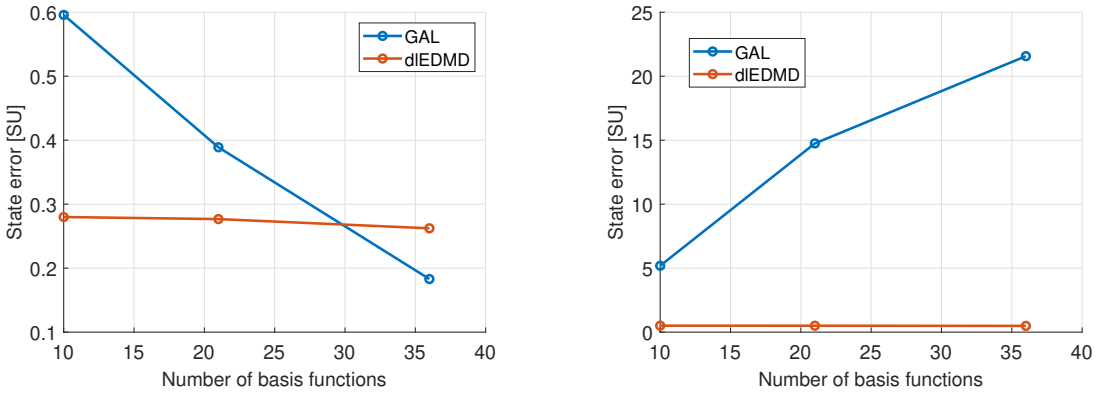
Consider the Duffing oscillator's dynamical system in eq. (2.30) [6], with one unstable equilibrium point in $[0, 0]^T$ SU and two asymptotically stable equilibrium points in $[\pm 1, 0]^T$ SU.

$$\frac{d}{dt}\mathbf{x}(t) = \frac{d}{dt}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ x_1(t) - x_1^3(t) - \frac{1}{2}x_2(t) \end{bmatrix} \tag{2.30}$$

Since the system is polynomial, EDMD is not applied, as it converges to GAL. The performances of the other two methods are evaluated for different numbers of basis functions; in particular, for GAL the degree of the Legendre polynomials is increased and the associated number of functions is used for dlEDMD. Data for dlEDMD was obtained by propagating 100 random initial conditions in the range $[-1, 1]^2$ SU for 10 TU and sampling with $\Delta t = 0.1$ TU.

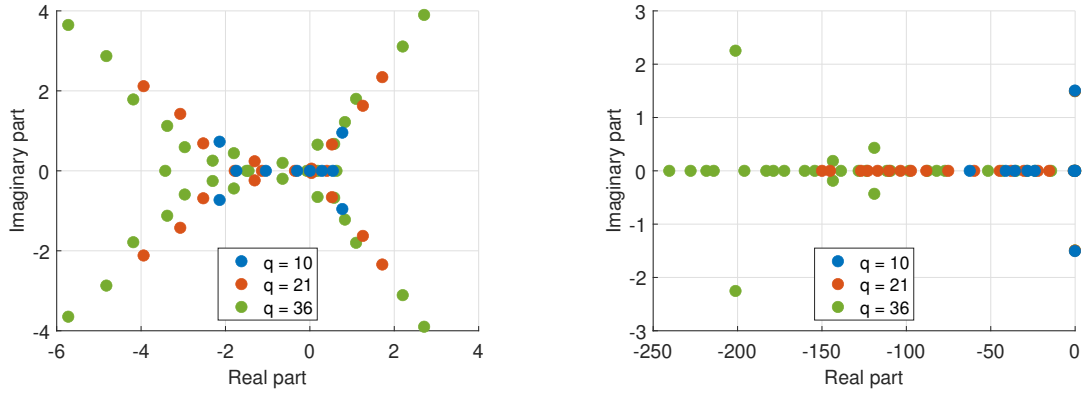
Figure 2.5 shows how the state reconstruction error changes depending on the propagation time T and the number of basis functions. One would expect the error to always decrease as more basis functions are used, but this is the case only for dlEDMD. For a longer propagation time, GAL performs worse when new functions are introduced because they change the Koopman spectrum; even if the single-step prediction is better, the new, spurious eigenvalues are highly divergent (fig. 2.6). This is due to the fact that no exact Koopman-invariant subspace can be found and, therefore, when new basis functions are

added, they have an influence on the dynamics of the old ones. In this case, they make the dynamics more unstable. On the other hand, dlEDMD is able to find a better subspace that results in a model more accurate than the GAL one: no divergence behavior can be observed, but, however, the improvement in accuracy with the number of basis functions is modest. An example trajectory can be seen in fig. 2.7.



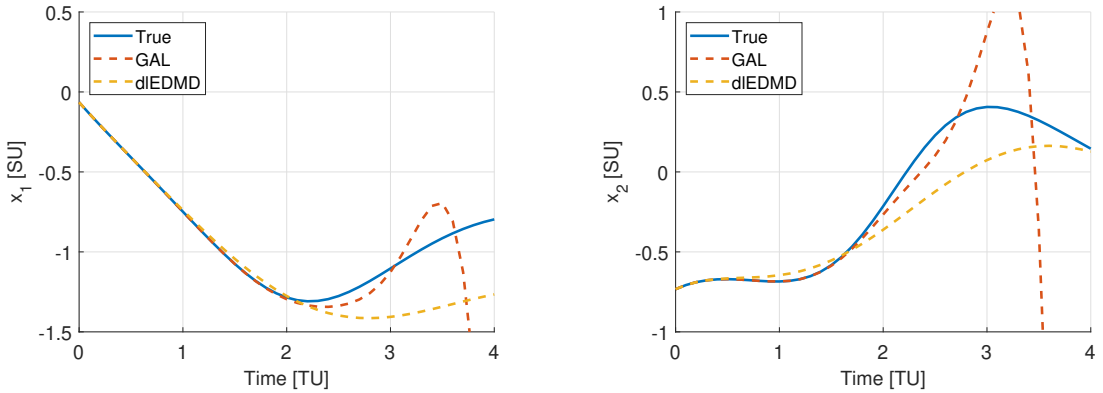
(a) Propagation time: 2 TU. (b) Propagation time: 4 TU.

Figure 2.5: Asymptotically stable Duffing oscillator mean error over 100 trajectories.



(a) GAL. (b) dlEDMD.

Figure 2.6: Asymptotically stable Duffing oscillator eigenvalues.



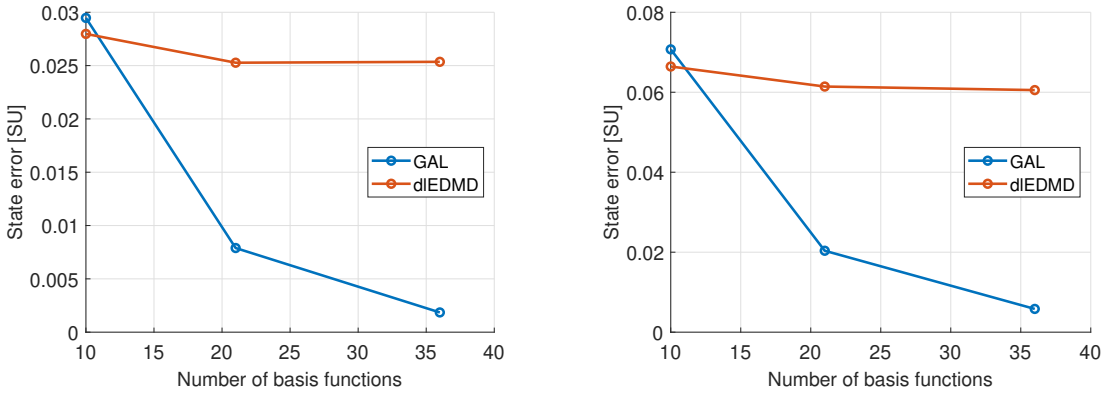
(a) (b)

Figure 2.7: Asymptotically stable Duffing oscillator example trajectory, 36 basis functions.

2.3.3. Stable Duffing Oscillator

Consider the Duffing oscillator's dynamical system in eq. (2.31) with one stable equilibrium point in $[0, 0]^T$ SU. It is the same example used in [3], but with a higher coefficient for the cubic term.

$$\frac{d}{dt}\mathbf{x}(t) = \frac{d}{dt}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -x_1(t) - x_1^3(t) \end{bmatrix} \tag{2.31}$$

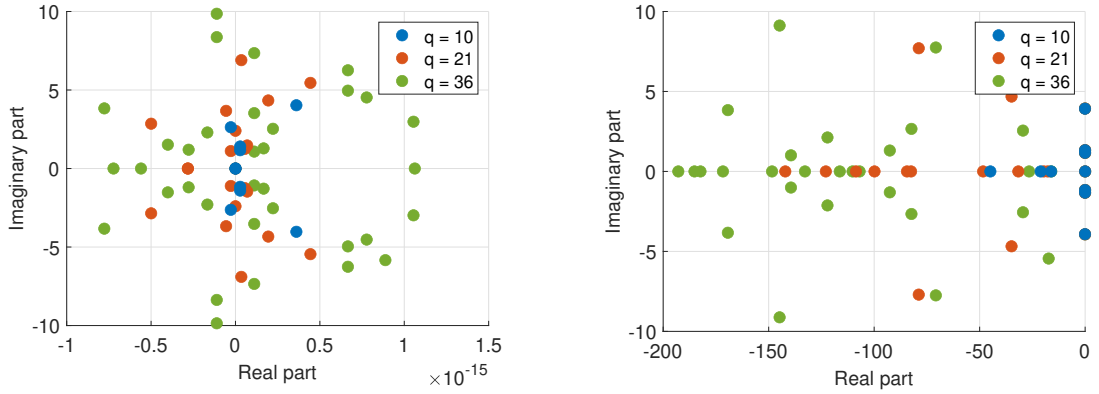


(a) Propagation time: 2 TU. (b) Propagation time: 4 TU.

Figure 2.8: Stable Duffing oscillator mean error over 100 trajectories.

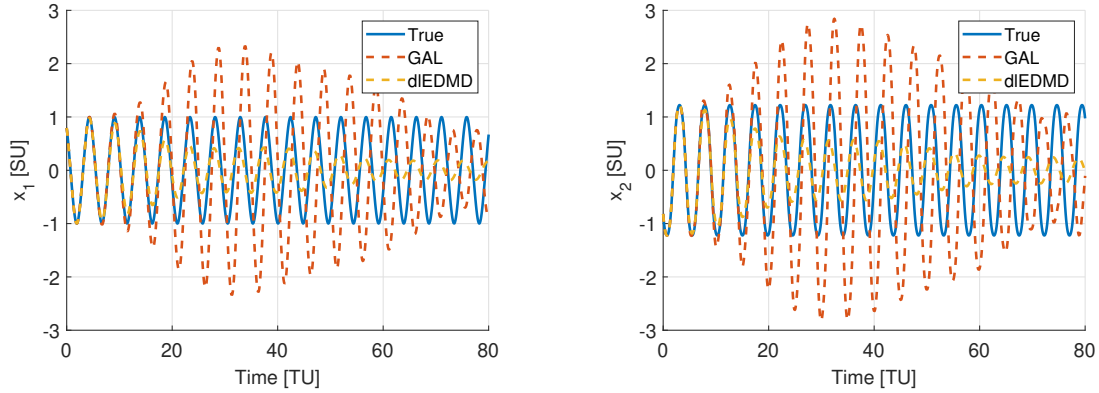
Also in this case, GAL and dlEDMD (with the same number of samples as in the previous case) are used. In fig. 2.8 it can be observed that, differently from before, the error does not

increase when higher-order polynomials are used for GAL. On the contrary, the accuracy improvement for dlEDMD is much more modest than it is for GAL. The eigenvalues in fig. 2.9 show that new, stable modes are introduced in the system, but they are wrong, as they do not appear in the oscillatory behavior of the original dynamics or, at least, not with that magnitude. In fig. 2.10 an example trajectory of duration 80 TU shows how these new frequencies appear when the model is propagated for longer times; the dlEDMD model, instead, manifests a damped behavior.



(a) GAL. (b) dlEDMD.

Figure 2.9: Stable Duffing oscillator eigenvalues.



(a) (b)

Figure 2.10: Stable Duffing oscillator example trajectory, 36 basis functions.

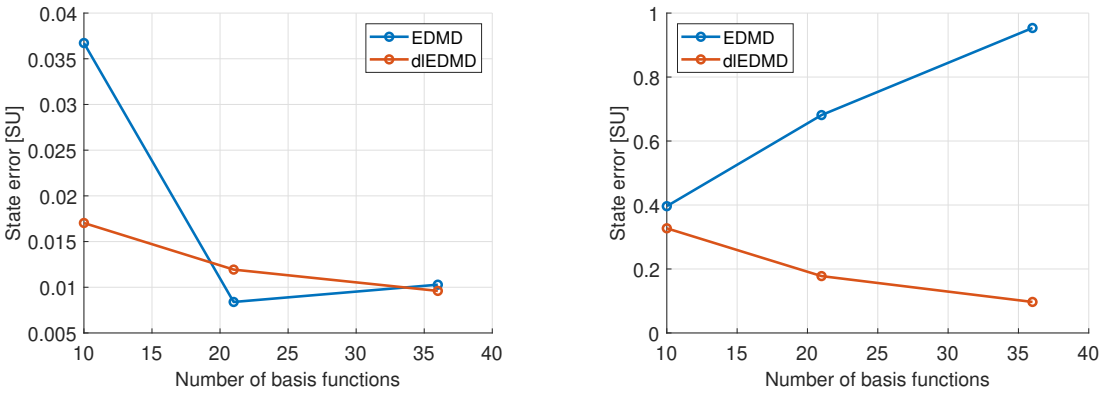
2.3.4. Non-polynomial System

Consider the non-polynomial and control-affine dynamical system in eq. (2.32), a modified version of the example in [7]. Analytic GAL with Legendre polynomials cannot be used

in this case (at least not in a straightforward way), so, instead, EDMD is applied and compared with dlEDMD.

$$\frac{d}{dt}\mathbf{x}(t) = \frac{d}{dt}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -0.3\sin x_1(t) + \dfrac{1}{100}x_2(t) + u(t) \end{bmatrix} \tag{2.32}$$

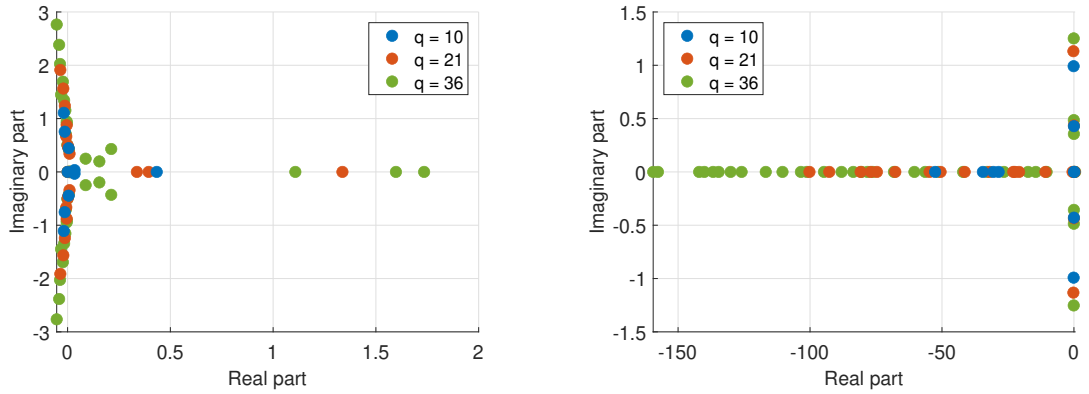
Free-dynamics Model First of all, the uncontrolled dynamics is approximated: 100 uniformly random initial conditions were generated in the range $[-1, 1]^2$ SU and propagated for 10 TU. Data was sampled with time step $\Delta t = 0.1$ TU, resulting in a total number of snapshot pairs $S = 10000$.
Figure 2.11 shows the state error for the two methods as more basis functions are used, and for different propagation times; in figs. 2.12 and 2.13 it can be seen that the spectrum approximated by EDMD gets polluted as the number of basis functions increases, making the dynamics unstable.



(a) Propagation time: 2 TU.

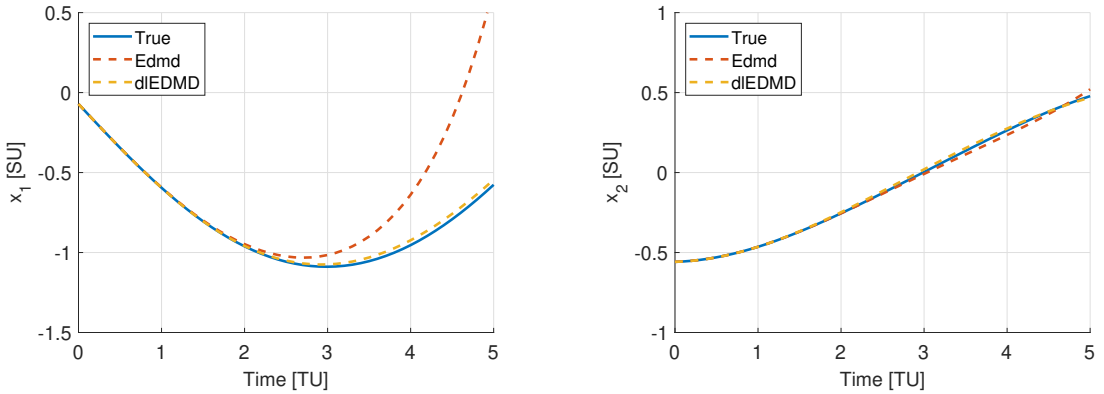
(b) Propagation time: 5 TU.

Figure 2.11: Uncontrolled non-polynomial system mean error over 100 trajectories.



(a) EDMD. (b) dlEDMD.

Figure 2.12: Non-polynomial system eigenvalues.



(a) (b)

Figure 2.13: Uncontrolled non-polynomial system example trajectory, 36 basis functions.

Bilinear Control Model By solving eq. (2.21), the control is introduced in bilinear form. The free-dynamics Koopman model with 10 basis functions is employed since it has the best performance between the EDMD models obtained. New 100 trajectories were generated by using random piecewise-constant control in $[-1, 1]$ SU/TU with time duration in $[1, 10]$ TU. To avoid having unbounded data, which could happen when introducing random control in the system, only trajectories with state in $[-1, 1]^2$ SU were used. The resulting system was tested on 100 trajectories of duration 5 TU; the results are shown in fig. 2.14, where it can be observed that the performance is consistent with the one of the uncontrolled system; an example trajectory is shown in fig. 2.15.

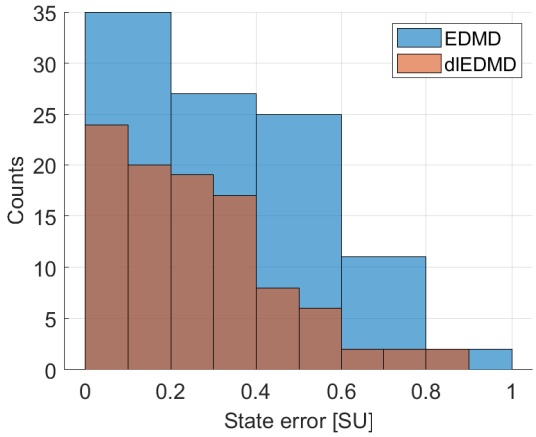


Figure 2.14: Bilinear control non-polynomial system error, $T = 5$ TU, 10 basis functions. EDMD mean 0.33 SU, dlEDMD mean 0.26 SU.

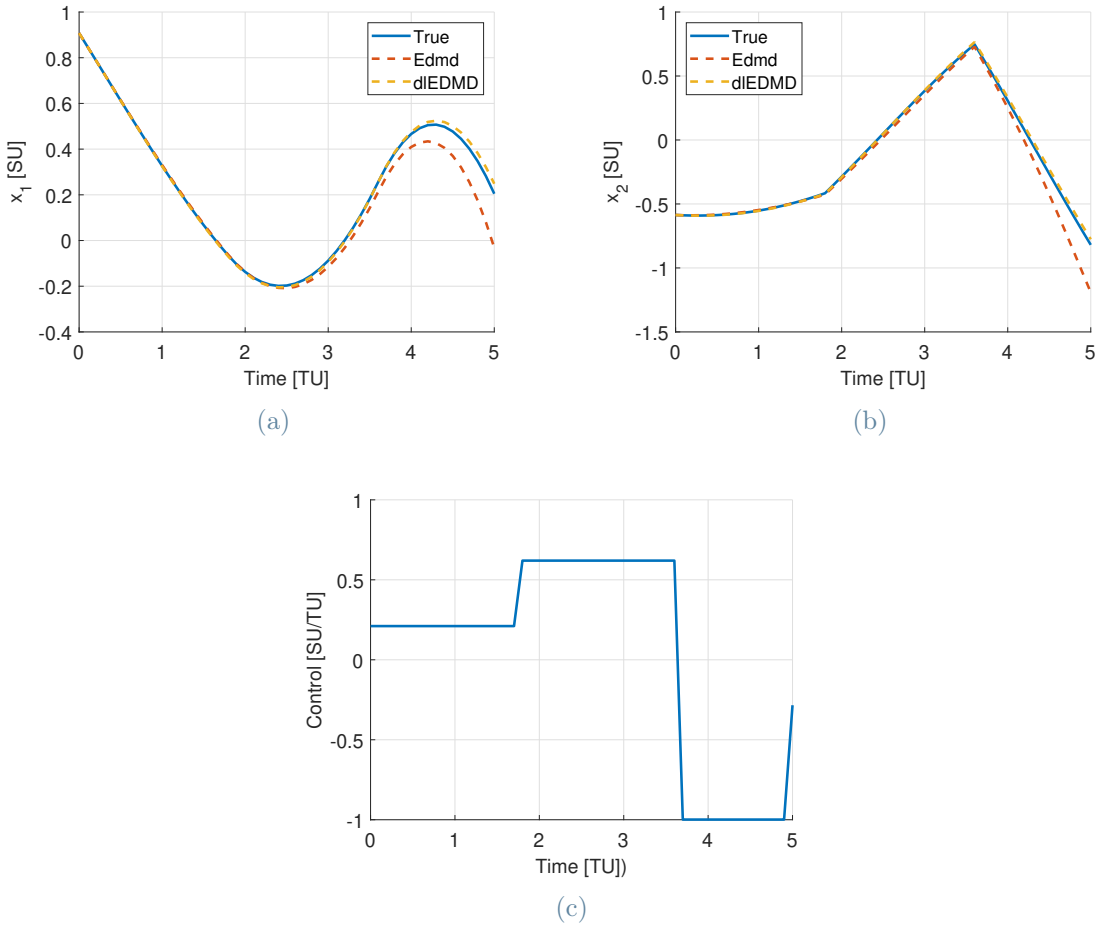


Figure 2.15: Bilinear control non-polynomial system example trajectory, 10 basis functions.

It must be pointed out that, for Legendre polynomials, $\mathcal{K}_i\boldsymbol{\psi}$ lies in the span of $\{\psi_1, \ldots, \psi_q\}$ (see eq. (2.7)), while this is not true in general for dlEDMD, especially with hyperbolic tangent as activation function; however, the learned subspace is large enough that the projection of controlled dynamics is still accurate.

Linear Control Model The control is introduced linearly with EDMD by solving eq. (2.19), with linear functions of the control and with data obtained in a similar way as for the bilinear model; the cost function in dlEDMD is modified accordingly. The condition in eq. (2.12b) requires that $\mathcal{K}_i\boldsymbol{\psi}$ lies in the span of the constant function, but this is not satisfied by either of the methods. Practically, they get the best approximation possible by projecting the Koopman generator associated with the control onto the span of the constant function.
The crucial difference between the two methods is that EDMD has a fixed dictionary (Legendre polynomials); dlEDMD, on the other hand, optimizes its dictionary in order to have the best result. This can be seen in figs. 2.16 and 2.17 where dlEDMD shows a smaller error than EDMD, and comparable with the previous models.

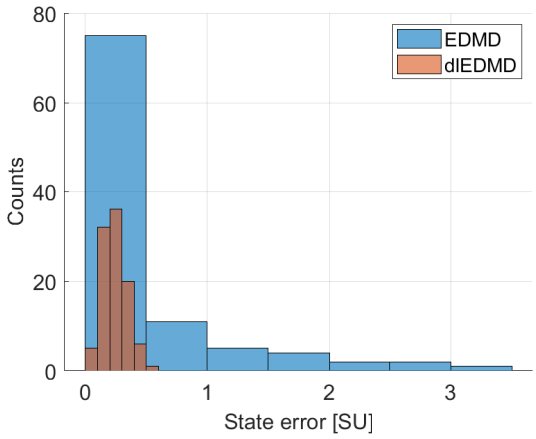


Figure 2.16: Linear control non-polynomial system error, $T = 5$ TU, 10 basis functions. EDMD mean 0.47 SU, dlEDMD mean 0.24 SU.

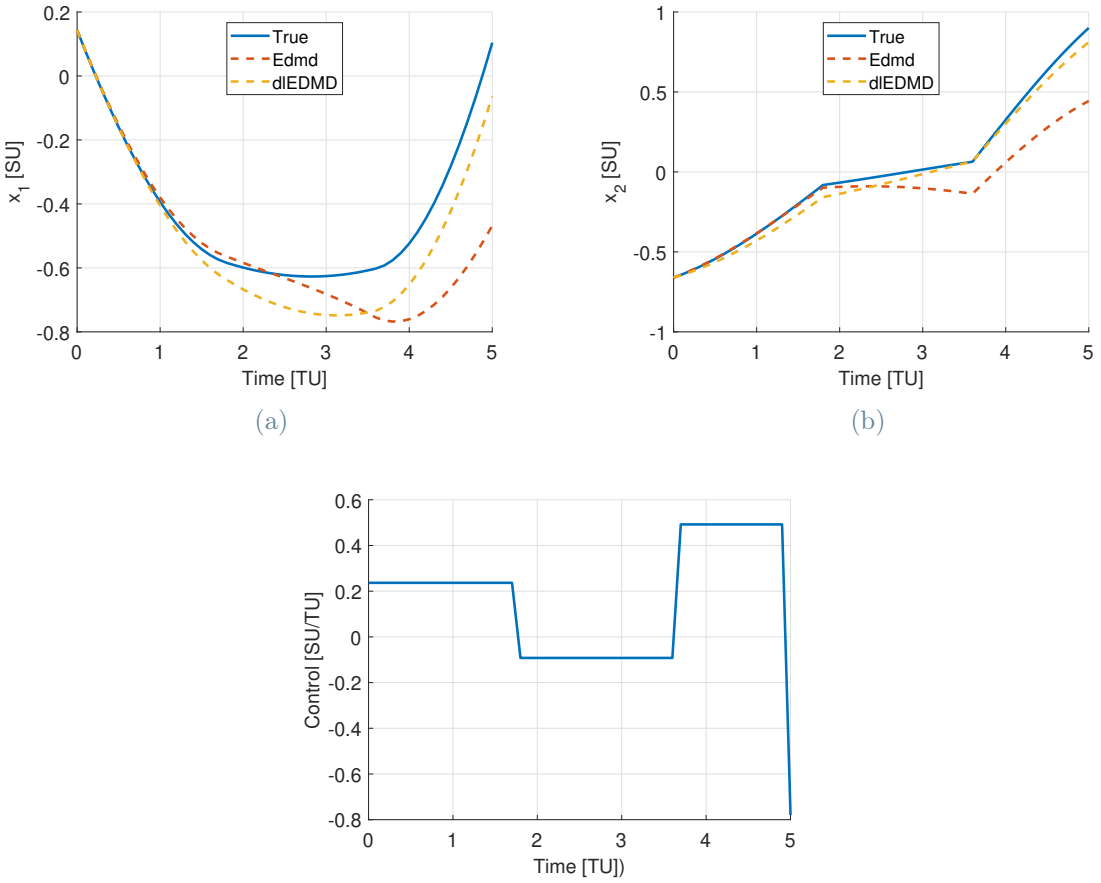
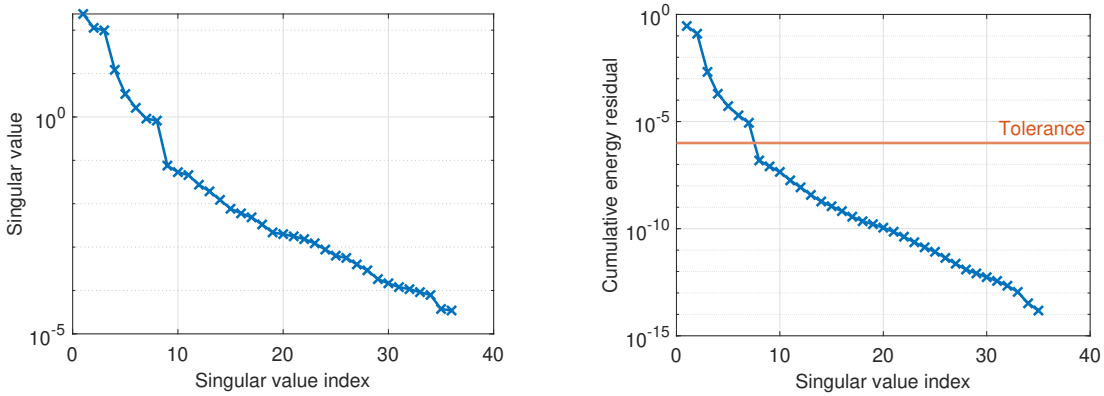


Figure 2.17: Linear control non-polynomial system example trajectory, 10 basis functions.

Truncated Singular Value Decomposition The dlEDMD dictionary is analyzed to check whether a truncated SVD is viable or not. The decomposition was applied to the data matrix used to obtain the free-dynamics Koopman matrix; the singular values and their cumulative energy residual are represented in fig. 2.18. With a tolerance of 10^{-6}, the resulting matrix can be truncated at the eighth singular value. The new system is approximated by following the steps in algorithm 2.2 and the resulting accuracy can be observed in fig. 2.19. Interestingly, the performance of the system is not influenced by the truncation which, on the other hand, reduces the dimension of the system and improves the conditioning of the matrices. Without the truncation the reciprocal of the condition of K in 1-norm is 2.6×10^{-8}; with the truncation it is 1.5×10^{-6}.



(a) Singular values. (b) Cumulative energy error.

Figure 2.18: Non-polynomial system truncated SVD analysis, dlEDMD.

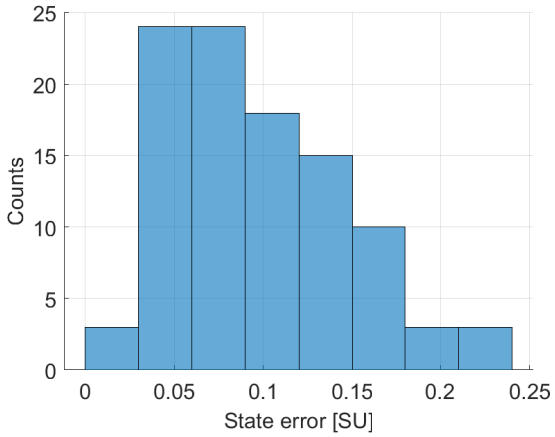


Figure 2.19: Uncontrolled non-polynomial system after truncated SVD error, $T = 5$ TU, 8 basis functions. dlEDMD mean 0.09 SU.

3 | Rocket Landing

This chapter sets out the rocket landing problem of interest: section 3.1 describes the mission scenario and the rocket properties, which are useful to understand the equations of motion in section 3.2; in sections 3.3 and 3.4 the OCPs are formulated and a Koopman-based LP is built from scratch.

3.1. Environment Modeling

The scenario involves a fuel-optimal rocket landing on the Mars' surface [23]. The atmosphere is considered, resulting in an aerodynamic force acting on the vehicle. The rocket must reach the target on the ground with zero velocity, while its mass decreases throughout the mission as a result of propellant expulsion.
It is useful to define three reference frames, also represented in fig. 3.1:

- an inertial, target-fixed reference frame e with its components locally defined, in order, as the *Up*, *East* and *North* directions;
- a non-inertial, body-fixed frame b with its first direction pointing from the engine to the tip and the other two components defined perpendicularly to the first one, according to the rotation matrix in eq. (3.3);
- a non-inertial, wind-fixed frame w with the direction of the rocket velocity relative to the inertial frame as its first component (no gusts are considered) and the other two defined perpendicularly to it, according to the rotation matrix in eq. (3.1).

The first reference frame is target-centered and can be considered inertial due to the short time of flight; the other two reference frames are body-centered; the Mars' curvature is neglected so that the altitude does not change with the horizontal position and the gravity can always be defined as aligned with the -*Up* direction. The i-th component of reference frame j, expressed in reference frame k, is denoted as $\hat{\mathbf{j}}_{i,k}$. The transformation of a vector from frame i to frame j can be expressed by the Direction Cosine Matrix (DCM) R_i^j: $\mathbf{a}_j = R_i^j \mathbf{a}_i$. The expression of the DCMs associated with the three reference frames is given in eqs. (3.1) to (3.3), where s_δ and c_δ stand for $\sin\delta$ and $\cos\delta$, respectively [23].

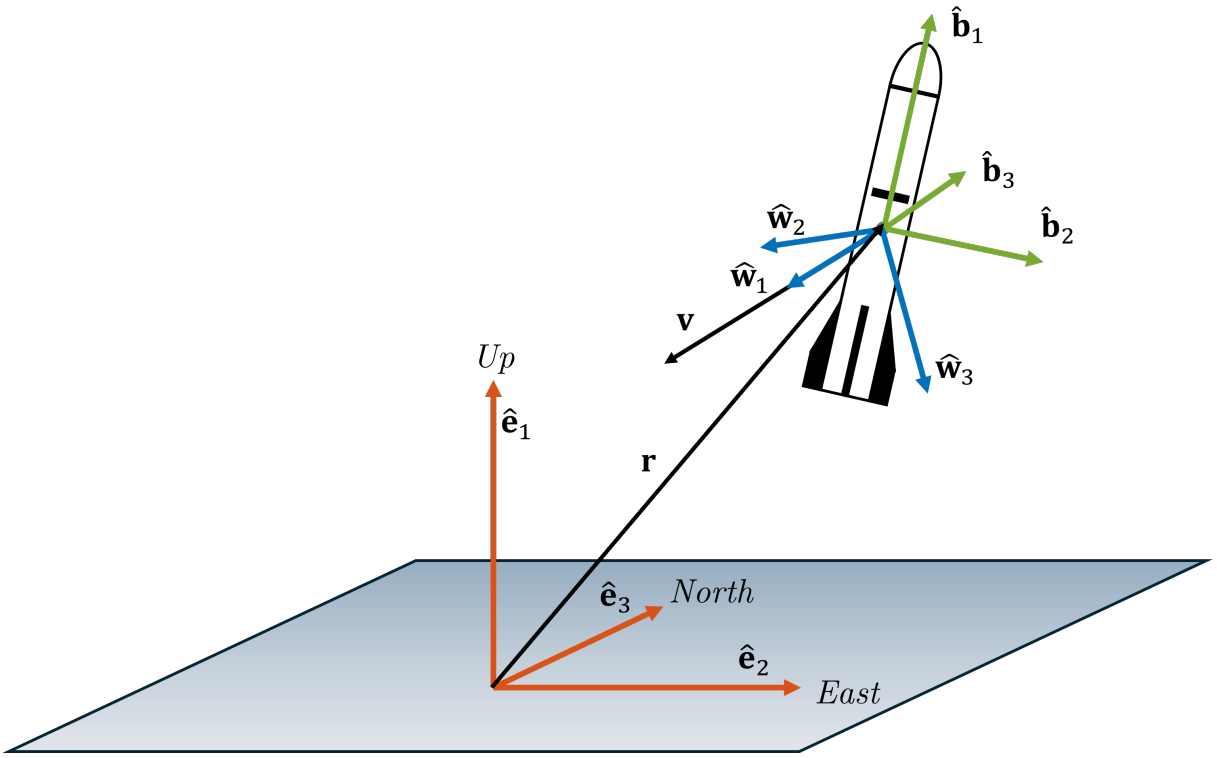


Figure 3.1: Rocket landing reference frames.

$$R_w^e = \begin{bmatrix} c_{\gamma_v} c_{\chi_v} & -s_{\chi_v} c_{\gamma_v} & -s_{\gamma_v} \\ s_{\chi_v} & c_{\chi_v} & 0 \\ s_{\gamma_v} c_{\chi_v} & -s_{\gamma_v} s_{\chi_v} & c_{\gamma_v} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{w}}_{1,e} & \hat{\mathbf{w}}_{2,e} & \hat{\mathbf{w}}_{3,e} \end{bmatrix} \tag{3.1}$$

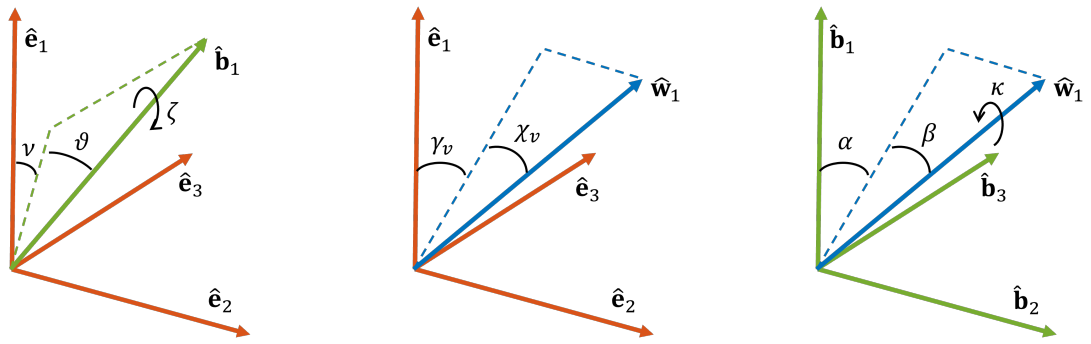
$$R_b^w = \begin{bmatrix} c_\alpha c_\beta & s_\beta & s_\alpha c_\beta \\ s_\kappa s_\alpha - c_\kappa c_\alpha s_\beta & c_\kappa c_\beta & -c_\kappa s_\alpha s_\beta - c_\alpha s_\kappa \\ -c_\alpha s_\kappa s_\beta - c_\kappa s_\alpha & c_\beta s_\kappa & c_\kappa c_\alpha - s_\kappa s_\beta s_\alpha \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{b}}_{1,w} & \hat{\mathbf{b}}_{2,w} & \hat{\mathbf{b}}_{3,w} \end{bmatrix} \tag{3.2}$$

$$R_b^e = \begin{bmatrix} c_\nu c_\theta & -c_\nu s_\theta s_\zeta - c_\zeta s_\nu & s_\nu s_\zeta - c_\nu c_\zeta s_\theta \\ c_\theta s_\nu & c_\nu c_\zeta - s_\nu s_\theta s_\zeta & -c_\zeta s_\nu s_\theta - c_\nu s_\zeta \\ s_\theta & c_\theta s_\zeta & c_\theta c_\zeta \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{b}}_{1,e} & \hat{\mathbf{b}}_{2,e} & \hat{\mathbf{b}}_{3,e} \end{bmatrix} = R_w^e R_b^w \tag{3.3}$$

In particular, ν, θ and ζ are, respectively, the yaw, pitch and roll angles that describe the rocket attitude with respect to the inertial frame (attitude angles). The aerodynamic angles α, β and κ are the angle of attack, the sideslip angle and the bank angle, respectively; they are important because they influence the aerodynamic force, and, for it to be defined, the angle of attack is bounded in the range $[150, 210]°$ and the sideslip angle in the range $[-30, 30]°$. To describe the transformation from wind to inertial frame, the vertical flight-path angle γ_v and the vertical azimuth angle χ_v are used [23]. They differ from the heritage flight path angle and azimuth angle which are usually applied in horizontal flight dynamics; in fact, they reach a singularity in vertical flight condition, when the azimuth angle varies a lot for small variations in the components of the horizontal velocity. The new angles are meant to work best for rocket landing applications and are defined as follows:

$$\gamma_v = \tan^{-1}\left(\frac{v_{3,e}}{v_{1,e}}\right), \quad \chi_v = \tan^{-1}\left(\frac{v_{2,e}}{\sqrt{v_{1,e}^2 + v_{3,e}^2}}\right) \tag{3.4}$$

where $v_{i,e}$ is the i-th component of the wind velocity vector in the inertial frame. Furthermore, the inverse tangent function has a unique value on the four quadrants, taking care of the signs of the numerator and denominator of the argument. A visual representation can be seen in fig. 3.2. Defining with i_δ a right-hand rotation of δ about the i-th axis, it is possible to obtain a new reference frame with a sequence of rotations about the intrinsic axes (Euler angles). The rocket attitude is obtained with the sequence $3_\nu 2_{-\theta} 1_\zeta$; the wind frame from the inertial frame with with the sequence $2_{-\gamma_v} 3_{\chi_v}$; the wind frame with respect to the body with the sequence $2_{-\alpha} 3_\beta 1_{-\kappa}$.



(a) Attitude angles. (b) Wind angles. (c) Aerodynamic angles.

Figure 3.2: Rocket landing angles representation. For simplicity, only the resulting first axis is represented.

When ν, ζ and $v_{2,e}$ are equal to zero, i.e. in the 2D motion in the $1-3$ plane, the sideslip and bank angles admit two values: $\beta = \kappa = a\pi$ for $a = 0, 1$. Consequently, also the angle of attack can have two values which differ by 180°. This means that the same attitude with respect to the wind can be obtained with two different sets of consecutive rotations. However, given the bounds on the aerodynamic angles defined before, the sideslip angle is considered equal to zero. The values of the aerodynamic angles in the planar motion are highlighted in eq. (3.5). The demonstration is given in appendix B. As expected, the body frame is obtained by a positive rotation of γ_v about the second axis, followed by a negative rotation about θ about the same axis.

$$\begin{cases} \alpha = \gamma_v - \theta \\ \beta = 0 \\ \kappa = 0 \end{cases} \tag{3.5}$$

The rocket is subjected to three forces: gravity, thrust and aerodynamic force. In this work, the gravity acceleration magnitude g is considered constant:

$$\mathbf{g}_e = \begin{bmatrix} -g & 0 & 0 \end{bmatrix}^T$$

The propulsive force depends on the throttle value η, for a given maximum thrust Γ_{max}, and its direction is the first body axis:

$$\mathbf{F}_e^P = \eta \Gamma_{max} \hat{\mathbf{b}}_{1,e}$$

The aerodynamic force depends on the rocket velocity and its magnitude $||\mathbf{v}_e||$, on the atmospheric density ρ and on the rocket attitude with respect to the wind:

$$\mathbf{F}_e^A = \frac{1}{2}\rho ||\mathbf{v}_e||^2 \left[R_b^e (G_1 \hat{\mathbf{b}}_{1,b} + G_2 \hat{\mathbf{b}}_{2,b} + G_3 \hat{\mathbf{b}}_{3,b}) \right]$$

It is denoted with G_i the aerodynamic coefficient in the i-th body direction that varies based on α, β, κ and the Mach number [23]. Furthermore, the Mars atmospheric model is exponential and it depends on the altitude r_1 (the first component of the position vector in the inertial frame) [57]:

$$\rho(h) = \rho_0 e^{-\frac{r_1}{R}}, \quad V_s = const$$

where ρ_0 is the reference density, R the reference altitude, V_s the speed of sound. The mass flow rate of the system depends on the throttle value:

$$\dot{m} = -\eta \frac{\Gamma_{max}}{V_{ex}}$$

where V_{ex} is the exhaust velocity of the engine. To the aim of keeping the model relatively simple, the complexity of a full 6 DoFs model is avoided since it would require the modeling of the rotational dynamics; instead, the attitude is considered as an external input. Moreover, it can be noted that, except for the aerodynamic force, all the other terms do not depend on the aerodynamic angles and only the first body vector is needed from the rocket attitude information. For this reason two sets of controls can be defined:

$$\mathbf{u} = \begin{bmatrix} \eta & \nu & \theta & \zeta \end{bmatrix}^T \qquad \text{or} \qquad \mathbf{u} = \begin{bmatrix} \eta & \hat{\mathbf{b}}_{1,e} \end{bmatrix}^T$$

The first one is more general and can be always used; the second set can be useful when the aerodynamic term is not considered, taking care of constraining the body vector to have unity norm. In the second case, all the dynamics is polynomial and GAL can be applied. In contrast to [23], where the aerodynamic angles are used as DoFs, the attitude angles are used in this work, preventing the singularity that occurs when the velocity becomes zero. In fact, the rocket attitude must be recovered to define the direction of the forces with respect to the inertial frame, but to obtain it from the aerodynamic angles, the wind angles, which are not defined for null velocity (eq. (3.4)), must be known.

3.2. Equations of Motion

In this section, the equations of motion are derived; to simplify the notation, vectors without subscript are considered as expressed in the inertial reference frame. Three different formulations are presented, but, in order to keep a similar shape between them, some common choices are made. The 3D dynamics of the rocket can be fully represented by the position vector $\mathbf{r}$, the velocity vector $\mathbf{v}$ and the mass m, such that the state can be grouped in the vector $\mathbf{x} = [\mathbf{r}, \mathbf{v}, m]^T$ and the differential equations read as follows, with the selected control vector $\mathbf{u}$:

$$\frac{d}{dt}\mathbf{x}(t) = \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{g} + \dfrac{\mathbf{F}^P(\mathbf{x}(t), \mathbf{u}(t))}{m(t)} + \dfrac{\mathbf{F}^A(\mathbf{x}(t), \mathbf{u}(t))}{m(t)} \\ -\eta(t)\dfrac{\Gamma_{max}}{V_{ex}} \end{bmatrix}$$

Mass Reciprocal A new variable is introduced: $m_r = 1/m$. Its dynamics can be derived, knowing the expression of the mass flow rate:

$$\frac{d}{dt}m_r(t) = \frac{d}{dt}\left(\frac{1}{m(t)}\right) = -\frac{1}{m^2(t)}\frac{d}{dt}m(t) = -m_r^2(t)\left(-\eta(t)\frac{\Gamma_{max}}{V_{ex}}\right) = m_r^2(t)\eta(t)\frac{\Gamma_{max}}{V_{ex}}$$

In this way, the term $1/m$ in the dynamics can be substituted by m_r, making the non-linearity polynomial instead of hyperbolic. Even if the differential equation associated with the mass, which is linear with respect to the throttle, is now replaced by a third-order polynomial term ($m_r^2\eta$), one could take advantage of this since, when no aerodynamic force is considered and when $\mathbf{u} = [\eta, \hat{\mathbf{b}}_{1,e}]^T$, the dynamics is completely polynomial. Therefore, GAL with Legendre polynomials can be applied.

Control Rate In section 2.1, it is explained that a generic system can be expressed in a control-affine form as in eq. (2.10), allowing one to obtain a bilinear model, when the other hypotheses are satisfied. For this reason, the original control is included in the state and its rate of change is considered as the new control:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} & \mathbf{v} & m_r & \eta & \nu & \theta & \zeta \end{bmatrix}^T, \quad \mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T = \begin{bmatrix} \dot{\eta} & \dot{\nu} & \dot{\theta} & \dot{\zeta} \end{bmatrix}^T$$

or

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} & \mathbf{v} & m_r & \eta & \hat{\mathbf{b}}_1^T \end{bmatrix}^T, \quad \mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T = \begin{bmatrix} \dot{\eta} & \dot{\hat{\mathbf{b}}}_1^T \end{bmatrix}^T$$

With these premises, it is possible to define the three models to be studied.

3.2.1. 1D Model with Aerodynamics

In this model (denoted as 1D-aero) only the vertical motion is considered, that is, $\nu = \theta = \zeta = 0$, or $\hat{\mathbf{b}}_1 = \hat{\mathbf{e}}_1$; furthermore, only the vertical component of the velocity is different

from zero. Since this is a particular case of the 2D motion, eq. (3.5) can be applied, resulting in the following value of the angle of attack:

$$\begin{cases} \alpha = 0^\circ & \text{if} \quad v_1 > 0 \\ \alpha = 180^\circ & \text{if} \quad v_1 < 0 \end{cases}$$

where the case of zero velocity is undefined, but also not a case of interest since the aerodynamic force would be zero. Given the bounds on the angle of attack, it is clear that only a negative vertical velocity can be accepted (in good agreement with the case of descent phase, under study). Any horizontal component of the aerodynamic force is neglected in order to keep the motion 1D. The state, the controls and the equations of motion read as:

$$\mathbf{x} = \begin{bmatrix} r_1 & v_1 & m_r & \eta \end{bmatrix}^T, \quad u = \dot{\eta}$$

and

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), u(t)) = \begin{bmatrix} v_1(t) \\ -g + m_r(t)\eta(t)\Gamma_{max} + m_r(t)F_1^A(\mathbf{x}(t)) \\ m_r^2(t)\eta(t)\dfrac{\Gamma_{max}}{V_{ex}} \\ u(t) \end{bmatrix}_{\nu=\theta=\zeta=v_2=v_3=0}$$

3.2.2. 2D Model with Aerodynamics

In this model (denoted as 2D-aero) a 2D motion in the $1-3$ plane is considered, that is, $\nu = \zeta = v_2 = 0$; α can be recovered from eq. (3.5), while the other aerodynamic angles are zero, as explained before; any out-of-plane component of the aerodynamic force is neglected to keep the motion 2D. Therefore:

$$\mathbf{x} = \begin{bmatrix} r_1 & r_3 & v_1 & v_3 & m_r & \eta & \theta \end{bmatrix}^T, \quad \mathbf{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T = \begin{bmatrix} \dot{\eta} & \dot{\theta} \end{bmatrix}^T$$

and

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) = \begin{bmatrix} v_1(t) \\ v_2(t) \\ -g + m_r(t)F_1^P(\mathbf{x}(t)) + m_r(t)F_1^A(\mathbf{x}(t)) \\ m_r(t)F_2^P(\mathbf{x}(t)) + m_r(t)F_2^A(\mathbf{x}(t)) \\ m_r^2(t)\eta(t)\dfrac{\Gamma_{max}}{V_{ex}} \\ \mathbf{u}(t) \end{bmatrix}_{\nu=\zeta=v_2=0}$$

3.2.3. 3D Model without Aerodynamics

Finally, the full 3D motion is considered, but, given its higher degree of complexity, the aerodynamic force is completely ignored. This model is denoted as 3D-noAero. In this case, the thrust vector can be used instead of the attitude angles, as explained before. Therefore:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} & \mathbf{v} & m_r & \eta & \hat{\mathbf{b}}_1^T \end{bmatrix}^T, \quad \mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T = \begin{bmatrix} \dot{\eta} & \dot{\hat{\mathbf{b}}}_1^T \end{bmatrix}^T$$

and

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) = \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{g} + m_r(t)\mathbf{F}^P(\mathbf{x}(t)) \\ m_r^2(t)\eta(t)\dfrac{\Gamma_{max}}{V_{ex}} \\ \mathbf{u}(t) \end{bmatrix}$$

3.3. Benchmark Rocket Landing Formulation

The cost function J to be minimized in a fuel-optimal problem is the fuel mass consumption; this is equivalent to minimize the integral of the throttle value [23]:

$$J(\eta(t), t_f) = \int_0^{t_f} \eta(t)dt$$

The benchmark OCP is stated in eq. (3.6), where the state and its dynamics depend on the selected model and the constraint on the first body vector is applied to its squared norm, since the derivative is linear with respect to the vector itself. Note that some constraints can be ignored because implicitly satisfied, depending on the model under study. For example, in the 2D-aero model the body vector depends on the attitude angles, but the

constraint on its norm is always satisfied through orthonormality of DCMs.

$$\text{find} \min_{\mathbf{u}(t),t_f} J(\eta(t),t_f) \text{ s.t.} \begin{cases} \frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)) \\ \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{r}(t_f) = \mathbf{r}_f \\ \mathbf{v}(t_f) = \mathbf{v}_f \\ \eta_{min} \leq \eta(t) \leq \eta_{max} \\ \alpha_{min} \leq \alpha(\mathbf{x}(t)) \leq \alpha_{max} \\ \beta_{min} \leq \beta(\mathbf{x}(t)) \leq \beta_{max} \\ \mathbf{u}_{min} \leq \mathbf{u}(t) \leq \mathbf{u}_{max} \\ \left|\left|\hat{\mathbf{b}}_1(t)\right|\right|^2 = 1 \end{cases} \tag{3.6}$$

The costate vector $\boldsymbol{\lambda} = [\lambda_{r_1}, \ldots, \lambda_{\hat{b}_{13}}]^T$ is defined; assuming that the problem is not abnormal ($\lambda_0 = 0$), the Hamiltonian and the augmented Hamiltonian functions read as:

$$H = \eta + \boldsymbol{\lambda}^T \mathbf{f}$$

$$\begin{aligned} H^a = H &+ \mu_{\eta,1}(\eta_{min} - \eta) + \mu_{\eta,2}(\eta - \eta_{max}) + \mu_{\alpha,1}(\alpha_{min} - \alpha) + \mu_{\alpha,2}(\alpha - \alpha_{max}) + \\ &+ \mu_{\beta,1}(\beta_{min} - \beta) + \mu_{\beta,2}(\beta - \beta_{max}) + \boldsymbol{\mu}_{u,1}^T(\mathbf{u}_{min} - \mathbf{u}) + \boldsymbol{\mu}_{u,2}^T(\mathbf{u} - \mathbf{u}_{max}) + \\ &+ \mu_{||\hat{\mathbf{b}}_1||}\left(\left|\left|\hat{\mathbf{b}}_1\right|\right|^2 - 1\right) \end{aligned}$$

Some necessary conditions for optimality can be derived by applying PMP. The right-hand side of the dynamics does not depend explicitly on time and, therefore, nor does the Hamiltonian. This means that it is constant along the optimal trajectory and, for the transversality condition of free final time problems, $H(t_F) = 0$.

$$H(t) = 0, \qquad \forall t \in [0, t_F] \tag{3.7}$$

The partial derivatives of the Hamiltonian with respect to the control variables are obtained in order to find a minimum according to PMP. Depending on the formulation:

$$\frac{\partial H}{\partial u_1} = \lambda_\eta, \qquad \frac{\partial H}{\partial u_2} = \lambda_\nu, \qquad \frac{\partial H}{\partial u_3} = \lambda_\theta, \qquad \frac{\partial H}{\partial u_4} = \lambda_\zeta$$

or

$$\frac{\partial H}{\partial u_1} = \lambda_\eta, \qquad \frac{\partial H}{\partial u_2} = \lambda_{\hat{b}_{11}}, \qquad \frac{\partial H}{\partial u_3} = \lambda_{\hat{b}_{12}}, \qquad \frac{\partial H}{\partial u_4} = \lambda_{\hat{b}_{13}}$$

Since the Hamiltonian has a linear relation with respect to the control, when the derivatives are different from zero, the associated control is at the boundary:

$$u_i = \frac{u_{min,i} + u_{max,i}}{2} + \text{sign}\left(\frac{\partial H}{\partial u_i}\right)\left[\frac{u_{min,i} + u_{max,i}}{2}\right] \quad \text{if} \quad \frac{\partial H}{\partial u_i} \neq 0$$

However, singular arcs may be present; therefore, further investigation is needed. Consider $\partial H/\partial u_1 = \lambda_\eta$: if it is constant and equal to zero in a finite range of time $[t_1, t_2]$, its derivative must be zero.

$$\begin{aligned} \frac{d}{dt}\lambda_\eta = -\frac{\partial H^a}{\partial \eta} = -1 - \boldsymbol{\lambda}_{\mathbf{v}}^T m_r \frac{\partial \mathbf{F}^P}{\partial \eta} - \lambda_{m_r} m_r^2 \frac{\Gamma_{max}}{V_{ex}} + \\ + \mu_{\eta,1} - \mu_{\eta,2} = \sigma(\mathbf{x}, \boldsymbol{\lambda}) + \mu_{\eta,1} - \mu_{\eta,2} = 0 \end{aligned} \quad \text{if} \quad \lambda_\eta = 0 \quad \forall t \in [t_1, t_2]$$

If $\sigma(\mathbf{x}, \boldsymbol{\lambda}) \neq 0$ in $[t_1, t_2]$, either $\mu_{\eta,1}$ or $\mu_{\eta,2}$ are strictly positive and $\eta = \eta_{min}$ or $\eta = \eta_{max}$.

$$\eta(t) = const \Leftrightarrow \frac{d}{dt}\eta(t) = 0 \Leftrightarrow u_1(t) = 0 \quad \text{if} \quad \frac{\partial H}{\partial u_1} = 0 \quad \text{and} \quad \sigma(\mathbf{x}, \boldsymbol{\lambda}) \neq 0 \quad \forall t \in [t_1, t_2]$$

Other considerations could be made, for example, by considering further derivatives of the costate or not disregarding the abnormality, but they are beyond the scope of this work.

3.4. Koopman-based Rocket Landing Formulation

The OCP stated in the previous section is now reformulated in order to be compliant with a Koopman model. Suppose to have a dictionary of functions defined by $\boldsymbol{\psi}(\mathbf{x}) = [\psi_1(\mathbf{x}), \dots, \psi_q(\mathbf{x})]^T$ and a dynamical model:

$$\frac{d}{dt}\boldsymbol{\psi}(\mathbf{x}(t)) = \mathbf{f}(\boldsymbol{\psi}(\mathbf{x}(t)), \mathbf{u}(t))$$

where $\mathbf{f}(\boldsymbol{\psi}, \mathbf{u})$ is either a bilinear model (eq. (2.9)) or a linear model (eq. (2.11)) with respect to the control. Furthermore, the original state can be recovered through the

projection matrix: $\mathbf{x} = P\boldsymbol{\psi}(\mathbf{x})$; it is denoted with $P_{\mathbf{x}_i}$ the matrix that contains the rows of P associated with the vector of states $\mathbf{x}_i$. A new set of independent states, representing the values of the dictionary functions, is defined as $\mathbf{L} = [L_1, \dots, L_q]^T$. The cost function can now be expressed as follows:

$$J(\mathbf{L}(t), t_f) = \int_0^{t_f} P_\eta \mathbf{L}(t) dt$$

The Koopman OCP is stated in eq. (3.8). The same consideration as before, on the redundancy of constraints, is applicable in this case. It is pointed out that, whereas for the benchmark problem the initial state could be only partially constrained, leaving free, for example, the initial throttle level, in this case all the initial state must be constrained. In fact, its components are interdependent through the expression of the dictionary functions with respect to the original state; they cannot be left free without imposing further constraints. For comparison purposes, the initial state is fully constrained in the benchmark formulation as well.

$$\text{find} \min_{\mathbf{u}(t), t_f} J(\mathbf{L}(t), t_f) \text{ s.t. } \begin{cases} \dfrac{d}{dt}\mathbf{L}(t) = \mathbf{f}(\mathbf{L}(t), \mathbf{u}(t)) \\ \mathbf{L}(0) = \boldsymbol{\psi}(\mathbf{x}_0) \\ P_{\mathbf{r}}\mathbf{L}(t_f) = \mathbf{r}_f \\ P_{\mathbf{v}}\mathbf{L}(t_f) = \mathbf{v}_f \\ \eta_{min} \leq P_\eta \mathbf{L}(t) \leq \eta_{max} \\ \alpha_{min} \leq \alpha(\mathbf{L}(t)) \leq \alpha_{max} \\ \beta_{min} \leq \beta(\mathbf{L}(t)) \leq \beta_{max} \\ \mathbf{u}_{min} \leq \mathbf{u}(t) \leq \mathbf{u}_{max} \\ \left|\left|P_{\hat{\mathbf{b}}_1}\mathbf{L}(t)\right|\right|^2 = 1 \end{cases} \tag{3.8}$$

Assuming again no abnormality, the Hamiltonian can be defined, with $\boldsymbol{\lambda} \in \mathbb{R}^q$ being the new vector of costates:

$$H = P_\eta \mathbf{L} + \boldsymbol{\lambda}^T \mathbf{f}$$

Similarly as before, some necessary conditions for optimality can be derived. The Hamiltonian has no explicit dependence on time, therefore eq. (3.7) still holds. The partial derivatives of the Hamiltonian with respect to the control variables are now addressed.

For a bilinear model:

$$\frac{\partial H}{\partial u_1} = \boldsymbol{\lambda}^T K_1 \mathbf{L}, \qquad \frac{\partial H}{\partial u_2} = \boldsymbol{\lambda}^T K_2 \mathbf{L}, \qquad \frac{\partial H}{\partial u_3} = \boldsymbol{\lambda}^T K_3 \mathbf{L}, \qquad \frac{\partial H}{\partial u_4} = \boldsymbol{\lambda}^T K_4 \mathbf{L}$$

For a linear model:

$$\frac{\partial H}{\partial \mathbf{u}} = \left[\boldsymbol{\lambda}^T K_c\right]^T$$

In both cases, the dependence of the Hamiltonian on the control is linear. Therefore, the optimal control is at the boundaries if no singular arcs are present; otherwise, further considerations (including abnormality) must be made. In any case, little meaning is expected to be found, given that the Koopman model is purely data-driven and many of its states might have no physical interpretation: their only purpose is to approximate well enough the original state dynamics.

3.4.1. Koopman-based Linear Program

If a Koopman linear control model can be obtained, the OCP can be rewritten, after discretization, as the LP in eq. (3.9).

$$\text{find} \ \min_{\mathbf{x}_{LP}} \mathbf{d}_f^T \mathbf{x}_{LP} \ \text{s.t.} \ \begin{cases} D\mathbf{x}_{LP} \leq \mathbf{d} \\ D_{eq}\mathbf{x}_{LP} = \mathbf{d}_{eq} \end{cases} \tag{3.9}$$

In particular, the time domain is discretized into segments of the same length, such that $n_s + 1$ nodes are obtained, separated by a constant time step Δt. The optimization variables are grouped in the vector $\mathbf{x}_{LP} \in \mathbb{R}^{(n_s+1)(q+m)}$ that includes the lifted state and the control at each node:

$$\mathbf{x}_{LP} = \begin{bmatrix} \mathbf{L}_0^T & \mathbf{u}_0^T & \cdots & \mathbf{L}_i^T & \mathbf{u}_i^T & \cdots & \mathbf{L}_{n_s}^T & \mathbf{u}_{n_s}^T \end{bmatrix}^T$$

It must be noted that the final time is fixed, because otherwise the problem would not be a LP; for the same reason, the constraint on the body vector norm (or other constraints that do not depend linearly on the state) cannot be imposed, thus this formulation can only be applied to models that do not require explicit imposition of these constraints. The vectors and matrices of eq. (3.9) are constructed hereafter.

Cost Function Instead of minimizing the integral of the throttle, which would require the use of quadrature rules, the final mass reciprocal is minimized (maximization of final mass). Therefore, the cost function vector only requires the reconstruction from the final state:

$$\mathbf{d}_f = \begin{bmatrix} \mathbf{0} \\ P_{m_r}^T \\ 0 \end{bmatrix}$$

Equality Constraints The equality constraints must take into account the initial conditions, the dynamics and the endpoint constraints. More explicitly, the dynamics constraints impose a single-step propagation step:

$$K_0^d \mathbf{L}_i + K_c^d \mathbf{u}_i - \mathbf{L}_{i+1} = \mathbf{0} \quad \forall i \in \{1, \dots, n_s - 1\}$$

where K_0^d and K_c^d are the matrices representing the Koopman discrete operator with time step Δt. Matrix D_{eq} and vector $\mathbf{d}_{eq}$ take the following expression:

$$C_{eq} = \begin{bmatrix} I_{q\times q} & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \\ K_0^d & K_c^d & -I_{q\times q} & 0 & 0 & & 0 & 0 & 0 & 0 \\ 0 & 0 & K_0^d & K_c^d & -I_{q\times q} & & 0 & 0 & 0 & 0 \\ & & & & & \ddots & & & & \\ 0 & 0 & 0 & 0 & 0 & & K_0^d & K_c^d & -I_{q\times q} & 0 \\ 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & P_{\mathbf{r}} \\ 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & P_{\mathbf{v}} \end{bmatrix}, \quad \mathbf{d}_{eq} = \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}_0) \\ \mathbf{0} \\ \mathbf{r}_f \\ \mathbf{v}_f \end{bmatrix}$$

Inequality Constraints As they are defined in eq. (3.9), the inequality constraints are only upper bounds on a linear combination of the optimization variables. Therefore, to take both the lower and upper bounds into account, the terms can be rewritten as follows:

$$D = \begin{bmatrix} -D_l \\ D_u \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} -\mathbf{d}_l \\ \mathbf{d}_u \end{bmatrix}$$

with

$$D_l = D_u = \begin{bmatrix} 0 & \mathbf{I}_{m\times m} & 0 & 0 & & 0 & 0 \\ 0 & 0 & P & 0 & & 0 & 0 \\ 0 & 0 & 0 & \mathbf{I}_{m\times m} & & 0 & 0 \\ & & & & \ddots & & \\ 0 & 0 & 0 & 0 & & P & 0 \\ 0 & 0 & 0 & 0 & & 0 & \mathbf{I}_{m\times m} \end{bmatrix}, \quad \mathbf{d}_l = \begin{bmatrix} \mathbf{u}_{min} \\ \mathbf{x}_{min} \\ \mathbf{u}_{min} \\ \vdots \\ \mathbf{x}_{min} \\ \mathbf{u}_{min} \end{bmatrix}, \quad \mathbf{d}_u = \begin{bmatrix} \mathbf{u}_{max} \\ \mathbf{x}_{max} \\ \mathbf{u}_{max} \\ \vdots \\ \mathbf{x}_{max} \\ \mathbf{u}_{max} \end{bmatrix}$$

It is pointed out that no bounds are imposed on the initial state, as it is already constrained by the equality constraints. In this case the bounds are imposed on all the state through matrix P, but any subset of the state or any linear combination of the lifted state can be bounded by changing the reconstruction matrix.

4 | Numerical Simulations

This chapter focuses on presenting the numerical solution of the OCPs described in chapter 3. In section 4.1 a Koopman model for the dynamics is obtained, following the approaches discussed in section 2.2, and its accuracy is evaluated; in section 4.2 the benchmark and Koopman OCPs are solved and the optimality is discussed. Some rocket-related constants, used throughout the chapter, are defined in table 4.1 [23, 57].

	Symbol	**Value**
Gravity acceleration	g	3.71 m/s^2
Exhaust velocity	V_{ex}	1966.07 m/s
Maximum thrust	Γ_{max}	16572.72 N
Reference density	ρ_0	1.57×10^{-2} kg/m^3
Reference altitude	R	9354.50 m
Speed of sound	V_s	220.00 m/s

Table 4.1: Rocket landing environment constants.

4.1. Koopman Model

A separated Koopman model must be obtained for each of the three sets of equations of motion in section 3.2. In order to keep data in the same order of magnitude, all the state is normalized according to the following relation:

$$\bar{x}_i = a_{i,1} x_i + a_{i,2} \quad \forall i \in \{1, \dots, n\}$$

where $\bar{x}_i$ is the i-th state after normalization; the equations of motion are modified accordingly. In particular, the scalars $a_{i,1}$ and $a_{i,2}$ are defined such that, for a normalized state comprised in $[-1, 1]$, the original state belongs to the intervals shown in table 4.2. The vertical position is not expected to go below zero; the vertical velocity is bounded

to be negative since the rocket should not raise its altitude again; the other variables are between typical values or in bounds imposed by their own definition [58, 59].

State	Bounds
r_1	$[0, 2000]$ m
r_2, r_3	$[-2000, 2000]$ m
v_1	$[-300, 0]$ m/s
v_2, v_3	$[-300, 300]$ m/s
m	$[2200, 1800]$ kg
η	$[0, 1]$
θ	$[-180, 180]^\circ$
$\hat{b}_{11}, \hat{b}_{12}, \hat{b}_{13}$	$[-1, 1]$

Table 4.2: Rocket landing bounds on state after normalization.

All the neural networks have almost the same architecture, with $n_h = 3$, $w_h = 100$, $const = 1$, while the output dimension depends on the desired number of basis functions; the regularization parameter depends on the case under study. Data was obtained by propagating random initial conditions sampled uniformly in $[-1, 1]^n$, with a the time step between two consecutive samples $\Delta t = 0.1$[1]. Trajectories were propagated until one of the states exceeded its bounds, to ensure normalization of data. The error is defined similarly as in eq. (2.28), but it was computed separately for each subset of state variables, after de-normalization. Note that the mass was considered, instead of the mass reciprocal, when errors were computed. The Monte Carlo campaigns were carried out over 100 trajectories.

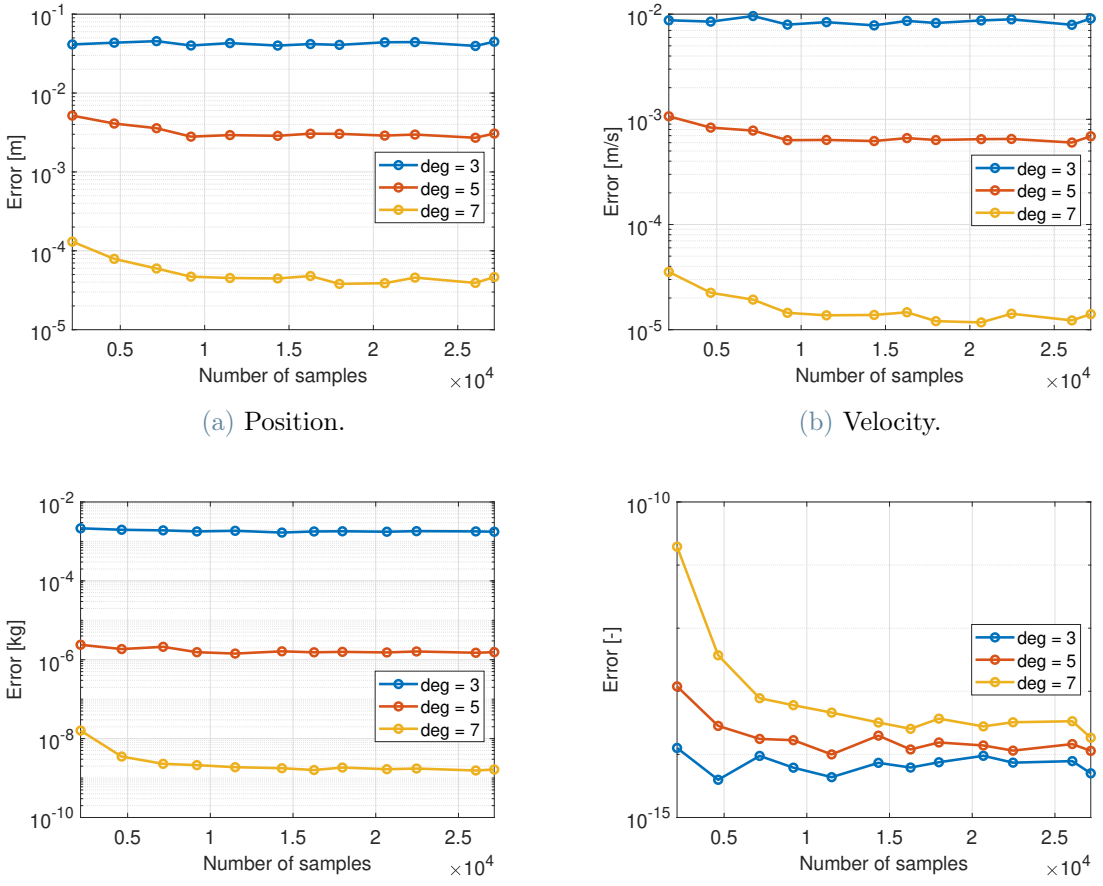
4.1.1. 1D Model with Aerodynamics

Since the nonlinearities are not only polynomial, due to the presence of the aerodynamic force, GAL with Legendre polynomials cannot not be applied: EDMD and dlEDMD are compared. First of all, a convergence analysis is performed for EDMD, for different numbers of samples and polynomial orders; when the needed number of samples is known, the same data set can be used to train the neural network.

Free-dynamics Model The EDMD error is shown for $T = 10$ s and $T = 40$ s in figs. 4.1 and 4.2; three different polynomial orders were studied: 3, 5, and 7. It can be

[1]Dimensionless time, corresponding to ≈ 0.67 s

observed that, in this case, higher-order polynomials do not make the system unstable (so there are no bad closure issues) and, except for the throttle which has an accuracy in the order of 1×10^{-8} at worse, the error decreases with the polynomial order, even when trajectories are propagated for a longer time. This is probably due to the fact that the 1D aerodynamic force (drag) is small compared to the gravity and propulsive forces; thus, the problem can be seen as a polynomial system (which in this case can be well-approximated by Legendre polynomials) with a small non-polynomial perturbation as in [5]. In general, the third-order model appears already at convergence, with a relatively small number of samples (5×10^3). For the other models, the convergence behavior can be observed; in fact, as the number of basis functions increases, more coefficients of the Koopman matrix must be estimated and more data is needed to have good accuracy. When the 40 s trajectories are analyzed, the convergence cannot be observed as the mean error still has a decreasing behavior with the number of samples. However, the seventh-order model has a smaller error than the other ones, even if not at convergence.



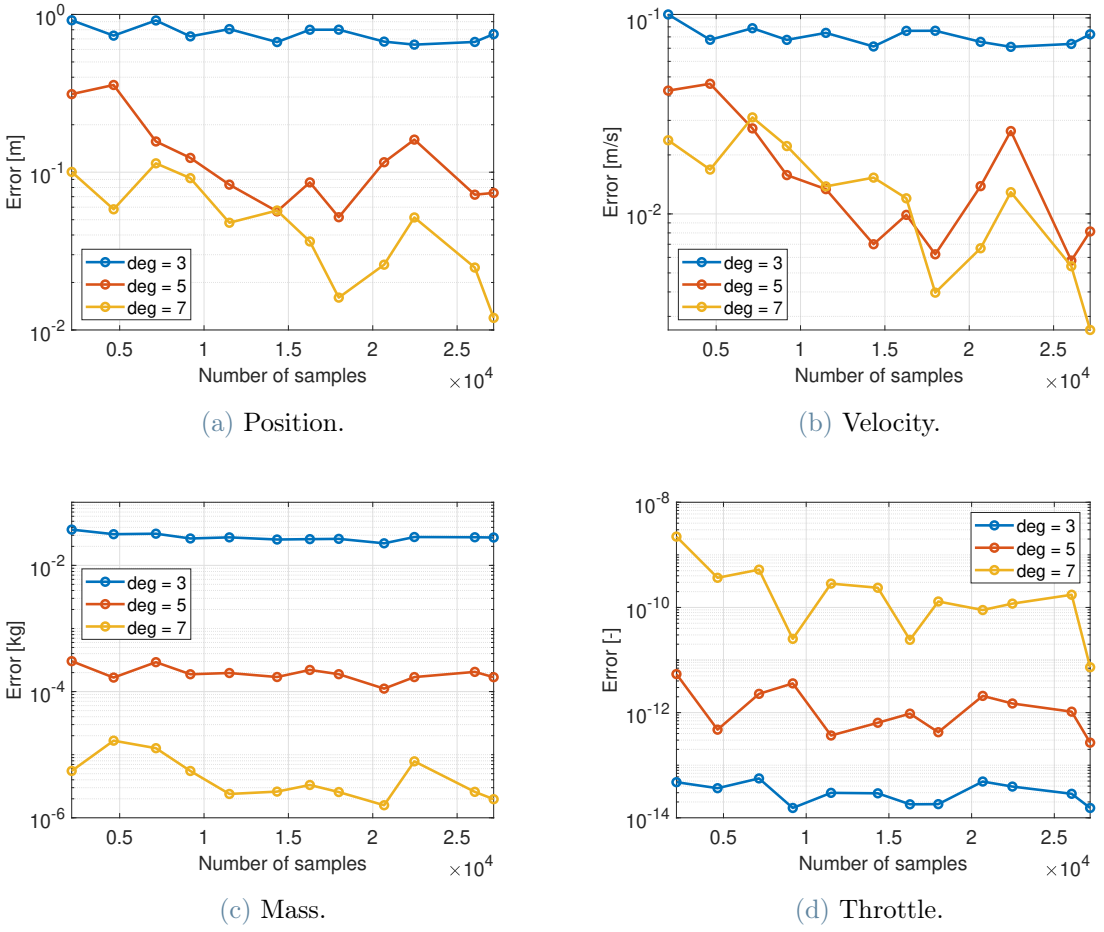
(a) Position.

(b) Velocity.

(c) Mass.

(d) Throttle.

Figure 4.1: Uncontrolled 1D-aero mean error, $T = 10$ s, EDMD.



(a) Position.

(b) Velocity.

(c) Mass.

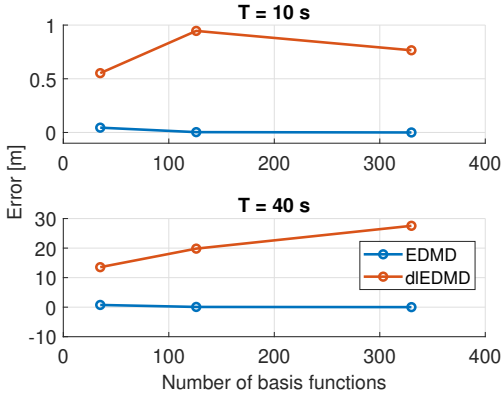
(d) Throttle.

Figure 4.2: Uncontrolled 1D-aero mean error, $T = 40$ s, EDMD.

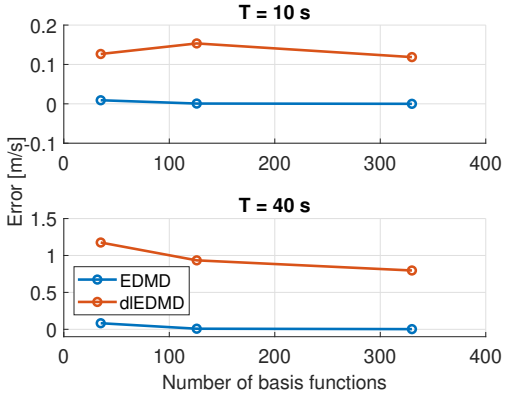
The EDMD model is compared with dlEDMD, using the number of basis functions associated with each polynomial order. The neural network was trained with 2.7×10^4 snapshot pairs, according to the convergence behavior observed for EDMD and regularization was introduced only for the two models with more basis functions, with $r = 0.01$. The mean error for the two methods is shown in fig. 4.3, for different number of basis functions. It can be observed how dlEDMD, in this case, performs worse than EDMD, with a mean error which is more than one order of magnitude higher than EDMD. Moreover, the error is not always decreasing when more basis functions are used, depending on the state variable under study; when an improvement is observed, it is, however, very modest. It is evident that obtaining a good dlEDMD model requires a good setup of the network architecture and of the hyperparameters (number of epochs, initialization of weights, learning rate); in any case, since Legendre polynomials are a proper set of basis functions in this case, it is not expected that dlEDMD performs better than EDMD with a generic dictionary of functions. An example trajectory, showing the propagation of the original dynamics and

of the two Koopman models with 35 basis functions (third-order polynomials), is reported in fig. 4.4.

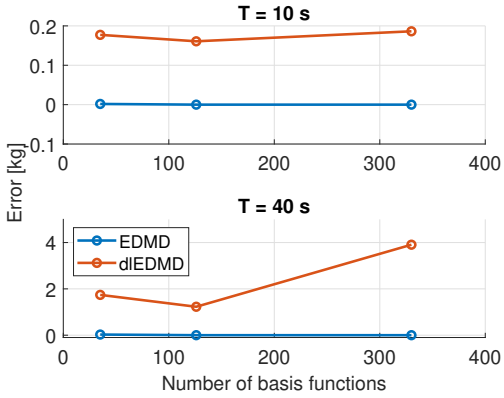
Nevertheless, the dlEDMD method cannot be completely discarded before introducing the control; in fact, even if a controlled model cannot perform better than the associated uncontrolled one, dlEDMD could still have a better accuracy than EDMD for a control model.



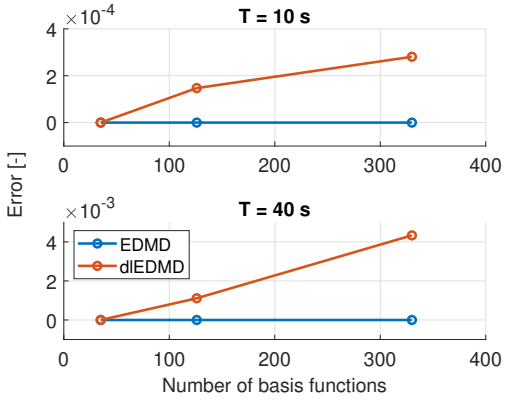
(a) Position.



(b) Velocity.

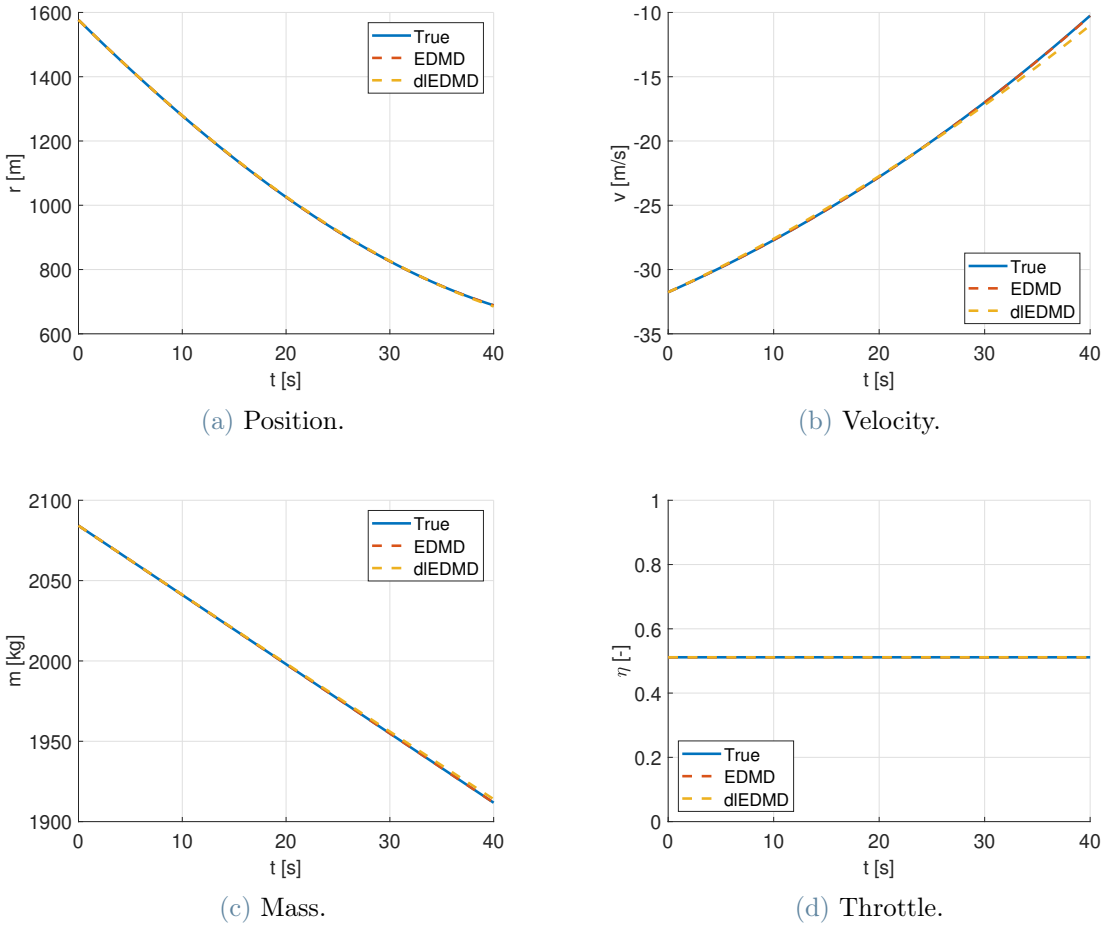


(c) Mass.



(d) Throttle.

Figure 4.3: Uncontrolled 1D-aero mean error, EDMD and dlEDMD.



(a) Position.

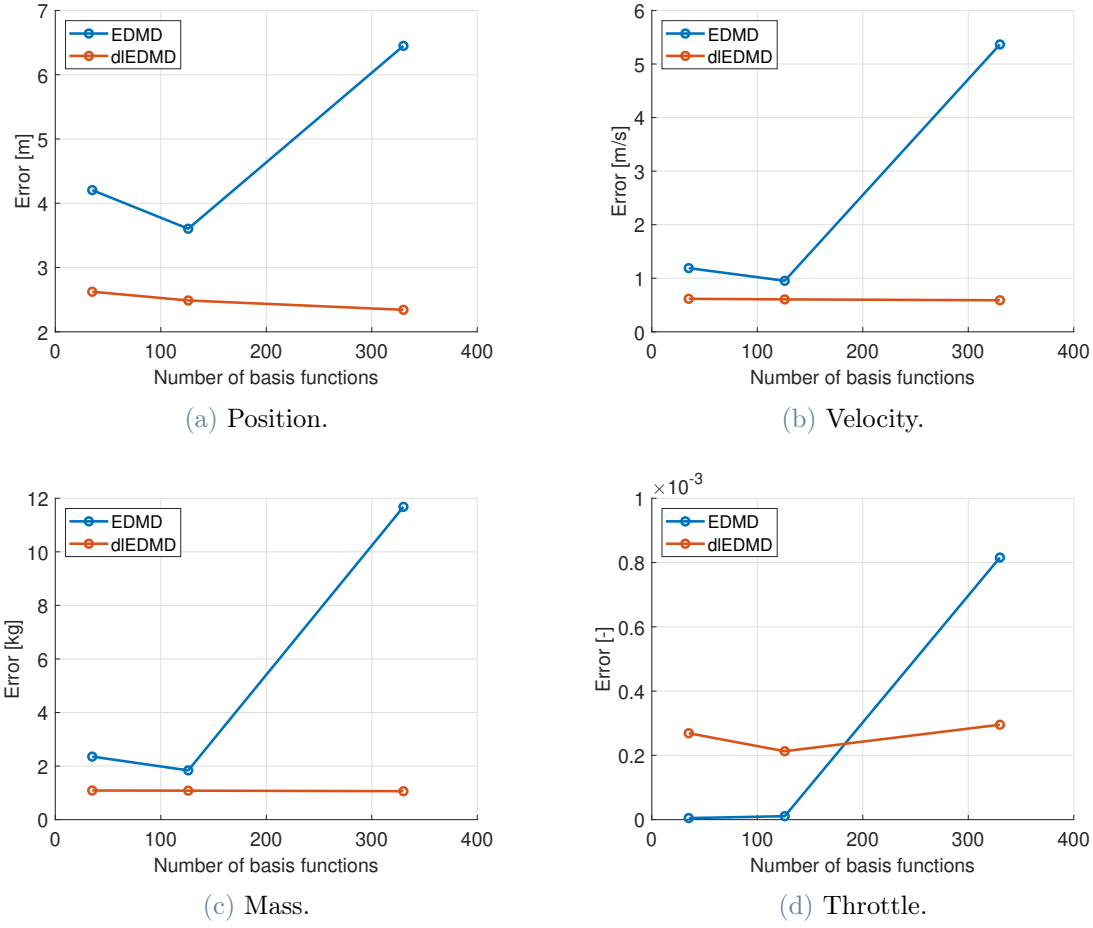
(b) Velocity.

(c) Mass.

(d) Throttle.

Figure 4.4: Uncontrolled 1D-aero example trajectory, 35 basis functions.

Linear Control Model A linear model is obtained following the steps presented in section 2.2, using the same amount of data as in the uncontrolled case. The control (throttle rate of change) is a uniform, random, piecewise-constant signal such that $|u| \leq 0.2$ 1/s, with random duration in the range $[0.2, 1.5]$ s for each piece. The bound on the control is compatible with a typical bandwidth and the same value can be used also for the other dynamical models (same value, possibly different units) [45]. A regularization with $r = 0.01$ was used for dlEDMD. The performances of the two methods are again compared, using different number of basis functions, but the Monte Carlo campaign was carried out on 10 s trajectories, because it is more difficult to obtain longer trajectories with a bounded state, when control is introduced; the results are shown in fig. 4.5. Differently from before, EDMD presents a divergent behavior when more basis functions are used; dlEDMD, on the other hand, is able to learn the dictionary of functions in order to have a better accuracy than EDMD for a linear control model, but with almost no improvement as more functions are used.



(a) Position. (b) Velocity.

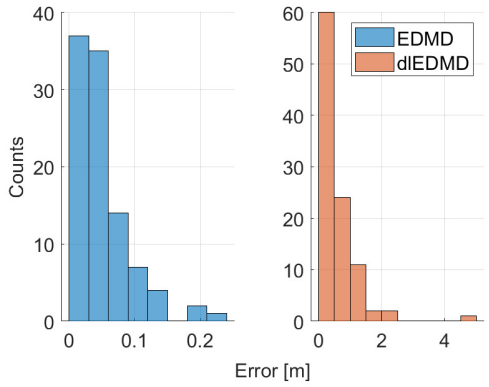
(c) Mass. (d) Throttle.

Figure 4.5: Linear control 1D-aero mean error, $T = 10$ s, EDMD and dlEDMD.

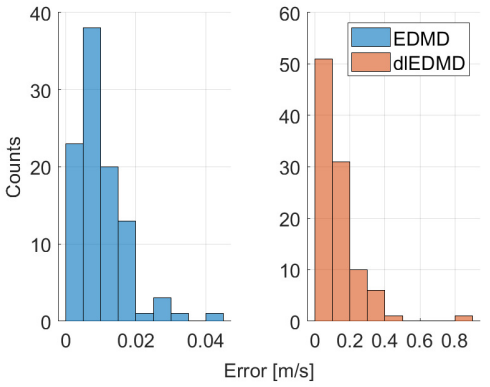
Bilinear Control Model The bilinear model can be obtained after the computation of the free-dynamics part; for simplicity, only the model with 35 basis functions is considered. Since the control vector field of the control-affine dynamics is constant (only the throttle is directly affected by control, and in a linear way: $\dot{\eta} = u$), the projection of the operator onto the Legendre polynomials can be easily computed analytically, without the need for data. However, the procedure in eq. (2.21), with the same amount of data as before, is followed to show its potentiality. Any subspace of Legendre polynomials is Koopman-invariant when considering a constant control vector field (see [3] for the expression of the Jacobian matrix of Legendre polynomials). The same cannot be said for the dlEDMD basis functions, thus the resulting model is, in general, a further approximation with respect to the free-dynamics one.

In fig. 4.6, it can be observed that the bilinear model has the same accuracy of the uncontrolled one. Although this is expected for EDMD, it shows that the dlEDMD basis functions subspace is wide enough that the projection error is negligible; in future works

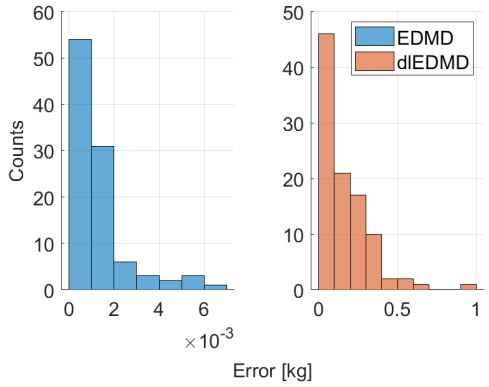
it could be proven that, given the ability of neural networks in approximating generic functions, the projection error of the derivative of the basis functions onto the functions themselves can be bounded. Figure 4.7 presents an example trajectory propagated with the two control models; the EDMD linear model is inaccurate, while the dlEDMD one is almost comparable with the bilinear models.



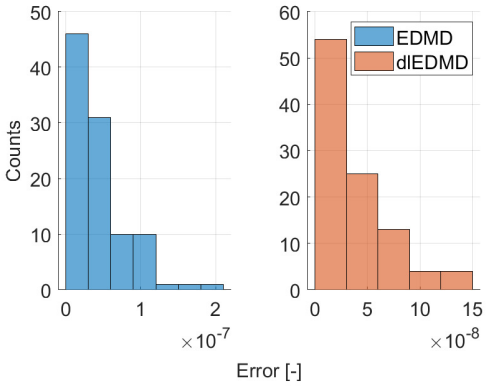
(a) Position. Mean $(0.051, 0.579)$ m.



(b) Velocity. Mean $(0.010, 0.127)$ m/s.

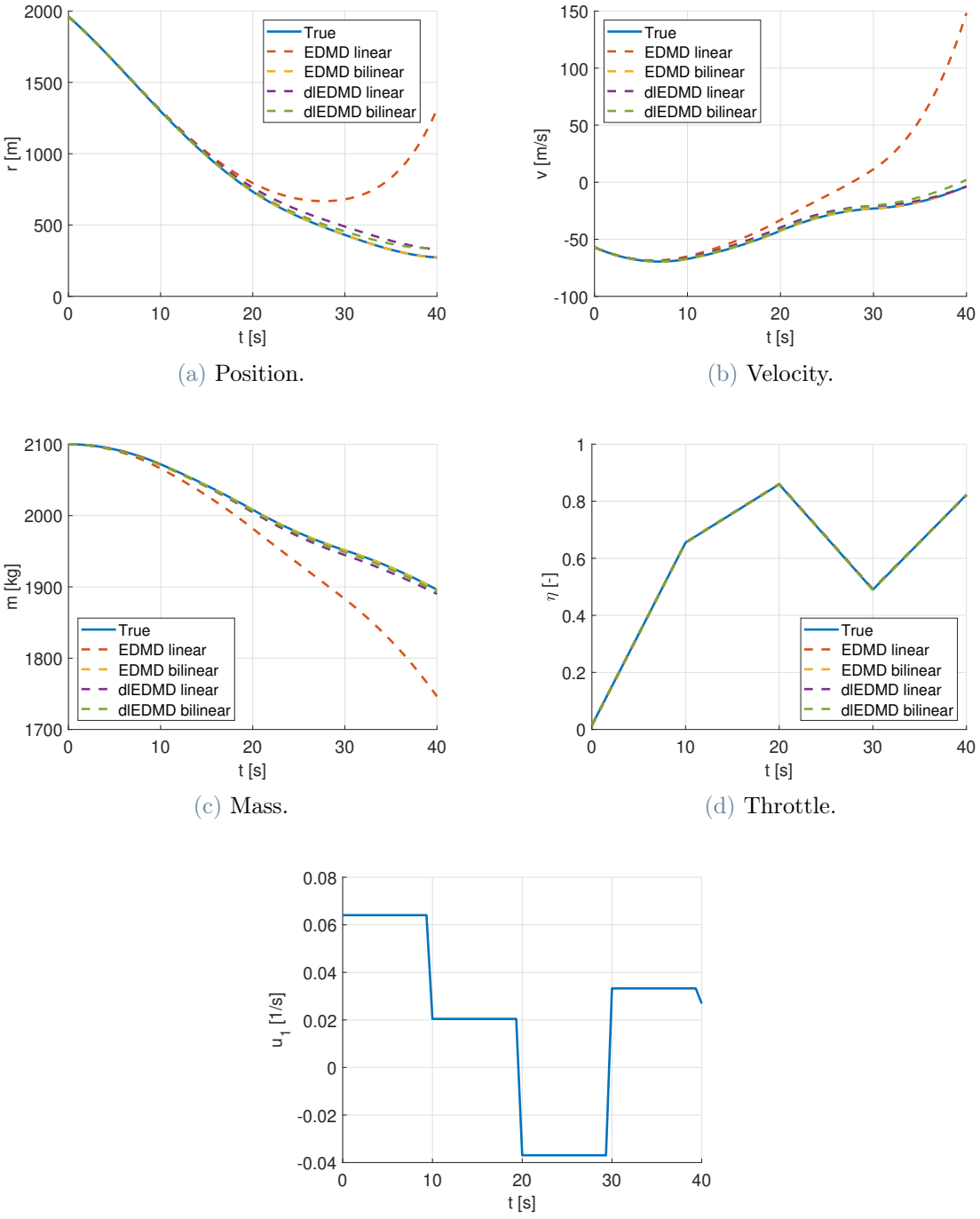


(c) Mass. Mean $(0.01, 0.165)$ kg.



(d) Throttle. Mean $(4.0, 3.7) \times 10^{-8}$.

Figure 4.6: Bilinear control 1D-aero error, $T = 10$ s, 35 basis functions, EDMD and dlEDMD. The mean error is reported, from left to right, for EDMD and dlEDMD.



(a) Position.

(b) Velocity.

(c) Mass.

(d) Throttle.

(e) Control.

Figure 4.7: Controlled 1D-aero example trajectory, 35 basis functions.

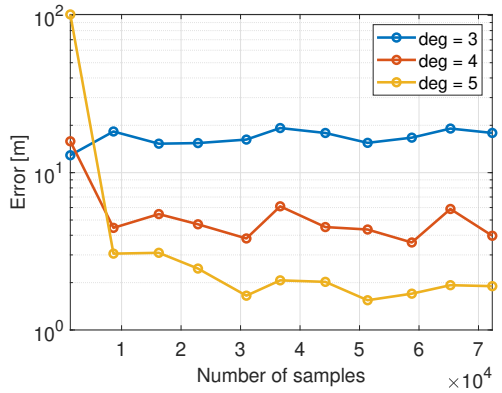
4.1.2. 2D Model with Aerodynamics

The 2D-aero model is a generalization of the 1D one to values of pitch angle different from zero. The Koopman model is obtained, again, with both EDMD and dlEDMD, to

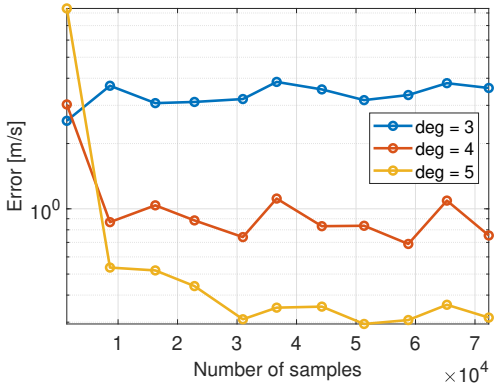
assess wether transversal velocity and aerodynamic forces have an impact on the EDMD accuracy or not.

Free-dynamics Model Similarly as before, a convergence analysis for EDMD is carried out; however, when propagating trajectories, it was ensured that also the angle of attack remained in $[150, 210]°$, so that the aerodynamic force can be defined (section 3.1). Moreover, the maximum polynomial order is 5, to avoid having a very large dictionary of functions; in fact, fifth-order Legendre polynomials for a seven-dimensional system results in 792 basis functions. The results are presented in figs. 4.8 and 4.9. The 10 s trajectories show convergence with increasing number of samples and the highest-order model has better accuracy. In the 40 s trajectories, convergence cannot be observed and, while this issue could be solved by using more data (not done here because of the increase in computational burden), it can be already observed that the fifth-order model exhibits a closure issue which makes the error higher than in the other models, in terms of position and velocity. This is probably caused by the aerodynamic force, which in this case has a magnitude comparable with the other forces, and by the pitch angle, which introduces trigonometric functions in the dynamics.
The dlEDMD method is applied with the lowest number of basis functions used for EDMD (120), with 7.2×10^4 data snapshots and with regularization parameter $r = 0.01$; it was then compared with the corresponding EDMD model. An analysis with variable number of functions was not performed, but, based on the previous results, no great increase in the error is expected: in the worst case, the error should stay in the same order of magnitude. Figure 4.10 shows that dlEDMD is able to achieve a greater accuracy than EDMD, in terms of position and velocity error; the mass, throttle and pitch angle errors are larger than the EDMD ones, but still small enough for the model to be used in practical applications; moreover, it is pointed out that the throttle and pitch angle are easier to predict for EDMD (they are constant when no control is applied).

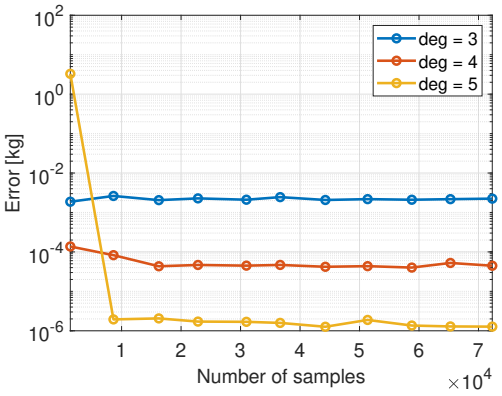
Linear Control Model A model is obtained with both EDMD and dlEDMD, with 120 basis functions. All the controls were generated as in the 1D-aero model, with the same bounds on each control variable. Even if a smaller error might be achieved by dlEDMD with a larger dictionary, it would not be convenient to have a very high-dimensional system in the OCP that must be solved subsequently. The performances are presented in fig. 4.11.



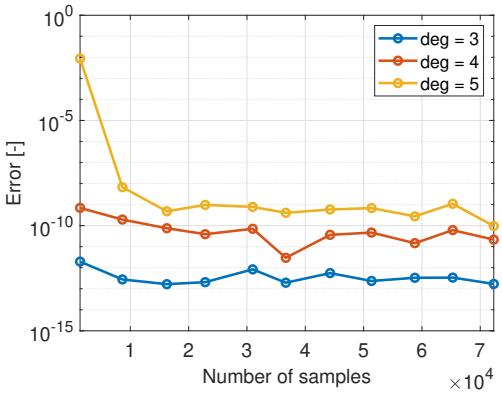
(a) Position.



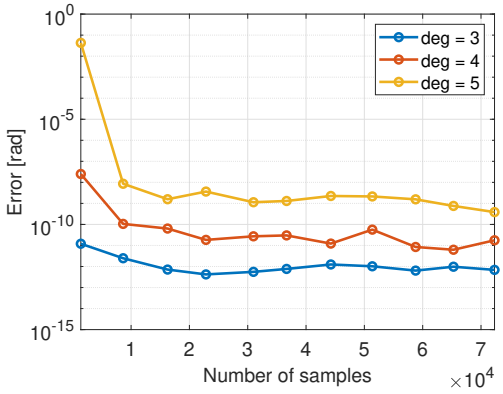
(b) Velocity.



(c) Mass.



(d) Throttle.

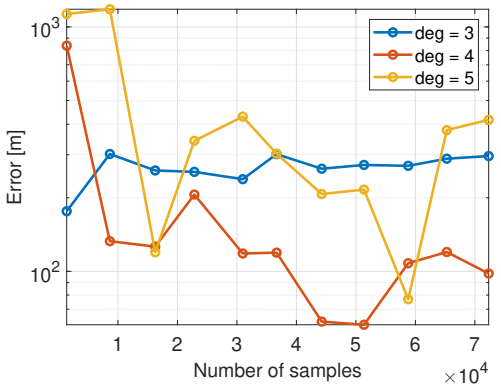


(e) Pitch angle.

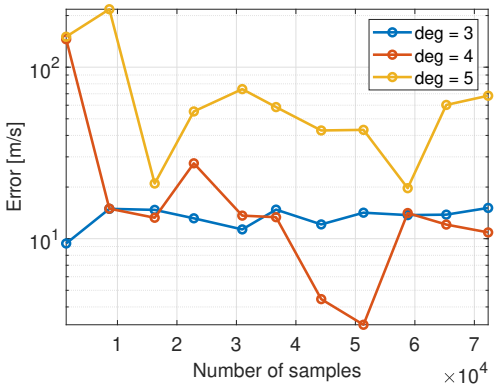
Figure 4.8: Uncontrolled 2D-aero mean error, $T = 10$ s, EDMD.

Bilinear Control Model The bilinear model is obtained similarly as in the 1D-aero case, with 120 basis functions. The results in fig. 4.12 and the example trajectory in fig. 4.13 show that the bilinear model maintains the same error as the free-dynamics model;

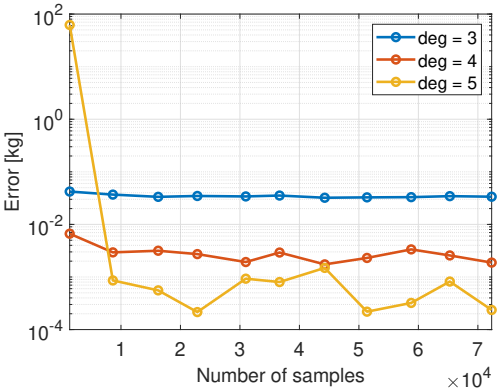
for what concerns the linear control model, while dlEDMD provides a more accurate system than EDMD, none of the two can be used for accurate propagation.



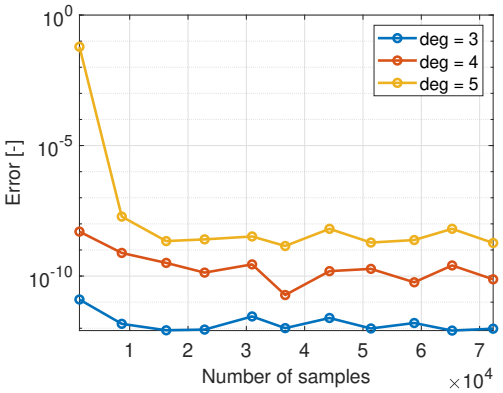
(a) Position.



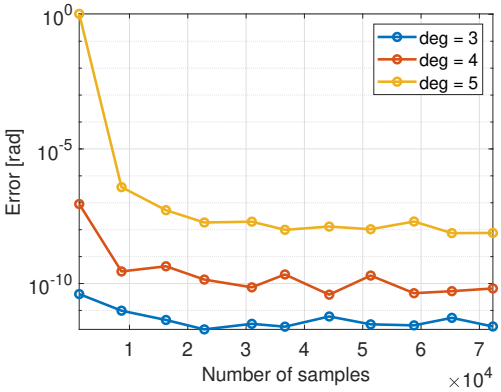
(b) Velocity.



(c) Mass.

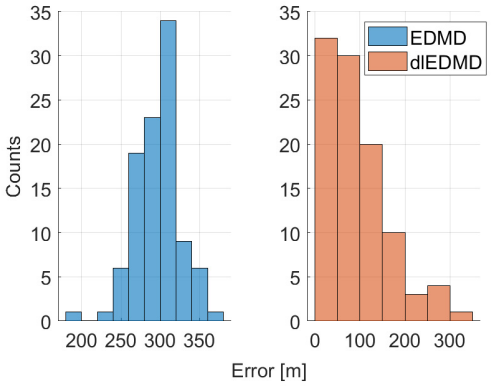


(d) Throttle.

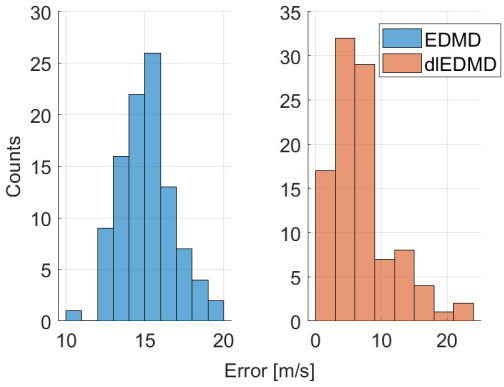


(e) Pitch angle.

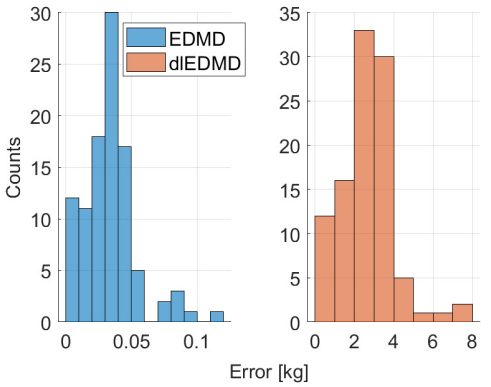
Figure 4.9: Uncontrolled 2D-aero mean error, $T = 40$ s, EDMD.



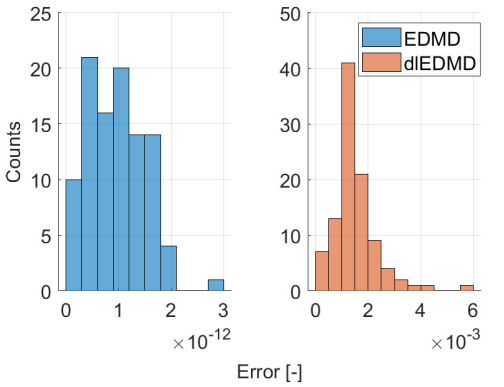
(a) Position. Mean $(296.22, 95.55)$ m.



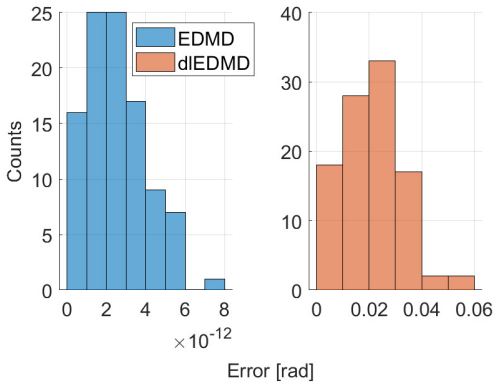
(b) Velocity. Mean $(15.11, 7.03)$ m/s.



(c) Mass. Mean $(0.03, 2.71)$ kg.

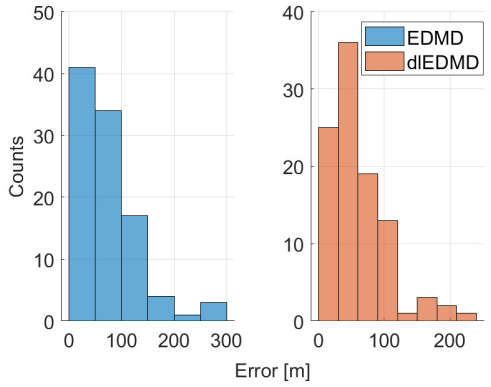


(d) Throttle. Mean $(9.63 \times 10^{-13}, 1.5 \times 10^{-3})$.

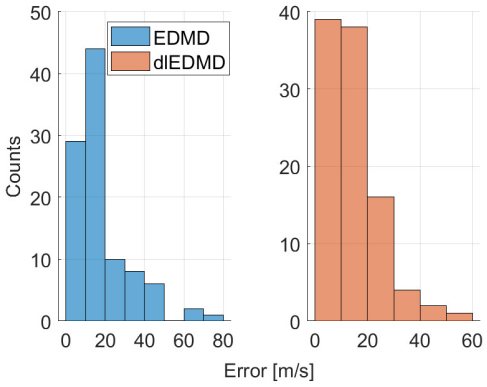


(e) Pitch angle. Mean $(2.53 \times 10^{-12}, 0.02)$ rad.

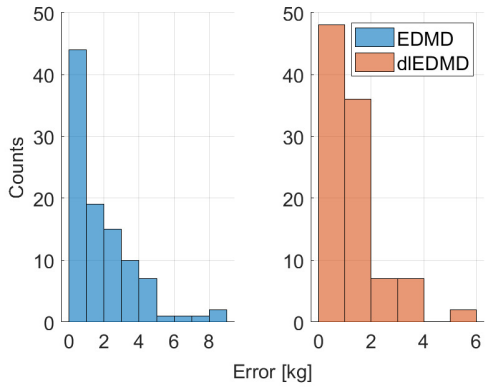
Figure 4.10: Uncontrolled 2D-aero error, $T = 40$ s, 120 basis functions, EDMD and dlEDMD. The mean error is reported, from left to right, for EDMD and dlEDMD.



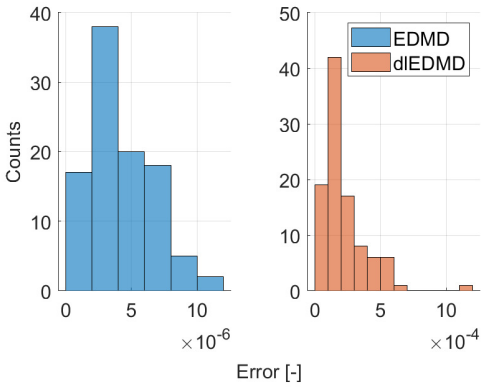
(a) Position. Mean $(72.04, 59.95)$ m.



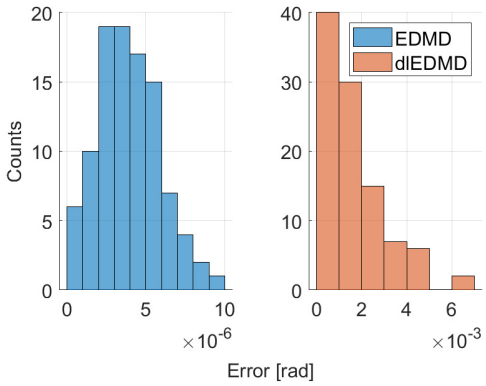
(b) Velocity. Mean $(18.30, 14.44)$ m/s.



(c) Mass. Mean $(1.91, 1.26)$ kg.

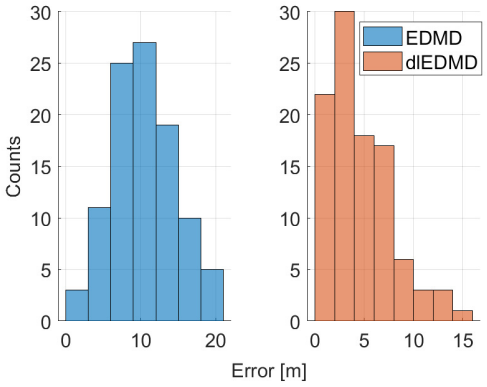


(d) Throttle. Mean $(4.31 \times 10^{-6}, 2.20 \times 10^{-4})$.

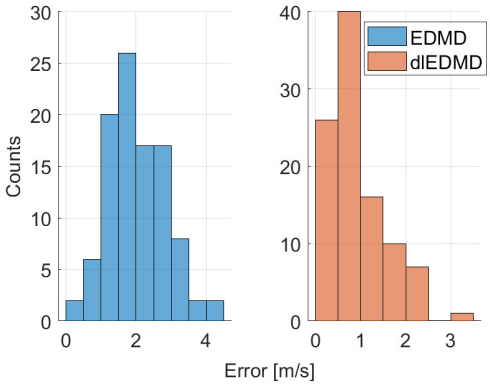


(e) Pitch angle. Mean $(3.88 \times 10^{-6}, 0.002)$ rad.

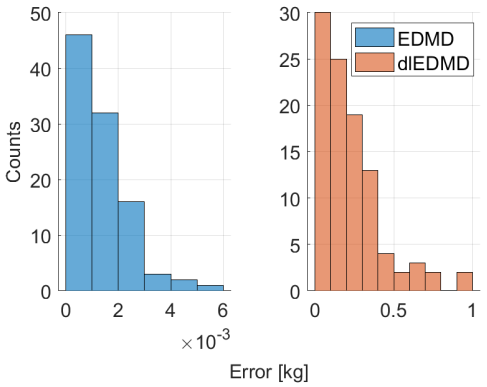
Figure 4.11: Linear control 2D-aero error, $T = 10$ s, 120 basis functions, EDMD and dlEDMD. The mean error is reported, from left to right, for EDMD and dlEDMD.



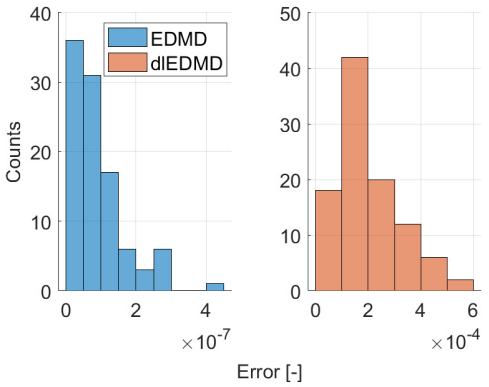
(a) Position. Mean $(10.40, 4.69)$ m.



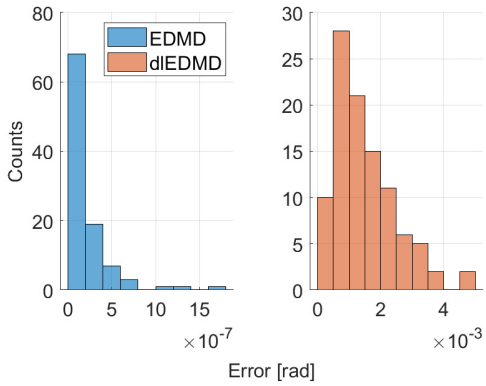
(b) Velocity. Mean $(2.02, 0.97)$ m/s.



(c) Mass. Mean $(0.001, 0.23)$ kg.

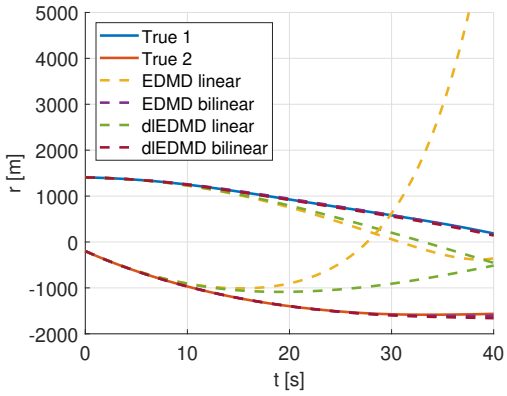


(d) Throttle. Mean $(9.06 \times 10^{-8}, 2.00 \times 10^{-4})$.

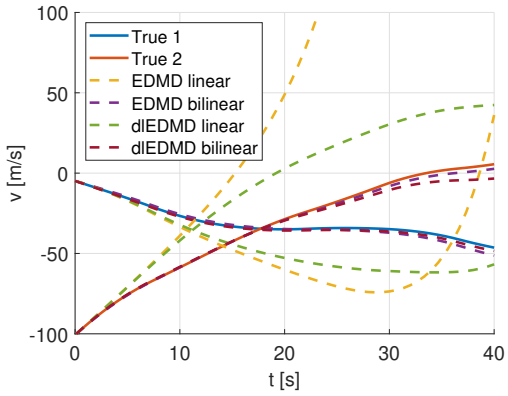


(e) Pitch angle. Mean $(1.92 \times 10^{-7}, 0.002)$ rad.

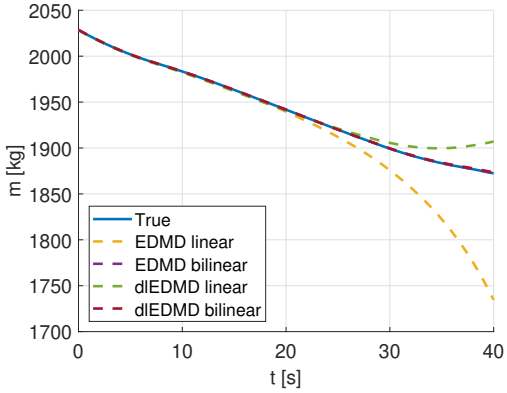
Figure 4.12: Bilinear control 2D-aero error, $T = 10$ s, 120 basis functions, EDMD and dlEDMD. The mean error is reported, from left to right, for EDMD and dlEDMD.



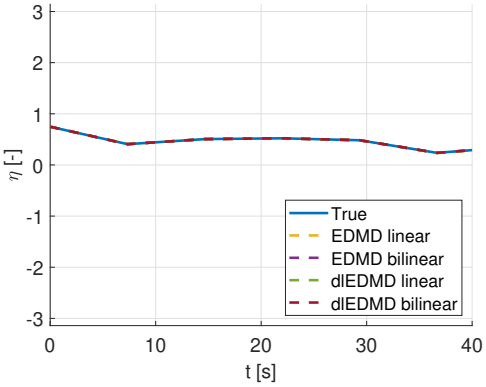
(a) Position.



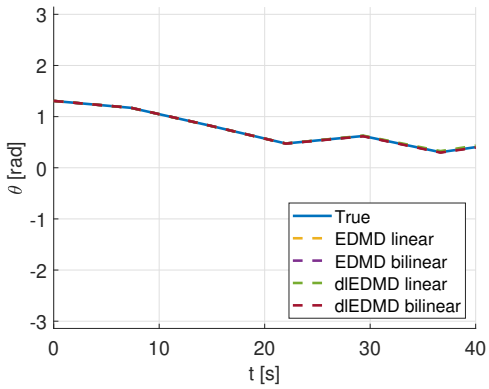
(b) Velocity.



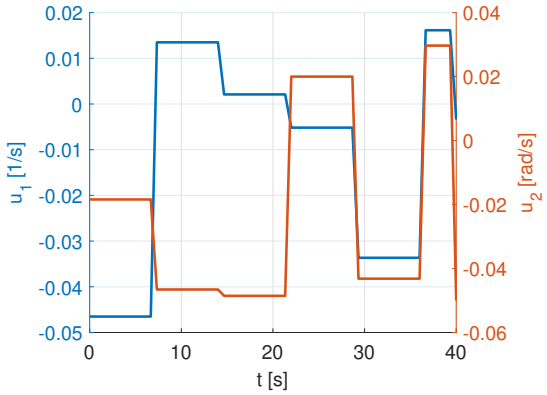
(c) Mass.



(d) Throttle.



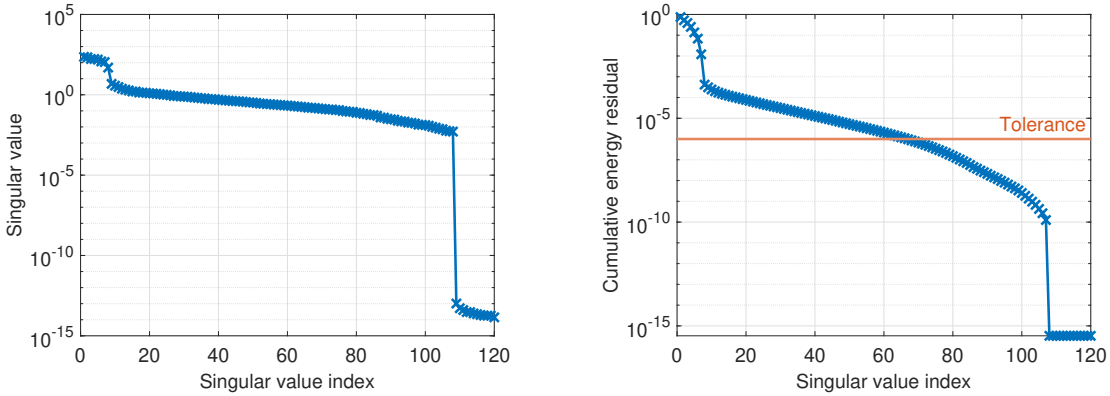
(e) Pitch angle.



(f) Control.

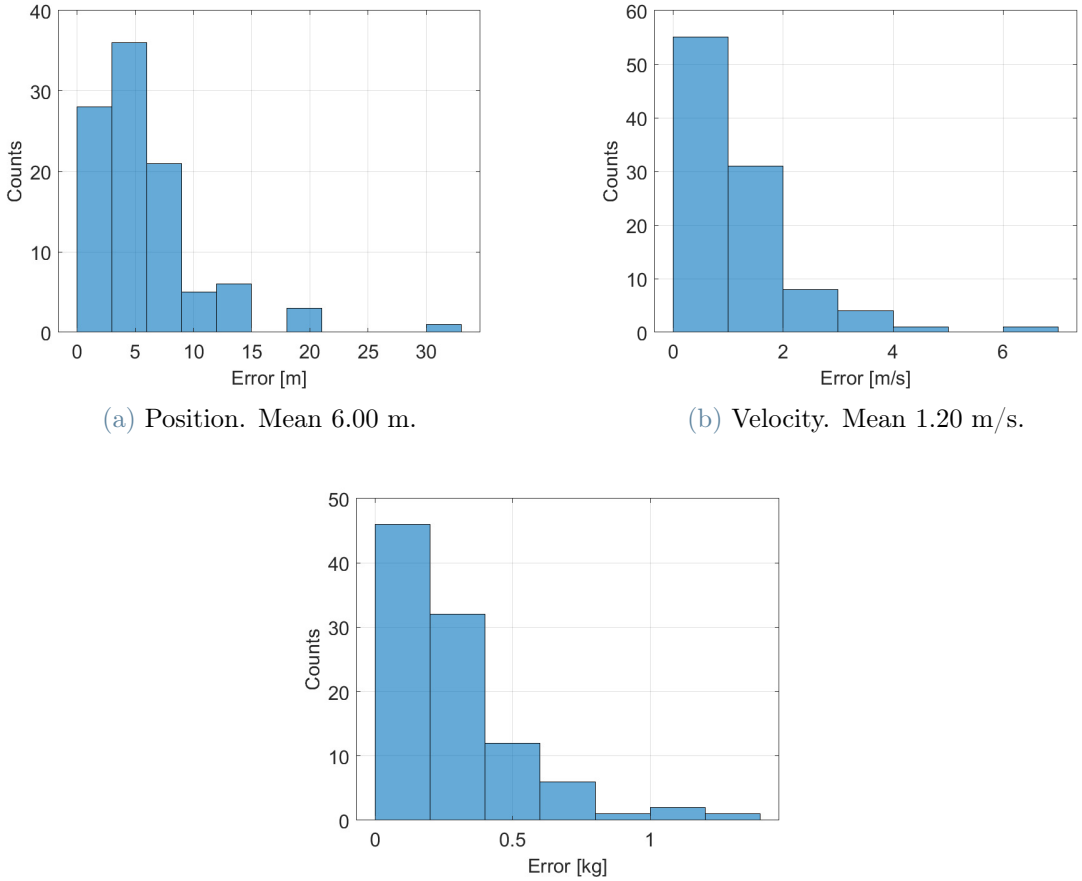
Figure 4.13: Controlled 2D-aero example trajectory, 120 basis functions.

Truncated Singular Value Decomposition The SVD can be performed for the dlEDMD model to try and reduce its dimension. With a tolerance $\varepsilon = 10^{-6}$, the basis functions can be reduced to 67 without affecting the accuracy (figs. 4.14 and 4.15).



(a) Singular values. (b) Cumulative energy.

Figure 4.14: 2D-aero truncated SVD analysis, dlEDMD.



(a) Position. Mean 6.00 m. (b) Velocity. Mean 1.20 m/s.

(c) Mass. Mean 0.29 kg.

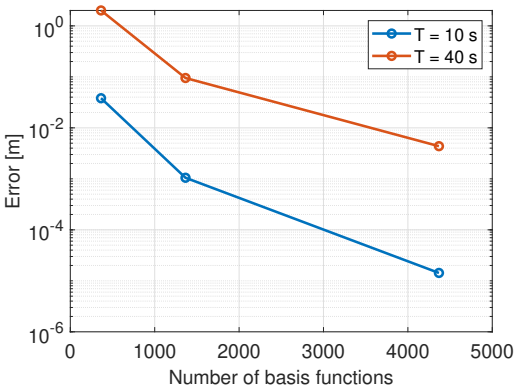
Figure 4.15: Bilinear control 2D-aero error after truncated SVD, $T = 10$ s, 67 basis functions, dlEDMD.

4.1.3. 3D Model without Aerodynamics

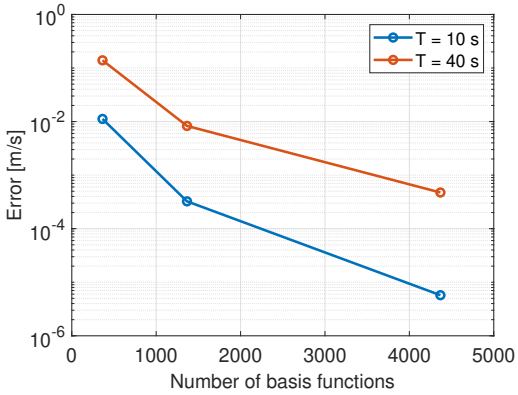
The differential equations of the 3D model are constructed such to have only polynomial nonlinearities; therefore, GAL can be applied and its potential in particular cases is shown. Given the higher complexity, dlEDMD is not employed, although the previous paragraphs give a good indication that it could outperform GAL, especially in terms of number of basis functions. No data is needed, since it is not a data-driven method; however, it is pointed out that, when computing error on trajectories, the constraint on the norm of the first body vector was not imposed for simplicity.
Furthermore, since a linear model for the control would be equivalent to compute the Koopman control matrix only with the constant Legendre polynomial (section 2.2), and since in the previous cases this did not result in a model accurate enough (when applying EDMD), only the bilinear model is approximated.

Free-dynamics Model The model is built with polynomials of order 3, 4 and 5. It must be noted that, with fifth-order polynomials, the dimension of the resulting system is 4368, thus, even if it is more accurate, it would not be useful for optimization. Figure 4.16 shows how, differently from the 2D-aero model with EDMD, the error decreases when more basis functions are used. However, the third-order model, with 364 basis functions, is already accurate enough, with a mean error on the velocity of 1×10^{-1} m/s at worse.
On the drawbacks side, the aerodynamic force is completely neglected, making it useless to use this Koopman model to solve the OCP, when there are other methods that can solve the same problem more efficiently [25], without increasing the state dimension. Furthermore, $q = 364$ basis functions might be too many to be handled by optimization algorithms, considering that the number of optimization variables is in the order of (# nodes $\cdot\, q$) after discretization of the continuous problem. On the other hand, dlEDMD could either take the aerodynamic force into account and, hopefully, use a smaller set of basis functions; however, the neural network optimization algorithm must be first refined to obtain a useful model.

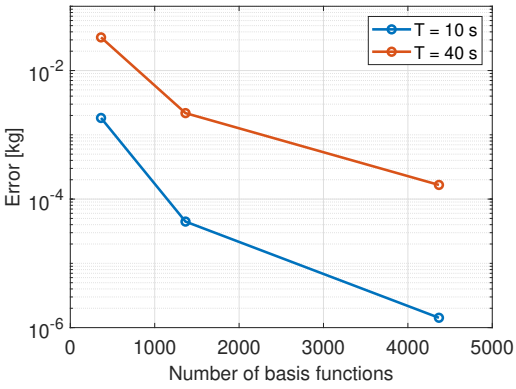
Bilinear Control Model The bilinear control model is approximated starting from the uncontrolled one with third-order polynomials. Data was generated using the same control bounds as the previous models. As it can be seen in fig. 4.17, the accuracy is the same of the free-dynamics model; an example trajectory can be observed in fig. 4.18.



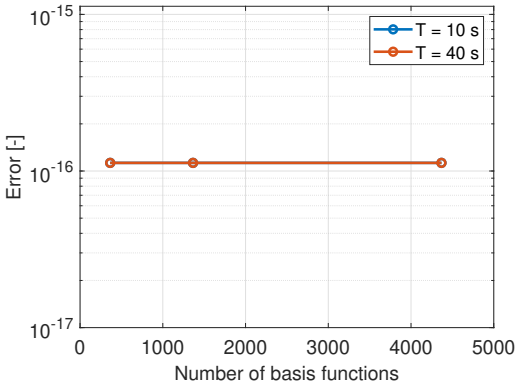
(a) Position.



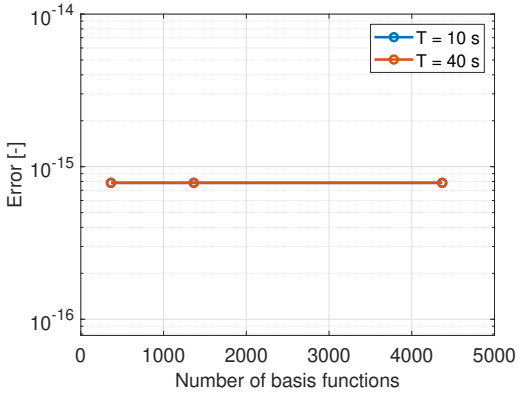
(b) Velocity.



(c) Mass.

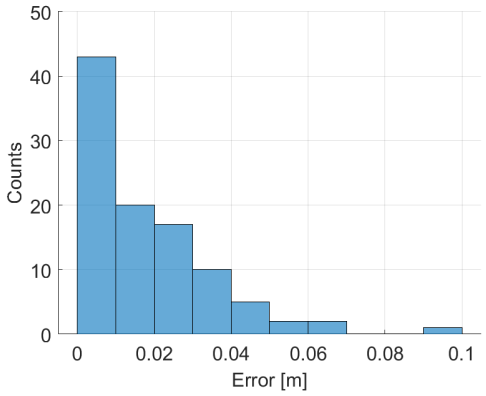


(d) Throttle.

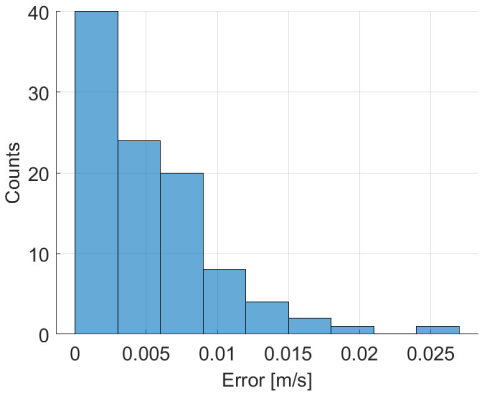


(e) Body vector.

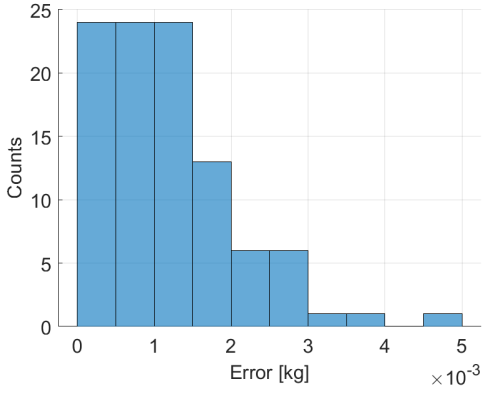
Figure 4.16: Uncontrolled 3D-noAero mean error, $T = 40$ s, GAL.



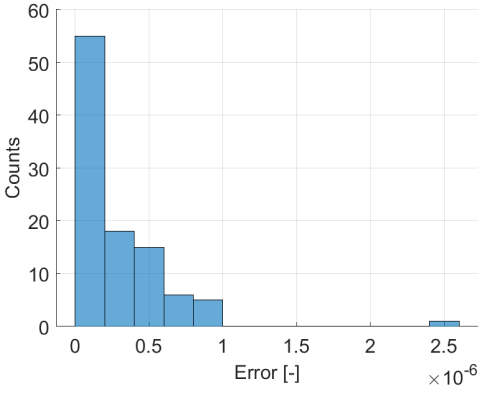
(a) Position. Mean 0.018 m.



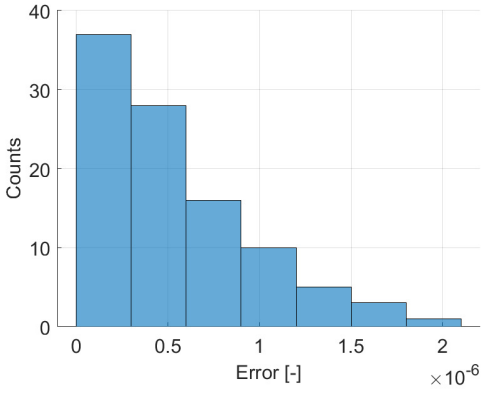
(b) Velocity. Mean 0.005 m/s.



(c) Mass. Mean 0.001 kg.

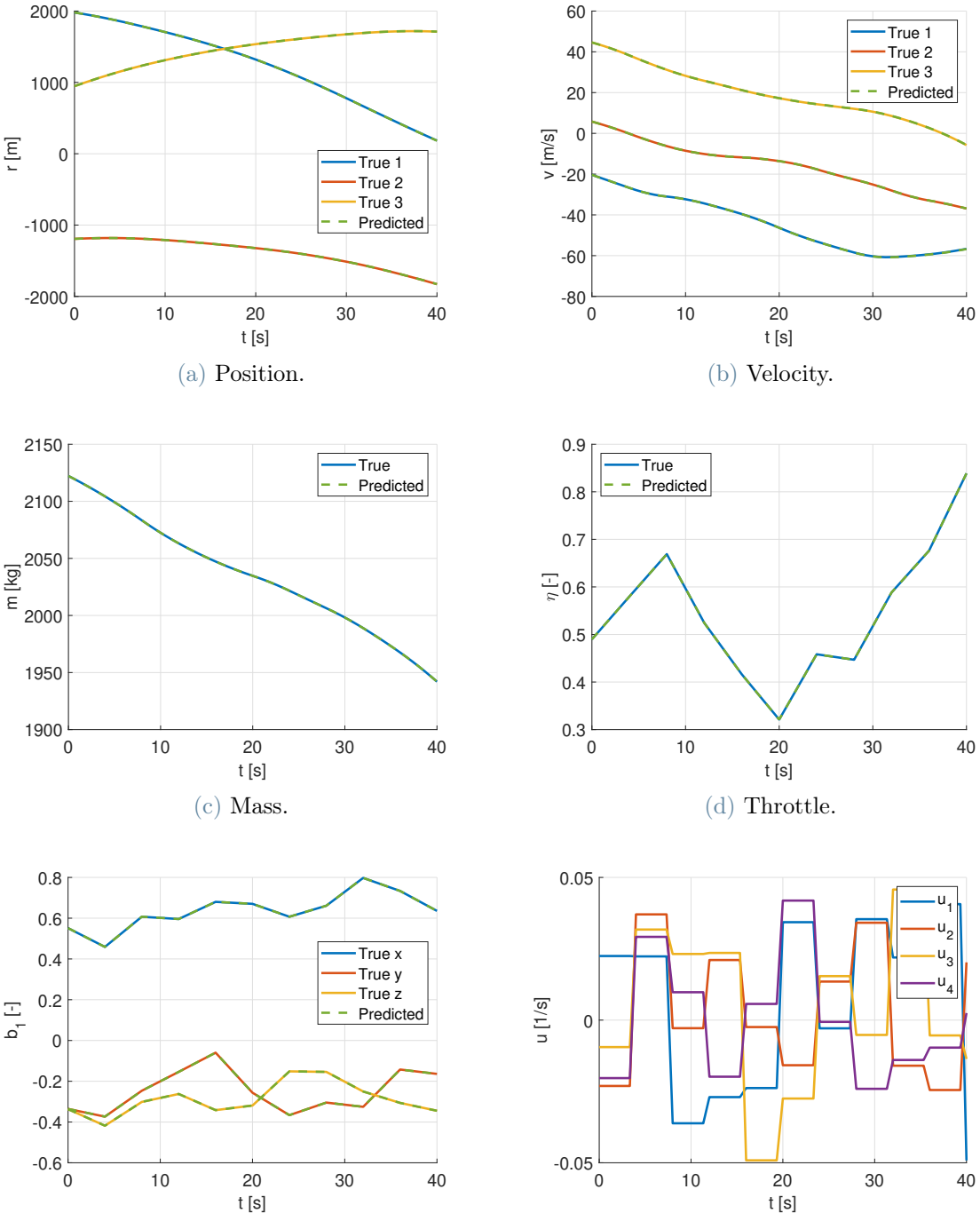


(d) Throttle. Mean 2.82×10^{-7}.



(e) Body vector. Mean 5.29×10^{-7}.

Figure 4.17: Bilinear control 3D-noAero error, $T = 10$ s, 364 basis functions, GAL.



(a) Position.

(b) Velocity.

(c) Mass.

(d) Throttle.

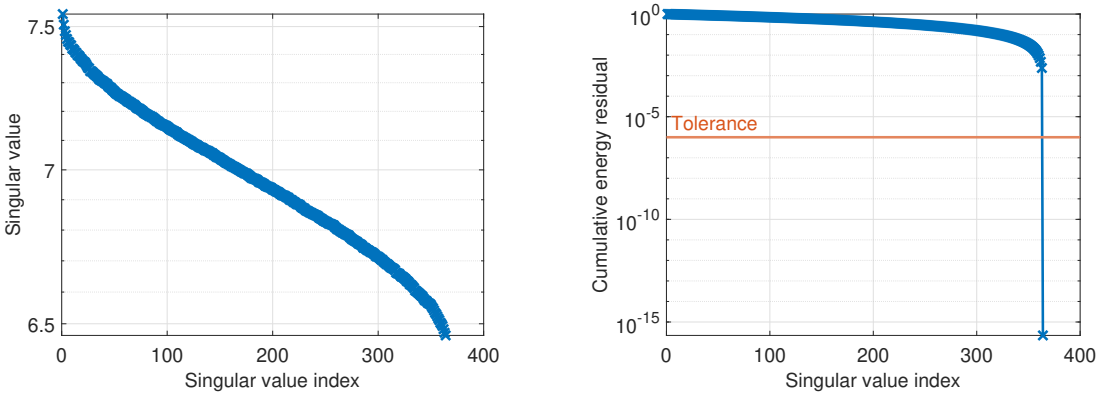
(e) Body vector.

(f) Control.

Figure 4.18: 3D-noAero example trajectory with control, 364 basis functions.

Truncated Singular Value Decomposition Figure 4.19a shows how, differently from the 2D-aero case, there are no zero singular values. This is expected, since the basis functions are well-defined in this case (Legendre polynomials) and they are independent

one from the other; also, no dimensionality reduction can be obtained, looking at the cumulative energy in fig. 4.19b, with a tolerance $\varepsilon = 10^{-6}$. Therefore, no truncation is performed for the 3D-noAero model.



(a) Singular values. (b) Cumulative energy.

Figure 4.19: 3D-aero truncated SVD analysis, GAL.

4.2. Optimal Control

The OCPs discussed in sections 3.3 and 3.4 are solved and compared in order to assess the optimality of the Koopman-based formulation and whether it is advantageous in terms of numerical efficiency or not. The solutions of the NLPs were obtained using GPOPS-II [60] with IPOPT solver [61], after discretization into a mesh composed by 40 segments with 5 collocation points each. The LP was solved with the MATLAB function *linprog* (with *interior-point-legacy* algorithm) using a mesh of 200 nodes, for comparison with the bilinear model. The machine used to carry out the simulations has a 3.20 GHz AMD Ryzen 7 7735HS CPU and 16 GB of RAM.

The solutions are compared in the following way: two control curves are obtained by solving the two different problems; the original dynamics is propagated with both solutions in order to show the actual optimal trajectory and how it behaves with the Koopman control; the Koopman dynamics is also propagated with its control to verify that the constraints are satisfied in the new formulation and to compare the trajectory with the original one.

The decision criteria for the Koopman models obtained in section 4.1 is to choose the best compromise in terms of number of basis functions and state accuracy. For example, if the model with less basis functions has an acceptable accuracy, it is used in the OCP.

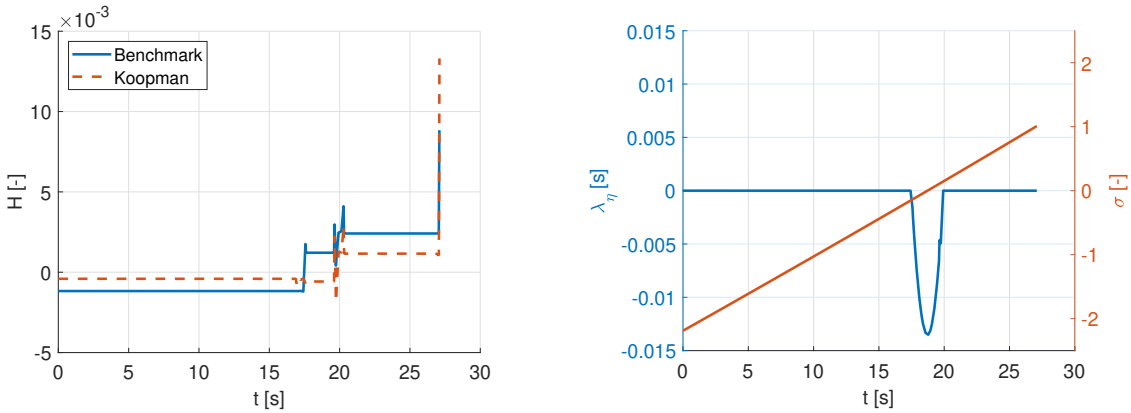
4.2.1. 1D Model with Aerodynamics

Two Koopman systems are selected for the 1D-aero model: the EDMD bilinear model and the dlEDMD linear model with 35 basis functions, because, with the same dictionary size, each of them has the best accuracy. The initial conditions, the endpoint constraints and the bounds used for the problem are listed in table 4.3.

State	Initial condition	Endpoint constraint	Bounds
r_1 [m]	500	0	free
v_1 [m/s]	-10	-0.01[2]	free
m [kg]	1905	free	free
η [-]	0.3	free	$[0.3, 0.8]$[3]
u_1 [1/s]	free	free	$[-0.2, 0.2]$
t [s]	0	free/28 (LP)	free

Table 4.3: 1D-aero optimal control parameters.

The results of the optimization are shown, for the two models, in figs. 4.20 to 4.22. The Hamiltonian is almost constant and equal to zero ($\approx 1 \times 10^{-3}$) in both the benchmark and the Koopman bilinear problem; however, the Hamiltonian has not been computed in the LP. The necessary condition of the benchmark problem shows that the throttle rate is at the bounds when $\lambda_\eta \neq 0$ and, instead, it is zero when the costate is zero and $\sigma(\mathbf{x}, \boldsymbol{\lambda}) \neq 0$.



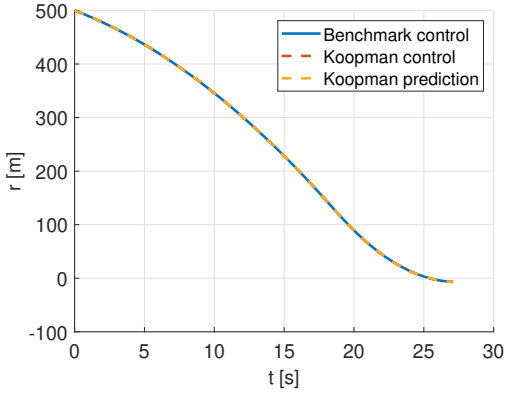
(a) Hamiltonian.

(b) Necessary condition on throttle rate.

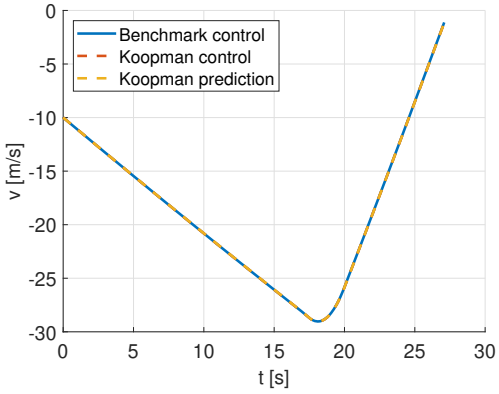
Figure 4.20: 1D-aero optimality conditions, bilinear model.

[2]In the 2D model a completely null velocity might cause issues in the computation of the wind angles. Therefore, the first component has been fixed to this value for all the three problems.

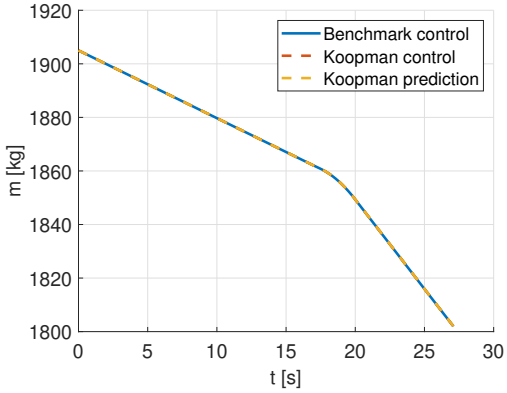
[3]The engine cannot be switched off after ignition and some operability limits have been introduced.



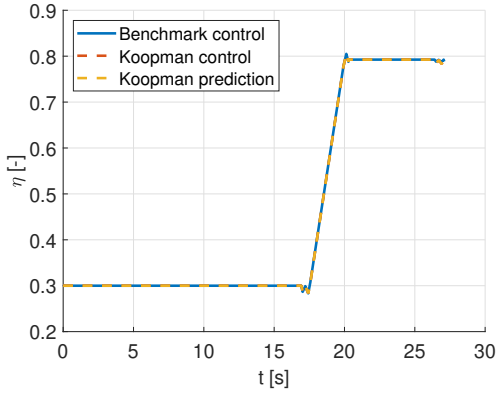
(a) Position.



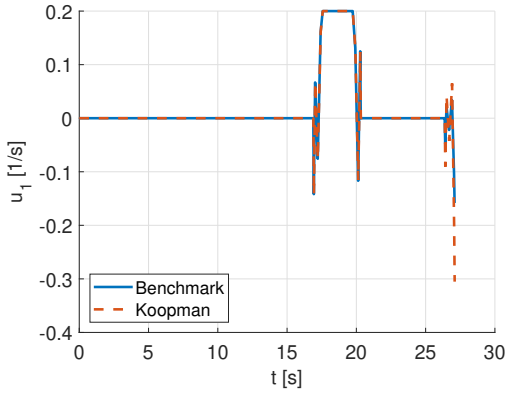
(b) Velocity.



(c) Mass.

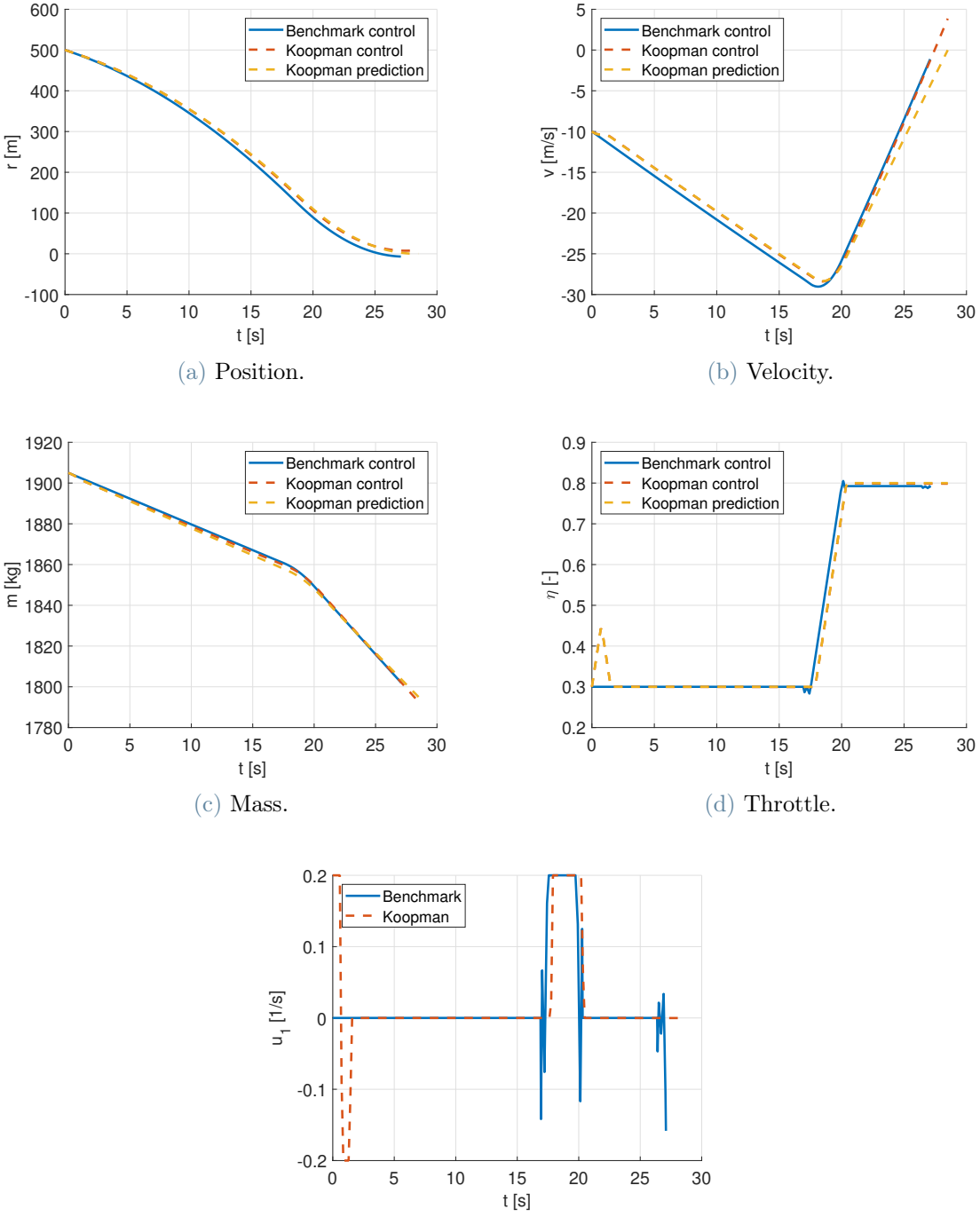


(d) Throttle.



(e) Control.

Figure 4.21: 1D-aero OCP solution, bilinear model.



(a) Position.

(b) Velocity.

(c) Mass.

(d) Throttle.

(e) Control.

Figure 4.22: 1D-aero OCP solution, linear model.

It can be observed that the Koopman bilinear model gives almost exactly the same results of the benchmark. On the other hand, the results of the linear model are slightly different, but it must be specified that in the LP the final time is fixed to 28 s, resulting

in a longer time of flight than the benchmark (27.09 s), which is a free final time problem. A solution for the linear program could not be obtained with the same final time of the benchmark (probably because on the border of the feasible region); likewise, a benchmark solution could not be obtained with any fixed final time. The difference in solution is mostly due to this difference; however, a line search algorithm for different final times could be implemented [25].
The performances of the models are assessed in terms of error in final position and velocity, number of iterations and computational time. For the Koopman-based formulation, the final error is obtained by propagating the original dynamics with the optimal control obtained. The results are summarized in table 4.4. The error of the benchmark is introduced by the discretization of the problem; the bilinear model has a similar error, but the computational time was much longer; the LP error is comparable with the other solutions, it required less iterations than the benchmark, but longer computational time. However, more efficient algorithms could be used to reduce the solver time. The final mass is lower (i.e. less optimal) than the benchmark mostly because of the longer time of flight.

	Benchmark	**Bilinear Koopman**	**Linear Koopman**
Final position error	6.79 m	6.94 m	9.54 m
Final velocity error	1.11 m/s	1.12 m/s	3.86 m/s
Final mass	1802 kg	1802 kg	1793 kg
Iterations	20	27	18
Computational time	0.56 s	18.22 s	2.81 s

Table 4.4: 1D-aero optimal control performance.

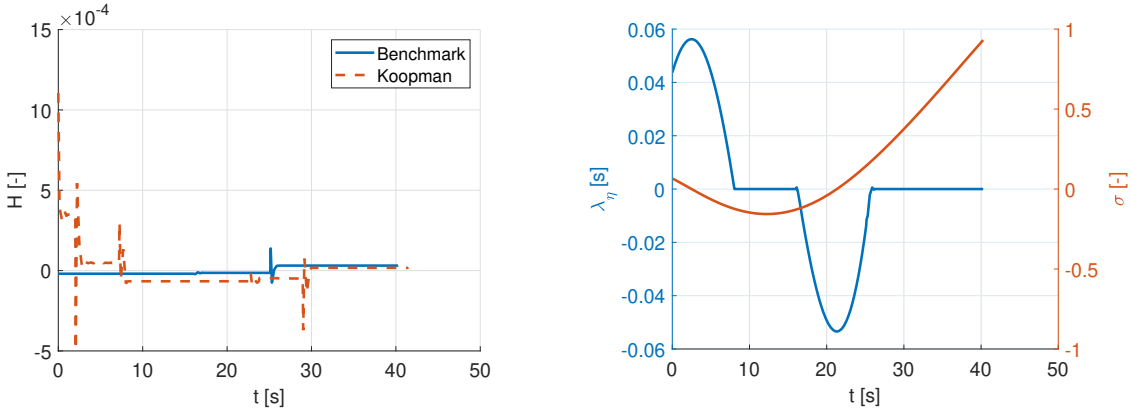
4.2.2. 2D Model with Aerodynamics

The bilinear dlEDMD model with 67 basis functions (after SVD truncation) is used for the Koopman-based formulation, since an accurate enough linear model could not be found. Note that the SVD truncation was crucial, not only because it reduced the dimension of the problem, but also because it made it well-posed. In fact, when no truncation is performed, the algorithm may try to optimize some lifted states which have no impact on the cost function or on the constraints on the original state; no solution could be found with the non-truncated model. In order to avoid a noisy oscillating behavior in the control solution, a regularization term was added to the cost function, such that the integrand became $\eta + 5 \times 10^{-2} u_1^2 + 5 \times 10^{-1} u_2^2$. The parameters for the OCP are listed in table 4.5 and the results are presented in figs. 4.23 and 4.24. The Hamiltonian is approximately

constant and equal to zero ($\approx 1 \times 10^{-4}$) for the two problems, and the necessary condition on the throttle is satisfied in the benchmark problem. The regularization changes the formulation of the problem, but with small impact on the necessary conditions, even if is smooths out the expected bang-bang behavior in the throttle curve.
However, the Koopman formulation results in a different control trajectory which cannot be superimposed on the benchmark one; consequently, the rocket follows a slightly different trajectory. This difference could be due to the approximation introduced by the Koopman model: when the Koopman operator is approximated by a finite subspace of functions, some modes of the original dynamics might be truncated, making the optimization problem converge to a different solution.

State	Initial condition	Endpoint constraint	Bounds
$\mathbf{r}$ [m]	$[1500, -1800]^T$	$[0, 0]^T$	free
$\mathbf{v}$ [m/s]	$[-10, 110]^T$	$[-0.01, 0]^T$	free
m [kg]	1905	free	free
η [-]	0.8	free	$[0.3, 0.8]$
θ [$\circ$]	-87.09	free	free
α [$\circ$]	182.44	free	$[150, 210]$
u_1 [1/s]	free	free	$[-0.2, 0.2]$
u_2 [rad/s]	free	free	$[-0.2, 0.2]$
t [s]	0	free	free

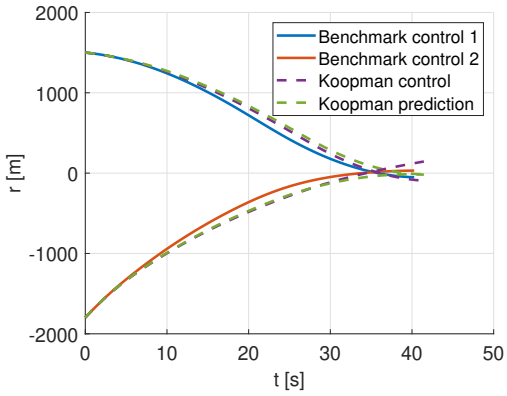
Table 4.5: 2D-aero optimal control parameters.



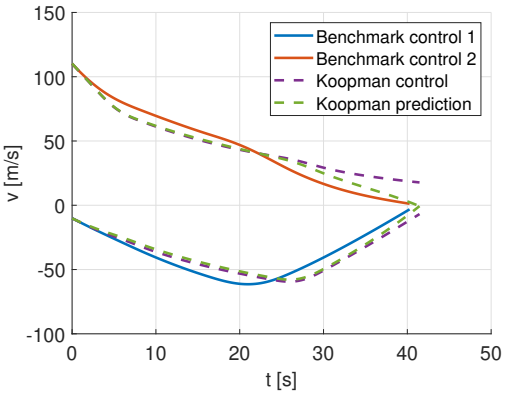
(a) Hamiltonian.

(b) Necessary condition on throttle rate.

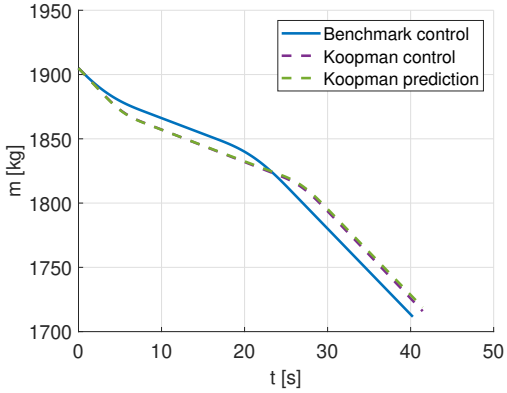
Figure 4.23: 2D-aero optimality conditions, bilinear model.



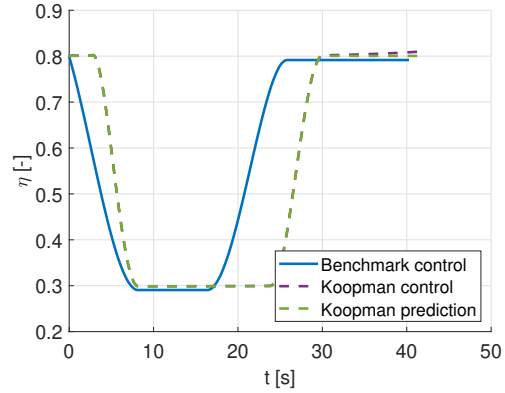
(a) Position.



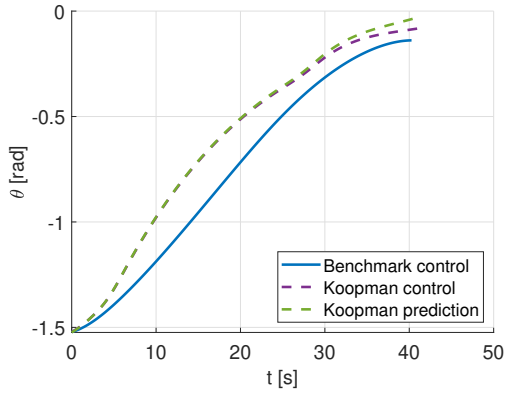
(b) Velocity.



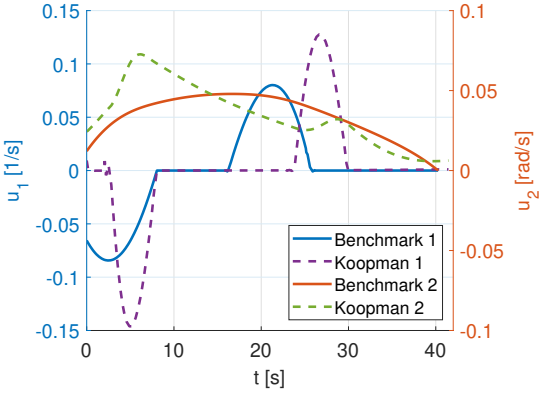
(c) Mass.



(d) Throttle.



(e) Pitch angle.



(f) Control.

Figure 4.24: 2D-aero OCP solution, bilinear model.

Table 4.6 shows the main results of the optimization problems; the error is intended as the 2-norm of the final position or velocity error vector. It can be noted that the Koopman optimization results in a larger error with a much longer computational time

(and iterations). While the final mass of the Koopman model is higher (more optimal) than the benchmark, it must not be deduced that the Koopman formulation allows a better result to be obtained, as the resulting trajectory is more infeasible than the benchmark one.

	Benchmark	Bilinear Koopman
Final position error	60.68 m	173.14 m
Final velocity error	3.38 m/s	19.03 m/s
Final mass	1712 kg	1716 kg
Iterations	57	76
Computational time	3.05 s	292.25 s

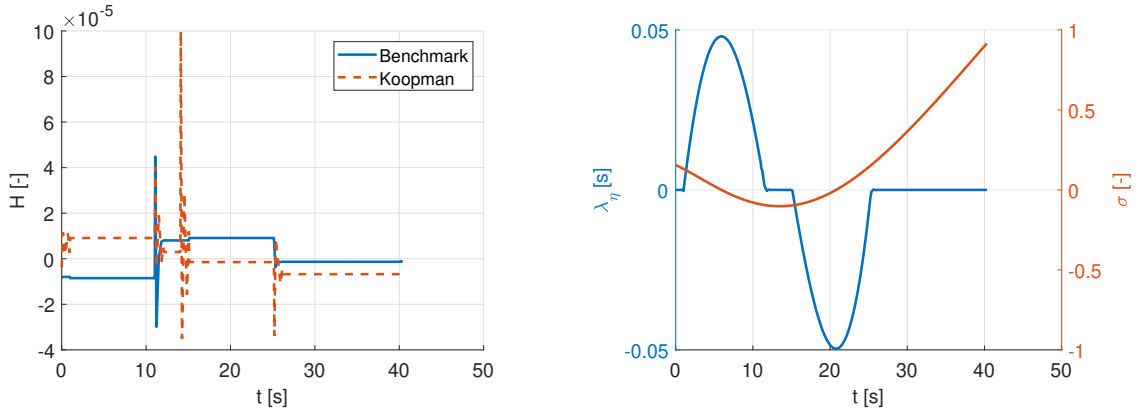
Table 4.6: 2D-aero optimal control performance.

4.2.3. 3D Model without Aerodynamics

The bilinear GAL model with 364 basis functions is used for the Koopman-based formulation. As for the 2D-aero model, a regularization was applied to the control, such that the integrand of the cost function became $\eta + 5 \times 10^{-2} u_1 + 5 \times 10^{-1} [u_2, u_3, u_4]^T$. In table 4.7 the parameters of the problem are listed; it must be noted that the constraint on the norm of the body vector has been imposed explicitly, differently from the other problems, where it was automatically satisfied. As it can be observed in fig. 4.25, the throttle necessary condition is again satisfied and the Hamiltonian is constant and approximately zero ($\approx 1 \times 10^{-5}$) for the benchmark and the Koopman formulations.

State	Initial condition	Endpoint constraint	Bounds
$\mathbf{r}$ [m]	$[1500, 1050, -1800]^T$	$[0, 0, 0]^T$	free
$\mathbf{v}$ [m/s]	$[-10, -70, 110]^T$	$[-0.01, 0, 0]^T$	free
m [kg]	1905	free	free
η [-]	0.8	free	$[0.3, 0.8]$
$\hat{\mathbf{b}}_1$ [-]	$[0.40, 0.51, -0.76]$	free	free
$\mathbf{u}$ [1/s]	free	free	$[-0.2, 0.2]$
t [s]	0	free	free

Table 4.7: 3D-noAero optimal control parameters.



(a) Hamiltonian.

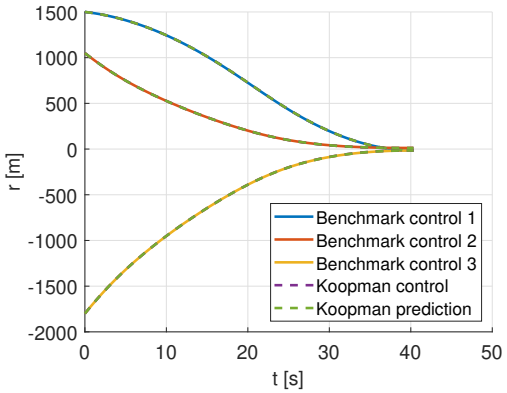
(b) Necessary condition on throttle rate.

Figure 4.25: 3D-aero optimality conditions, bilinear model.

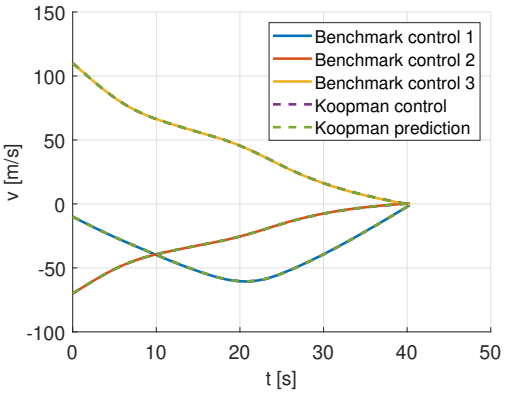
Figure 4.26 shows the resulting trajectory. The Koopman solution is superimposed on the benchmark one, as expected, given the small mean error of the Koopman system obtained in section 4.1. The characteristic bang-bang behavior of rocket landing problems can be observed in the throttle level, although smoothed out by the regularization imposed on the control through the cost function. Also table 4.8 shows that the landing error and the final mass of the two problems are comparable. However, the Koopman formulation required 31 iterations more than the benchmark and more than 75 min to obtain a solution. This is due to the high dimension of the Koopman system; in fact, 364 states, 4 control variables, 1 final time variable and 200 nodes result in $(364 + 4) \cdot 200 + 1 = 73601$ optimization variables. Even if the bilinear control system might decrease the complexity of the dynamics nonlinearities, the lifting of the state invalidates all the simplification.

	Benchmark	**Bilinear Koopman**
Final position error	26.47 m	24.77 m
Final velocity error	1.14 m/s	1.06 m/s
Final mass	1693 kg	1694 kg
Iterations	74	105
Computational time	1.91 s	4606.67 s

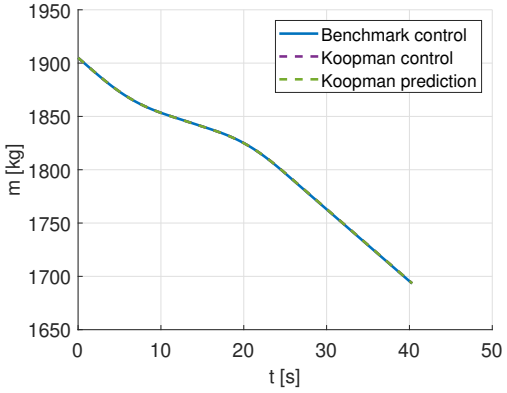
Table 4.8: 3D-noAero optimal control performance.



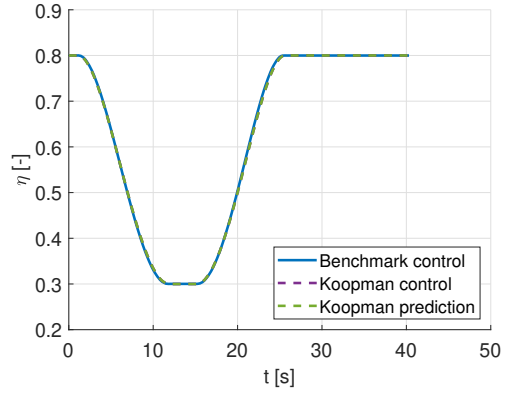
(a) Position.



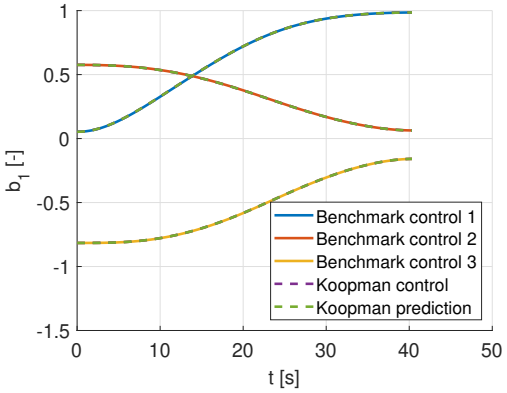
(b) Velocity.



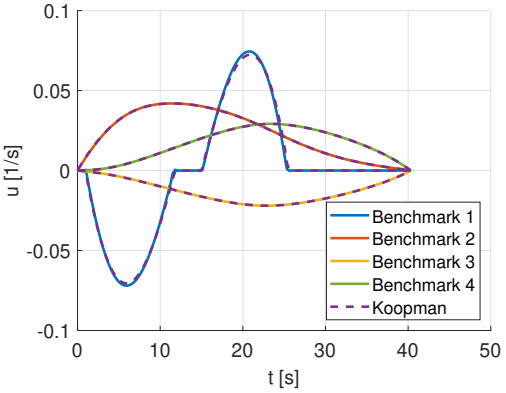
(c) Mass.



(d) Throttle.



(e) Body vector.



(f) Control.

Figure 4.26: 3D-noAero OCP solution, bilinear model.

5 | Conclusions

5.1. Lessons Learned

KOT is promising because it can, theoretically, globally linearize an uncontrolled autonomous dynamical system. The objective of this work was to demonstrate its applicability in a atmospheric, rocket landing, optimal control framework. Two well-established approaches (GAL and EDMD) were compared to a more recent method: dlEDMD. It was shown that GAL can be used with Legendre polynomials as basis functions to obtain an analytical solution when the dynamics is polynomial; EDMD, instead, can approximate the operator with a fixed dictionary of functions by providing a set of data, when a closed-form solution cannot be obtained. Through some motivational examples in section 2.3, the limitations of fixed-dictionary approaches have been discussed: in particular, they suffer from closure issues in some cases, leading to a divergent behavior as the number of basis functions increases. Therefore, it is crucial to select a proper set of basis functions. In this context, dlEDMD is an innovative method that extends EDMD by learning an optimal dictionary of functions through neural network optimization, and it proved to be effective where the other techniques failed.

While the main hope is to being able to include the control in a linear way, the hypotheses to be satisfied are very restrictive. In this case, it is critical to optimize the basis functions in order to minimize this limitation; dictionary learning techniques are able to achieve this and the experiments showed that the accuracy of dlEDMD models was always better than the associated GAL and EDMD ones, when a linear control model was sought.

More in general, a bilinear control model can be obtained with almost no further approximations. Again, when no analytic solution is available, and GAL cannot be applied, data-driven methods must be employed. However, some care is needed when building it with EDMD-like methods that aim at minimizing the single-step prediction of the discrete-time Koopman system: in fact, the bilinearizability is referred to continuous-time systems. In this work, for the first time to the best of the author's knowledge, the EDMD framework has been modified to minimize the prediction of the continuous controlled dynamics *after* having already obtained the free-dynamics Koopman matrix.

This approach is similar to gEDMD [55], that directly computes the free-dynamics Koopman infinitesimal generator, and to SINDy [56], that includes generic functions of the state and control without separating the uncontrolled part from the controlled one. With the approach of this work, an accurate enough free-dynamics Koopman operator can be obtained first (thus making sure that the set of basis functions is appropriate) and the control can be included consequently without additional losses. The effectiveness of the method was demonstrated by the motivating examples in section 2.3. Moreover, all the EDMD-based methods to include control have been successfully included in the dlEDMD algorithm, which was originally built only for uncontrolled systems [6].
In section 4.1 three different dynamical models of the rocket landing problem, with different levels of complexity, have been analyzed. The EDMD method showed good performance with the 1D model with aerodynamic force, with increasing accuracy as the order of the Legendre polynomials increases; on the other hand, dlEDMD performs worse and with less improvement with a larger dictionary of functions. However, to the aim of constructing a linear control model, a flexible dictionary helps in the accuracy of the resulting model, while EDMD is not accurate enough. The 2D model with aerodynamic force showed a divergence issue when EDMD was applied, while with dlEDMD the accuracy was better; moreover, SVD truncation was crucial to decrease the number of basis functions and to have a dynamics with well-conditioned matrices. An accurate enough linear control model could not be found in this case, even with dlEDMD. Lastly, the 3D model without aerodynamic force was tested with GAL, given its polynomial dynamics, to show its potential in particular cases. In all cases, however, the proposed method to obtain a bilinear control model proved to be effective, and can be used in future works.
One main conclusions is that, in general, it is better to start with GAL/EDMD methods, because they are easier to be applied and can be effective with some dynamical models. Only when they fail, it is worth to apply dlEDMD.
For what concerns optimal control, all the Koopman models are able to provide a solution similar to the benchmark. The only exception is the 2D model, in which the Koopman approximation truncates the dynamics, making the solver to converge to a slightly different solution. The bilinear model, in general, showed worse performance than the benchmark, in terms of both optimization time and number of iterations. Despite the fact that nonlinearities are less complex the the ones of the original model, the increased dimensionality of the problem makes it not useful to use a bilinear Koopman model for optimal control. The Koopman linear model could be successfully used to build a LP; a solution could be found with relative good performance, even if a proper comparison with the benchmark could not be carried out, due to the different formulations of the two problems.

5.2. Future Directions

While the dlEDMD method proved to be effective when GAL and EDMD fail, little improvement in the accuracy of the Koopman model can be observed when the number of the basis functions increases. With the current architecture of the neural network, nothing prevents the algorithm from introducing *useless* functions in the output. In fact, when the dictionary grows larger, it is more convenient to add trivial functions in the output, such as constant functions, that have exact single-step prediction. This *pushes* down the value of the cost function. Additionally, linear combinations of the already present functions might be introduced, simply because nothing impedes it. This behavior is suggested by the SVD analyses performed in this work: when dlEDMD was used, some singular values were zero, meaning that some output functions could be discarded; instead, with GAL there were no null singular values, because the Legendre polynomials are orthonormal. For this reason, an interesting development of this work would be to tackle this problem, because in this way the accuracy of the Koopman model could be increased even more; furthermore, it might be crucial in order to obtain a linear model, since even dlEDMD struggles to approximate a good model in some cases. One idea would be to take redundancy into account in the loss function, penalizing the deviation of the Gram matrix from the identity (the Gram matrix can be built on a set of vectors and is equal to the identity if all the vectors are orthonormal [62]). Moreover, in this work the neural network architecture was kept fixed; a more thorough analysis is needed to assess the impact of the hyperparameters (activation function, number and width of hidden layers, etc.) on the network's performance. These two improvements can, hopefully, lead to a Koopman system for a 3D model with aerodynamic force, or even for a full 6 DoFs model.

Regarding optimal control, the presented bilinear Koopman-formulation of the OCP is not advantageous with respect to the benchmark problem; however, algorithms that specifically exploit the structure of the system may be employed. For instance, the bilinear dynamics can be easily linearizable and a sequence of LPs could be solved.

The Koopman-based LP here discussed has been built from scratch and the final time of the problem was kept fixed: more performing algorithms can be used to solve the LP and iterative methods can be applied to find the optimal final time. Furthermore, although not done in this work, the eigendecomposition of the linear model could result in a further increase of the algorithms' computational efficiency.

Bibliography

[1] B. O. Koopman. Hamiltonian systems and transformations in hilbert space. *Proceedings of the National Academy of Sciences of the United States of America*, 17(5): 315–318, 5 1931.

[2] S. Servadio, D. Arnas, and R. Linares. Dynamics near the three-body libration points via koopman operator theory. *Journal of Guidance, Control, and Dynamics*, 45(2): 1800–1814, 7 2022.

[3] S. Servadio, D. Arnas, and R. Linares. A koopman operator tutorial with othogonal polynomials, 7 2022. URL `https://arxiv.org/abs/2111.07485`.

[4] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley. A data-driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25:1307–1346, 6 2015.

[5] C. Hofmann, G. Lavezzi, D. Wu, S. Servadio, and R. Linares. Comparative analysis of analytical and data-driven koopman operators for the j2 problem with atmospheric drag. In *Astrodynamics Specialist Conference*, Broomfield, CO, USA, 8 2024. AAS/AIAA.

[6] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 10 2017.

[7] D. Goswami and D. A. Paley. Bilinearization, reachability, and optimal control of control-affine nonlinear systems: A koopman spectral approach. *IEEE Transactions on Automatic Control*, 67(6):2715 – 2728, 6 2022.

[8] S. E. Otto and C. W. Rowley. Koopman operators for estimation and control of dynamical systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:59–87, 2 2021.

[9] D. Bruder, X. Fu, and R. Vasudevan. Advantages of bilinear koopman realizations

for the modeling and control of systems with unknown dynamics. *IEEE Robotics and Automation Letters*, 6(3):4369–4376, 7 2021.

[10] S. M. Rajkumar, S. Cheng, N. Hovakimyan, and D. Goswami. Linear model predictive control for quadrotors with an analytically derived koopman model, 2024. URL `https://arxiv.org/abs/2409.12374`.

[11] J. Wang, B. Xu, J. Lai, Y. Wang, C. Hu, H. Li, and A. Song. An improved koopman-mpc framework for data-driven modeling and control of soft actuators. *IEEE Robotics and Automation Letters*, 8(2):616–623, 2 2023.

[12] C. Hofmann, S. Servadio, R. Linares, and F. Topputo. Advances in koopman operator theory for optimal control problems in space flight. In *Astrodynamics Specialist Conference*, Charlotte, NC, USA, 8 2022. AAS/AIAA.

[13] J. M. Longuski, J. J. Guzmán, and J. E. Prussing. *Optimal Control with Aerospace Applications*. Microcosm Press and Springer, 2014.

[14] E. T. Bell. *Men of Mathematics*. Simone & Schuster, 2023.

[15] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishechenko. *The Mathematical Theory of Optimal Processes*. John Wiley & Sons, 1962.

[16] A. E. Bryson Jr. and Y. Ho. *Applied Optimal Control*. Taylor & Francis Group, 1975.

[17] J. T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2010.

[18] R. F. Hartl, S. P. Sethi, and R. G. Vickson. A survey of the maximum principles for optimal control problems with state constraints. *SIAM Review*, 37(2):181–218, 6 1995.

[19] M. Sagliano. *Development of a Novel Algorithm for High Performance Reentry Guidance*. PhD thesis, University of Bremen, 2016.

[20] L. Huneker, M. Sagliano, and Y. E. Arslantas. Spartan: An improved global pseudospectral algorithm for high-fidelity entry-descent-landing guidance analysis. In *30th International Symposium on Space Technology and Science*, Kobe-Hyogo, Japan, 7 2015. ISTS/JSASS.

[21] M. Sagliano. Performance analysis of linear and nonlinear techniques for automatic scaling of discretized control problems. *Operation Research Letters*, 42(3):213–216, 3 2014.

[22] M. Sagliano, D. Seelbinder, and S. Theil. Six-degree-of-freedom rocket landing optimization via augmented convex–concave decomposition. *Journal of Guidance, Control, and Dynamics*, 47(1):20–35, 1 2024.

[23] M. Sagliano, P. Lu, D. Seelbinder, and S. Theil. Analytical treatise on endo-atmospheric fuel-optimal rocket landings. *Journal of Guidance, Control, and Dynamics*, 48(3):450–469, 3 2025.

[24] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[25] B. Açıkmeşe and S. R. Ploen. Convex programming approach to powered descent guidance for mars landing. *Journal of Guidance, Control, and Dynamics*, 30(5): 1353–1366, 9-10 2007.

[26] M. Sagliano. Pseudospectral convex optimization for powered descent and landing. *Journal of Guidance, Control, and Dynamics*, 41(2):320–334, 2 2018.

[27] M. Sagliano. Generalized hp pseudospectral-convex programming for powered descent and landing. *Journal of Guidance, Control, and Dynamics*, 42(7):1562–1570, 7 2019.

[28] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

[29] F. V. Bennett. Apollo lunar descent and ascent trajectories. In *Aerospace Sci. Meeting*, New York, NY, USA, 3 1970. AIAA.

[30] M. W. Harris and M. B. Rose. *Optimal Spaceraft Guidance*. Utah State University, 2023.

[31] J. A. Jungmann. Gravity turn trajectories through planetary atmospheres. In *Guidance, control and Flight Dynamics Conference*, Huntsville, AL, USA, 8 1967. AIAA.

[32] R. R. Sostaric and J. R. Rea. Powered descent guidance methods for the moon and mars. In *Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, CA, USA, 8 2005. AIAA.

[33] R. Ingoldby. Guidance and control system design of the viking planetary lander. In *Guidance and Control Conference*, Hollywood, FL, USA, 8 1977. AIAA.

[34] B. A. Steinfeldt, M. J. Grant, D. M. Matz, and R. D. Braun. Guidance, navigation, and control technology system trades for mars pinpoint landing. In *Atmospheric Flight Mechanics Conference and Exhibit*, Honolulu, HI, USA, 8 2008. AIAA.

[35] U. Topcu, J. Casoliva, and K. D. Mease. Minimum-fuel powered descent for mars pinpoint landing. *Journal of Spacecraft and Rockets*, 44(2):324–331, 3-4 2007.

[36] D. Garg, M. Patterson, W. W. Hager, A. V. Rao, D. A. Benson, and G. T. Huntington. A unified framework for the numerical solution of optimal control problems using pseudospectral methods. *Automatica*, 46(11):1843–1851, 11 2010.

[37] I. M. Ross and F. Fahroo. A direct method for solving nonsmooth optimal control problems. *IFAC Proceedings Volumes*, 35(1):479–484, 2002.

[38] S. R. Ploen, B. Açıkmeşe, and A. Wolf. A comparison of powered descent guidance laws for mars pinpoint landing. In *Astrodynamics Specialist Conference and Exhibit*, Keyston, CO, USA, 8 2006. AIAA/AAS.

[39] F. Najson and K. D. Mease. A computationally non-expensive guidance algorithm for fuel efficient soft landing. In *Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, CA, USA, 8 2005. AIAA.

[40] F. Najson and K. D. Mease. Computationally inexpensive guidance algorithm for fuel-efficient terminal descent. *Journal of Guidance, Control, and Dynamics*, 29(4): 955–964, 7-8 2006.

[41] B. Açıkmeşe, J. M. Carson III, and L. Blackmore. Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem. *IEEE Transactions on Control Systems Technology*, 21(6):2104–2113, 11 2013.

[42] L. Blackmore, B. Açıkmeşe, and D. P. Scharf. Minimum-landing-error powered-descent guidance for mars landing using convex optimization. *Journal of Guidance, Control, and Dynamics*, 33(4):1161–1171, 7-8 2010.

[43] R. Yang and X. Liu. Fuel-optimal powered descent guidance with free final-time and path constraints. *Acta Astronautica*, 172:70–81, 7 2020.

[44] X. Liu and Z. Shen. Entry trajectory optimization by second-order cone programming. *Journal of Guidance, Control, and Dynamics*, 39(2):227–241, 2 2016.

[45] M. Sagliano, A. Heidecker, J. M. Hernández, S. Farì, M. Schlotterer, S. Woicke, D. Seelbinder, and E. Dumont. Onboard guidance for reusable rockets: Aerodynamic descent and powered landing. In *Scitech 2021 Forum*. AIAA, 1 2021.

[46] J. Meditch. On the problem of optimal thrust programming for a lunar soft landing. *IEEE Transactions on Automatic Control*, 9(4):477–484, 10 1964.

[47] E. Kaiser, J. N. Kutz, and S. L. Brunton. Data-driven discovery of koopman eigenfunctions for control. *Mach. Learn.: Sci. Technol.*, 2(3):035023, 6 2021.

[48] R. A. DeCarlo. *Linear Systems: A State Variable Approach with Numerical Implementation*. Prentice Hall, 1989.

[49] D. Goswami and D. A. Paley. Global bilinearization and controllability of control-affine nonlinear systems: A koopman spectral approach. In *Annual Conference on Decision and Control*, pages 6107–6112, Melbourne, Australia, 12 2017. IEEE.

[50] M. Korda and I. Mezić. Optimal construction of koopman eigenfunctions for prediction and control. *IEEE Transactions on Automatic Control*, 65(12):5114–5129, 12 2020.

[51] M. J. Colbrook, L. J. Ayton, and M. Szőke. Residual dynamic mode decomposition: Robust and verified koopmanism. *Journal of Fluid Mechanics*, 955:A21, 1 2023.

[52] S. L. Brunton and J. N. Kutz. *Data Driven Science & Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[53] C. Nantabut. Analysis of truncated singular value decomposition for koopman operator-based lane change model, 9 2024.

[54] W. A. Manzoor, S. Rawashdeh, and A. Mohammadi. Vehicular applications of koopman operator theory—a survey. *IEEE Vehicular Technology Society Section*, 11: 25917–25931, 3 2023.

[55] S. Klus, F. Nüske, S. Peitz, J. Niemann, C. Clementi, and C. Schütte. Data-driven approximation of the koopman generator: Model reduction, system identification, and control. *Physica D Nonlinear Phenomena*, 406:132416, 5 2020.

[56] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 3 2016.

[57] J. Benito and K. D. Mease. Reachable and controllable sets for planetary entry and landing. *Journal of Guidance, Control, and Dynamics*, 33(3):641–654, 5-6 2010.

[58] M. Sagliano, A. Heidecker, S. Farì, J. M. Hernández, M. Schlotterer, S. Woicke, D. Seelbinder, and E. Dumont. Powered atmospheric landing guidance for reusable rockets: the callisto studies. In *Scitech 2024 Forum*, Orlando, FL, USA, 1 2024. AIAA.

[59] J. Carradori, M. Sagliano, and E. Mooij. Transformer-based robust feedback guidance

for atmospheric powered landing. In *Scitech 2025 Forum*, Orlando, FL, USA, 1 2025. AIAA.

[60] M. A. Patterson and A. V. Rao. A general-purpose matlab toolbox for solving optimal control problems using variable-order gaussian quadrature collocation methods, 1 2014.

[61] J. Nocedal, A. Wächter, and R. A. Waltz. Adaptive barrier strategies for nonlinear interior methods. *SIAM Journal on Optimization*, 19(4):1674–1693, 1 2009.

[62] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 2013.

A | Neural Network Jacobian Matrix

The Jacobian matrix of the output of the neural network of eq. (2.24) with respect to the state $\mathbf{x}$ is derived, since it is needed to compute the time derivative of the basis functions obtained through dlEDMD. The Jacobian matrix of the network's output $\boldsymbol{\psi}(\mathbf{x})$ can be easily derived:

$$\nabla\boldsymbol{\psi}(\mathbf{x}) = \begin{bmatrix} 0_{1\times n} \\ I_{n\times n} \\ \nabla\mathbf{z}_f(\mathbf{x}) \end{bmatrix}$$

The term $\nabla\mathbf{z}_f(\mathbf{x})$ can be obtained with the chain rule:

$$\begin{cases} \nabla\mathbf{z}_0(\mathbf{x}) = W_i \\ \nabla\mathbf{z}_j(\mathbf{x}) = \left(I_{w_h\times w_h} + D_j^{sh}(\mathbf{x})W_j\right)\nabla\mathbf{z}_{j-1}(\mathbf{x}) \quad \forall j \in \{1,\dots,n_h\} \\ \nabla\mathbf{z}_f(\mathbf{x}) = W_f\nabla\mathbf{z}(\mathbf{x}) \end{cases}$$

where the matrix $D_j^{sh}(\mathbf{x})$ is used to introduce the derivative of the hyperbolic tangent with respect to its argument, knowing that $d(\tanh a)/da = \sinh^2 a$.

$$D_j^{sh}(\mathbf{x}) = \operatorname{diag}\left(\sinh^2\left(W_j\mathbf{z}_{j-1}(\mathbf{x}) + \mathbf{p}_j\right)\right) \quad \forall j \in \{1,\dots,n_h\}$$

with $\operatorname{diag}(\mathbf{a})$ denoting the diagonal matrix with $\mathbf{a}$ on the diagonal and with the hyperbolic sine applied element-wise. It becomes evident that the dot product between the Jacobian matrix of the network's basis functions and the generic control-affine vector field does not belong to the span of the basis functions themselves, exception made for particular cases.

B | Expression of the Aerodynamic Angles

In the case of planar motion $\nu = \zeta = 0$, furthermore also $v_{2,e} = \chi_v = 0$. The DCMs can be rewritten as follows:

$$\begin{bmatrix} c_\alpha c_\beta & s_\beta & s_\alpha c_\beta \\ s_\kappa s_\alpha - c_\kappa c_\alpha s_\beta & c_\kappa c_\beta & -c_\kappa s_\alpha s_\beta - c_\alpha s_\kappa \\ -c_\alpha s_\kappa s_\beta - c_\kappa s_\alpha & c_\beta s_\kappa & c_\kappa c_\alpha - s_\kappa s_\beta s_\alpha \end{bmatrix} = R_b^w = R_e^w R_b^e =$$

$$= \begin{bmatrix} c_{\gamma_v} & 0 & s_{\gamma_v} \\ 0 & 1 & 0 \\ -s_{\gamma_v} & 0 & c_{\gamma_v} \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} = \begin{bmatrix} c_\theta c_{\gamma_v} + s_\theta s_{\gamma_v} & 0 & -s_\theta c_{\gamma_v} + c_\theta s_{\gamma_v} \\ 0 & 1 & 0 \\ s_\theta c_{\gamma_v} - c_\theta s_{\gamma_v} & 0 & c_\theta c_{\gamma_v} + s_\theta s_{\gamma_v} \end{bmatrix}$$

By equating, for example, the first row and the central element of the leftmost and rightmost matrices, the following system of equations is obtained:

$$\begin{cases} \cos\alpha\cos\beta = \cos\theta\cos\gamma_v \\ \sin\beta = 0 \\ \sin\alpha\cos\beta = -\sin\theta\cos\gamma_v + \cos\theta\sin\gamma_v \\ \cos\beta\cos\kappa = -\sin\theta\cos\gamma_v + \cos\theta\sin\gamma_v \end{cases} \Rightarrow \begin{cases} \beta = a\pi \\ \kappa = a\pi \\ \alpha = \gamma_v - \theta + a\pi \end{cases} \quad \text{for} \quad a = 0, 1$$

List of Figures

2.1 dlEDMD neural network scheme. 28
2.2 Closed system Koopman sparsity pattern. 35
2.3 Closed system Koopman eigenfunctions evolution example. 36
2.4 Closed system mean error. 37
2.5 Asymptotically stable Duffing oscillator mean error. 38
2.6 Asymptotically stable Duffing oscillator eigenvalues. 38
2.7 Asymptotically stable Duffing oscillator example trajectory. 39
2.8 Stable Duffing oscillator mean error. 39
2.9 Stable Duffing oscillator eigenvalues. 40
2.10 Stable Duffing oscillator example trajectory. 40
2.11 Uncontrolled non-polynomial system mean error. 41
2.12 Non-polynomial system eigenvalues. 42
2.13 Uncontrolled non-polynomial system example trajectory. 42
2.14 Bilinear control non-polynomial system error, $T = 5$ TU, 10 basis functions. EDMD mean 0.33 SU, dlEDMD mean 0.26 SU. 43
2.15 Bilinear control non-polynomial system example trajectory. 43
2.16 Linear control non-polynomial system error, $T = 5$ TU, 10 basis functions. EDMD mean 0.47 SU, dlEDMD mean 0.24 SU. 44
2.17 Linear control non-polynomial system example trajectory. 45
2.18 Non-polynomial system truncated SVD analysis. 46
2.19 Uncontrolled non-polynomial system after truncated SVD error, $T = 5$ TU, 8 basis functions. dlEDMD mean 0.09 SU. 46

3.1 Rocket landing reference frames. 48
3.2 Rocket landing angles representation. 49

4.1 Uncontrolled 1D-aero mean error, $T = 10$ s, EDMD. 63
4.2 Uncontrolled 1D-aero mean error, $T = 40$ s, EDMD. 64
4.3 Uncontrolled 1D-aero mean error, EDMD and dlEDMD. 65
4.4 Uncontrolled 1D-aero example trajectory. 66

4.5 Linear control 1D-aero mean error, $T = 10$ s, EDMD and dlEDMD. 67
4.6 Bilinear control 1D-aero error, $T = 10$ s, EDMD and dlEDMD. 68
4.7 Controlled 1D-aero example trajectory. 69
4.8 Uncontrolled 2D-aero mean error, $T = 10$ s, EDMD. 71
4.9 Uncontrolled 2D-aero mean error, $T = 40$ s, EDMD. 72
4.10 Uncontrolled 2D-aero error, $T = 40$ s, EDMD and dlEDMD. 73
4.11 Linear control 2D-aero error, $T = 10$ s, EDMD and dlEDMD. 74
4.12 Bilinear control 2D-aero error, $T = 10$ s, EDMD and dlEDMD. 75
4.13 Controlled 2D-aero example trajectory. 76
4.14 2D-aero truncated SVD analysis, dlEDMD. 77
4.15 Bilinear control 2D-aero error after truncated SVD, $T = 10$ s, dlEDMD. . . 77
4.16 Uncontrolled 3D-noAero mean error, $T = 40$ s, GAL. 79
4.17 Bilinear control 3D-noAero error, $T = 10$ s, GAL. 80
4.18 Controlled 3D-noAero example trajectory. 81
4.19 3D-aero truncated SVD analysis, GAL. 82
4.20 1D-aero optimality conditions, bilinear model. 83
4.21 1D-aero OCP solution, bilinear model. 84
4.22 1D-aero OCP solution, linear model. 85
4.23 2D-aero optimality conditions, bilinear model. 87
4.24 2D-aero OCP solution, bilinear model. 88
4.25 3D-aero optimality conditions, bilinear model. 90
4.26 3D-noAero OCP solution, bilinear model. 91

List of Tables

2.1 Comparison of Koopman approaches. 30

4.1 Rocket landing environment constants. 61
4.2 Rocket landing bounds on state after normalization. 62
4.3 1D-aero optimal control parameters. 83
4.4 1D-aero optimal control performance. 86
4.5 2D-aero optimal control parameters. 87
4.6 2D-aero optimal control performance. 89
4.7 3D-noAero optimal control parameters. 89
4.8 3D-noAero optimal control performance. 90

List of Symbols

In the text, the following symbols can also appear in bold, meaning that they denote vectors. In any case, the type of variable they represent is the same. The SI units of variables associated with well-defined physical quantities are specified. Symbols associated with typical mathematical or physical objects are not listed (e.g. the N-dimensional euclidean space $\mathbb{R}^N$).

Variable	Description	
n	State dimension	-
m	Control dimension	-
q	Vector subspace dimension	-
S	Number of data samples	-
n_s	Number of Linear Program time segments	-
deg	Polynomial degree	-
r	Position	m
v	Velocity	m/s
m	Mass	kg
m_r	Mass reciprocal	1/kg
η	Throttle level	-
ν	Yaw angle	rad
θ	Pitch angle	rad
ζ	Roll angle	rad
g	Gravity acceleration	m/s^2
Γ	Thrust	N
F^P	Propulsive force	N
V_{ex}	Engine exhaust velocity	m/s
F^A	Aerodynamic force	N
ρ	Air density	kg/m^3

Variable	Description	
ρ_0	Reference density	$\mathrm{kg/m^3}$
R	Reference altitude	m
V_s	Speed of sound	m/s
G_i	Aerodynamic coefficient in i-th body direction	$\mathrm{m^2}$
γ_v	Vertical flight-path angle	rad
χ_v	Vertical azimuth angle	rad
α	Angle of attack	rad
β	Sideslip angle	rad
κ	Bank angle	rad
e	Target-fixed reference frame	
b	Body-fixed reference frame	
w	Wind-fixed reference frame	
t	Time	s
Δt	Time step	s
T	Trajectory time duration	s
ε_T	State error on trajectory of time duration T	
x	State	
y	Single-step propagation of state	
u	Control	
f	Continuous-time dynamics	
F	Discrete-time dynamics	
Ψ	Vector subspace	
ψ	Basis function of vector subspace	
w_Ω	Weighting function for vector subspace	
Ω	Vector subspace domain	
$\mathcal{K}$	Koopman infinitesimal generator	
$\mathcal{K}^d$	Koopman discrete operator	
$\mathcal{K}_u$	Koopman generator parametrized by the control	
$\mathcal{K}_u^d$	Koopman operator parametrized by the control	
K	Koopman continuous matrix	
K^d	Koopman discrete matrix	

Variable	Description
K_c	Koopman continuous matrix for linear control
K_c^d	Koopman discrete matrix for linear control
K_ξ	Koopman continuous matrix associated with function ξ
K_ξ^d	Koopman discrete matrix associated with function ξ
φ	Koopman eigenfunction
V	Matrix of Koopman right eigenvectors
Λ	Diagonal matrix of Koopman continuous eigenvalues
L	Koopman lifted state variable
P	Koopman projection matrix
λ	Costate
μ^f	Endpoint constraint multiplier
μ	Path constraint multiplier
H	Hamiltonian function
H^a	Augmented Hamiltonian function
Υ	Endpoint function
S_w	Switching function
σ	Optimality condition function fo rocket landing
x_{LP}	Linear Program optimization variable
d_f	Linear Program cost function coefficient
d_{eq}	Linear Program equality constraint constant
D_{eq}	Linear Program equality constraint coefficient
d	Linear Program inequality constraint constant
D	Linear Program inequality constraint coefficient
d_l	Linear Program lower bound constant
D_l	Linear Program lower bound coefficient
d_u	Linear Program upper bound constant
D_u	Linear Program upper bound coefficient
ω	Neural network parameter
l	Neural network external layer
h	Neural network hidden layer
n_h	Neural network hidden layers number

Variable	Description
w_h	Neural network hidden layers width
l_r	Neural network learning rate
z	Neural network layer output
W	Neural network weight matrix
p	Neural network bias
D^{sh}	Neural network matrix of hyperbolic sine
V_Σ	Matrix of right singular vectors
U_Σ	Matrix of left singular vectors
$U_{\Sigma,i}$	Matrix of left singular vectors truncated at i-th value
Σ	Rectangular diagonal matrix of singular values
E	Singular values total energy
E_i	Cumulative energy up to i-th singular value
J	Generic cost function
ε	Generic tolerance
ξ_x	Data matrix of function ξ evaluated at samples of x
r	Tikhonov regularization parameter
s_δ	Sine of generic angle δ
c_δ	Cosine of generic angle δ
R_i^j	Matrix to rotate vector from frame i to frame j

Acknowledgements

I am really thankful to the German Aerospace Center (DLR) in Bremen for hosting me during my work, and in particular to Dr. Marco Sagliano for supporting me in the development of this thesis. I am also thankful to Prof. Francesco Topputo for supervising this study. Lastly, I want to thank my family and my friends who supported me during all my academic life and especially in the last period.