

Thesis report

Integrating space mapping and machine learning techniques for enhanced optimal control mapping

AE5211: Thesis

Guy Maré

Thesis report

Integrating space mapping and machine
learning techniques for enhanced optimal
control mapping

by

Guy Maré

4670132

Supervisor: C. Varriale
Faculty: Faculty of Aerospace Engineering, Delft

Cover: five-jets-on-sky by María Noel Rabuñal Cantero

Summary

This study explores the integration of machine learning with space mapping techniques to enhance the mapping of optimal control sequences between low- and high-fidelity flight mechanic models. Space mapping is a methodology that simplifies the control optimisation process by approximating a high-fidelity model using less computationally demanding low-fidelity models, which are then iteratively corrected to converge towards high-fidelity outputs. The main research question investigates how the integration of space mapping with sequence-to-sequence neural networks can improve control sequence mapping compared to traditional model predictive control (MPC) methods, particularly in managing the trajectory differences in non-linear flight regimes.

In the pursuit of sustainable aviation, with a sharp focus on reducing emissions through innovative designs and enhanced flight mechanics, the computational cost of high-fidelity models becomes a significant limitation. These models, crucial for capturing complex interactions in advanced aircraft designs, often require simplification to reduce computational demands. This research proposes a novel approach by combining the strengths of machine learning, particularly sequence-to-sequence neural networks like Gated Recurrent Units (GRUs) and transformers, with space mapping techniques to bridge the gap between low- and high-fidelity models effectively.

The study delves into two main machine learning architectures: GRUs and transformers. GRUs excel in managing sequences with fewer changes, maintaining stable predictions with minimal error. Transformers on the other hand are well suited at handling complex sequences with frequent changes, thanks to their ability to process entire sequences simultaneously through self-attention mechanisms. This capability makes transformers particularly suitable for dynamic scenarios where anticipating future states is crucial.

A significant contribution of this study is the implementation of the Prior Knowledge Input-Difference (PKI-D) architecture, which uses the low-fidelity model output as a baseline that the neural network corrects, providing a robust framework for the machine learning models to accurately predict trajectory adjustments. This architecture not only enhances the predictive accuracy but also optimises computational efficiency by reducing the dependency on extensive high-fidelity simulations.

Comparative analyses reveal that MPC methods typically provides superior mapping performance for trajectories requiring no anticipation, while the hybrid machine learning-space mapping approach offers improved performance comparably or better in complex scenarios requiring advanced anticipation. This study highlights the critical role of active learning in adapting the machine learning models to new data dynamically, a feature that proves essential in maintaining accuracy over prolonged operational periods.

In conclusion, this research demonstrates that integrating space mapping with machine learning can significantly enhance the mapping of control sequences in aerospace applications. It provides a starting point for future studies to explore tailor made machine learning solutions using extremely small data sets in situations where data availability is sparse. This research could further open up avenues where the advanced capabilities of machine learning can be applied to problems in aerospace engineering previously inaccessible.

Contents

| | |
|--|-----------|
| Summary | i |
| Nomenclature | iv |
| 1 Introduction | 1 |
| 2 Theoretical Foundation | 3 |
| 2.1 Optimal Control Theory | 3 |
| 2.2 Complexity Reduction | 6 |
| 2.3 Space Mapping | 9 |
| 2.4 Sequence-to-Sequence Neural Networks | 16 |
| 3 Space Mapping & Machine Learning: Research Direction & Integration Challenges | 27 |
| 3.1 Research direction | 27 |
| 3.2 Integration challenges | 28 |
| 3.3 Space mapping problem formulation | 30 |
| 3.4 Test cases | 31 |
| 4 Mapping Function Implementation | 35 |
| 4.1 MPC mapping function | 35 |
| 4.2 Mapping function: Machine Learning | 36 |
| 4.3 Machine learning model robustness | 40 |
| 5 Model Predictive Control Based Space Mapping | 47 |
| 5.1 Number of segments | 50 |
| 5.2 Planning horizon | 52 |
| 6 Sequence-to-Sequence Network Based Space mapping | 55 |
| 6.1 Comparing network types | 55 |
| 6.2 Influence of active-learning & retraining | 62 |
| 7 Results Analysis & Discussion | 67 |
| 7.1 Results summary | 67 |
| 7.2 Answering the research question | 68 |
| 7.3 Discussion | 70 |
| 8 Conclusion & suggestions for future research | 72 |
| 8.1 Conclusion | 72 |
| 8.2 Suggestions for future research. | 73 |
| References | 74 |
| A Low-Fidelity Flight Mechanics Models | 79 |
| A.1 Open loop | 80 |
| A.2 Closed loop | 80 |
| A.3 High-fidelity model | 81 |
| B Error Determination Methods | 82 |

| | | |
|----------|--|-----------|
| C | MPC method supporting results | 85 |
| C.1 | Base implementation | 85 |
| C.2 | Number of segments analysis | 87 |
| C.3 | Planning horizon analysis | 89 |
| D | Machine Learning method results | 91 |
| D.1 | Network type analysis | 91 |
| D.2 | Active learning analysis | 96 |

Nomenclature

Abbreviations

| Abbreviation | Definition |
|--------------|--|
| DoF | Degree of Freedom |
| RNN | Recurrent Neural Network |
| LSTM | Long-Short Term Memory |
| GRU | Gated Recurrent Unit |
| HBVP | Hamiltonian-Boundary-Value-Problem |
| PMP | Pontryagin's Maximum Principle |
| LSED | Lock-Step Euclidean Distance |
| LCSS | Longest Common Sub-Sequence |
| DFD | Discrete Fréchet Distance |
| DTW | Dynamic Time Warping |
| RBF | Radial Base Functions |
| GPU | Graphics Processing Unit |
| PKI-D | Prior Knowledge Input-Difference model |

1

Introduction

To meet the International Air Transport Association (IATA)’s goal of achieving carbon neutrality by 2050, the aviation sector must significantly reduce its emissions. Current projections estimate that annual CO₂ emissions could reach nearly two thousand megatons by 2050, nearly double today’s levels, without effective reduction measures [1][2]. Research is focusing on replacing fossil fuels with sustainable aviation fuels and enhancing aircraft efficiency. Innovative designs, such as distributed propulsion systems and unconventional configurations like blended wing bodies or Prandtl planes, are being explored to improve aerodynamic and propulsion synergy, potentially reducing drag and improving fuel economy [3].

However, moving away from the traditional tube-with-wing aircraft configuration introduces numerous challenges, especially in the design and simulation phases. The increased complexity in aerodynamic-propulsion coupling in these new models makes simulation computationally intensive, a significant concern in design processes relying on iterative methods like multidisciplinary design optimisation. Here, each cycle of design adjustments and analyses aims to converge to an optimum, including assessing aircraft handling and determining optimal control input sequences for certain predefined manoeuvres—a process falling within optimal control theory. Using detailed models for this optimisation quickly becomes impractical due to the high number of states and control inputs, compounded by the non-linear aerodynamics near the flight envelope’s edge.

A common approach to manage computational demands is employing simplified or low-fidelity models, which, while easier to calculate, often lack the ability to accurately represent complex interactions between subsystems. A key step in utilising low-fidelity surrogates is to make sure that the output of these models align. A technique that can be used to align models of varying fidelity is through the use of space mapping. This method aims to seek a map between the input and output, or parameter space of models to align their outputs. The key advantage of space mapping is that they offer tailor-made solutions to surrogate based problems using relatively simple algorithms while keeping the computational cost to a minimum. Another emerging solution lies in machine learning, where networks can learn complex behaviours through training. Once trained, these networks can rapidly analyse intricate inputs and seek out existing patterns.

In theory, both space mapping and sequence-to-sequence based neural networks are designed as mapping functions, their approach to mapping diverge fundamentally. As mentioned before, space mapping offers bespoke solutions, whereas machine learning seeks to find a general mapping functions. Furthermore, machine learning methods often require large data sets, whereas space mapping methods aims to minimise the amount of data. The central challenge of this research is to merge the advanced sequence-to-sequence mapping performance of machine learning, with the low computational cost required for

space mapping. The goal is to effectively bridge the gap between low and high-fidelity models within the realm of control optimisation, thereby addressing a critical need for more efficient and accurate modelling techniques for unconventional aircraft and complex flight scenarios.

This report starts with a review of existing literature to build a foundational understanding in optimal control, space mapping, and machine learning in chapter 2. Then, chapter 3 breaks down how the different core principles of space mapping and machine learning can be combined into a single framework and introduces the main research question. Then, chapter 4 presents a deep dive into both the space mapping and a hybrid space mapping-machine learning implementation which leads into a robustness analysis of the machine learning approach. The method specific results are presented in chapter 5 and chapter 6. Following the presentation of the main results, chapter 7 combines the main takeaways to answer the main research question. Lastly, a general conclusion is given in chapter 8, followed by recommendations for future research based on the lessons learned during this project.

2

Theoretical Foundation

This chapter explores the theoretical foundations necessary for understanding space mapping and machine learning in the context of control theory. It begins by discussing the roles of low- and high-fidelity models in control optimisation, followed by an in-depth exploration of space mapping techniques. The chapter then transitions to an examination of machine learning, with a focus on sequence-to-sequence networks, crucial for enhancing model integration and performance. This literature review lays the groundwork for appreciating current advancements and pinpointing the research gaps that this study seeks to explore.

2.1. Optimal Control Theory

As the name suggests, optimal control theory is about calculating the best set of control inputs for an aircraft to satisfy a certain objective. The objective in question can range from determining the optimal flight path considering weather circumstances [4], to evaluating the landing performance of rotorcraft vehicles [5]. Independent of its specific application, the pursuit of optimising control strategies for air and spacecraft has been a longstanding endeavour.

Typically, control optimisation starts with a flight dynamics model of the aircraft in question containing all the relevant performance metrics. The equations of motion can be of varying complexity depending on the level of fidelity that is desired. In general for a continuous time model, the evolution of the state variables $\underline{x}(t)$ of the system can be written as a set ordinary differential equations:

$$\dot{\underline{x}} = \underline{f}(\underline{x}(t), \underline{u}(t), t) \quad (2.1)$$

Here, $\underline{u}(t)$ represents the control inputs to the system, for example stick movement done by the pilot. Starting at some initial point, the evolution of the states and inputs are incorporated into the objective function:

$$J(\underline{x}(t), \underline{u}(t)) = \int_{t_0}^{t_f} \underline{L}(\underline{x}(t), \underline{u}(t), t) dt + \Phi(\underline{x}(t_f), t_f) \quad (2.2)$$

subject to:

$$\underline{\phi}(\underline{x}(t_0), t_0, \underline{x}(t_f), t_f) = \underline{0} \quad (2.3)$$

$$\underline{C}(\underline{x}(t), \underline{u}(t), t) \leq \underline{0} \quad (2.4)$$

The objective function J can be divided into the endpoint cost, and the running cost. The endpoint can be interpreted in the context of flight trajectory as the penalty in case the desired target is not reached. The running cost, also called Lagrangian term, represents the cost of performing movements. This

term is given as the Lagrangian L integrated over the time-frame of the trajectory. The terms $\underline{\phi}$ and \underline{C} represent the boundary and path constraints imposed on the trajectory and control inputs. These can be incorporated in the Equation 2.2 to arrive at the constrained objective function:

$$J = \underline{\Phi} - \underline{\nu}^T \underline{\phi} + \int_{t_0}^{t_f} [L - \underline{\mu}^T \underline{C} + \underline{\lambda}^T (\underline{f} - \underline{\dot{x}})] dt \quad (2.5)$$

Here ν and μ are the Lagrange multipliers for the boundary and path constraints. Furthermore the term λ is incorporated. This is the co-state, an important parameter that relates the change in state to the associated cost for the objective function. The term $\underline{f} - \underline{\dot{x}}$ represents the difference between the state dynamics and the actual rate of state change. This term is added when performing discrete optimisation to compensate between any discrepancies between the system dynamics and the actual trajectory changes. Equation 2.5 can be reduced to its augmented form by introducing the Hamiltonian function H [6]:

$$\underline{J}_a = \underline{\Phi} - \underline{\nu}^T \underline{\phi} + \int_{t_0}^{t_f} [\underline{H} - \underline{\lambda}^T \underline{\dot{x}}] dt \quad (2.6)$$

with:

$$\underline{H} = \underline{L} - \underline{\mu}^T \underline{C} + \underline{\lambda}^T \underline{f} \quad (2.7)$$

Pontryagin's Maximum Principle

Using calculus of variation, Pontryagin's Maximum Principle (PMP) states that given for an optimal trajectory x^* and for any admissible small perturbation of the optimal trajectory δx :

$$\delta \underline{J}(\underline{x}^*, \delta \underline{x}) = 0, \forall \delta \underline{x} \quad (2.8)$$

This condition ensures that for a given solution, the derivative at that point is zero, meaning it is at least a local minimum if not a global minimum [7]. Note that even if the solution proves to be a suitable candidate, it does not automatically mean that it is a global optimal trajectory. This is a limit of PMP. Further test such as examining the second derivative of J must be done to assess the nature of the solution. The dynamic constraint, Equation 2.1, is also included to enforce that the solution adheres to the system dynamics. Because this theorem is assumed to be in continuous time and not in discrete, the term $\underline{f} - \underline{\dot{x}}$ is simply replaced by \underline{f} . The set of conditions described above form a Hamiltonian-Boundary-Value-Problem (HBVP). Solving this problem can be done via two ways which will be discussed next.

2.1.1. Solving HBVPs

Like any optimisation problem, there exist various methodologies to address the problem, each way having its benefits depending on the nature of the problem. Similarly, solving optimal control problems in the form of a HBVP has its own challenges depending on how the system is defined.

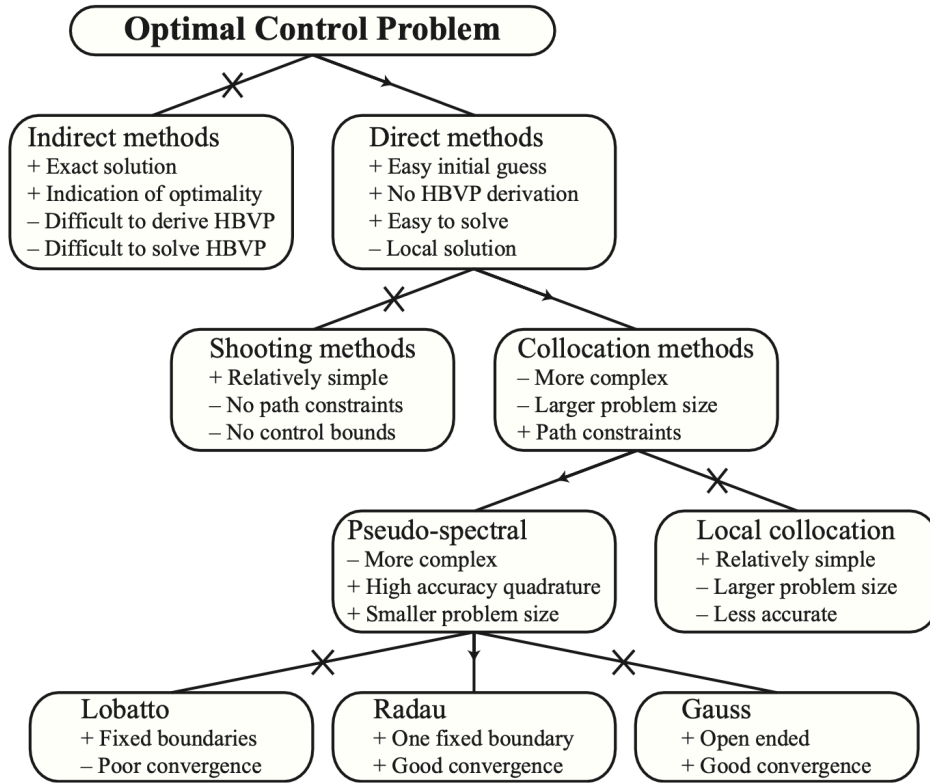


Figure 2.1: Overview of different optimisation methods based on problem properties. Adapted from [6].

According to [6], classical optimal control approaches can roughly be divided into direct and indirect methods. Indirect methods use PMP to solve the problem. These methods are dubbed indirect because they rely on deriving first order optimality conditions to reach an optimal trajectory, instead of directly optimising the trajectory.

One of the problems that Figure 2.1 shows is that in practice it is often difficult to derive the HBVP equations. While this method yields an exact solution, deriving the optimality conditions is very difficult. If the constraints are changed, which is often the case in system level optimisation, the whole derivation process has to be repeated. Another issue is that the optimiser is very sensitive to initial starting point and the costate. As the costate is not easily interpretable, this could lead to slow convergence if not properly set.

Direct methods on the other hand rely on numerical methods to find an optimal solution. These are in general more easy to work with and more robust, at the downside that it is not possible to determine if a solution is a global or local optimum.

2.1.2. Practical limits of optimal control theory

Optimal control theory inherently intersects with non-linear optimisation, especially when addressing complex, real-world control problems. Even moderately high-fidelity flight mechanics models can introduce non-linear behaviour which imposes various challenges and restrictions to the problem. These are not just theoretical concerns but directly impact the practical application of optimal control theory.

Like with any optimisation problem, there are limits in terms of size and complexity of the problem to what can be optimised. A big part of the problem has to do with the so called "curse of dimensionality" first coined by Richard Bellman in 1957 [8]. The problem is that computational effort does not scale linearly with the size of the problem. A fine example is that matrix operations typically scale between $\mathcal{O}(n^{2.5})$ and $\mathcal{O}(n^3)$ [9]. Increasing the amount of states therefore has a significant impact on the computational cost. This is combined with the fact that higher dimension models are often more

taxing to evaluate adding to the total evaluation time.

In the world of optimisation, there is a notion that regarding optimisers and the problems that they solve that *"there is no free lunch"* [10]. This means that there is no general optimisation algorithm that is good in any task. In more detail this means that an optimiser designed for a specific task will under perform if applied to another problem set. Applying this concept to optimal control, this means that depending on which approach is chosen in Figure 2.1, a different optimiser might be better suited. The following commonly used optimisation methods exemplify the challenges faced when performing complex optimisation.

Gradient based optimisation

Gradient based methods are a popular choice for optimisation due to their intuitive operation, ability to handle constraints, and relatively straight forward approach. If second derivative information is available, gradient based methods become an even more efficient optimisation tool [9]. A large caveat to this approach is that if gradient information is not readily available, some of its advantages diminish. Finite difference methods which are used to calculate gradients are prone to inaccuracies depending on the problem structure. The objective function might be non-convex, discontinuous, or very noisy [11]. In such a case, the calculated gradient based on finite difference might not reflect the real gradient. This could lead to oscillations in the search process or prevent convergence entirely. Another point is that computing gradients can become computationally expensive if the objective function is difficult to evaluate.

Nevertheless for lower dimension, well-posed problems are an excellent candidate for gradient based methods. One such an example is the interior point method employed in [4]. Interior point methods traverse the space bounded by constraints to search for an optimum. In more detail, interior point optimisation was applied through an open source python library called "IPOPT"¹. This library is a popular choice for solving non-linear optimisation problems.

Bayesian optimisation

Another popular approach is to treat the objective function as a black box without gradient information. Using Gaussian Processes, a predictive model is constructed by strategically sampling the objective function. This predictive model can make estimations with a certain degree of confidence. By strategically weighing exploratory and exploitative sampling, the optimiser can refine the optimisation in regions near the optimal point, while also exploring the whole design space [12]. Especially problems where the evaluations cost is high, Bayesian optimisation is a good candidate. A downside however is that Bayesian optimisation does not fare well under high dimensional problems. This is because of the need of repeated inversion of sample matrices. The more samples are taken the larger this matrix gets. Typically, Bayesian optimisation is limited to 15 - 20 dimensions [12] [11]. Similar to the IPOPT package, there exist packages for Bayesian optimisation. For lower dimension problems or parameterised problems Bayesian optimisation can still be used for trajectory optimisation [13].

2.2. Complexity Reduction

In the realm of optimal control optimisation, a natural strategy when faced with complex modelling tasks is to explore possibilities for simplification. This approach is not only for convenience, in some cases it is also a necessity if the problem is too complex to handle. However, this simplification process comes with its own set of trade-offs. While making a model more computationally manageable, there is an inherent sacrifice in model accuracy. This section will dive into intricacies of low- and high-fidelity models.

¹<https://coin-or.github.io/Ipopt/>

2.2.1. High-fidelity flight mechanics model

High-fidelity models, typically represented by 6-DoF frameworks, are based on thirteen equations implemented in closed-form. These models simulate the physical forces and movements affecting an aircraft, offering a detailed and precise depiction of flight dynamics:

Conservation of Linear Momentum Equations (CLMEs)

$$\dot{u} = rv - qw + \frac{1}{m} (W_x + F_x^{(A)} + F_x^{(T)}) \quad (2.9)$$

$$\dot{v} = -ru - pw + \frac{1}{m} (W_y + F_y^{(A)} + F_y^{(T)}) \quad (2.10)$$

$$\dot{w} = qu - pv + \frac{1}{m} (W_z + F_z^{(A)} + F_z^{(T)}) \quad (2.11)$$

Here, W represents the components of the weight vector, $F^{(A)}$ represents the aerodynamic force, and $F^{(T)}$ represents the thrust force expressed in the body frame of reference. u, v, w represent the components of the velocity vector. $p, q,$ and r represent the angular velocity components. Specific rules can be setup for the thrust and aerodynamic models. Every force acting on the aircraft can be constructed from simple polar curves or user defined functions. These can be simple values, to complex equation, and even 3D lookup tables for complex aerodynamics for a wide range of scenarios.

Conservation of Angular Momentum Equations (CAMEs)

Similar to the CLMEs, the forces and moments can be user defined functions which take the aircraft states as inputs. These equations can be written as:

$$\dot{p} = (C_1 r + C_2 p) q + C_3 \mathcal{L} + C_4 \mathcal{N} \quad (2.12)$$

$$\dot{q} = C_5 p r - C_6 (p^2 - r^2) + C_7 \mathcal{M} \quad (2.13)$$

$$\dot{r} = (C_8 p - C_2 r) q + C_4 \mathcal{L} + C_9 \mathcal{N} \quad (2.14)$$

Here, C_1 to C_9 are model inertia constants calculated with respect to the body axis. $\mathcal{L}, \mathcal{M},$ and \mathcal{N} are the sum of moments for roll, pitch, and yaw respectively. These moments are calculated around the centre of mass and are determined by aerodynamic and thrust forces.

Flight Path Equations (FPE)

The flight path equations are needed to project the aircraft dynamics to the earth based reference frame. This transformation is given by:

$$\begin{bmatrix} \dot{x}_{E,G} \\ \dot{y}_{E,G} \\ \dot{z}_{E,G} \end{bmatrix} = [T_{EB}] \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (2.15)$$

Here the subtext E, G means the location of centre of gravity G in the earth reference frame E . T_{EB} is a matrix that is based on the aircraft's attitude quaternion. The components of this quaternion are calculated from the CLME and CAME equations and the kinematic equations described in the next section.

Kinematic Equations (KE)

The kinematic equations describe how the attitude quaternion is updated:

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{bmatrix} \quad (2.16)$$

The set of CLME, CAME, FPE, and KE gives the complete set of thirteen equations for the aircraft state [14]. Combined with a user defined input set this describes the motion of the aircraft when simulated in JSBSim. Based on the definition of the aerodynamic and thrust forces, this can quickly become a complex model to do optimisation with.

2.2.2. Low-fidelity flight mechanics model

A common method to simplify a model involves reducing the complexity of the aerodynamic and propulsive models. This is an inherently difficult task due to the complex interplay of aerodynamic forces and propulsion dynamics. Due to the non-linearity, this is often computationally very intensive. This complexity is often compounded by the need to accurately capture the effects of these reductions on the overall system behaviour. Alternatively, other approaches focus on simplifying system dynamics by reducing the degrees of freedom to three, limiting the system to translation only. For instance, some models limit the scope to longitudinal motion only, involving two translational movements and one rotation (pitch), which offers a more manageable but still representative analysis of the system's dynamics. Additionally, simplifications to symmetric flight are frequently employed, where the focus is on conditions that assume symmetric behaviour of the aircraft about its longitudinal axis. For instance, some models limit the scope to longitudinal motion only, involving two translational movements and one rotation (pitch), which offers a more manageable but still representative analysis of the system's dynamics. In practice, a general 3-DoF system is commonly defined as:

$$\frac{d}{dt} \begin{bmatrix} X \\ Y \\ h \\ V \\ \psi \\ \phi \\ \gamma \\ W \end{bmatrix} = \begin{bmatrix} V \cos(\psi) \cos(\gamma) + \omega_x \\ V \sin(\psi) \cos(\gamma) + \omega_y \\ V \sin(\gamma) + \omega_z \\ \frac{g}{W} [(T \cos(\theta - \gamma) - D) - W \sin(\gamma)] \\ \frac{g \sin(\phi)}{WV} [L + T \sin(\alpha)] \\ \frac{g}{WV} [(L + T \sin(\theta - \gamma)) \cos(\phi) - W \cos(\gamma)] \\ -CT \end{bmatrix} \quad (2.17)$$

Here x, y, z represent translation of the aircraft. θ, ϕ, Ψ present aircraft pitch, roll, and heading angle respectively. V represent the velocity and W the weight of the aircraft. Furthermore γ represent the flight path angle, and α represents the angle of attack. θ is the geometric pitch angle of the aircraft. Drag is calculated using a quadratic drag polar using the lift coefficient [15].

This approach can be adapted to meet specific modelling requirements, as exemplified in [16]. The equations governing thrust can be refined to account for altitude differences using empirical relations. Another example is improved drag modelling. Based on the phase of the flight, the aircraft configuration and thus drag is changed. This allows for better accuracy in scenarios where the effect of drag on aircraft performance is significant. Despite the simplistic nature of the model, targeted improvements can improve accuracy in key areas. An example of such an improvement is to add the inputs as states to the model, and specify the rates of change as inputs. This mimics the interaction between control surface deflection and resulting attitude change more closely at limited cost.

2.2.3. Model limits

As stated in the introduction of this section, simplifications are a matter of trading accuracy for computational speed. When a low-fidelity model is used as described in subsection 2.2.2 instead of the model described in subsection 2.2.1, there will be a limit to which trajectories can be simulated. These difference stem from the following differences.

Point mass assumption

When reducing a model's weight distribution to a point mass, rotational dynamics are disregarded. Manoeuvres involving substantial rotational changes are no longer accurately represented within the model. This extends to aerodynamic modelling as well. Aerodynamic moments which are a dominant contribution to in the whole flight envelope, but are particularly sensitive near the edge of the flight envelope, are not incorporated. This limits the use cases for these types of models.

Simplified propulsive modelling

A prevalent methodology in flight mechanics involves setting the thrust as a constant value, modulated by a pilot-controlled setting to determine the actual thrust generated by the aircraft. However, this approach often overlooks several critical environmental factors that affect thrust production, such as altitude and inlet velocity. [16] demonstrated that the impact of these environmental influences could be effectively modelled using empirical relationships derived from aircraft parameters and state data. Incorporating these influences as scaling factors to the thrust used during optimisation can significantly enhance model accuracy while incurring minimal computational overhead. Furthermore, engine specific behaviour such as spool up time is often not modelled as well.

Simplified aerodynamic modelling

Another simplification is the use of linear aerodynamics. While this is a valid approach for small angle perturbations, it ignores the effect of stall for higher angles of attack. This limitation becomes particularly pronounced in scenarios characterised by low velocities or high angles of attack, such as during landing and take-off phases, slow climbs, and slow turns. This means that for low velocities or high angles of attack this approach ignores a dominant factor. Examples of scenarios that operate within these conditions are landing and take-off, slow climb, and slow turns. Under these conditions, the linearised approach neglects a dominant aspect of aerodynamic behaviour, potentially leading to significant discrepancies in model predictions.

In practical applications, the strategy of reducing model complexity is a commonly adopted technique, despite its inherent limitations in accuracy. By carefully defining the scope of the problem and weighing which aspects to exclude, many of the shortcomings associated with low-fidelity models can be effectively mitigated. A notable example of this approach is shown in [17]. In this particular study, the goal was to design an algorithm to assist in deep stall recovery using optimal trajectory planning using a 3-DoF model. This model choice was strategic, as the primary dynamics in this context involved rotational motion rather than translational. As a result, the model was specifically adapted to compute dynamic moments, rather than forces. Subsequent validation using a comprehensive 6-DoF model confirmed the adequacy of the results. This case demonstrates that a well-defined and simplified problem can yield reliable outcomes. However, the effectiveness of such an approach is inherently limited by the structural constraints of the problem. In contrast, employing a 6-DoF model directly offers a broader range of capabilities, allowing for a more diverse and detailed exploration of flight dynamics.

2.3. Space Mapping

A crucial part of employing surrogate based optimisation is that the low-fidelity model's output should accurately represent the behaviour of the high-fidelity model. The process of aligning the low-fidelity model with the high-fidelity model in an iterative manner is called space-mapping (SM).

2.3.1. Space Mapping Principles & Theory

SM seeks to remedy model misalignment via several different approaches. First, the input space of the low-fidelity model can be adjusted to account for modelling simplifications. An example of this is adjusting control variables to compensate for modelling differences. Alternatively, space mapping can be applied to the model's output through the application of a correction factor or bias to rectify any differences between the predicted responses of the low- and high-fidelity models. Lastly, the parameters of the low-fidelity model itself can also be tuned to more accurately reflect the high-fidelity model. In the context of aircraft trajectory planning, the low-fidelity model could adjust its weight, wingspan, or other physical parameter to align better with the high-fidelity model. Note that space mapping originally was intended to fine and coarse meshes, hence the notation "fine" and "coarse". A great way to visualise the principle of space mapping is through the following illustration [18]:

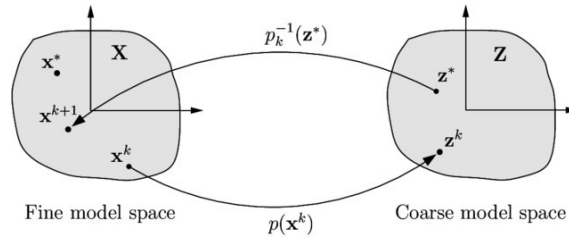


Figure 2.2: The concept of space mapping visualised. Adapted from [19].

At its core, space mapping is an optimisation problem, striving to minimise the discrepancies between coarse and fine models [19]. The overarching aim of space mapping is to *achieve a satisfactory solution with a minimal number of computationally expensive "fine" model evaluations*[20]. Herein lies also one of the key characteristics of SM. This technique produces tailor-made solutions for surrogate modelling and optimisation problems. Generating a general mapping would mean that more high-fidelity function evaluations than necessary would be needed to generate a complete mapping, as opposed to only mapping the point of interest. In formal terms, space mapping can be stated as:

$$x_c = P(x_f) \quad (2.18)$$

$$R_c(P(x_f)) \approx R_f(x_f) \quad (2.19)$$

Here $R(x)$ represents the model output for a given input point x , usually in the fine model space. By taking the inverse of Equation 2.18 and optimising the low-fidelity model, a solution suitable for the high-fidelity model can be found. The initial implementation of this concept used a linear mapping between the low- and high-fidelity space[20].:

$$x_c = P^{(j)}(x_f) = B^{(j)}x_f + c^{(j)} \quad (2.20)$$

with $B^{(j)} \in \mathbb{R}^{n \times n}$, and $c^{(j)} \in \mathbb{R}^{n \times 1}$. To establish a proper mapping, an initial m points in x_f are evaluated. Then, through parameter extraction, the corresponding x_c are found:

$$x_c^{(j)} = \arg \min_{x_c} \| R_f(x_c^{(j)}) - R_c(x_c) \| \quad (2.21)$$

Effectively, for a given set of high-fidelity points, the corresponding low-fidelity points are obtained through minimisation of the prediction error. Then, with the high-fidelity points and the corresponding low-fidelity points, the parameters in B and c are iteratively updated:

$$\bar{x}_f \approx x_f^{(m_j+1)} = (P^{(j)})^{-1}(x_c^*) \quad (2.22)$$

Here j denotes the iteration number. If the initial m high-fidelity points are not enough to establish a sufficient mapping, additional high-fidelity evaluations can be performed. This iterative method, while simple in nature, comes with several considerations. The relationship between low- and high-fidelity models is rarely linear, complicating the mapping process. Furthermore, an initial set of fine model evaluations, necessary for creating a mapping, might not be ideally positioned, leading to a sub-optimal starting point. Additionally, simplifications inherent in the model may result in non-unique solutions during the parameter extraction phase.

Aggressive Space Mapping

Aggressive Space Mapping (ASM) is an improvement of the original space mapping technique making use of a quasi-Newton iterative approach. The goal of ASM is to solve the non-linear system:

$$f(x_f) = 0 \quad (2.23)$$

Here, $f^{(j)}(x)$ is the error vector at iteration J . Each iteration, both the low- and high-fidelity model are evaluated. Then, using the mapping Jacobian $J_{(p)}$, the parameters in x_f are updated and a new high-fidelity point is found:

$$J_{(p)}^{(j)} h^{(j)} = -f^{(j)} \quad (2.24)$$

$$x_f^{(j+1)} = x_f^{(j)} + h^{(j)} \quad (2.25)$$

The mapping Jacobian can be estimated using finite difference. The algorithm terminates if the mapping error is sufficiently small. In pseudo code this approach is formulated as follows:

Algorithm 1 Aggressive Space Mapping (ASM)

Require: High-fidelity model R_f , low-fidelity model R_c , initial guess $x_f^{(0)}$, tolerance ϵ , maximum iterations N_{\max}

Ensure: Optimised parameters x_f^*

- 1: Evaluate initial discrepancy $f^{(0)} \leftarrow R_f(x_f^{(0)}) - R_c(x_f^{(0)})$
 - 2: **while** $\|f^{(j)}\| > \epsilon$ and $j < N_{\max}$ **do**
 - 3: Calculate the mapping Jacobian $J_{(p)}^{(j)}$ at $x_f^{(j)}$
 - 4: Solve for update step $h^{(j)}$ such that $J_{(p)}^{(j)} h^{(j)} = -f^{(j)}$
 - 5: Update parameters $x_f^{(j+1)} = x_f^{(j)} + h^{(j)}$
 - 6: Evaluate new discrepancy $f^{(j+1)} = R_f(x_f^{(j+1)}) - R_c(x_f^{(j+1)})$
 - 7: **end while**
 - 8: **return** $x_f^{(j)}$ as x_f^*
-

A common adaptation of this method is the inclusion of trust regions. Based on the accuracy of the low-fidelity model, the model can locally be trusted to be an accurate representation of the high-fidelity model. This allows the optimiser to locally optimise the model, instead of only taking a single step during the iteration. In the implementation above, this adaptation would be inserted in line 7, and would increase the overall efficiency of the whole approach. Depending on if the solution is satisfactory, the trust region can be expanded or decreased.

Compared to the original implementation of space mapping, this adaptation tends to converge faster while requiring less high-fidelity function evaluations. This is done by remedying one of the main critiques of the original method; mainly the use of selective evaluations compared to evaluations beforehand. The problem formulation of ASM also opens up the opportunity to use more advanced optimisation techniques such as, but not limited to, trust region methods

Space Mapping as a framework

While earlier discussions have centred on common implementation techniques like ASM, the underlying principles of space mapping can be extended into a broader, more adaptive framework. This higher-level approach allows for the adaptation of the methodology to suit specific application needs rather than adhering strictly to traditional methods. An example of such a case where this adaptability becomes essential is in managing control sequences between low- and high-fidelity flight mechanics models.

As noted in previous sections, these models exhibit significant non-linear behaviour, making it challenging to encapsulate their dynamics within a limited set of parameters. The complexity inherent in these models often surpasses the capability of conventional space mapping techniques, which are typically more effective in scenarios where the models and the to-be-mapped variables can be described with fewer, more direct parameters.

2.3.2. Multi Model Steering Algorithm

The challenge around non-linear trajectories and control sequences inspired the development of the Multi-Model Steering Algorithm (MMSA) [21]. While not proposed as a space mapping technique, it embodies the core principles of space mapping to a high degree, making it highly relevant as a benchmark method in this study to which other methods are compared to. The core principle behind MMSA framework is to divide complex motion planning problems where controls cannot be guessed a priori due to the complexity of the models involved. The MMSA framework is structured as follows:

1. *Strategic layer*: This layer's objective is to define the objective of the motion planning problem. In terms of control optimisation, this layer is about defining the cost function parameters, start- and final state, and other parameters of relevance.
2. *Tactical layer*: With the control optimisation problem defined, a low-fidelity model is used to solve the optimisation problem. The idea is that the low-fidelity model captures the global dynamics of the high-fidelity model accurate enough to generate a realistic trajectory and accompanying set of controls. This is also where the first similarity with space mapping can be seen. Instead of performing control optimisation with a high-fidelity model, a low-fidelity model is used as a surrogate which will be mapped in the next layer.
3. *Reflexive layer*: The final layer focuses on planning the actual control inputs for the high-fidelity model (read: finding a mapping function to map a low-fidelity trajectory to a high-fidelity model by adjusting the inputs). The planning is realised through Model Predictive Control (MPC). The calculated optimal trajectory is taken as a reference trajectory for the MPC controller to follow, also called Model Predictive Tracking (MPT). The way this is done is by splitting the trajectory in segments. For each segment a smaller optimal control problem is solved using a predictor model that locally approximates the high-fidelity model. Once a solution is found, the controls are passed to the high-fidelity model and the segment is simulated. Using the final state as a starting point for the next segment, the cycle is repeated. This again mimics the principle of space mapping by using a low-fidelity predictor model in lieu of a high-fidelity simulator.

While the cycle above accounts for the "multi-model steering" part of MMSA, the "adaptive" component draws even more parallels with space mapping, ASM in particular. Due to inherent misalignment between the low- and high-fidelity model, the control optimisation step in the tactical layer may not actually produce an optimal trajectory for the high-fidelity model. Through iterative updates to the model parameters and subsequent MPT step, the models are gradually aligned. This is highlighted in the following flowchart of MMSA [21]:

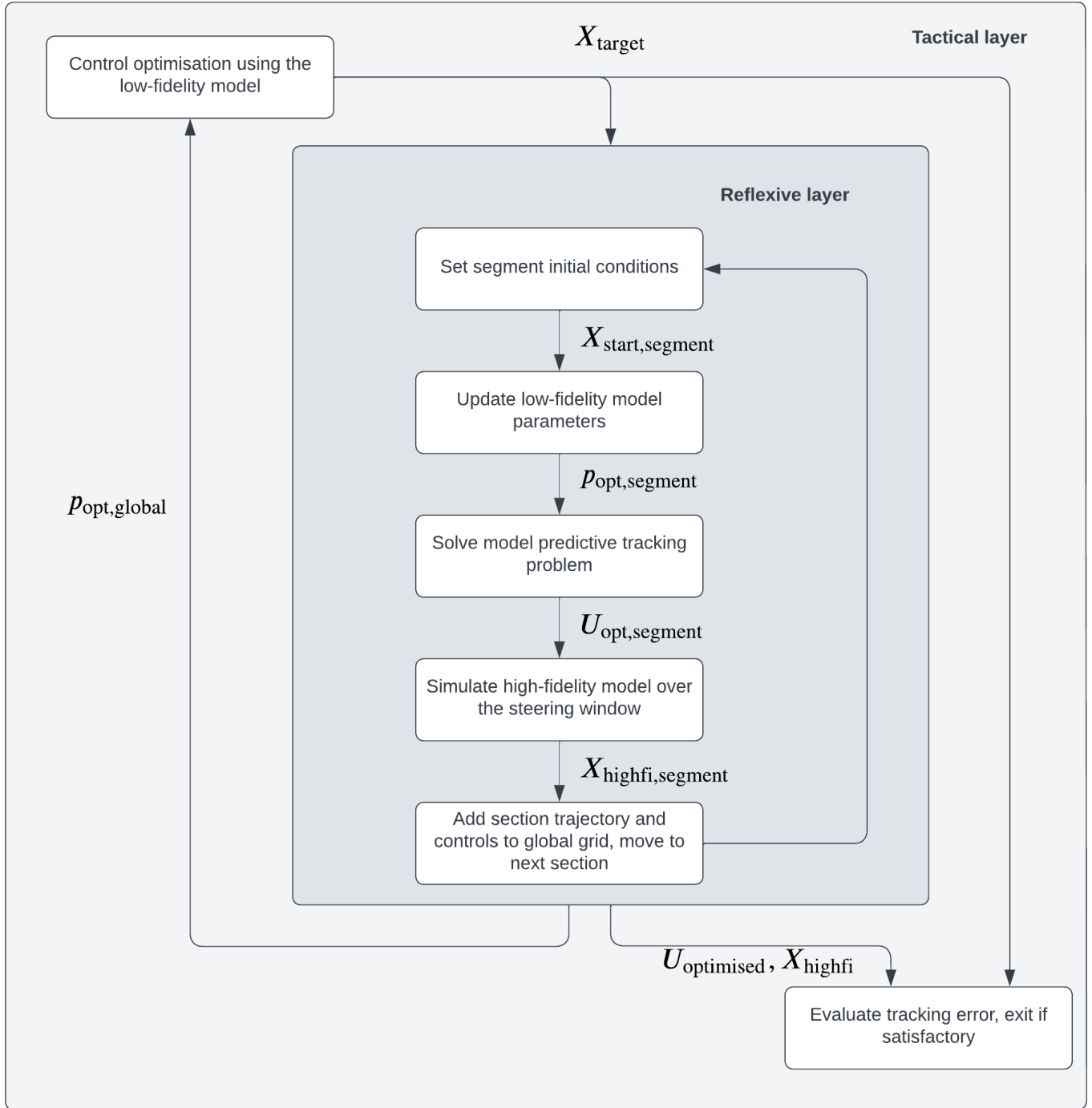


Figure 2.3: Flowchart of the MMSA process.

In this simplified implementation, the similarity with ASM can be seen in the iterative updates of p to improve the low-fidelity control optimisation problem. Important to note is that the low-fidelity model parameters are optimised for each section, and are collectively used during control optimisation to produce a trajectory that is also optimised for the high-fidelity model. Furthermore, the mapping error is defined by: $e = \int_{T_0}^{T_{\text{track}}} \|X_{(\text{target})} - X_{(\text{highfi})}\| dt$. The algorithm converges if the error between the optimal trajectory computed in the tactical layer and the trajectory computed in the reflexive layer is below a certain threshold.

Model Predictive Control

A key part of MMSA is understanding how MPC address model discrepancies during motion planning and how local parameters affect mapping performance. A useful illustration of the MPC concept is given as follows:

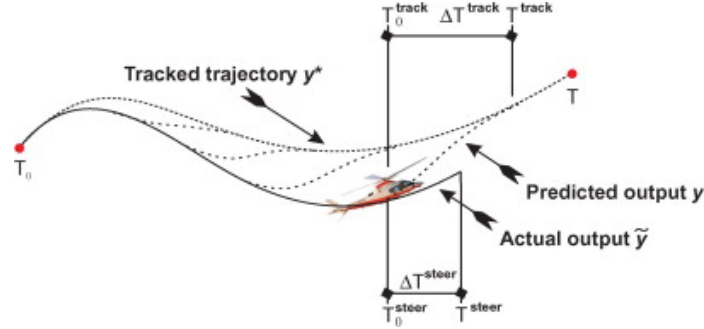


Figure 2.4: Visualisation of Model Predictive Tracking. The original paper focused on mapping critical trajectories for multi-body rotor craft. Adapted from [21].

Looking at Figure 2.4, a few key terms jump out. Starting from above, the tracking window, denoted by starting time T_0^{track} , signals the start of the local control optimisation problem to be solved. A key part of why MPC is a popular choice for motion planning is that it can easily deal with model discrepancies. The optimisation problem is structured such to reduce trajectory errors over a longer time window and is formulated as follows:

$$\min_{y,u} J^{track} \quad (2.26)$$

with:

$$J^{track} = \int_{T_0^{track}}^{T^{track}} M(y, y^*, u) dt \quad (2.27)$$

subject to:

$$f(\dot{y}, y, u, p^*) = 0 \quad (2.28)$$

$$g^{track}(y, u) \in [g_{min}^{track}, g_{max}^{track}] \quad (2.29)$$

$$y(T_0^{track}) = \tilde{y}_0 \quad (2.30)$$

$$M(y, y^*, u) = \|y - y^*\|_{S_y^{track}} + \|u\|_{S_u^{track}} + \|\dot{u}\|_{S_{\dot{u}}^{track}} \quad (2.31)$$

The control optimisation problem extends its horizon to future states as well, using the predicted trajectory y as part of its cost function. The dynamics of the predictor model are given by Equation 2.28, and are dependent on local parameters p^* . Looking again at Figure 2.4, the high-fidelity model's simulation extends only through a segment of the tracking window, emphasising the short-term validity of the predictor model and the substantial impact of local parameter updates. Over time, the motion planning problem becomes increasingly more representative of the actual dynamics, with the high-fidelity model serving as validation. Moving on, J^{track} represent the total cost of the trajectory and is defined by M at every time step. The individual cost terms in M are further scaled by weights S .

To summarise, the key principle behind MPC and MPT is to not only plan for the current time segment, but to incorporate future segments into the motion planning problem enhancing the realism and applicability of the trajectory planning. Low-fidelity predictor models play a crucial role in approximating the high-fidelity model and to reduce computational costs.

2.3.3. Modern Space Mapping Adaptations

With space mapping gaining traction after its introduction in 1996, the need for more advanced methods became clear due to the development of more complex simulations. Techniques such as knowledge-based modelling and neural networks promise to reduce computational demands while maintaining accuracy, particularly in areas where extrapolation and rapid data processing are crucial. This section will overview these advanced methods, setting the stage for their more detailed exploration and application in contemporary design practices.

Prior Knowledge Input

Similar to space mapping, machine learning is another emergent method in enhancing surrogate models. A logical next step is explore where these two methods compliment each other. One such method is Prior Knowledge Input (PKI) models. A typical PKI model architecture is structured as follows::

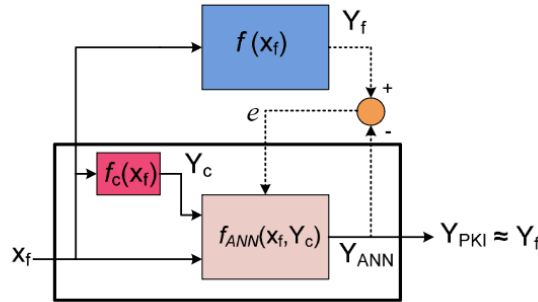


Figure 2.5: Architecture of a typical PKI model. Adapted from [22].

The main principle behind PKI is to incorporate prior knowledge in neural network prediction. In the depiction above, a high-fidelity control sequence x_f is passed to the low-fidelity model f_c and directly to the neural network F_{ANN} . The output of the coarse model is then used by the neural network as extra information to base its prediction on, hence the name prior knowledge. The networks output Y_{PKI} is a direct approximation of the high-fidelity model's output Y_f . This way, the model does not have to start from building knowledge about the problem from scratch but has the low-fidelity model to provide it with an initial prediction. Model training is done by comparing the network output to the actual output of the high-fidelity model $f(x_f)$ and adjusting its weights accordingly. This drastically improves learning performance and efficiency, and also reduces the amount of data required.

Prior knowledge Input-Difference method

Another adaptation of the PKI method is through the use of difference methods. Instead of directly outputting a prediction, the low-fidelity model's output Y_c is used as a base output, to which a predicted correction Y_d is applied to. The advantage is that this aids in overall mapping stability, especially early on in training. Even more than before, the network is already 'aimed' in the right direction by the low-fidelity model when generating an output:

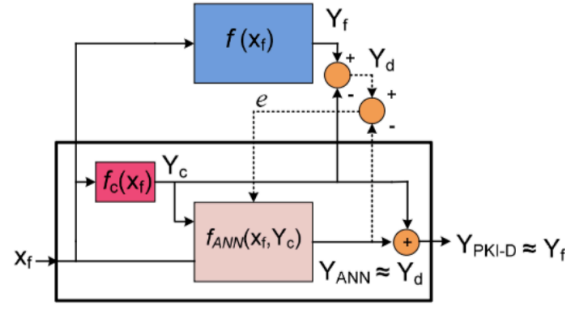


Figure 2.6: An adaptation of the PKI method using a difference correction method. Adapted from [22].

As illustrated, the PKI-D relies even more on prior information of the low-fidelity model by having its output serve as the basis of the network prediction, and having the machine learning network learn the discrepancies between the low- and high-fidelity network. In the context of trajectories, the PKI-D method learns which controls (x_f) produce the largest inaccuracy by the low-fidelity model's trajectory y_c .

The main take-away with methods such as a PKI and PKI-D is that they leverage the strengths of machine learning, while keeping in mind the main principles of space mapping.

2.4. Sequence-to-Sequence Neural Networks

Neural networks have been around for some time already, however only in the last decade have picked up steam due to a surge in computing power. This section will dive into the fundamentals of machine learning, specifically sequence based networks.

2.4.1. Fundamentals of Machine Learning

Machine learning has been a revolutionary way to process data, being able to digest vast amounts of data to search for complex patterns. Given certain attributes of the data set, these models learn from the data to make predictions or decisions on their own using relatively simple algorithms. This approach still requires some degree of problem knowledge to define the correct features and problem structure.

Deep learning however takes it a step further. Inspired by human neurons these types of "deep learning neural networks" are able to learn even more complex tasks without requiring prior knowledge. This approach forms the main differentiation between regular machine learning and deep learning. Deep learning can be seen as a branch of machine learning, and is responsible for many revolutions in the field of computer vision, speech, and natural language processing. These capabilities however do not come for free. In general, deep learning networks are more data and computationally expensive compared to regular machine learning algorithms.

As alluded to before, deep learning took its inspiration from the way neurons process information. This is best seen when looking at how a node or neuron connects to inputs in a network:

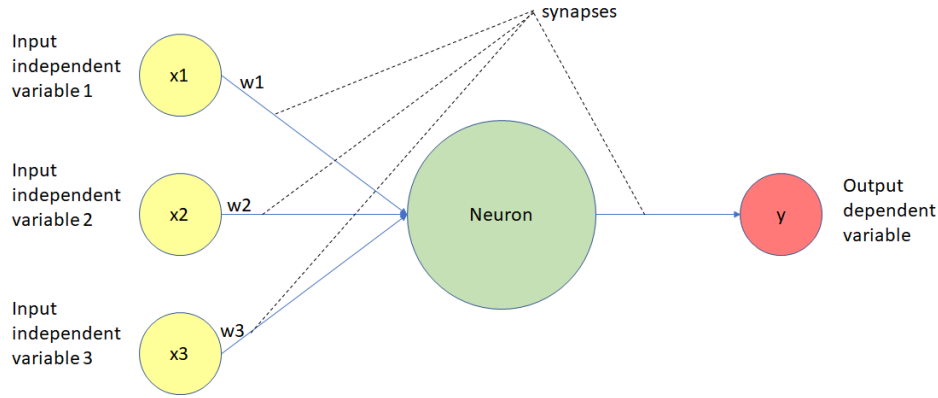


Figure 2.7: Layout of a single layer, single neuron network. In a network, layers of these neurons are stacked to enable complex patterns to be unravelled. Adapted from [23].

Indicated in Figure 2.7 are the connections between inputs and the neuron itself called synapses. In real neurons, the connection strength determines how well a signal travels between two neurons. In deep learning, this strength is determined by weights. In a network there are usually multiple layers stacked on top of each other, with multiple neurons per layer. The input signal into a single neuron consists of a weighted sum of all the outputs of the previous layer of neurons. This signal is then passed through an activation functions which determines the output of the neuron:

$$\bar{y} = \sigma(w^t \bar{x} + \bar{b}) \quad (2.32)$$

The weights are specific for each neuron and have a great influence on how the network performs. An important part of deep learning is the network training. This training process of adjusting the weights of a network such that the correct output is given in the end. Adjustment is made on the basis of the loss function which is usually given as the mean squared error of the output and the ground truth. Using techniques such as gradient descent or other optimisation methods the weights are adjusted [24].

One of the most important developments in deep learning is the back propagation algorithm. This algorithm allows a network to update its weights based on an the output error. This error is fed in reverse order through the network. Using the gradient of the loss function, the weights of the neurons with a strong contribution to the correct output are increased, and neurons who negatively contribute get their weights reduced. The magnitude of the adjustment is called the learning rate and is usually varied over the course of training to speed up convergence. It is similar to the step size of an optimiser in regular optimisation [25].

Another important aspect is the frequency of updates. A common practice is to bundle suggested updates of multiple data inputs, and combine them into a single update. This is called a batch, and is often used to promote stability of convergence. During the training, the data set is divided into multiple batches. Each batch represents an iteration of updates to the network. When the whole data set has been evaluated, this is called an epoch. Usually a careful trade-off is made between the number of epochs, batch size, and learning rate to prevent over-fitting of the data and speed of convergence [25].

2.4.2. Recurrent Neural Networks

RNNs are a branch of machine learning which specialises in time sensitive inputs. Conventional neural networks are designed to process single data entries, where each entry is independent of the order in which they are given. This also means that individual data points are not influenced by each other. RNNs on the other hand excel at capturing temporal patterns. Depending on previous inputs a RNN builds up a "hidden state", akin to a memory. The difference between a conventional neural network and a recurrent network can best be described as:

'Recurrent neural networks are connectionist models that capture the dynamics of sequences via cycles in the network of nodes. Unlike standard feed-forward neural networks, recurrent networks retain a state that can represent information from an arbitrarily long context window.' [26]

During each time step evaluation, the hidden state is updated to include the information of the current time step. This way, information is preserved while evaluating new steps. During each cycle, the hidden state is updated and passed on to the next time step. A similarity can be drawn to the integral term of a PID-controller. Both the I term and the hidden state integrate historical information to inform current output. The I term sums past errors for better control adjustments, while the hidden state in an RNN accumulates data from past inputs, influencing its response to new information. The way information is integrated can be seen in the architecture of the RNN:

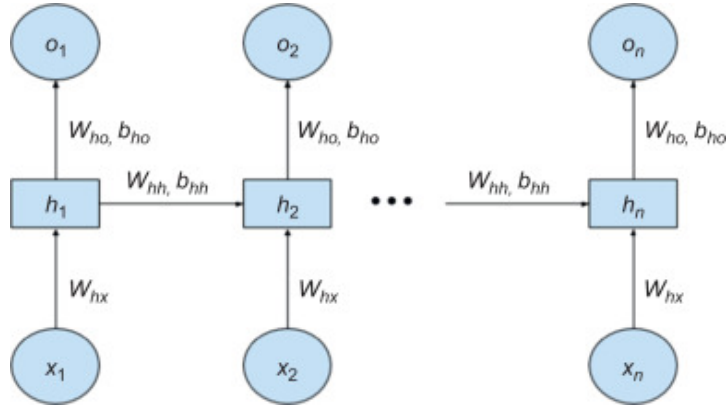


Figure 2.8: Architecture of a recurrent neural network. Each "column" represents the network during an iteration of data processing. For clarity, multiple instances of the same network at different instances of data processing are shown. x_i and o_i denote the input and output at time step i . h_i denotes the hidden state at each time step, updated by the network with weights W_{hx} , W_{hh} , W_{ho} , and biases b_{hh} , b_{ho} . The set of weights and biases are the same for each time step of the input, but get updated during training. Adapted from [27].

Common tasks for which RNNs are used are time series prediction and sequence-to-sequence mapping. The first task is about predicting future states given a certain starting sequence. During processing of the initial time series, the hidden state is constructed. Then using the last entry in the sequence and the built up hidden state, a prediction is made. Each subsequent prediction is then used to make the next prediction.

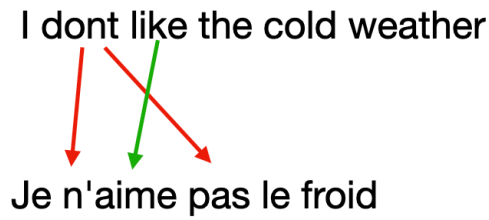
Where time series prediction tacks on subsequent predictions to the end of a sequence, mapping of sequences is also an area where RNNs excel. Here, an encoder-decoder structure is used to extract the essence of a sequence to be able to translate it to another sequence [28]. This will be covered in more detail in section 2.4.3.

Universal sequence-to-sequence function approximator

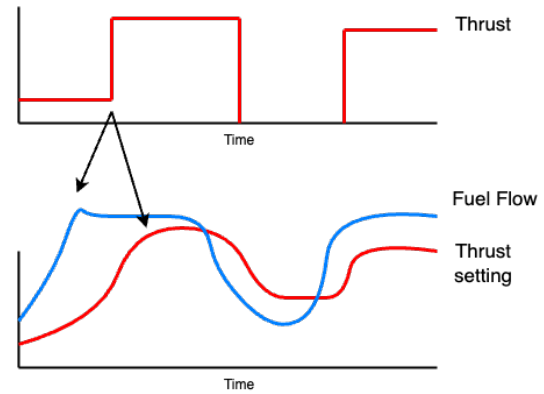
RNNs originally gained traction in the field of natural language processing. When dealing with language, the order of words is critical for the meaning of a sentence. Advances in RNN architectures led to significant improvement in translation benchmarks and revolutionised natural language processing through machine learning [28][29][30].

Mapping input sequence from low to high-fidelity models could be viewed in the same way. Similar to a word appearing multiple times in a sentence, a certain elevator deflection input could occur multiple times in an input sequence. In both scenarios, their time of occurrence significantly impacts their effects. Furthermore if certain patterns are present in a sequence, it is important that these are preserved during translation. This is visualised in the following example:

I dont like the cold weather
 Je n'aime pas le froid



(a) Natural language translation from English to french. Note how translation is not a one-to-one conversion task. The order changes depending on certain patterns present in a language.



(b) Example of an input sequence translation. Similar to a sentence, a direct conversion may not be possible due to modelling differences. The low-fidelity aircraft model might use instantaneous thrust increase, while the complex aircraft model might model fuel flow and thrust setting to control thrust.

Figure 2.9: Visualisation of the similarity between natural language translation tasks and control sequence mapping.

This abstraction of sentences to general sequences is supported by [30]. This source suggests that transformer models, a type of RNN, can be viewed as a universal sequence-to-sequence function approximator. One of the ways this is showed is by proving that a network architecture is *Turing Complete*. This property means that given enough computational resources, any algorithmic task such as sequence-to-sequence mapping can be performed regardless of complexity. The implication of this statement is that RNN networks are theoretically capable of performing tasks such as control input mapping, and are not limited to language processing. These findings were corroborated by [31][28] for simplified RNN models. In practice any network is limited by the computational power available. However for limited size problems this could be a viable approach to sequence mapping for optimal control problems.

Limits of basic RNNs

While Turing completeness is an important theoretical foundation, it does not describe the real-life performance of a model. The fact that more specialised architectures have emerged signifies that there is much to be gained in efficiency and performance for certain problem sets. Improvements in network architecture have been made to specifically address the problem of the vanishing/exploding values, and the problem of limited memory[26].

Like with any neural network, the way input values move through the nodes is through multiplication by the weights between nodes. These weighted values are fed into an activation function which produces the input value for the next layer. In an RNN, values in the hidden state are passed through to each subsequent input. This is done through a connected layer which has its own set of weights, see Figure 2.8. In the case of a vanishing gradient, the set weights has a low value. The effect that this has is that the influence of an early data entry gets progressively diminished compared to later weights. With exploding weights, the opposite happens [32].

A side effect of the vanishing value problem is that for long sequences, information at the beginning is forgotten because its contribution also vanishes compared to later information. As a remedy different activation functions have been proposed. The idea being that other activation functions might preserve better, however these have met similar drawbacks [33].

2.4.3. Long-Short Term Memory Neural Networks

Another source of memory limitation is the fact that information in the hidden state is constantly overwritten by subsequent updates. In a sense, information is "flushed out" when new information is added. A work around for this problem is the addition of a long term memory, where selective bits of infor-

mation are stored regarding long term patterns. The long term memory is also called the "cell-state", similar to the hidden state. This is the essence of a "Long-Short Term Memory or LSTM for short. A typical LSTM network architecture is as follows:

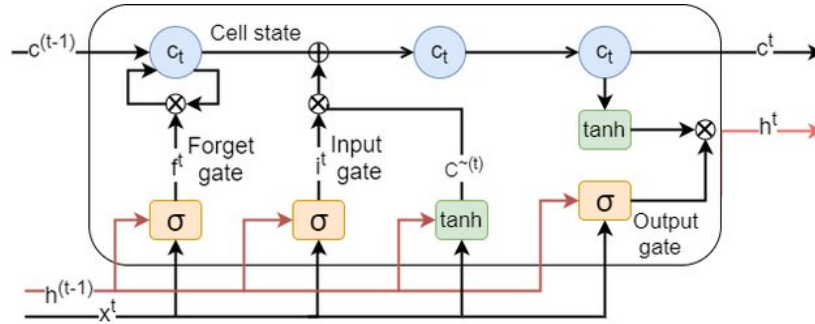


Figure 2.10: Architecture of an LSTM model. Input and forget gates determine which information gets passed to the cell state. Not shown are the weights between input states and activation functions similar to Figure 2.8. Adapted from [34].

The way information gets passed to the cell state is through the inclusion of gates. Gates work on the basis of activation functions and weights, similar to normal network layers. The forget gate reduces the values that should be forgotten in the cell based on a weighted sum of the hidden state and input at that time step, fed into an activation layer. In a similar fashion, the input gate updates the cell state by adding values to certain cell states. This is done through the input gate which decides *which values* should be updated. The candidate value denoted by $\tilde{C}^{(t)}$ determines by *how much* the cell state should be updated. The hidden state, which acts as the short term memory, is updated in the same way as in regular RNNs [34][35].

Based on these improvements, LSTM networks are able to achieve greater performance over longer time series compared to regular RNNs. The sequence length they are able to process is still limited to 500-1000 entries [36]. This improvement comes at the cost of being more difficult to train due to the added complexity of the gates.

Encoder-Decoder structure

Regarding sequence-to-sequence mapping, LSTM based networks utilise an encoder-decoder structure to capture the essence of a sequence and properly translate it. This structure is similar to the encoder-decoder structure used in standard RNNs:

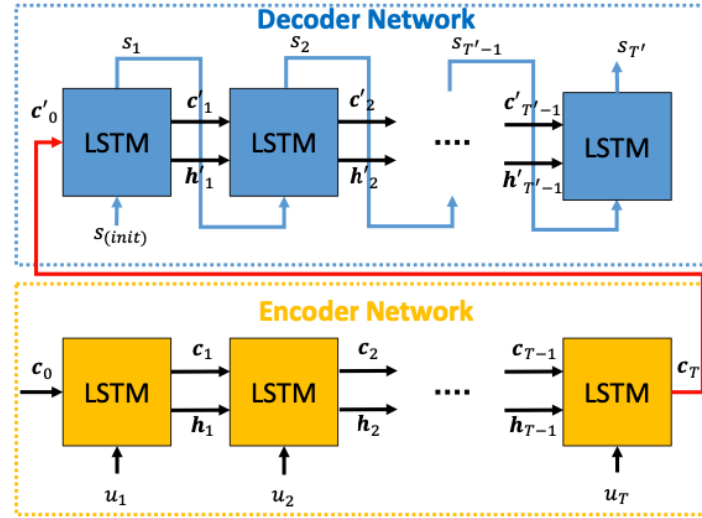


Figure 2.11: Typical architecture of an encoder-decoder structure using LSTM networks. The network itself consist of a single LSTM block for encoding and a single block for decoding. Each output is reused as an input for the same block to create future outputs. The composition of an LSTM block is the same as in Figure 2.10 [37].

The encoder ingest the data sequence step by step. This builds up the hidden and cell state. Once every step is processed, the cell state containing a memory of the entire sequence is passed to the decoder network. Using a sequence starting token, which can be a known initial value, outputs are generated using the previous output as input for the current step. The advantage of this approach is that it allows for variable length sequences. Standard mapping algorithms require a fixed input and output sequence length [36][28].

One of the first works pioneering this concept on English to French translation noted that even for long sentences the network performed well [28]. Furthermore, adding more layers to the network lead to better translation, implying that more complex patterns could be captured. Another notable observation is that reversing the source sequence led to better performance. By reversing the sequence, the beginning of a sequence is processed last, keeping its information more immediate and more present in memory when starting to generate the target sequence. This can help in better contextual alignment between the source and target sequences.

In the pioneering studies of machine translation, such as the translation from English to French, it was observed that networks like LSTMs could handle long sequences effectively, capturing intricate patterns in sentences [28]. This capability is analogous to the task of translating control sequences from a low to high-fidelity flight dynamics model. For instance, consider an LSTM trained to translate a sequence of control inputs optimised for a low-fidelity model of an aircraft's ascent into a sequence that would produce the same ascent trajectory in a high-fidelity model.

Just as adding more layers to a network improved language translation by capturing more complex linguistic structures, increasing the depth of an LSTM network for flight control translation allows it to better understand and replicate the complex interactions between various control inputs and aircraft responses. For example, a deeper LSTM might better grasp how throttle settings and pitch attitude influence the latter stages of ascent in terms of speed and altitude.

Moreover, just as reversing the order of words in a source sentence led to better translation outcomes, reversing the sequence of control inputs—processing the final inputs first—can keep the most critical information fresh in the network's memory. In practice, this might mean that for a complex manoeuvre sequence, the network first processes the control inputs needed for the final position of the manoeuvre. This way, when the LSTM begins generating the corresponding high-fidelity control inputs, the crucial final inputs are more present in its memory, allowing for better contextual alignment. For example, by reversing the control input sequence for a landing manoeuvre, the network can more accurately predict

the precise throttle and pitch inputs needed during the final approach, enhancing the accuracy and safety of the landing. It must be noted that these findings are purely empirical observations, and highlight that machine learning still operates as a black box.

2.4.4. Transformer Networks

While proving to be much more capable than standard RNNs, LSTM networks still struggle with long sequences. This is where transformer networks seek to find improvement. The approach transformers take is different compared to RNNs as they eliminate the hidden state, and introduce two new concepts: positional encoding and self attention. First introduced in 2017, transformer networks drastically improved translation tasks even for long sentences [38]. Their function is best explained when looking at the model architecture.

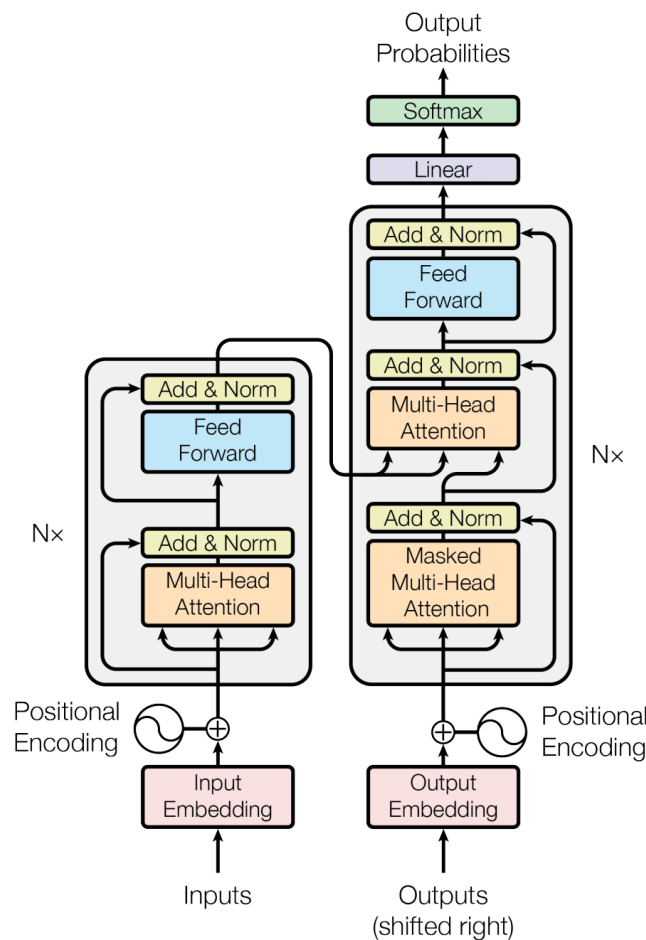


Figure 2.12: Architecture of an transformer model. An input sequence is encoded in the block on the left. When an output is produced, the attention map formed during encoding is used to decoded the sequence. Adapted from [38]

Similar to LSTM sequence-to-sequence models, transformer models also make use of the encoder-decoder structure. A big advantage of transformer models is that the input can be ingested in parallel instead of sequentially. This gives a large performance boost when training the model on multiple GPUs. Output generation is done similarly to LSTM models where the current output is used as input for the next output.

Positional encoding

As input sequences in a transformer model are not processed sequentially, there is no way for the network to know the order of the data points. Positional encoding is a way to "time stamp" the data points using a sum of sine and cosine functions. This way the order of the sequence is preserved. These time stamps are added as an extra dimension to the input sequence. Through the "add & norm" layers the positional encoding is preserved while data travels through the layers. This is called residual connection.

Self Attention

Self attention is arguably the most important part of a transformer network, as it allows the network to learn patterns in the data. This is also what makes transformers excel at sequence-to-sequence tasks [28]. The original paper on transformer models introduced the concept of multi-head attention as can be seen here:

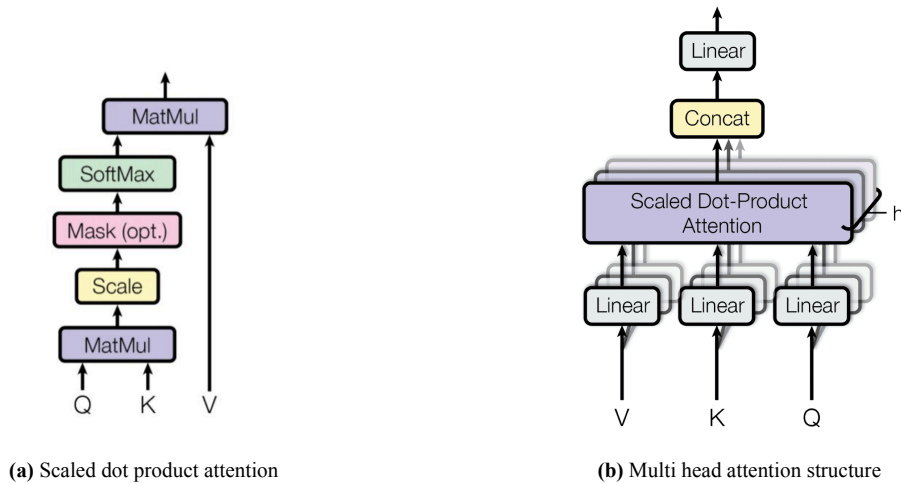


Figure 2.13: Self-attention mechanism in transformer models. Each head of multi head attention is made up of a single scaled dot product producing an attention map. Adapted from [38]

Looking at Figure 2.13a, the inputs Q , K , and V can be seen. These stand for Query, Key, and Value respectively. This structure has its origin in natural language processing. An analogy for the query key value system would be as follows. Suppose you search the internet using a search term. This would be the search *query*. Then search results that are most relevant are returned. These are the search *keys*. The information behind these keys are the *values*, and ultimately is what you wish to know. Mapping the right keys to a certain value is what a transformer is trying to learn during training. In essence, self-attention is about determining the correct correlation between entries in a sequence, thereby "learning" the correct patterns in data [39]. When applying this concept to the field of flight mechanics, the relevance of self-attention becomes clear in the context of the previous ascent trajectory example. In such manoeuvres, yaw and roll inputs typically exert a lesser influence on the aircraft's behaviour than pitch inputs. Through self-attention, the transformer network learns to assign greater importance to the pitch inputs, thus reducing the influence of less critical factors such as yaw and roll. This focused attention enables the network to generate a more accurate translation of control sequences, honing in on the inputs that significantly affect the trajectory.

In practice this is done through the use of weight matrices, where each entry in the matrices is a learnable parameter. Each of the inputs Q , K , and V in Figure 2.13a, are the multiplication of, for example matrix W^Q , with the input matrix. Note that the input sequence is enhanced with an extra dimension due to the positional encoding. The input matrix has size $T \times (D + P)$. Where T is the amount of time steps, D is the amount of dimensions in the input sequence, and P is the positional encoding. This allows transformers to map any sequence based data [38] [39]. The result of the dot

product between Q and K is also called the attention filter, highlighting which values are closely related. The values of this $T \times T$ matrix indicate the relative connection strength between each entry in the sequence. By multiplying this filter with the values V , an augmented matrix of size $T \times (D + P)$ is acquired with important data points being highlighted for future processing. This process is repeated in each of the attention heads seen in Figure 2.13. The resulting matrices are then concatenated horizontally into a single large matrix, and passed into a linear layer to reduce the output to the original dimension size. The output of this attention mechanism is a filtered matrix containing information about only the most relevant pieces of information in a sequence. A great visualisation of self-attention is given by [40]:

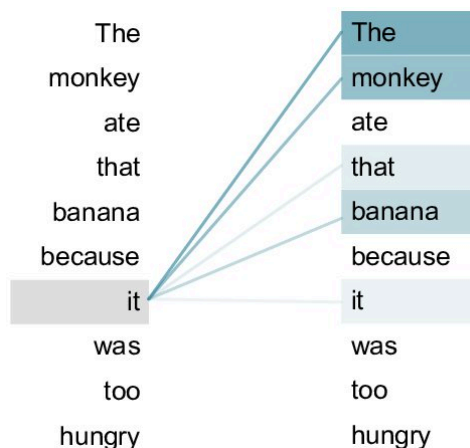


Figure 2.14: Example of self-attention in natural language processing. Related words in a sentence are highlighted with a darker colour connection. Adapted from [41].

Using multiple attention heads in a model allows for more complex and different patterns to be picked up. The output of multiple heads is concatenated and using a linear transformation adjusted to the correct output dimensions. This can be seen in Figure 2.13. An added benefit that self-attention is evaluated instantaneously and can be computed in parallel. This greatly enhances computational efficiency.

2.4.5. Training strategies

Training machine learning networks to capture complex behaviours critically depends on both the volume and structure of the dataset employed. In the realm of seq2seq models, which are instrumental in tasks such as language translation, the distinction between labelled and unlabelled data underpins different training methodologies.

Labelled data typically comprises pairs of corresponding inputs and outputs. For instance, in a translation task, an input sequence might be a sentence in English, with the corresponding label being the equivalent sentence in another language. Similarly, in image classification, an input would be an image, and the label would be the description or category that the image fits into. Training with labelled data is categorised as supervised learning, where the model learns to predict outputs directly from labelled examples, using a predefined loss function to measure accuracy against a known ground truth.

Conversely, unlabelled data lacks such direct annotations. In this scenario, the learning algorithm must infer structures or patterns within the data autonomously, often guided by a loss function that evaluates the quality of predictions in the absence of explicit labels. This approach requires the model to develop its own interpretation of what constitutes a ‘correct’ prediction, navigating through the data’s inherent complexities without predefined answers.

This distinction highlights the fundamental challenges and strategies inherent in training machine learning models, particularly in how they generalise from training data to perform tasks in real-world scenarios. The choice between supervised and unsupervised training methods can significantly influ-

ence the effectiveness and applicability of the resulting models[42].

Transitioning from the foundational understanding of training with labelled data, it is crucial to acknowledge the practical challenges associated with acquiring such data. Labelled data can often be difficult to obtain and expensive to annotate, typically requiring significant labour or high computational loads to acquire ground truth labels.

These challenges underline the importance of exploring alternative training paradigms, such as semi-supervised learning, active learning, and transfer learning, which can leverage unlabelled data more effectively or reduce the dependency on large initial datasets. Such methods aim to mitigate the resource-intensive demands of traditional supervised learning, offering pathways to more efficient and model training[43].

Diving deeper into active learning, instead of labelling every data point in advance, this method starts with an initial smaller labelled data set, and selects the most useful samples using an oracle. An oracle is the name for the labelling function or person doing the labelling each iteration. In practice, the iterative approach can look as follows:

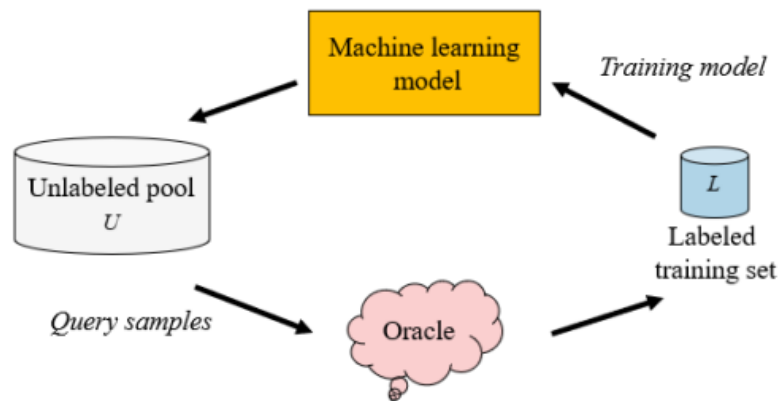


Figure 2.15: Active learning loop, adapted from [43]

As can be seen, active learning iteratively adds data from the unlabelled pool of data which is then labelled by the oracle. An important step here is how the network decides which data points to query, this is called the query strategy. Amongst others, three methods are highlighted[43][44]:

- **Uncertainty sampling:** Using a model uncertainty score, new samples are based on how confident the model is in its prediction. A downside is that this method does not consider the sample itself, which can cause oversampling in certain regions.
- **Bayesian active learning:** This method uses the Bayesian framework to reduce uncertainty in machine learning models. This process focuses on iteratively selecting and labelling data points that are expected to provide the greatest informational return, thereby optimising the learning process and improving model performance with fewer labelled instances. This approach is valuable in scenarios where data is scarce or expensive to label, and where managing prediction uncertainty is crucial [45].
- **Diversity sampling:** Rather than looking at uncertainty, diversity-based sampling primarily focuses on the diversity of the selected samples. This method ensures that the chosen samples cover a wide range of data characteristics, aiming to improve the generalisation ability of the model across various parts of the data space. Unlike uncertainty sampling, diversity-based methods prioritise a broad and representative dataset to mitigate the risk of overfitting and reduce biases in the

learned model. However, for optimal results, many active learning strategies combine elements of both diversity and uncertainty to balance informativeness with broad data coverage.

To summarise this section, training machine learning models such as seq2seq models requires large upfront labelled data sets. Labelled data supports supervised learning, allowing models to learn and predict based on known input-output pairs. Due to the high costs and effort required for labelling data, an alternative approach such as active learning is employed to reduce dependency on large labelled datasets. Active learning iteratively selects the most informative samples for labelling using various strategies to enhance data efficiency, and enable machine learning in data sparse environments.

2.4.6. Contemporary research

As discussed Figure 2.4.2, RNNs have broad applications beyond natural language processing. This section offers an overview of notable contemporary works that have effectively utilised these networks in trajectory related tasks.

Originally, RNNs were mainly applied to predictive tasks such time series prediction. In the context of trajectories, several papers have explored the use of LSTMs combined with auto-encoder-decoder for trajectory prediction. An example of this combination was applied to predict plausible car trajectories based on relative positions, demonstrating the capability of LSTMs to generate accurate trajectories [37]. Similarly in [46], the same structure, enhanced with an attention mechanism, was used to avert ship collisions in high traffic areas, resulting in performance that surpassed classical collision avoidance models. The integration of the attention mechanism highlights the capability to manage complex sequence prediction tasks with enhanced accuracy.

Focusing on trajectory adjustment, a recent study done by [47] used LSTM networks to learn aircraft movement patterns under dynamic weather conditions. By integrating convolutional layers, the model effectively learned both temporal and spatial patterns, leading to improved flight paths. The joining of LSTMs and CNNs showed comprehensive understanding crucial for accurate predictions in dynamic environments.

In robotics, LSTMs have addressed complex movement planning issues. In [48], an LSTM was utilised to augment robotic arm movement to account for sliding of objects, a challenge compounded by kinematic and dynamic constraints. The network learned from physics simulations to improve sequence planning, paralleling the complexities encountered in flight mechanics.

In a similar fashion, transformer networks have been used to improve spacecraft trajectories as shown in [49]. Through the inclusion of system dynamics in the network, the quality of the generated trajectories improved significantly. This study highlights the potential of incorporating neural networks in complex dynamic systems to enhance spatial-temporal understanding of trajectories, and improve generation of optimal control sequences.

In summary, these studies collectively demonstrate the wide-ranging utility of RNNs in trajectory prediction and planning. Their applications, from generating realistic trajectories in automotive and maritime contexts to enhancing control in aerospace and robotics, signify a substantial advancement in the field of trajectory modelling and sequence-related tasks.

3

Space Mapping & Machine Learning: Research Direction & Integration Challenges

This chapter outlines the integration of space mapping and machine learning, focusing on the research directions and challenges of integrating these methodologies. Here, the potential for enhancing space mapping with machine learning is explored, addressing both the opportunities and hurdles. First the research direction will be covered, followed by addressing the specific integration challenges this study faces. Then, the space mapping problem formulation will be presented, followed by an overview of the test cases to which both MPC as well as the machine learning method will be benchmarked to.

3.1. Research direction

Despite recent advancements in machine learning and space mapping, significant gaps remain in efficiently mapping optimal control problems and sequences. Existing studies have demonstrated the potential of RNNs in trajectory prediction or augmentation of trajectories, however none have covered the direct mapping of complete sequences. On the other hand, various space mapping techniques have demonstrated to be able to map control sequences, but lack sophistication provided by modern machine learning techniques.

To summarise the previous chapter, the primary research gap lies in effectively integrating machine learning and space mapping techniques to map optimal control sequences across different fidelity flight models, particularly in non-linear flight conditions. This is however where the crux of the problem lies. The core principles that make space mapping and machine learning powerful tools is also where they diverge fundamentally. The distinct nature of space mapping and machine learning presents unique opportunities, but it also brings forth significant challenges when attempting to merge them. Addressing this research gap and integration challenge could lead to significant advancements in the accuracy and efficiency of trajectory planning in complex flight scenarios. With the research gap outlined above, the following research question is posed:

How does the integration of space mapping and sequence-to-sequence networks facilitate the mapping of optimal control sequences between low and high-fidelity flight mechanic models to minimise trajectory differences in non-linear flight regimes?

Besides being a collection of techniques, space mapping can also be seen as a general framework where other techniques such as model predictive control act as a mapping function. This has also been de-

scribed in subsection 2.3.2. In putting the research question in its proper context, it is important to realise that space mapping is seen here as a framework in which machine learning is placed to perform as the mapping function. The goal of this research is to investigate if an out-performance can be achieved compared to existing space mapping techniques which in this case is model predictive control. To help guide answering this research question, the following sub-questions are defined:

- (a) Which type of network is best suited for a hybrid space mapping-machine learning implementation?
- (b) How does active learning influence the predictive accuracy and adaptability of machine learning networks in hybrid space mapping implementations?
- (c) How does hybrid space mapping-machine learning compare to existing trajectory mapping techniques such as MMSA in terms of computational cost and mapping error?

The research questions outlined above serve as a guide to steer a comparative study between traditional space mapping techniques and their integration with modern machine learning technologies. In determining the scope of this research, it is equally important to determine what is not included.

Although space mapping and machine learning represent a spectrum of mapping solutions with contrasting methodologies, this study will not encompass a purely machine learning-based approach to the mapping problem. While such an inclusion could provide a broader perspective, it would diverge significantly from the core principles of space mapping, potentially complicating direct comparisons. Therefore, the scope of this research is limited to methodologies that incorporate the core principles of space mapping, either as a standalone method or through minor adjustments that fall within its natural capabilities such as active learning. This deliberate limitation to space mapping-focused methodologies gives space for a thorough overview of the various integration challenge that arise. Another limitation of this research is to limit trajectory mapping to 2D cases only. This simplification aids in result analysis, visualisation, and understanding.

The subsequent section will break down these challenges, providing an overview of the complexities of merging traditional space mapping with advanced machine learning techniques. The following section continues with developing a methodology to help answer the main research questions

3.2. Integration challenges

As stated before, integrating space mapping and machine learning has tremendous potential to improve surrogate based optimisation. When thinking about possible implementations, an ideal solution would harness the power of seq2seq models and their ability to make complex relations, and the low data requirement of space mapping methods. To get a better grip on the scope of the problem, the following comparison can be made:

Table 3.1: Comparison of Space Mapping and Seq2Seq Machine Learning Networks

| Feature | Space Mapping (SM) | Machine Learning (ML) - Seq2Seq Networks |
|--------------------------------------|--|--|
| Objective | Tailor-made solution specific to problem | Generalised model for varying problems within scope |
| High-Fidelity Data Requirement | Minimal, only generated when needed | Extensive; large datasets required beforehand |
| Adaptability | None, only valid for the specific design point | High, general model |
| Accuracy | High within scope, low outside outside of scope. | Medium-High data point is similar to training set |
| Learning Strategy | Gradient based, single data points | Batch based, stochastic gradient descent |
| Applicability to sequence based data | Medium, possible through MPC based methods | High, state of the art sequence based models available |

Reviewing Table 3.1 highlights a fundamental difference in approach. Space mapping is designed to be precise for specific scenarios, creating tailor-made solutions that are highly accurate at certain points. On the other hand, machine learning aims to be more versatile, building models that can apply broadly across various scenarios.

In integrating space mapping and machine learning, the goal is to create a symbiotic approach that benefits the strengths of each method, rather than forcing one to adapt beyond its intended capabilities. Any attempt at combining these methods should therefore not go against each method's fundamental principles. With this in mind, a solution that favours both methods will be leaning more towards a tailor made space mapping implementation as opposed to a more generalised machine learning implementation.

Space mapping is designed, and functions best when it's fine-tuned with specific data points. This does not naturally extend to multiple scenarios without significant alterations to its core mechanics. On the other hand, machine learning can be more readily adapted.

Normally when a model is extensively trained on a small data set, it can suffer from overfitting. This is where a model fits exactly to its training data but performs poorly on new data. In the context of integrating with space mapping, this could instead be advantageous. The active learning mechanic, introduced in subsection 2.4.5, could mimic space mapping's strategy of tailoring models to fit precise data points closely.

In an integrated setup, leveraging active learning allows machine learning to mesh with space mapping in an organic way. Still, in order to test if an implementation is satisfactory, the following conditions for integration are defined:

1. **Accuracy Improvement:** The integrated method must demonstrate superior mapping performance resulting in a reduced error compared to standalone methods.
2. **Data Efficiency:** The integrated method should minimise the need for extensive initial datasets, and only gather new data when needed.
3. **Robustness in implementation:** The integrated method should provide reliable and consistent mapping performance.
4. **Flexibility:** The integrated method should be flexible, allowing for straightforward adaptations to different trajectory optimisation problems without significant model adaptations.

To ensure that a hybrid space mapping-machine learning network effectively leverages the strengths of both methods, these conditions will serve as guidelines for the development of new frameworks. Regarding specific measurable quantities, accuracy and robustness are easily quantifiable by looking at the error metrics defined in Appendix B. The magnitude of the error correlates with accuracy, whereas robustness correlates with spread in error for different conditions. This will be further discussed in chapter 6. Data efficiency is also easily determined by looking at the amount of high-fidelity model evaluations are required. Lastly, flexibility involves the model's adaptability to new optimisation challenges, requiring both fine-tuning and modifications to suit different problems.

One aspect that is not taken into account is the computational load the mapping functions themselves. Machine learning networks are notorious for requiring a lot of computational power to train, however it important to contextualise this within the landscape of the optimisation problem. Space mapping is particularly useful when the high-fidelity model is significantly more computationally demanding than the low-fidelity model, such as in computational fluid dynamics or complex flight simulators. The resources utilised during mapping are generally overshadowed by the high costs associated with high-fidelity evaluations. Although reducing computational overhead is beneficial, it is essential to balance this with the accuracy and efficacy of the mapping process.

3.3. Space mapping problem formulation

Navigating the complexities of integrating space mapping and machine learning begins with a proper problem formulation. This starts with defining the space mapping problem to which machine learning is applied to. An important step is the parameterisation of control inputs. For this purpose, Hermite spline interpolation provides a robust solution by enabling smooth transitions between specified control points. A downside of regular spline interpolation is that it can be susceptible to overshooting in an attempt to fit through all the control points:

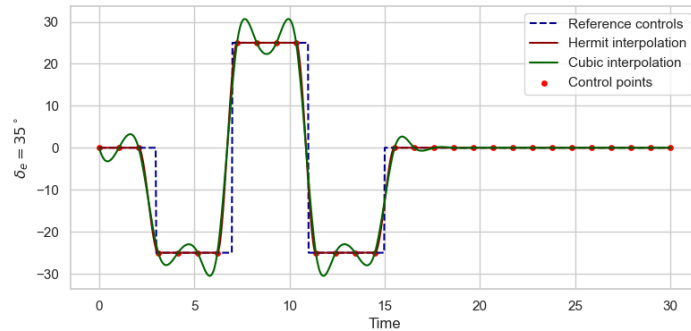


Figure 3.1: Comparison between different spline interpolation methods for estimating a sampled reference signal.

For the same set of control points denoted in red (sampled from the reference control sequence), it can clearly be seen that hermit spline interpolation is the superior way to reconstruct a sequence from a set of point. Especially when the control points find themselves near the edge of the control space, hermit interpolation limits the overshoot into infeasible ranges.

With the control parameterisation set, the space mapping problem can be introduced along side which the various mapping functions will be bench marked. Referring to Equation 2.3.1, a space mapping problem can be formulated as an optimisation problem:

$$\min J(u) \quad (3.1)$$

with:

$$J(u) = \epsilon(x_{ref}, x_f) \quad (3.2)$$

Here ϵ represents the error between the reference trajectory (x_{ref}) and the high-fidelity trajectory (x_f). Note that x_f is the high-fidelity trajectory which is acquired by simulating control u . Different methods of estimating error can be used as introduced in Appendix B. Regardless of method, the goal of space mapping is to reduce the error between a reference or target trajectory and the high-fidelity trajectory.

From this starting point, this research will explore two different approaches to address this mapping problem. The first method focuses on an adaptation of the MMSA algorithm introduced in subsection 2.3.2, whereas the second method integrates machine learning networks to enhance the mapping process. These will be discussed in the next chapter.

3.4. Test cases

In order to prove that a novel approach actually works as intended, it is essential to demonstrate satisfactory performance in multiple scenarios. This section will introduce the different test cases which will be used in this research to investigate the working of the MPC and machine learning methods.

While the primary focus of this research is on mapping optimal control sequences and trajectories, it's important to recognise that real-world scenarios can often exhibit limited variation in control actuation and flight profiles. This can reduce the complexity and diversity needed for a robust analysis. Ideally, a control sequences with plenty of control actuation is used which yields a trajectory that reaches into the non-linear part of the flight regime. It is in this region that simplified mapping architectures struggle to accurately map trajectories. In turn, the following standard trajectories are introduced:

Test case 1: Isolated elevator deflection

For the first test case, constant thrust with varying elevator deflection has been chosen. The main reason behind these control is to isolate the effect of the elevator on the aircraft's trajectory. This setup highlight the difference in pitch dynamics between the different models, and consequently shows that the network is able to correct such differences:

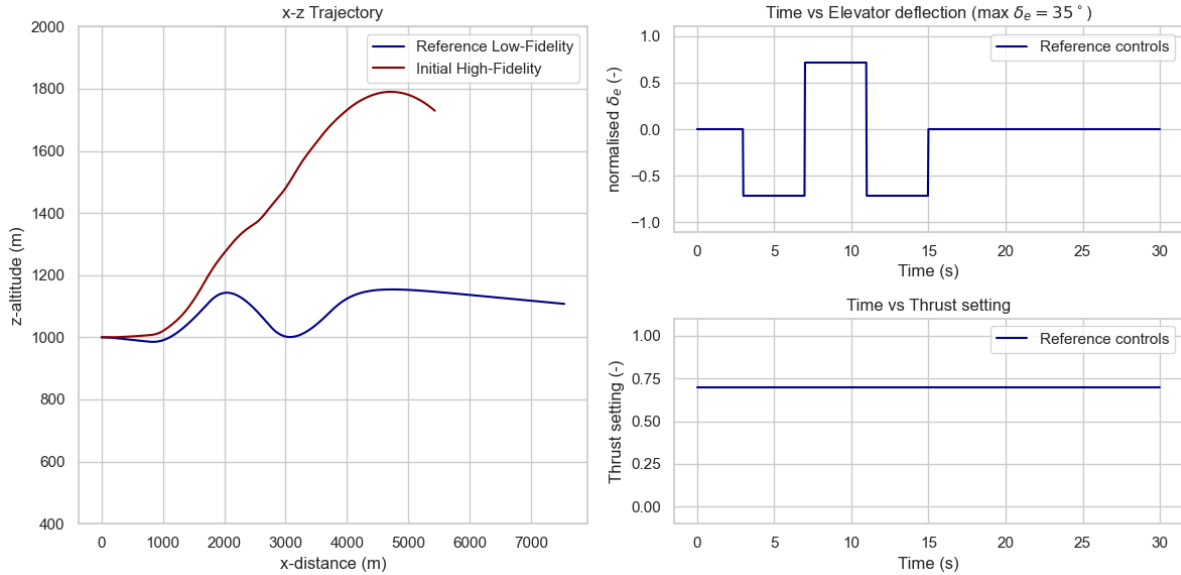


Figure 3.2: Test control case 1. The elevators are deflect to 25 degrees, or about 70% of the maximum deflection allowed. The thrust setting is fixed to 70%. The left plot shows the respective x-z trajectory of the low and high-fidelity model.

The plot reveals significant behavioural differences between the two models. The low-fidelity model, potentially using less representative control gains, reacts differently compared to the high-fidelity model, which incorporates more complex dynamics and does not assume a simplistic point mass model. This results in less instantaneous pitch changes compared to the low-fidelity model. In the experimental

setup, each aircraft model initiates at the same altitude, with a starting velocity of 240 m/s, which approximates the cruising speed of the reference Airbus A320, and a zero-degree flight path angle. These settings are chosen to maintain both stability and relevance to typical flight operations. Operating at cruise speed provides a stable, steady-state condition that serves as a realistic point of departure for analysis. At this speed, even minor deviations in control inputs can shift the aircraft into non-linear performance regions, offering an ideal scenario for testing the robustness and efficacy of various mapping strategies. Furthermore, it is important to note that both models start in untrimmed conditions. Starting each test case from an untrimmed state is crucial since the high-fidelity and low-fidelity models inherently possess different trim points due to their distinct internal dynamics. Starting from these models' individual trim conditions could introduce biases in the results, affecting the fairness and accuracy of the comparisons. By beginning with both models untrimmed, the experiment ensures a controlled and consistent baseline, allowing any observed differences in performance to be attributed directly to the unique ways each model handles flight dynamics.

Test case 2: Isolated thrust variation

Conversely, the next case keeps the elevator deflection constant while varying thrust level. This isolates the difference in thrust modelling used by the respective models:

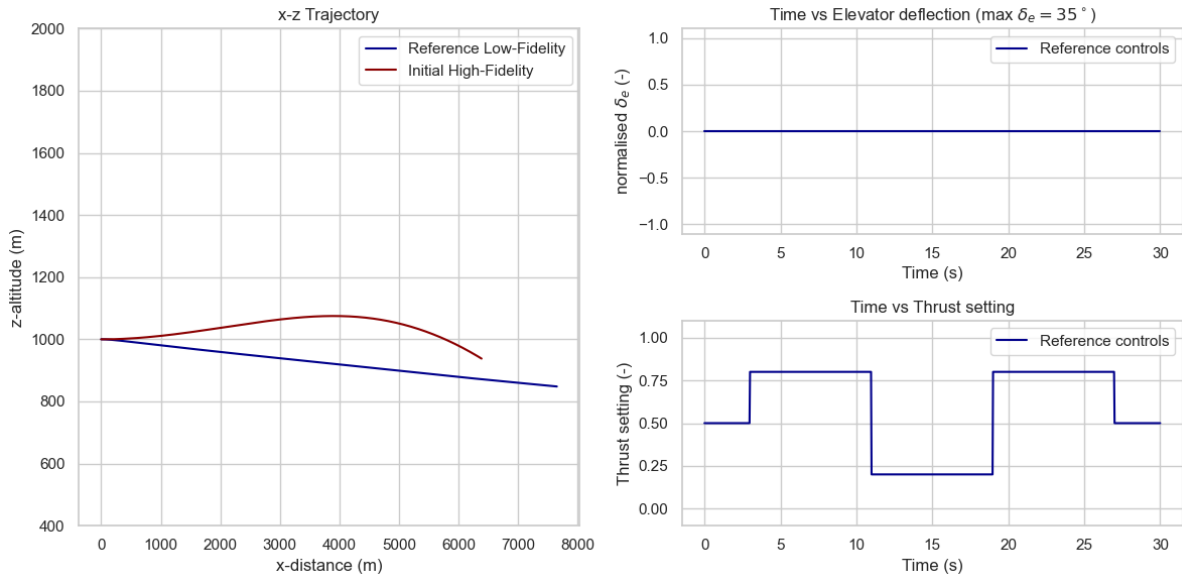


Figure 3.3: Test control case 2. The elevators are kept at 0 degrees deflection. The thrust setting is varied between 20% and 80%, with an average of 50%. The left plot shows the respective x-z trajectory of the low and high-fidelity model.

Comparing to the first test case, the control steps for the thrust setting are increased in duration. Compared to the elevator deflection, engines require some time to spool up, and time is required to have a meaningful acceleration. Elevator inputs have a more immediate effect on the path of the aircraft, so to compensate this, the step duration has increased. Again comparing to the first test case, this case has significantly less features in the reference trajectory. However this is also a good test for the machine learning networks ability to track straight lines and changes in velocity.

Test case 3: Combined controls 3-2-1-1 pattern

The last case combines elevator and thrust settings to investigate the combined pitch and acceleration dynamics:

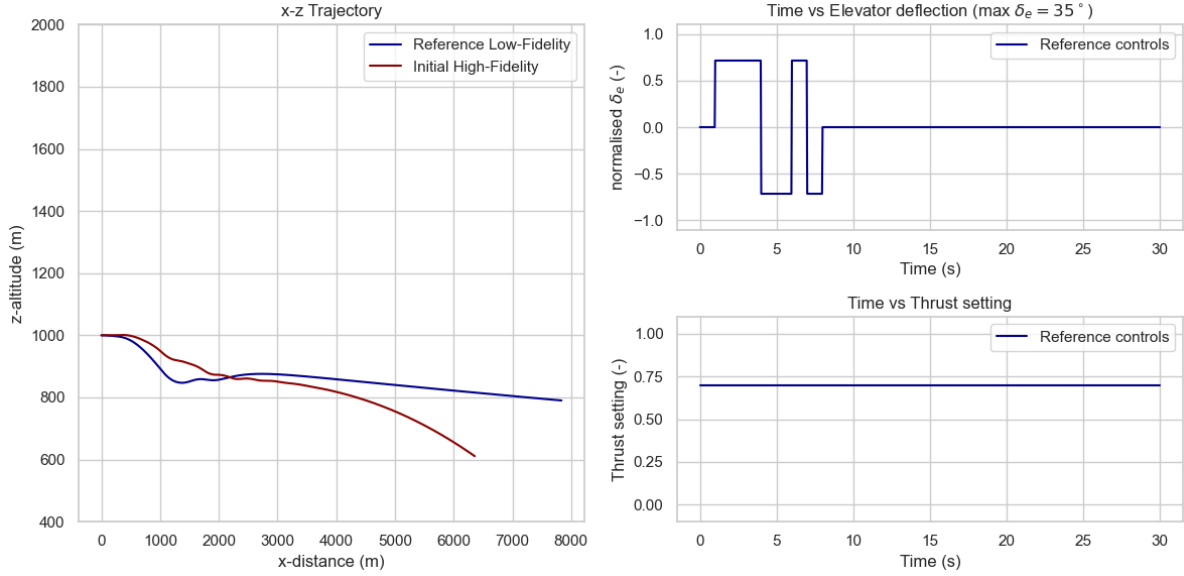


Figure 3.4: Test control case 3. The elevator is excited using a 3-2-1-1 actuation pattern with constant thrust.

For this case a 3-2-1-1 signal has been chosen as it is a common test signal. This signal is chosen due to its widespread use in control system evaluation, which facilitates bench-marking and comparison across different studies. This specific pattern provides a sequence of varying duration amplitudes that challenges the system's response mechanisms, making it ideal for testing the robustness and adaptability of control algorithms. In this scenario, thrust remains constant, as variable engine thrust does not realistically contribute to understanding the dynamic response, nor is it practical for rapid control changes.

Test case 4: Combined controls varying sine wave input

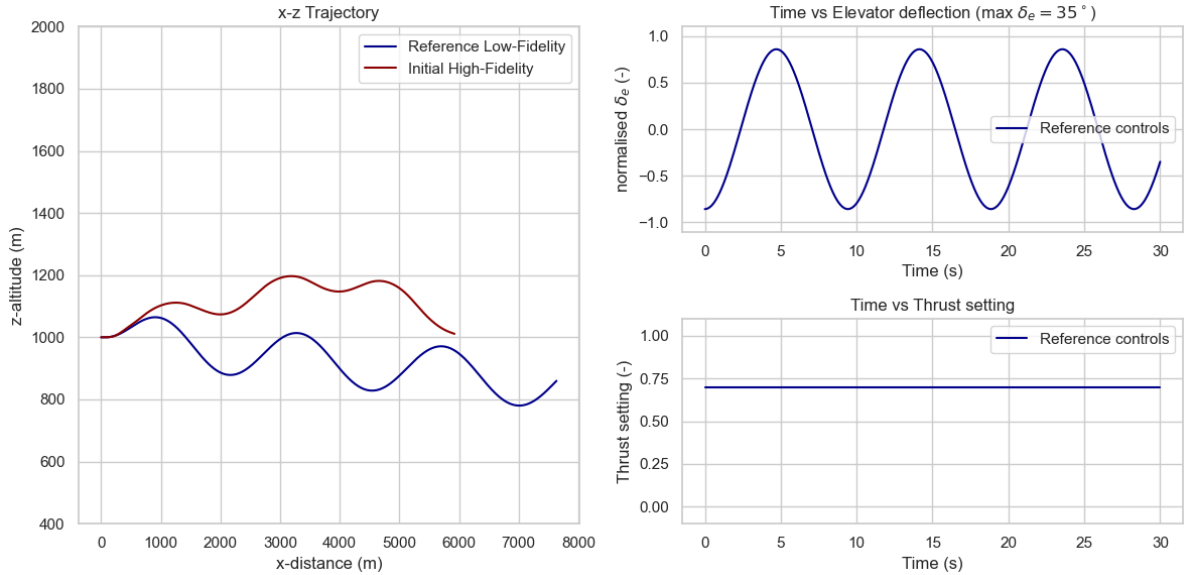


Figure 3.5: Test control case 4. The elevator is varied using sine waves. Thrust is kept constant to isolate the most dominant longitudinal effects.

The sinusoidal elevator deflection pattern tests the control optimiser's ability to anticipate and adjust for the recurring dynamic effects introduced by this oscillatory control input. By employing a sinusoidal

signal, the setup not only challenges the system's response to continuous changes but also its capacity to foresee the effects of these changes over time. This is particularly important in ensuring the control strategy is not merely reactive but proactive, optimising control actions based on predicted future states rather than solely reacting to past and present states.

Combined, these test cases cover four very different points in the control space of both models, each displaying different characteristics for the models to adapt to.

4

Mapping Function Implementation

This chapter delves into the practical implementation of the MPC and machine learning methods, providing a structured approach to their integration and application. The initial sections will outline the specific implementations of MPC and machine learning models, discussing the unique configurations and considerations for each approach. Following the detailed introduction of these implementations, the chapter transitions into a focused case study on machine learning robustness. This case study examines the robustness of the machine learning implementation through the lens of one of the test cases, highlighting the challenges associated with prediction variance and reliable predictive performance.

4.1. MPC mapping function

When adapting the MMSA algorithm discussed earlier, it is important to review what the goal of the implementation is. In this case, the goal is to adjust the control u to minimise the LCSS error. This actually corresponds to a single iteration of the MMSA algorithm, or a single pass through the *reflexive layer*. The reason why only this layer is chosen, and not the whole MMSA algorithm is to restrict the scope of this research, and focus on the mapping performance only. Effectively, this approach will serve as a benchmark to which the machine learning approach is compared to. The fundamental operation of MPC involves two horizons, the steering- and planning-window. These horizons are used in the following step wise process:

- **Model fitting:** The first step in a typical MPC iteration step is the model fitting, where a predictor model is locally fitted with the high-fidelity model. This stage is crucial for ensuring that the MPC's predictions accurately reflect the dynamics of the system. For this implementation, fitting is done over the steering window.
- **Manoeuvre planning:** Following the local fitting in the previous step, the next stage involves solving control optimisation using the fitted predictor model to steer the model along the reference track. This planning step is done for an extended planning window, not just over the steering window. This is an essential concept, as this allows the algorithm to somewhat anticipate on future movements.
- **Model steering:** Once the optimal controls are determined for the immediate planning window, these controls are applied to the high-fidelity model for the duration of the steering window. An important consideration here is the length of the steering window, as short windows are more accurate, however require more subsequent optimisations. Long windows however might not be accurate due to modelling differences between the predictor and high-fidelity model.

In pseudo-code, this becomes:

Algorithm 2 Adapted Multi-Model Steering Algorithm (MMSA)

```

1: Initialize iteration counter  $n_{\text{iter}} \leftarrow 0$ 
2: Solve planning problem to yield tracking trajectory  $\hat{y}_h$  on  $T_{\text{plan}_h}$  and final time  $T$ 
3: Set manoeuvre start:  $T_{\text{steer}} = T_{\text{track}} = T_0, l = 0, p_0 = p^*$ 
4: for  $i = 0$  to number of segments do
5:   Set interval initial conditions  $x_0$ 
6:   for  $k = 0$  to number of control update iterations do
7:     locally fit low-fidelity model parameters  $p_{(\text{opt}, \text{segment})}$ 
8:     Solve model predictive tracking problem on the tracking horizon to yield controls
        $u_{(\text{opt}, \text{segment})}$ 
9:     Project controls onto global control grid from  $T_{\text{start}}$  to  $T_{\text{steer}}$ 
10:    Forward simulate using the high-fidelity model with controls  $u_{(\text{opt}, \text{segment})}$  to yield
       $x_{(\text{opt}, \text{segment})}$ 
11:    end for
12:    Project outputs  $x_{(\text{opt}, \text{segment})}$  from  $T_{\text{start}}$  to  $T_{\text{steer}}$  onto global trajectory grid
13:    Increment update counter:  $l = l + 1$ 
14:    Local parameter correction yields new parameter estimate  $p_l$ 
15:  end for
16: Evaluate tracking error:  $e = \int_{T_0}^{T_{\text{track}}} \|\tilde{y}_h - \tilde{y}_s\| dt$ 

```

A critical aspect of implementing MPC is the strategic balance between the steering window and the planning horizon. This balance represents a trade-off between achieving short-term accuracy and maintaining effective long-term planning capabilities. For instance, employing a long steering window paired with a short planning horizon can enhance the accuracy of the predictor model due to the availability of abundant fitting data. However, the drawback of this configuration is that only a relatively short-term planning problem is addressed, which might not adequately account for future state requirements.

Conversely, a shorter steering window combined with a longer planning horizon may result in a less accurate predictor model due to limited data for fitting. Yet, this setup can be advantageous for manoeuvre planning because it allows the MPC to consider a larger portion of the reference trajectory, potentially enhancing the control strategy's responsiveness to changes. The downside is that the relevance of the computed manoeuvre decreases over time as the certainty of maintaining accuracy diminishes the further out the plan extends.

Additionally, the quality of model fitting and the realism of the reference trajectory play significant roles in determining the overall efficacy of the MPC algorithm.

4.2. Mapping function: Machine Learning

The machine learning-based approach closely follows the principles of trust-region based ASM, which in turn is adapted to leverage the sequential data processing capabilities of machine learning. This adaptation incorporates the PKI-D architecture, as detailed in Figure 2.3.3, combining sparse evaluations typical of ASM with the robust predictive capacity of PKI-D. The PKI-D model employs a machine learning network to refine the output of a low-fidelity model, enhancing its approximation to the high-fidelity model. This refined model is then utilised within an aggressive space mapping framework to optimise control sequences for the high-fidelity model. This approach harnesses the strengths of both machine learning and space mapping techniques, aiming to achieve high accuracy in control mapping with minimal high-fidelity evaluations. The flowchart below shows how this process works:

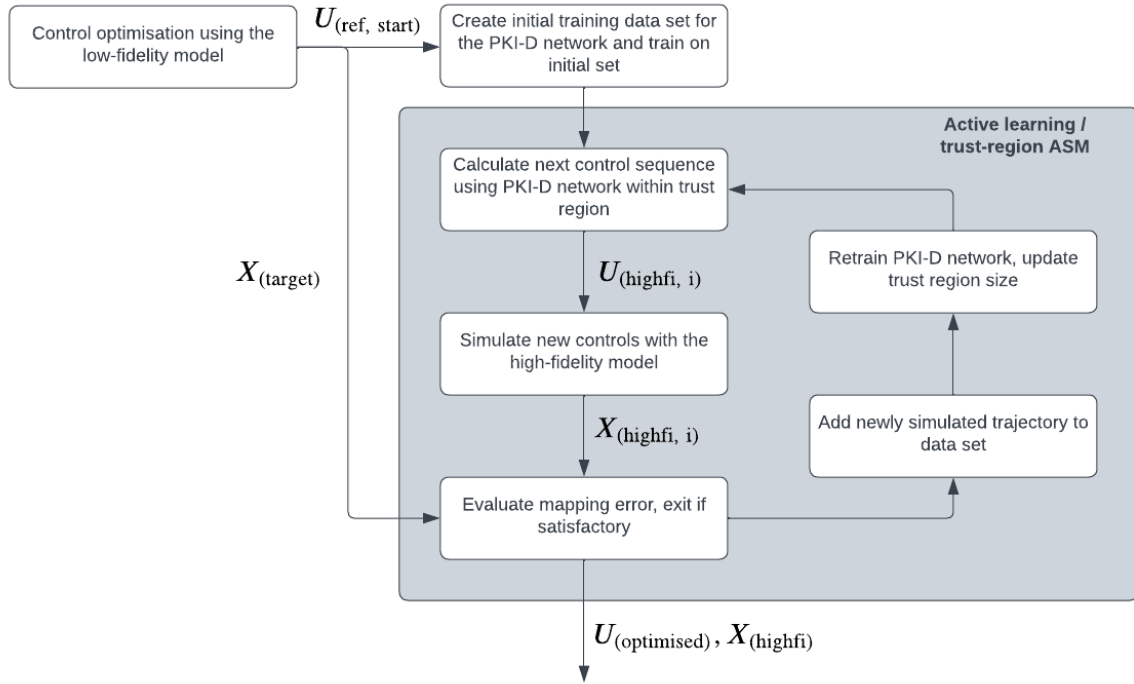


Figure 4.1: Process diagram of the hybrid trust-region ASM / PKI-D architecture.

In the proposed machine learning-based space mapping framework above, active learning plays an important role, particularly in addressing the initial need for training data. Machine learning networks are notorious for requiring large data sets to base its predictions on. By using active learning, the network can be initially trained on a minimal dataset, precisely tailored to closely approximate the high-fidelity model within a localised space around the initial control sequence. Later during the optimisation process, the network can be further refined by adding additional data points. This approach leverages the strengths of machine learning to approximate any function, adapted to the principles of space mapping. A few points to note is that in this study, the upper left block containing control optimisation is replaced with the test cases introduced in the previous chapter. Furthermore, mapping error evaluation can be done with any of the methods employed in Appendix B.

During each iteration of the ASM process, the model employs active learning to incorporate new data points to refine the network's prediction outside of the initial data set, allowing it to adapt and maintain accuracy as the control sequence evolves. Such an approach aligns with space mapping principles, which prioritises reducing high-fidelity evaluations to the bare minimum. Nonetheless an initial data set is required for the network to train on. The main idea here is that this hybrid ASM - machine learning approach is able to outperform conventional mapping techniques in terms of data requirement, even with the initial high-fidelity simulations required to train the model. Next, each step in Figure 4.1 will be discussed in more detail.

4.2.1. Data set generation

The generation of an initial dataset is a critical element in machine learning applications, especially in specialised scenarios such as this study. As outlined in section 3.3, the parameterisation of control sequences is achieved using control points and Hermite splines, which fundamentally support the development of the initial dataset around a designated control sequence. An illustration of this process is provided in Figure 4.2.

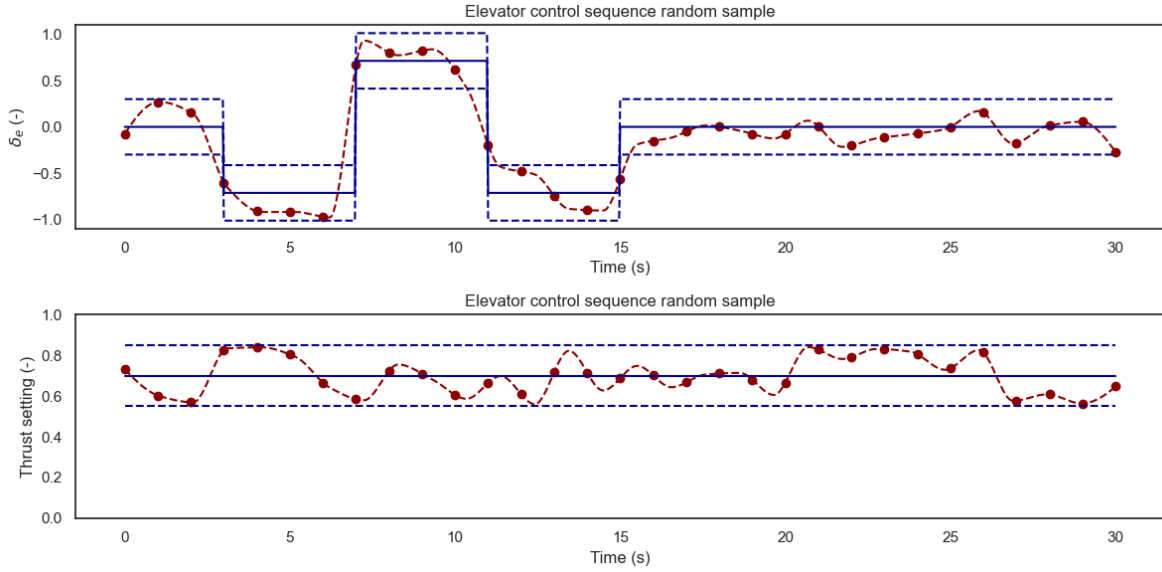


Figure 4.2: Example of a randomly generated control sequence around a starting control sequence. The solid blue line represents the reference control sequence, and the dashed blue line represent the limits of the random uniform distribution. The new sample is fitted through red control points

In this implementation, new control points are generated using a uniform random distribution around each reference control point to ensure a uniform spread within a controlled radius of the initial control sequence. The utilisation of Hermite splines guarantees that the generated sequences adhere to specified boundaries, facilitating the creation of smooth and realistic control trajectories. These trajectories subsequently serve to simulate the low- and high-fidelity models, providing the necessary paired data for the initial training of the machine learning model.

Unlike conventional machine learning applications that often segregate datasets into training, validation, and testing sets to evaluate model performance, this study adopts a different approach. Because of the high costs associated with generating high-fidelity simulations, it is impractical to allocate separate datasets for validation and testing. Consequently, the network's performance is evaluated solely based on its training on an initial dataset and its adaptation during a predetermined number of iterations. This methodology aligns with the primary goal of space mapping to minimise high-fidelity evaluations, ensuring that every data point used directly contributes to the optimisation process without the need for external validation.

4.2.2. PKI-D model implementation

As mentioned in section 4.2, this research focuses on implementing the PKI-D architecture to enhance a low-fidelity model with the advanced capabilities of seq2seq networks. The choice of PKI-D architecture is motivated by its efficient and strategic use of training data, the smooth integration of seq2seq networks, and the effective utilisation of all available data.

Processing all high-fidelity data upfront allows the machine learning training to proceed efficiently without the need for repeated high-fidelity simulations. This method not only minimises computational demands but also enables parallel processing, leveraging the full capabilities of modern machine learning frameworks. Additionally, the architecture accommodates the on-the-fly addition of data, similar to active learning, which enhances data efficiency.

The PKI-D architecture fully utilises seq2seq models by providing them access to the entire trajectory and control sequence at once. The following diagram illustrates the data flow within the PKI-D setup, highlighting how low- and high-fidelity models interact through the machine learning network during both training and application phases:

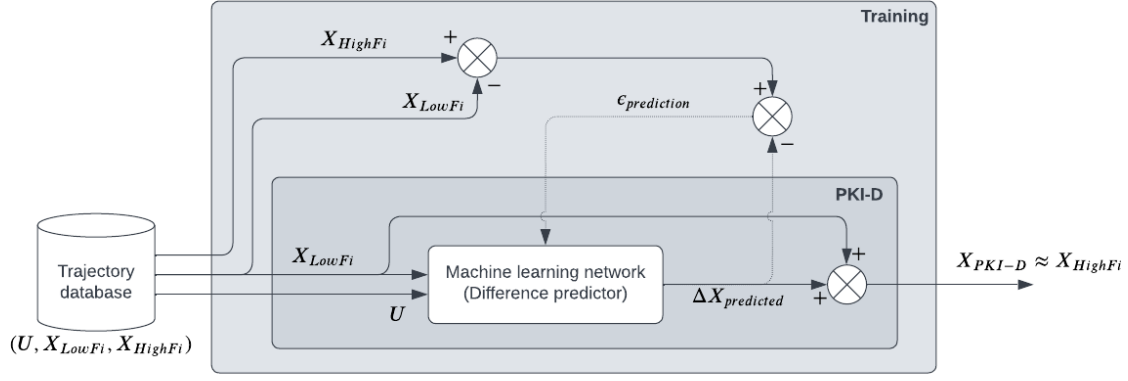


Figure 4.3: Data flow within the PKI-D architecture, illustrating the interaction between the low- and high-fidelity models through the machine learning network during training and usage.

Armed with an initial trajectory from the low-fidelity model and the initial control sequence, the network is tasked with predicting differences relative to the high-fidelity model. The integration of a low-fidelity model as a starting point for predictions ensures that the network's outputs remain grounded, minimising potential deviations from actual conditions and enhancing predictive accuracy.

Here, the closed-loop implementation of the low-fidelity is used to prevent significant prediction drift over time, which is essential for maintaining accuracy in long-term predictions. This configuration is especially critical in the PKI-D architecture, where the initial 'guess' provided by the low-fidelity model lays the groundwork for the machine learning network's output.

A large advantage is that the network can be trained using a set of pre-computed high-fidelity trajectories with their low-fidelity counterparts. These trajectories and control sequences can be crafted on the spot, or retrieved from a pre-existing collection. This allows for flexible implementation later on. During network inference, the control sequence and low-fidelity trajectory are used by the network to predict the difference between the high- and low-fidelity model. This also plays to the strengths of network architectures such as transformer networks, which are able to evaluate the whole control sequence and low-fidelity trajectory at once to predict any differences.

4.2.3. Network selection

As discussed in section 2.4, sequence-based data can be managed through two principal methodologies within neural networks, each developed to address specific challenges in sequence data processing:

- **Recurrent approach:** This approach ingests data step by step. Introduced as LSTM in section 2.4, this study will make us Gated Recurrent Units (GRU) which are a streamlined version of LSTMs. GRUs merge the forget and input gates into a single update gate. This simplification reduces the number of parameters and computational steps, enhancing processing efficiency while retaining the ability to model complex temporal dynamics.
- **Parallel approach:** Introduced as transformer networks, these networks processes a sequence all at once using positional encoding to capture temporal dynamics, and attention networks to capture important features in the input. In general, these networks excel in longer sequences with complex dependencies between sequences.

The deliberate selection of GRU and Transformer models for this study highlights their contrasting approaches to data processing—sequential versus parallel. By exploring these opposites, the study aims to provide a nuanced perspective on the integration of machine learning techniques with space mapping, assessing their efficacy in various operational scenarios where these underlying dynamics play a crucial role.

4.2.4. Optimiser choice

The control optimiser also plays an important role in the space mapping algorithm, serving not only to determine the next set of controls for the high-fidelity model but also to (indirectly) select the next data points for the PKI-D network's learning process. This function is crucial, especially considering the limited initial dataset. For this study, a steepest gradient trust region optimiser was selected for several reasons. Firstly, a steepest gradient descent optimiser is inherently designed to locate a local minimum, which is sufficient for this study's objective of demonstrating the integration of space mapping and machine learning, rather than optimising for the best possible mapping function. Secondly, the trust region approach is well-established for use with local surrogate models, where it adjusts the trust region size based on the surrogate's performance. If the surrogate model performs poorly, the trust region is reduced, thereby preventing poor predictions. The implementation of this optimiser is guided by the following pseudocode, which aligns with the operational flow depicted in Figure 4.1:

Algorithm 3 Steepest descent optimiser

```

1: Set  $max\_iter$ 
2: Train model on initial dataset for  $n\_training\_epoch\_init$  epochs
3: Calculate initial high-fidelity and difference model error for  $U_{opt}$ 
4: while iteration count <  $max\_iter$  do
5:   Calculate gradient of error with respect to control points using the PKI-D model
6:   Perform line search to find optimal step size  $\alpha$  within trust region
7:   Update control points using the optimal step size found
8:   Run high-fidelity model with new control sequence
9:   Add outputs from high-fidelity simulation to database
10:  Compute new errors for the high-fidelity and PKI-D model
11:  Calculate trust region performance ratio  $r$  based on actual and predicted reductions in error
12:  if  $r \geq 1.0$  then
13:    Increase size by 10%
14:  else if  $1.0 > r > 0.5$  then
15:    Keep size unchanged
16:  else
17:    Reduce size by 10%
18:  end if
19:  if  $r \leq 0.5$  then
20:    Retrain network if the
21:  end if
22:  Constrain trust region size within predefined minimum and maximum bounds
23:  Increment iteration count
24: end while

```

The approach above closely follows the process flow in Figure 4.1. It must be emphasised that the optimiser plays a crucial role in the mapping process, however in the interest of limiting the scope of this research, only a single optimiser will be used. Furthermore, it will be treated as a static object for the network to interact with. As a reminder, the focus of this research is on exploring active learning in the context of space mapping.

4.3. Machine learning model robustness

The implementation of machine learning models described in section 3.3 utilises networks trained on small datasets, as dictated by space mapping methodology. This section focuses on the robustness of these networks, a critical quality ensuring they maintain stable performance despite input data variations.

This becomes even more challenging due to this study’s small data sets. While robustness is key to network and algorithm efficacy, this discussion will not explore specific influencing parameters but instead provide context to this study’s approach.

Robustness in machine learning is defined as the model’s ability to deliver consistent performance irrespective of input- or training data changes. This is essential for systems employing finite difference gradient calculations, where consistent outputs are crucial for ensuring both accuracy and reliability. A robust model prevents minor input perturbations from causing significant output errors:

Machine learning model robustness denotes the capacity of a model to sustain stable predictive performance in the face of variations and changes in the input data[50].

Achieving robustness typically requires extensive datasets, guided by principles like the “rule of 10” or the more stringent “rule of 50,” which recommend that the number of samples be up to 50 times the number of trainable parameters. While typically applied to image based networks, the same order of magnitude is expected for sequence based networks. For networks with 300,000 to 600,000 parameters, this would require a prohibitively large number of high-fidelity model evaluations [51]. This study however shifts from general robustness to focus on stability only for very specific cases. This greatly reduces the need for vast datasets, emphasising only the most relevant training samples.

Variability, a key aspect of robustness, refers to how much a model’s performance or output changes under different conditions. To introduce variability and combat overfitting, techniques such as dropout are employed during network training. Dropout randomly deactivates a subset of neurons, encouraging the network to develop redundant pathways and enhance generalisation. Although disabled during inference, the stochastic nature of neuron deactivation during training influences overall network performance, especially when trained on smaller datasets.

To achieve repeatable results, fixed random seeds are often used. This method stabilises the inherent randomness in algorithmic training, ensuring reproducible outcomes by eliminating variations in operations like data shuffling and dropout between runs. The influence of randomness is less pronounced in larger datasets where the volume of data averages out specific behaviours and reduces fluctuations, diminishing the impact of random variability [52].

While robustness and variability are not the primary focus of this study, they remain crucial considerations in any discussion about machine learning and space mapping applications. The next section will present a case study from the first test case to illustrate how random variability can affect model performance and robustness. For context, the following diagram outlines the major components of the machine learning aspect of the study:

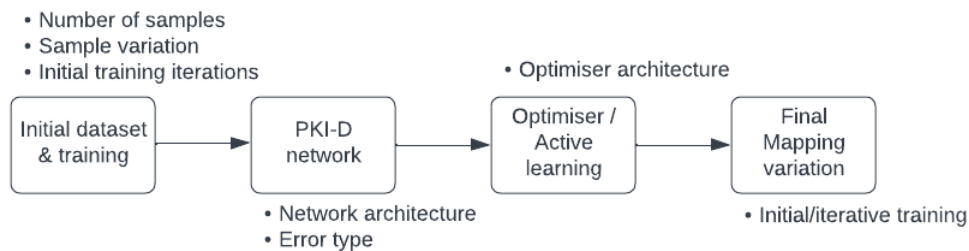


Figure 4.4: Schematic overview of influential components in the machine learning mapping process

where this study mostly focuses on the network type and relation between initial and iterative training, this case study will briefly cover the first block of the mapping process chain, as well as the error type

in the second block. The specific implementation of the machine learning model as well as data set generation will be discussed in chapter 6.

4.3.1. Initial data set robustness

As discussed in the previous sections, random variability is a major factor when dealing with small data sets. Where most datasets' size are roughly on the same order as the number of trainable parameters, this study will make use of extremely small datasets on the order of 10-20 samples. This makes the model vulnerable for variations between random seeds, as shown in:

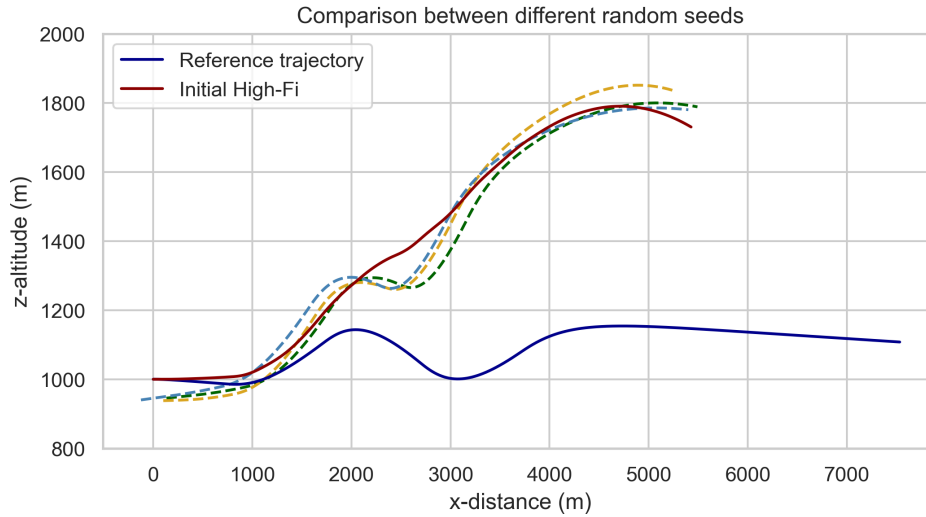


Figure 4.5: Visualisation of the effect of different random seeds on the prediction of the machine learning network (dotted lines) for the reference case controls after initial training.

The dashed lines represent the network's prediction after a fixed amount of initial training for different random seeds, while they generally all follow the same trend, clearly there is quite some variation between. Ideally the prediction for different random seeds is consistent, meaning that any optimisation based on these results is not dependent on picking the right random seed. To quantify the effect of the random seed, the following graph can be used:

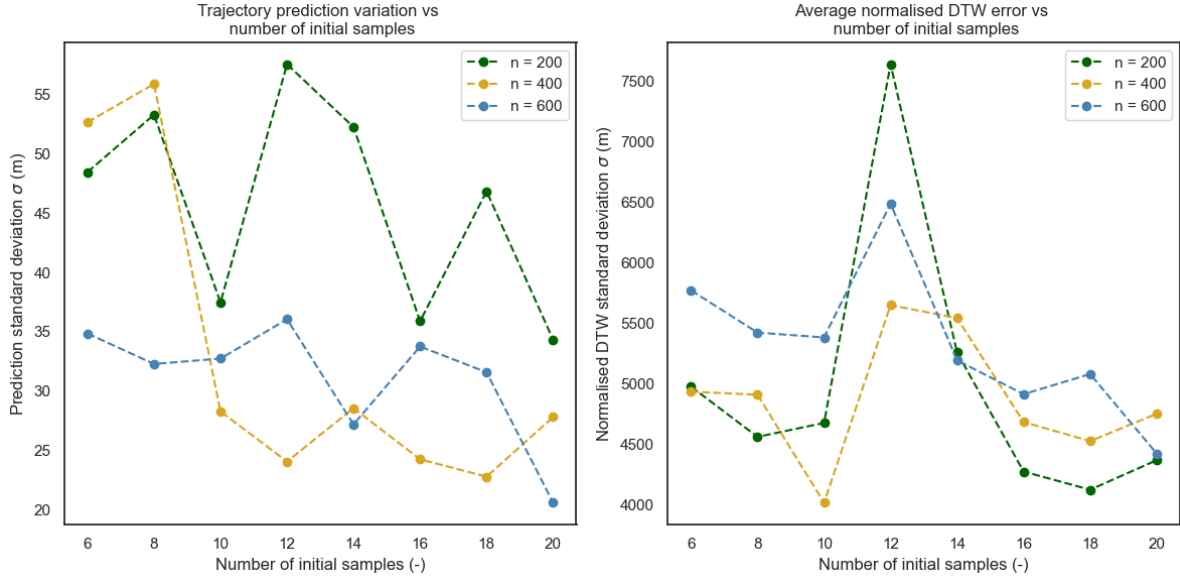


Figure 4.6: Variation in prediction performance as a function of sample size and training iteration (n). These graphs illustrate how the standard deviation in predictions changes with different numbers of initial samples and training iterations. Each data point represents the aggregate outcome from eight distinct random seeds and an initial data perturbation level of 0.15 using a transformer model as predictor network.

The left graph illustrates the average prediction variation at each network time step, highlighting prediction stability and the variability among different random seeds for the same input. There is no clear trend relating initial training iterations and sample size, though generally, higher training iterations result in lower prediction variability. However, networks initially trained for $n = 400$ iterations sometimes show even lower variability, except at very low sample counts, illustrating the complexities in preparing the initial dataset.

The right graph shows the average normalised DTW error relative to the number of sequences. Notably, higher training iterations do not guarantee better performance with a low number of samples. Furthermore, a significant spike in prediction error occurs across all training iterations, confirming that these results, averaged over multiple seeds, reflect consistent patterns rather than anomalies from individual seeds.

In summary, Figure 4.6 shows that while initial training iterations and sample size significantly affect prediction accuracy and variability, the relationships are complex. The impact of dataset perturbation is explored in the subsequent graph.

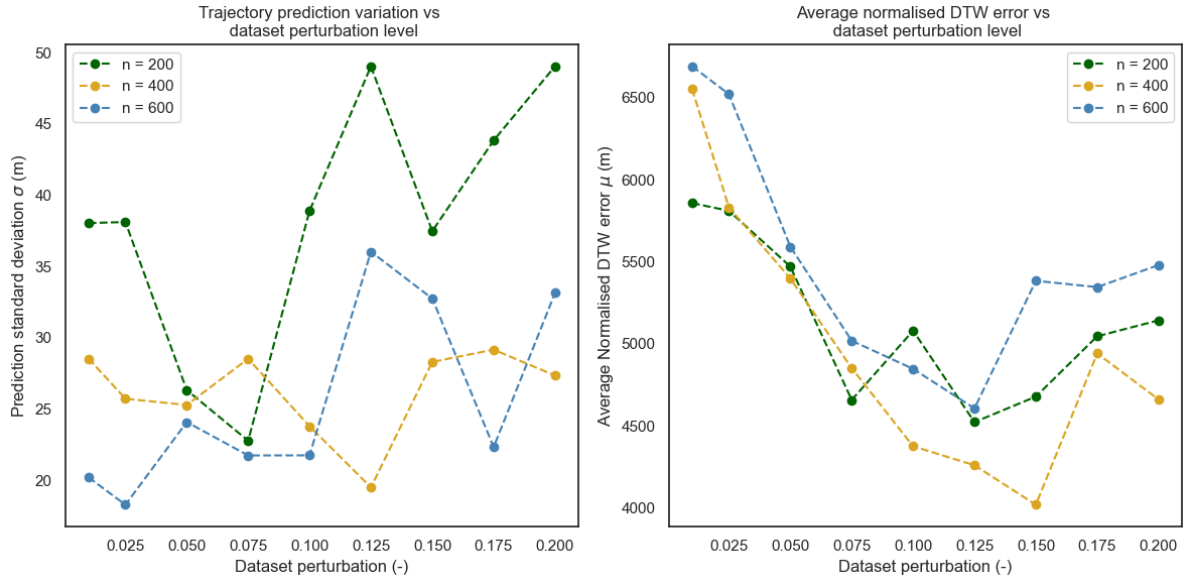


Figure 4.7: Variation in prediction performance as a function of data set perturbation and training iteration (n). This graph shows the impact of initial sample size perturbation for a fixed number of samples. Each data point represents the aggregate outcome from eight distinct random seeds using a transformer model as predictor network.

Continuing from previous analyses, the left graph in Figure 4.7 emphasises the role of initial training iterations in managing network variance. Like Figure 4.6, identifying a clear trend in training iterations is difficult suggesting that multiple factors influence outcomes. A more defined trend appears in the impact of dataset perturbation on average prediction error, where excessive training iterations degrade performance at higher perturbation levels. This may indicate that a network that is over-trained on large variations struggles with finer details. The unclear trends in other graphs highlight an area for future research.

4.3.2. Error choice robustness

Regarding model robustness, this case study also investigates the influence of error method on overall convergence. Especially for long sequences and when values quickly diverge, establishing a fundamental understanding of how the error estimation method influences the optimiser is crucial. In Appendix B, four methods for establishing errors are presented. Regarding the LCSS method, special attention should be given to setting the correct threshold value to include the whole domain of the problem. The effect of a threshold value that is too small can be seen as in the following graph:

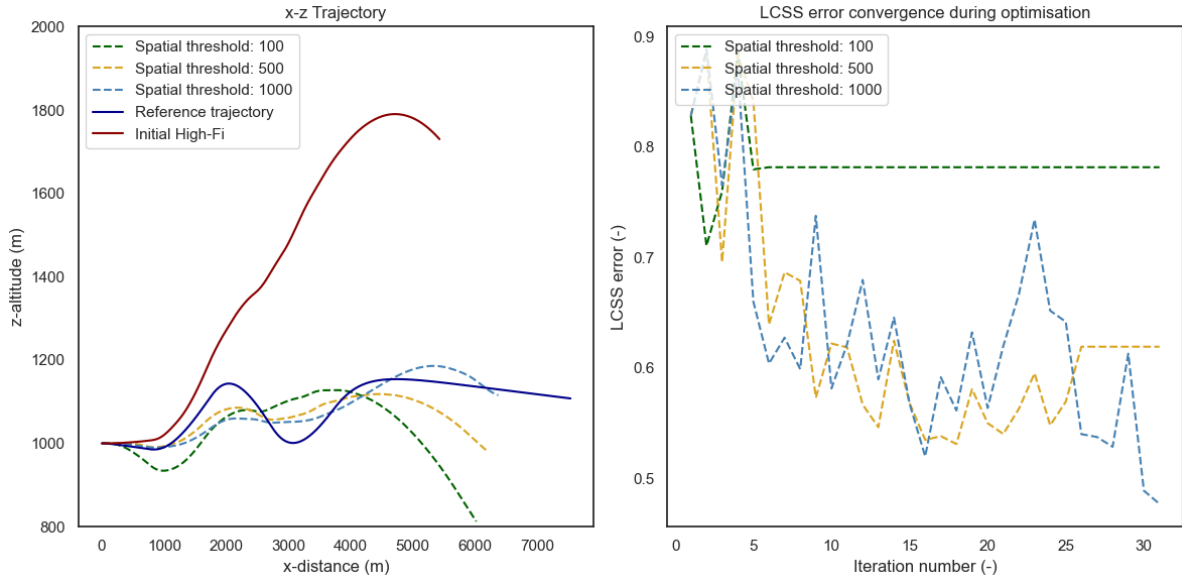


Figure 4.8: Influence of LCSS threshold on optimiser convergence. For this graph, a steepest gradient optimiser using a transformer network has been used.

As can be seen, the optimiser gets stuck in a local minimum with a 0 gradient when the threshold is too restrictive. As control perturbations introduced by the finite difference method change the trajectory, if the change happens outside the threshold this is not recorded leading to a zero gradient.

Other methods such as ELS, DTW, and DFD do not have this limitation as these methods don't have to deal with thresholds. Their respective impact on convergence can be found in the following graph:

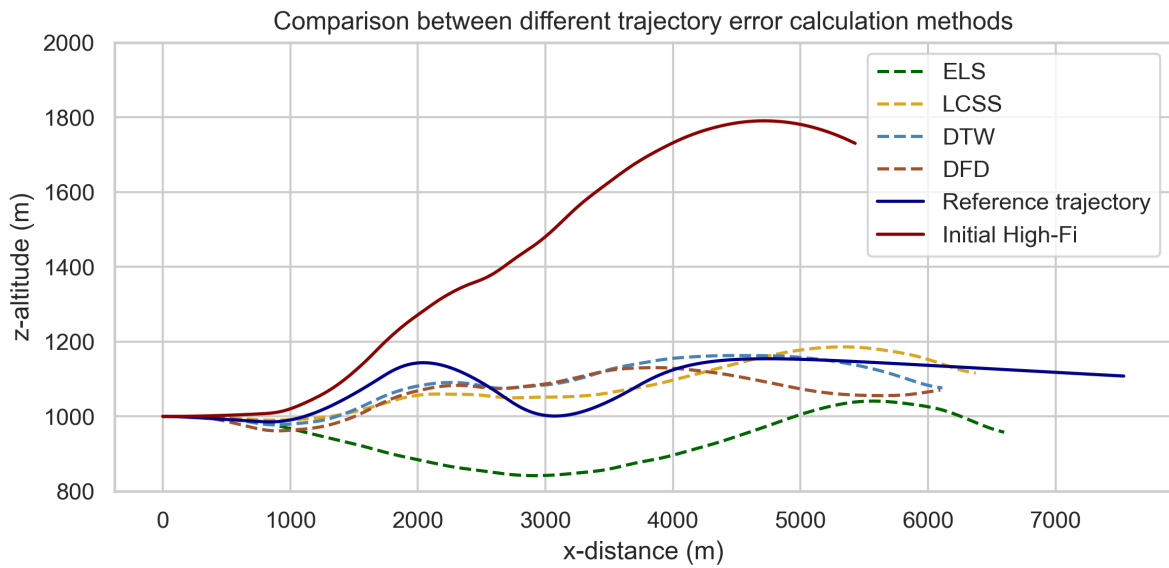


Figure 4.9: High-fidelity trajectories resulting from control optimisation using various error calculation methods.

Table 4.1: Optimisation results for different error calculation results, recalculated using each respective method

| Method used during optimisation | ELS (1e3 m) | DTW (1e3 m) | DFD (m) | LCSS error (-) |
|---------------------------------|--------------|--------------|--------------|----------------|
| ELS | 310.7 | 182.3 | 949.1 | 0.81 |
| LCSS | 334.8 | 101.6 | 1160.5 | 0.47 |
| DTW | 441.6 | 144.5 | 1442.3 | 0.58 |
| DFD | 457.7 | 165.7 | 1438.4 | 0.59 |

The analysis of the error calculations in Table 4.1 and Figure 4.9 demonstrates that the LCSS error method is a suitable error estimation method for optimisation. However this does highlight another vulnerability in the mapping chain as incorrect settings can still lead to subpar performance. Regarding the other error estimation methods, this highlights another point of attention, mainly that optimiser performance should not be judged based on error magnitude alone. In the table above, the ELS method achieves the lowest error for both ELS and DFD error, while visually producing the worst results out of the different error methods. This contradiction between numerical values and visual interpretation will be covered in more detail in future section.

To summarise the robustness case study, robustness is a key aspect in performance and requires careful consideration. Enhancing robustness requires a balance between the number of samples, initial training iterations, and the degree of sample perturbation. Moreover, choosing the right error methodology and setting appropriate threshold values is crucial for improving robustness and significantly affects the outcomes of the optimisation. While this analysis is not comprehensive, it highlights the complexities of integrating space mapping with machine learning, which extend beyond the scope of this study.

5

Model Predictive Control Based Space Mapping

This chapter will present the results of the sensitivity study of the MPC based approach. As explained in chapter 5, a crucial aspect of MPC is the balance between short-term accuracy and long-term planning. This balance will be examined by varying the following parameters:

- **Number of segments:** In MPC, the entire manoeuvre horizon is typically divided into equal-length segments, each optimised sequentially. Adjusting the number of segments directly affects the length of each segment, thereby modifying the duration of the steering window. This alteration allows exploration of how finer or coarser segmentation impact the controller's responsiveness and planning precision.
- **Planning horizon:** The length of the planning horizon is critical in determining the extent to which future states influence current decisions. By varying this parameter, the implications of different horizon lengths on the effectiveness of the control strategy are examined.

By examining these parameters introduced above, this sensitivity study aims to uncover the dominant factors involved in MPC mapping, laying the groundwork for the eventual comparison with machine learning models. Before diving into the sensitivity study, the results of test case 3 and 4 are highlighted, the results of test case 1 and 2 are covered in section C.1:

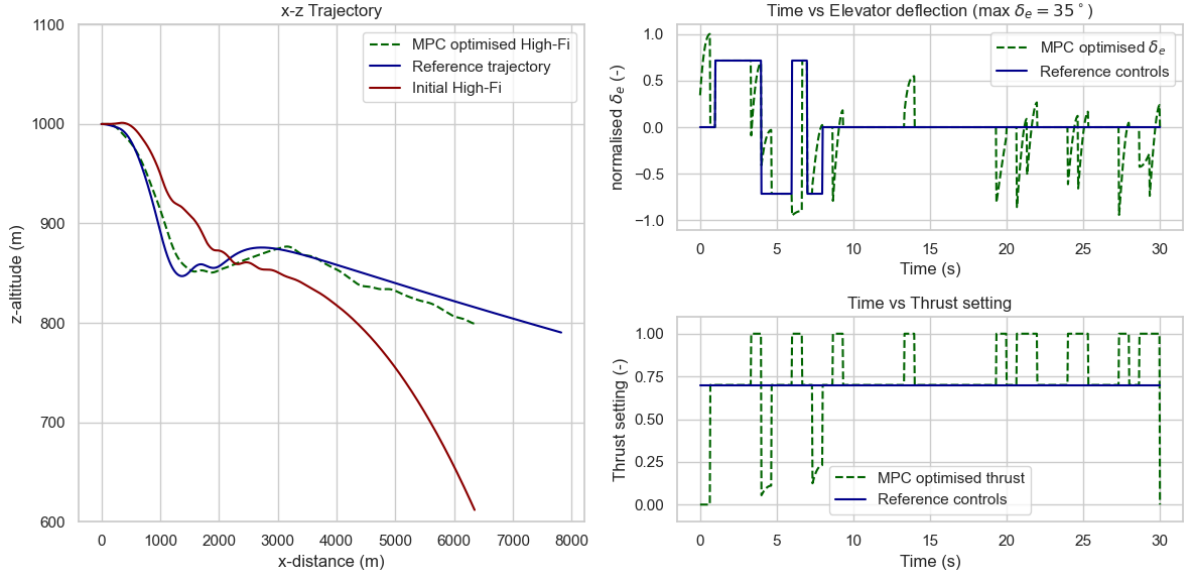


Figure 5.1: MPC optimised high-fidelity trajectory for test case 1. Furthermore, $n = 45$ segments were used, and a planning window of $T = 3s$ and a manoeuvre duration of 30 seconds.

Immediately notable is that overall, the MPC mapping function is able to track the reference trajectory to a very high degree. The consecutive control impulses are well preserved in the mapped trajectory, only deviating by a small amount. The choice of untrimmed models is also visible here. Immediately from the start, the controls are altered as they do not have to start from a predefined trim point.

Table 5.1: Error comparison between the initial and MPC optimised high-fidelity trajectory error compared to the reference trajectory for the first test case. For the LCSS error, a spatial threshold of 100 meter was used, and a temporal threshold of 30 steps were used, which is equivalent to 1 second.

| Error Type | Initial error | Optimised error | Percentage decrease w.r.t initial |
|------------------------------|---------------|-----------------|-----------------------------------|
| DTW error (1e3 m) | 175.2 | 121.6 | 30.6% |
| DFD (leash length) error (m) | 1485.2 | 1445.6 | 2.7% |
| LCSS (-) | 0.58 | 0.57 | 2.8% |
| ELS (1e3 m) | 487.2 | 463.0 | 5.0% |

Looking at the performance metrics in Table 5.1 only the DTW error seems to decrease by a significant amount, matching the visual difference between the initial and optimised trajectory. This can be explained by taking a look at what the dominating factory in the trajectory is and how that effects each error method. Starting with DTW and ELS, these are both ways to tracks cumulative error, however DTW decreased over twice as much, implying that temporal differences have quite an impact on trajectory matching. Furthermore, the DFD error has also decreased less then one would expect looking at the flight path. This reveals a deceptive trait, mainly that the biggest error is in covered distance and not in vertical distance. The low-fidelity model is able to reach much further than the high-fidelity model. This can be attributed to modelling differences. Lastly, the LCSS suffers from a slightly different problem, mainly the fact of boundary tuning. For the first part, the two trajectories are within the distance threshold. Later on however, the difference in covered distances consistently falls outside the allowable threshold, even though it is at least visually better aligned. Moving on to the fourth test case:

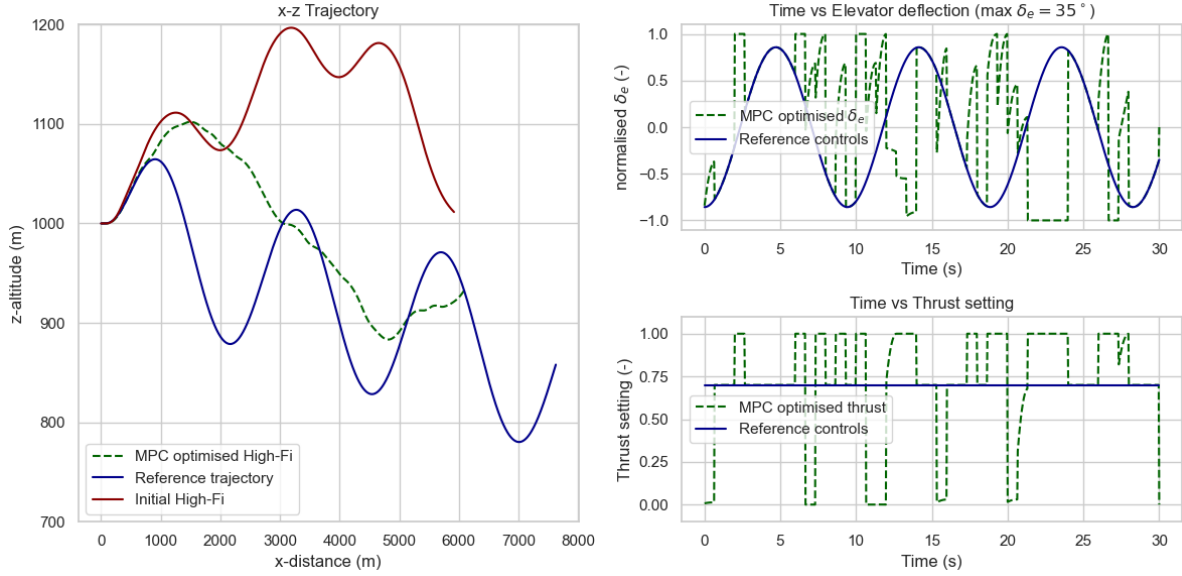


Figure 5.2: MPC optimised high-fidelity trajectory for test case 1. Furthermore, $n = 45$ segments were used, and a planning window of $T = 3s$ and a manoeuvre duration of 30 seconds.

Table 5.2: Error comparison between the initial and MPC optimised high-fidelity trajectory error compared to the reference trajectory for the first test case. For the LCSS error, a spatial threshold of 100 meter, and a temporal threshold of 30 steps which is equivalent to 1 second were used.

| Error Type | Initial error | Optimised error | Percentage decrease w.r.t initial |
|------------------------------|---------------|-----------------|-----------------------------------|
| DTW error (1e3 m) | 319.4 | 183.7 | 42.5% |
| DFD (leash length) error (m) | 1714.2 | 1524.0 | 11.1% |
| LCSS (-) | 0.81 | 0.78 | 3.6% |
| ELS (1e3 m) | 587.7 | 504.4 | 14.2% |

Looking at the results above for the combination of number of segments, some of the weak points of the MPC method are shown. The sine wave input causes large vertical variations which requires anticipation not only of the nearest hump, but also following humps. The repeating pattern highlights the need for anticipation. An overshoot due to a combination of controls and initial conditions lead to a phase offset compared to the reference trajectory which is not compensated later on. A similar trend is observed in Figure C.1. That is not to say that the MPC method cannot be used in increasingly complex trajectories, only that careful tuning the MPC method is required as will be done in the coming section. Regarding the error metrics, a similar story unfolds compared to the previous case. Obviously the DTW error decreases significantly, however the LCSS error decreases by a much smaller amount. Initially the model seems to fit well, however later on it only coincides with the reference trajectory a handful of times which contributes to a reduction in score.

To summarise the result of the initial MPC mapping results, the MPC method is quite able to correctly steer the high-fidelity model along the reference trajectories. However, in cases where anticipation is required such as in test case 4, this method initially lacks. Furthermore, the resulting error needs to be critically evaluated, keeping in mind the way error is calculated, low- and high-fidelity model differences, and which threshold values are used. The coming sections will investigate two important MPC metrics, those being the number of segments and planning horizon time to gather a complete view of the MPC method.

5.1. Number of segments

As stated before, the number of steering segments is an important factor in the MPC based methods. The effect that this has is two-fold, the high-fidelity model is steered for a shorter time, and the low-fidelity model is also fitted for a shorter time. Varying the amount of segments for each reference case yields the following plots:

Test case 1

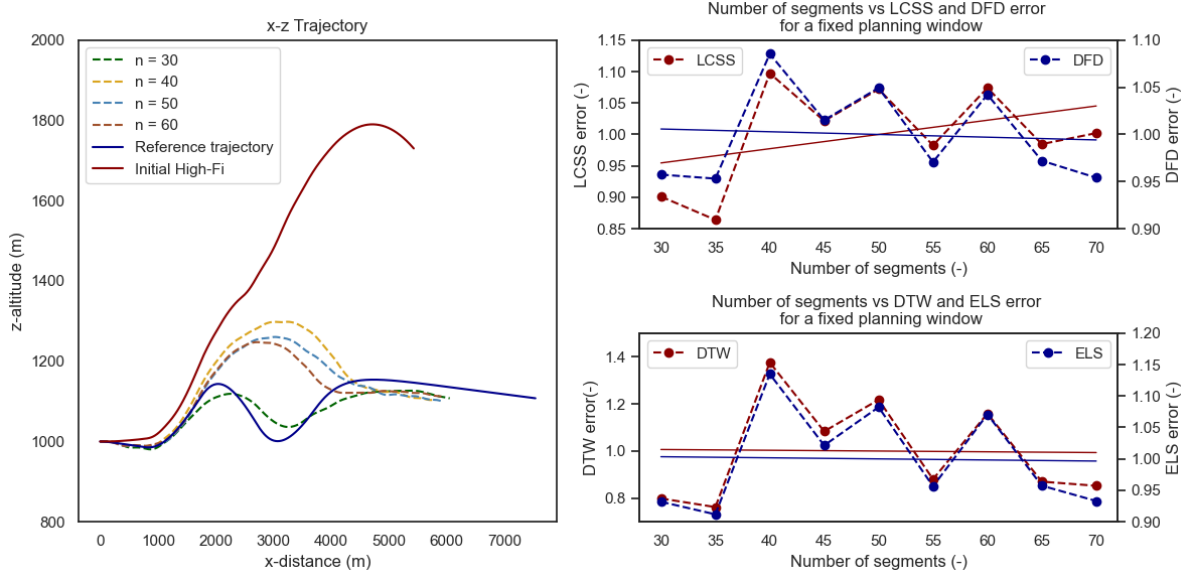


Figure 5.3: Effect of varying the number of segments for a fixed planning time of 3 seconds. Each respective error is normalised using the mean value of that error type.

Table 5.3: Comparison between the different plot regression lines statistical values for the relation between number of segments and error for test case 1

| Error Type | Average value | R^2 | p-value |
|-----------------------------|---------------|-------|---------|
| DTW error (1e3) | 145.5 | 0.000 | 0.958 |
| DFD error (leash length)(m) | 1363.9 | 0.006 | 0.837 |
| LCSS (-) | 0.61 | 0.156 | 0.293 |
| ELS (1e3) | 412.1 | 0.001 | 0.940 |

When evaluating the results of regression analysis such as in Figure 5.3, two statistical measures are crucial: the R^2 value and the p-value. The R^2 value provides insight into how much of the variability in the dependent variable can be explained by the independent variables used in the model. A higher value of R^2 such as 0.95 indicates a model that effectively captures the underlying patterns in the data, whereas a lower R^2 such as 0.1 suggests a poor fit. The p-value, on the other hand, tests the statistical significance of the observed relationships, helping to distinguish effects that are likely true from those that might occur simply by chance or inherent model variation. Here a high p-value suggest no statistical significance, whereas a low p-value suggest low chance that the acquired results are the result of random variation.

Looking at these metrics in Table 5.3, none of the error metrics indicate a significant relation between number of segments and mapping performance. This is further reinforced by looking at the actual error values in the plot, where both for high and low number of segments good results can be achieved. Looking at the DTW error specifically, this parameter sees the largest variance, which is accompanied

by the lowest R^2 value. To summarise these results, the MPC method is quite able to be tuned to a good fit at multiple settings as can be seen for the $n=30$ line in Figure 5.3. However finding a good segments can be a matter of trail and error for this particular case as there is no clear trend in any parameter to base adjustments on.

Test case 3

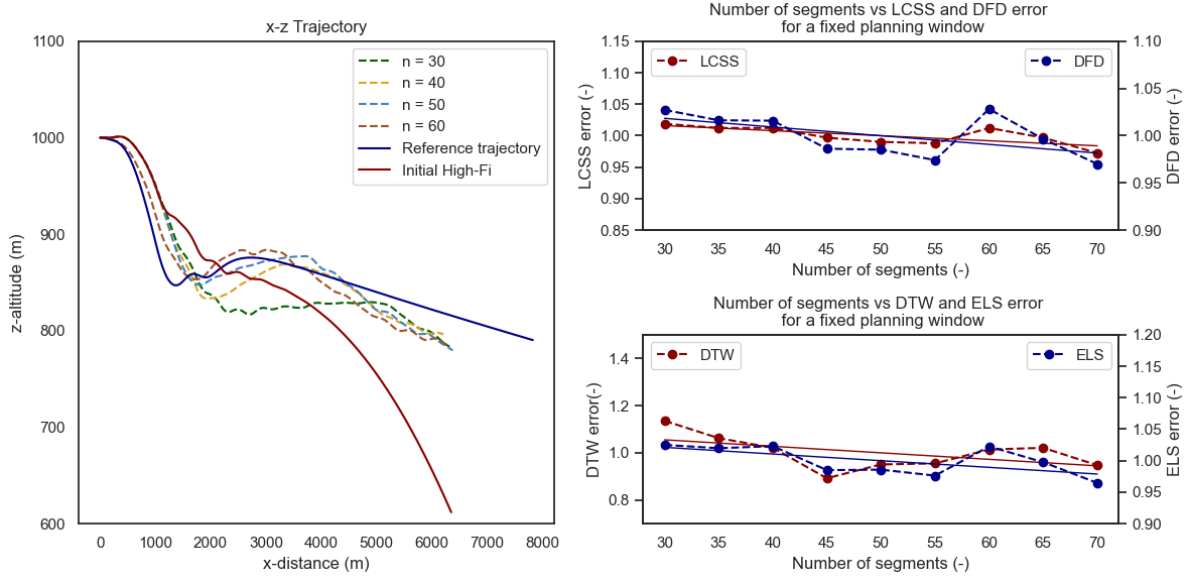


Figure 5.4: Effect of varying the number of segments for a fixed planning time of 3 seconds.

Table 5.4: Comparison between the different plot regression lines statistical values for the relation between number of segments and error for test case 3.

| Error Type | Average value | R^2 | p-value |
|-----------------------------|---------------|-------|---------|
| DTW error (1e3 m) | 107.8 | 0.271 | 0.151 |
| DFD error (leash length)(m) | 1293.7 | 0.313 | 0.117 |
| LCSS (-) | 0.54 | 0.529 | 0.026 |
| ELS (1e3 m) | 385.8 | 0.391 | 0.072 |

When looking at the third test case, a stark contrast can be drawn compared to the previous test cases. Due to the increased complexity of the control inputs and the resulting trajectory, increasing the amount of segments has a slightly positive effect on reducing the error. However, it must be noted that the amount of error reduction percentage wise is small as indicated by the y-axis. Compared to the first test case, there are no large variations in error noted for the different number of segments. Looking at the statistical metrics, the combination of high R^2 values combined with a low p-value suggests that in this case there is a relation between the number of segments. This relation however is relatively weak as can be seen in the plot. At most, the error drops a few percentages, and due to random variability, the same situation as in test-case 1 arises where multiple numbers of segments are able to provide good results. A similar observation is made for test case 4 which can be found in section C.2.

Summarising the main findings above reveals that overall the MPC method is quite able to optimise the high-fidelity model, however it is not quite able to deal with situations where large amounts of anticipation is required such as in test case 1. Furthermore, no significant relation exists between number of segments and optimiser performance, tuning the amount of segments is therefor more a case of trail and error instead of an informed relation. This could be explained by the fact that using more but shorter

segments leads to more incremental updates, but each update is fuelled by an predictor model that is fitted to a shorter sequence which means an inferior fit. However because the segment is also short, the impact of a bad fit is mitigated as only a small portion of the generated controls are executed. On the other hand, using fewer but longer segments yields a better fit, however now more of the generated control sequence is executed by the high-fidelity model. As the predictor model loses accuracy for long sequence, this partially cancels out the increased accuracy of the initial fit. All in all these effects seem to cancel out over the duration of the total sequence leading to a roughly equal fit.

5.2. Planning horizon

Previous analysis has focused on a constant planning horizon or fixed ratio for varying amount of segments. This part will dive into the effect of varying the planning horizon duration.

Test case 3

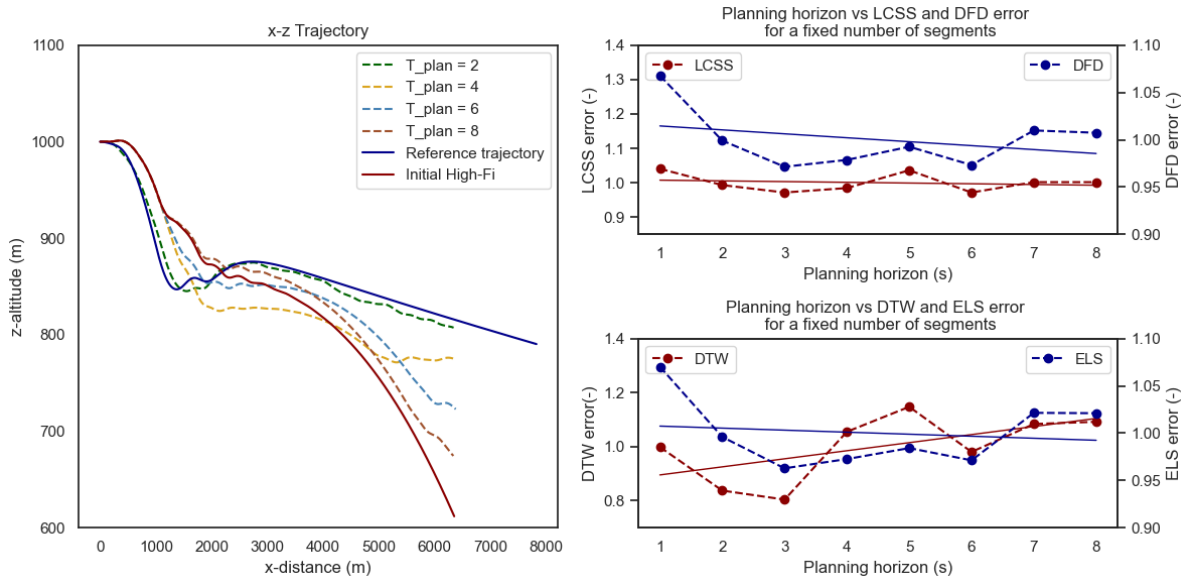


Figure 5.5: Effect of varying planning horizon duration for a fixed number of 45 segments and a manoeuvre duration of 30 seconds.

Table 5.5: Comparison between the different plot regression lines statistical values for the relation between planning horizon and error for test case 3

| Error Type | Average value | R^2 | p-value |
|-----------------------|---------------|-------|---------|
| DTW (1e3 m) | 119.7 | 0.355 | 0.1191 |
| DFD (leash length)(m) | 1313.5 | 0.107 | 0.4282 |
| LCSS (-) | 0.524 | 0.038 | 0.6427 |
| ELS (1e3 m) | 394.7 | 0.021 | 0.7307 |

Surprisingly, increasing the planning horizon yields to an overall worse fit. This can partially be explained by how the MPC optimiser functions. If it is not able to find an improved set of controls, it sticks to the starting controls, which explains that the optimised trajectory initially does not deviate from the initial high-fidelity trajectory. Additionally, the increased planning horizon might actually be a detrimental factor because of the fact that the predictor model loses accuracy over the course of the planning segment. Even if the model is refitted for each segment. This means that the newly found controls for long planning horizons are optimised for a model that in actuality does not capture the

high-fidelity. By prioritising long term planning by increasing the planning horizon, the whole control sequence suffers.

In general, the optimiser is able to generate the best results for a modest planning horizon, where moving beyond a certain point yields worse results which resembles test case 2 found in the appendix. Looking at Table 5.5, no clear trend can be established, except for the DTW error with a high data fit and high statistical significance (low P-value). The trend line for the LCSS error remaining constant can be explained by the fact that all trajectories are within the spatial threshold.

Test case 4

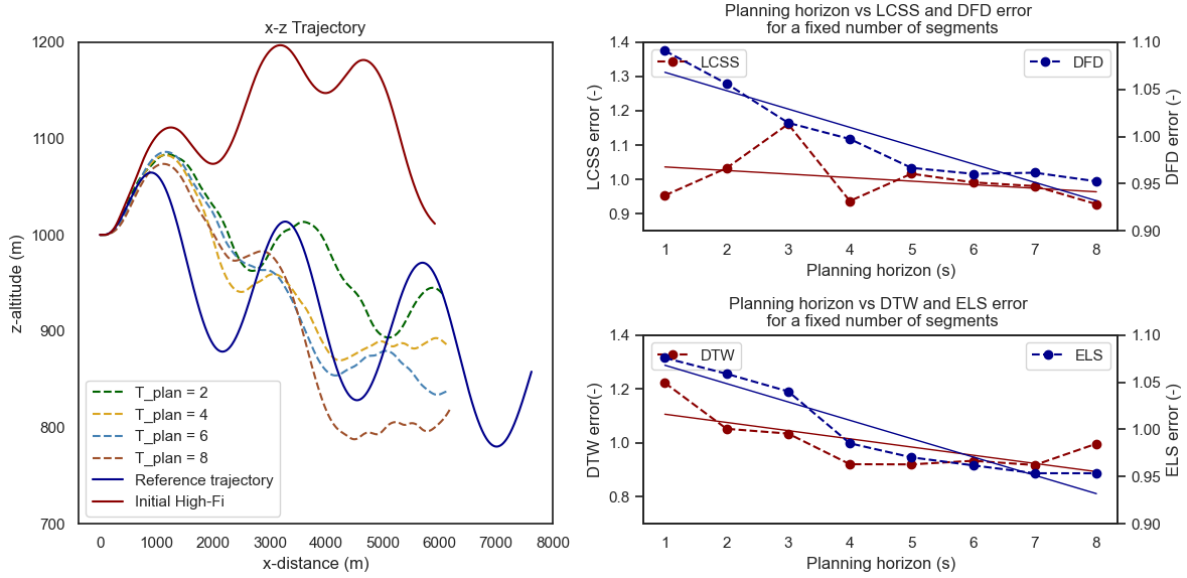


Figure 5.6: Effect of varying planning horizon duration for a fixed number of 60 segments and a manoeuvre duration of 30 seconds.

Table 5.6: Comparison between the different plot regression lines statistical values for the relation between planning horizon and error test case 4

| Error Type | Average value | R^2 | p-value |
|-----------------------------|---------------|-------|---------|
| DTW error (1e3 m) | 146.4 | 0.503 | 0.0488 |
| DFD error (leash length)(m) | 1322.7 | 0.882 | 0.0005 |
| LCSS (-) | 0.66 | 0.115 | 0.411 |
| ELS (1e3 m) | 400.7 | 0.896 | 0.0004 |

Regarding the last test case which was designed to test anticipatory performance, increasing the planning horizon seems to have a favourable effect on model fit. Especially the LCSS and DTW error seem to decrease the most, suggesting a better form fit for a higher planning horizon. This is also confirmed by the statistical relations with a high model fit parameter R^2 and a very low p-value. The trajectory for $T = 6$ s represents the point where further increases in planning horizons yields no improved results. Beyond this point the predictor model is not accurate enough to generate useful results. Another observation is that the longer planning horizon trajectories seem to converge to an average of the reference, and are less willing to locally adapt to altitude variations. For a mix of short term fit and long term anticipation, a sweet-spot around $T = 2$ s and $T = 4$ s is found.

Summarising the main results in the sensitivity study above reveals that the MPC method is a solid approach for control optimisation of high-fidelity models. Statistical analysis reveals that planning hori-

zon duration has a significant impact in mapping performance especially when anticipatory behaviour is required. Low planning horizons are favourable in most situations, whereas in cases where anticipation is required, a longer planning horizon duration is better. Again it is crucial to critically review the problem on a case-by-case basis as there is no clear cut relation.

6

Sequence-to-Sequence Network Based Space mapping

The purpose of this section is to investigate the key dynamics to integrating machine learning with space mapping techniques. The analysis is organised around the sub-questions outlined in section 3.1 to help guide answering the main research question. The analysis start with sub-question a, focusing on determining which type of network is most suitable for hybrid space mapping-machine learning implementations. This section introduces results for each test case, setting the stage for detailed comparisons. Following the comparison of different network types, the study proceeds to sub-question b, exploring the extent to which active learning affects the optimisation results. This examination is crucial for understanding how iterative retraining impacts the model's adaptability and accuracy, thereby informing the practical deployment of the hybrid modelling approach. With a thorough understanding of both the optimal network architecture and the efficacy of active learning, the machine learning based implementation can be compared to the MPC based method, thereby answering sub-question c.

6.1. Comparing network types

Starting with the comparison between different network types, a few performance metrics are important to keep in mind. As discussed in section 4.3, due to the extremely small sample size the machine learning implementation is vulnerable for random variations between different pseudo-random seeds. In order to effectively gauge the performance of a network it is therefor key to consider the average prediction, as well as the spread in prediction for different random seeds. Combining these two parameters tell a lot about a networks ability to generate correct results through the average, and consistency through training. This is reminiscent of the difference between precision and accuracy, adapted from ¹:

¹<https://wp.stolaf.edu/it/gis-precision-accuracy/>, accessed 04/07/2024

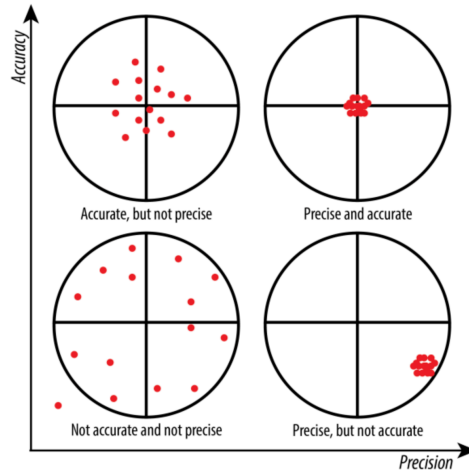


Figure 6.1: Difference between accuracy and precision.

In a similar way, a network can on average generate accurate estimations, but if they vary between random seeds this still is not very useful. In the coming sections, precision will be visualised as a 95% confidence interval around a certain mean. Relating this to the graph above, the mean error will represent accuracy, whereas the magnitude of the variance will represent precision, where a low variance implies high precision.

Network settings

In preparation for the upcoming sensitivity studies, specific network settings were established to ensure stable and repeatable results. The primary goal in configuring the network was not to optimise it for peak performance in trajectory mapping, but rather to establish a consistent baseline suitable for investigating the integration of machine learning with space mapping techniques. Consequently, this approach has dictated the selection of a fixed network configuration that supports reliable experimentation. This rationale also underpins the decision to forego a hyper-parameter tuning study, which, while common in machine learning applications aimed at maximising model performance, would not align with the exploratory and demonstrative nature of this project. The chosen configuration allows the focus to be put on demonstrating the potential and challenges of integrating machine learning within the space mapping framework without the variability introduced by extensive parameter optimisation:

Table 6.1: Overview of network parameters used during this research

| Network parameter | GRU | Transformer |
|--------------------------------------|-----|-------------|
| Initial training epochs | | 400 |
| Retraining epochs | | 80 |
| Number of control points | | 30 |
| Hidden dimension size | | 128 |
| number of layers | | 3 |
| dropout ratio | | 0.05 |
| number of initial samples | | 10 |
| Maximum number of optimisation steps | | 20 |
| Sample perturbation | | 0.15 |
| number of attention heads | - | 8 |
| Trajectory error calculation method | | DTW |
| Network learning rate | | 1.0e-03 |

Both the GRU and transformer networks are configured with the same number of layers and hidden

dimensions. However, the transformer network incorporates additional complexity with approximately 50% more trainable parameters, totalling around 595 thousand compared to 295 thousand for the GRU model. This increase is characteristic of transformer architectures, which inherently require more parameters due to the implementation of attention mechanisms. Unlike the GRU, the transformer also utilises positional encoding, which introduces an additional dimension to the input. It's important to note that positional encoding does not contribute to the count of trainable parameters, as it provides a fixed, non-trainable input addition. Furthermore, for the error determination method, DTW was chosen as it provides a good balance between robustness and being able to allow for some spatial and temporal differences.

Test case 1

Starting with the first test case, the optimised controls and resulting trajectory are presented for varying random seeds:

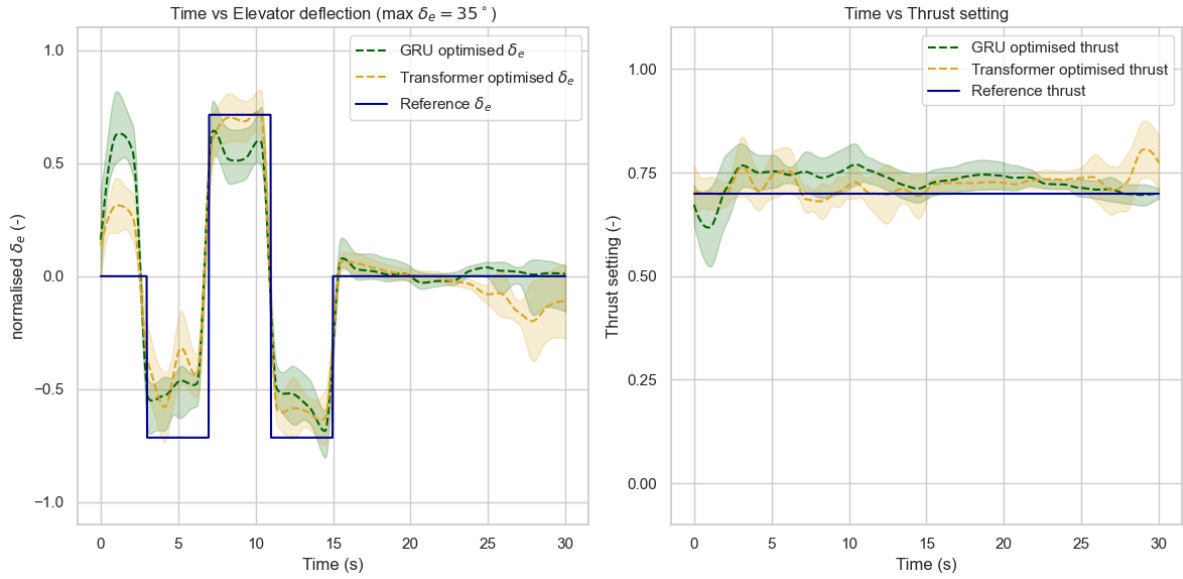


Figure 6.2: Machine learning optimised controls for test case 1. The controls optimised using a GRU network have a variance (expressed as a percentage of the maximum control value) of 18.5% and 5.2% for the elevator and thrust setting respectively. For the transformer network the variance is 19.5% and 5.4% respectively.

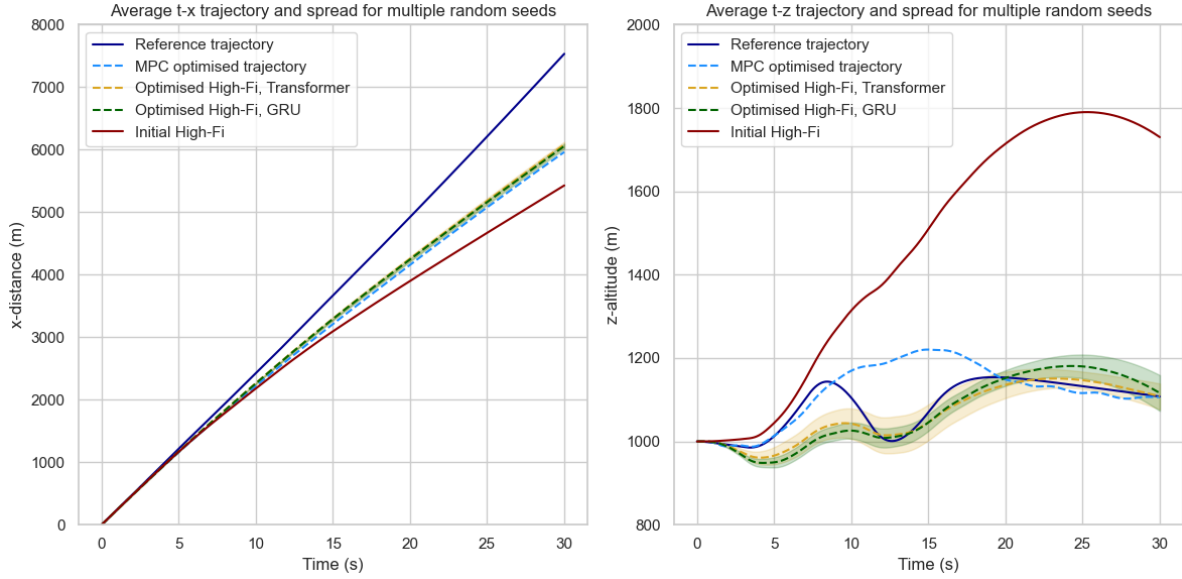


Figure 6.3: Machine learning optimised high-fidelity trajectory for test case 1. For the GRU network, the x and z variance is 15.9m and 33.9m respectively. For the transformer network the variance is 28.8m and 40.0m.

For the results above, the time history for both the x and z parameter are shown instead of the flight path (or x - z path). This has been chosen to facilitate the visualisation of prediction spread. Taking in the results above, the optimised high-fidelity trajectories are very similar in overall shape. This similarity implies that on average both networks perform equally well, and that the control optimiser does not appear to favour any one of the network's prediction in particular. Where some differences can be noticed is in the average variance in high-fidelity. The GRU-optimised trajectories yielded a lower variance meaning more prediction stability. This average is however not reflected in the controls, where both networks have equal variance.

Table 6.2: Optimised high-fidelity trajectory error metrics for test case 1. For the LCSS error, a spatial threshold of 100 meter, and a temporal threshold of 30 steps which is equivalent to 1 second were used.

| (a) GRU network | | | (b) Transformer network | | |
|------------------------|---------------------|--------------|-------------------------|---------------------|--------------|
| Error Type | Mean error μ | STD σ | Error Type | Mean error μ | STD σ |
| DTW (1e3 m) | 163.4 | 20.3 | DTW (1e3 m) | 160.1 | 16.9 |
| DFD (leash-length) (m) | 1477.9 | 58.5 | DFD (leash-length) (m) | 1451.5 | 63.2 |
| LCSS (-) | 0.642 | 0.064 | LCSS (-) | 0.64 | 0.057 |
| ELS (m) | 462.2 | 15.6 | ELS (m) | 453.0 | 28.4 |

Looking at the calculated error metrics in Table 6.2, both the results and variance is consistent between the transformer and gru network. Comparing the results to Table C.1 however, similar values can be seen, even though the underlying trajectories are not similar. This again highlights the short coming of the error calculation method and the complexity of the situation. Interpreting the trajectories on a more qualitative base, a noticeable aspect is that the MPC method where it is able to tends to follow the trajectory more closely. The machine learning approach however fits the general trajectory better. A point of attention again is in this scenario, the x -distance contributes the most to the error. The maximum deviation in z -direction is roughly in the neighbourhood of 200 meters, whereas the maximum z -distance is about 1450m. This undoubtedly has a large impact on the control optimiser.

Where the results above discuss the resulting high-fidelity trajectory using the different networks, another aspect of assessing network performance is how well the networks themselves perform in their role of approximating the high-fidelity model. This is done by looking at the average prediction error and error variance for the optimised results:

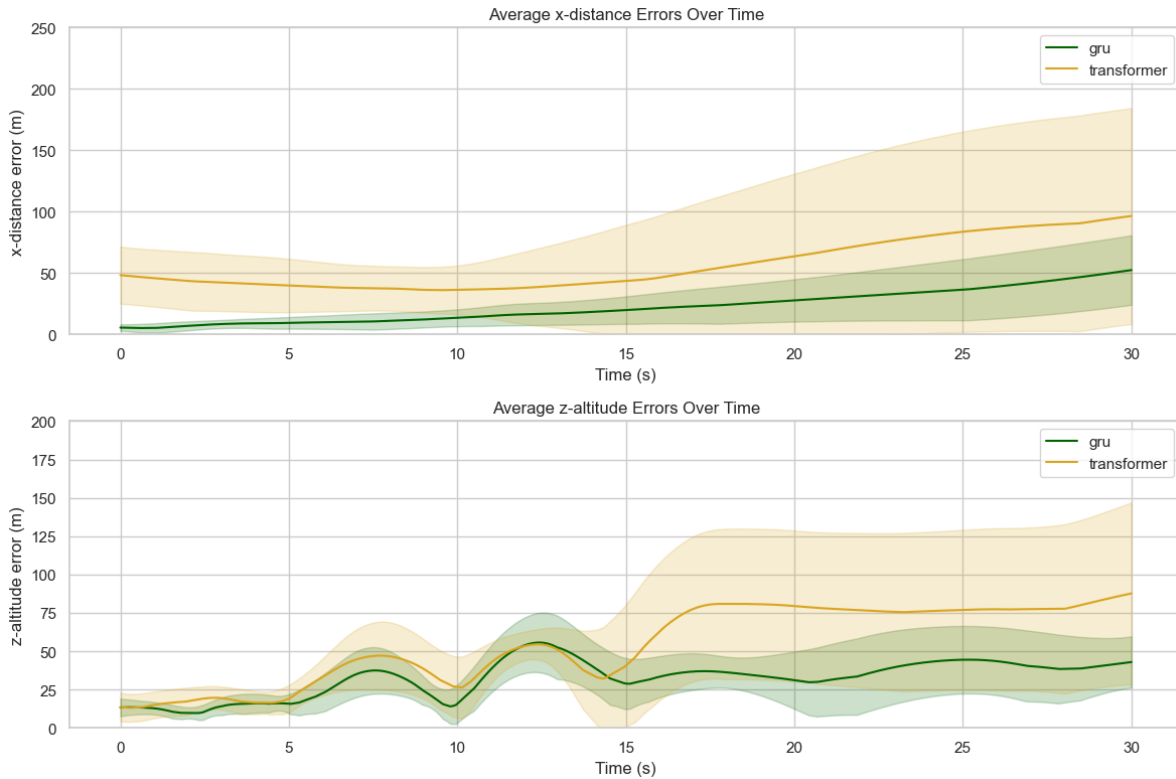


Figure 6.4: Average prediction error and error variance of the GRU and transformer network for test case 1. The average GRU prediction variance is 19.5m and 22.9m for x and z dimension respectively. The average transformer network variance is 70.3m and 47.5m.

The figure above tells a much different story compared to the previous figures. Here it can clearly be seen that the GRU network produces a much more consistent prediction and overall is more precise and more accurate. This superior consistency likely explain the lower variance in the final optimisation results. Another notable observation is that the GRU network steadily loses accuracy and precision over the course of the prediction. This resonates with one of the weaknesses of recurrent neural networks, where the hidden state memory is depleted too much for long prediction sequences. Furthermore, for both networks, there seems to be an increase in predictive error after every major change in controls, suggesting that the networks have a hard time accurately processing the immediate effects of control inputs, and that only after a short bit that this is compensated.

Test case 3

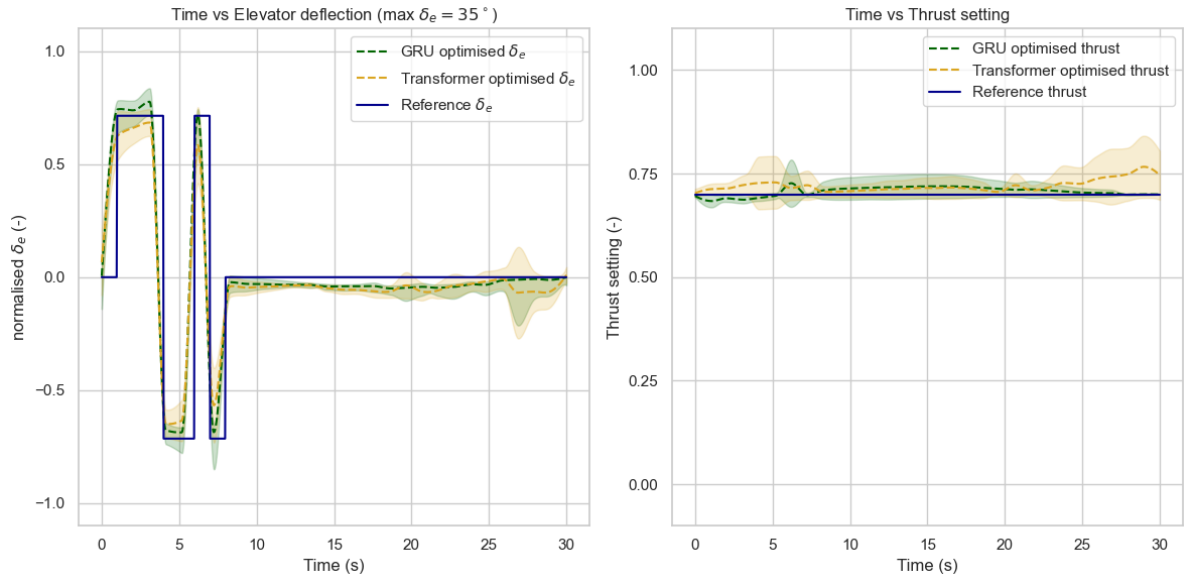


Figure 6.5: Machine learning optimised controls for test case 3. The controls optimised using a GRU network have a variance (expressed as a percentage of the maximum control value) of 2.75% and 2.62% for the elevator and thrust setting respectively. For the transformer network the variance is 7.77% and 4.52% respectively.

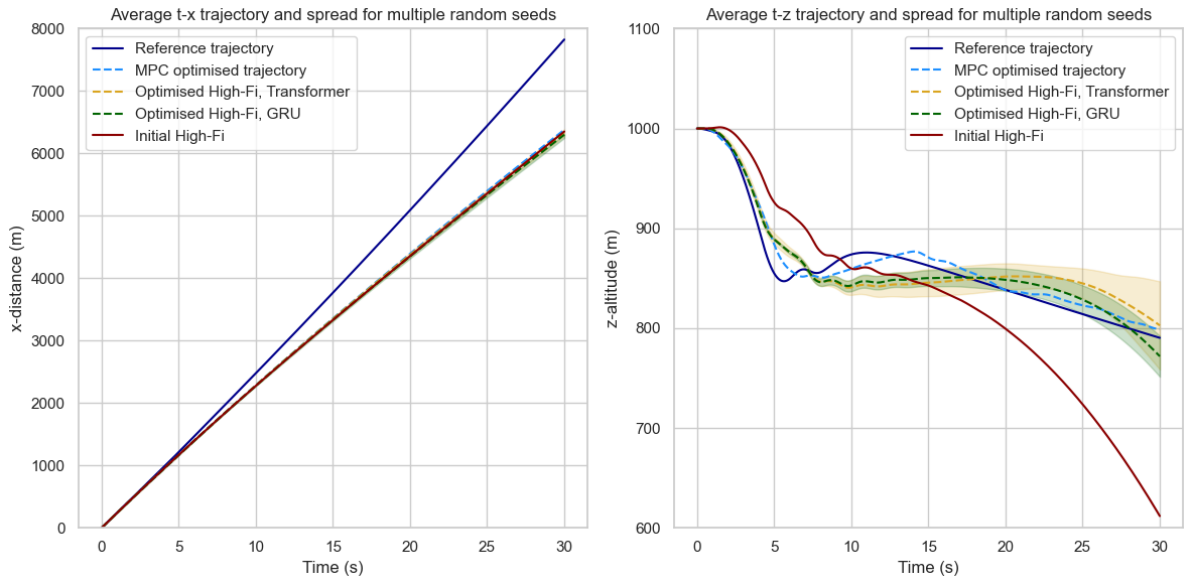


Figure 6.6: Machine learning optimised high-fidelity trajectory for test case 3. For the GRU network, the x and z variance is 4.79m and 32.2m respectively. For the transformer network the variance is 12.1m and 17.5m.

Table 6.3: Optimised high-fidelity trajectory error metrics for test case 3.

| (a) GRU network | | | (b) Transformer network | | |
|------------------------|---------------------|--------------|-------------------------|---------------------|--------------|
| Error Type | Mean error μ | STD σ | Error Type | Mean error μ | STD σ |
| DTW (1e3 m) | 143.2 | 3.8 | DTW (1e3 m) | 142.0 | 11.0 |
| DFD (leash-length) (m) | 1525.0 | 13.5 | DFD (leash-length) (m) | 1495.4 | 76.7 |
| LCSS (-) | 0.585 | 0.001 | LCSS (-) | 0.575 | 0.018 |
| ELS (m) | 490.5 | 3.8 | ELS (m) | 475.7 | 28.5 |

For the third case, a different scenario can be seen. This test case features a more complex initial control sequence and resulting trajectory. Surprisingly, the transformer network results in superior mapping performance, both in terms of precision and average. This highlights one of the strengths of transformer networks. Because they are able to consider the whole sequence at once, they are able to learn better from complex repeating patterns. Looking at the reference MPC case, it can be seen that this trajectory actually fits the general shape better, capturing more of the stepped decreases in altitude compared to the machine learning models. This becomes even more apparent when the parameters are properly tuned as can be seen in Figure 5.5 where the MPC optimiser was able to steer the high-fidelity model almost perfectly along the reference trajectory

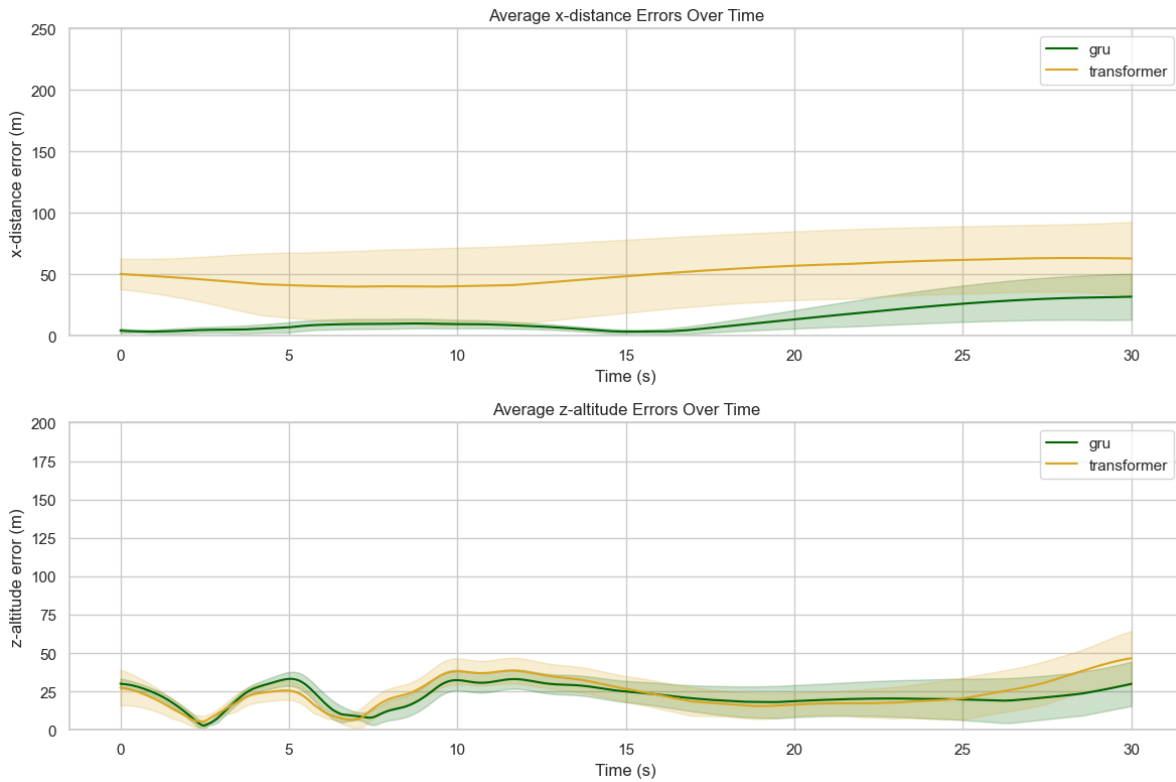


Figure 6.7: Average prediction error and error variance of the GRU and transformer network for test case 3. The average GRU prediction variance is 9.96m and 11.9m for x and z dimension respectively. The average transformer network variance is 39.3m and 12.7m.

Looking at the graph above, the difference in how sequence data is processed is highlighted. The GRU network quickly loses prediction precision over the course of the manoeuvre, suggesting that it is less capable in dealing with complex trajectories. The transformer network actually beats the GRU network,

especially near the end of the trajectory. Notable is that the average error and variance stays relatively constant in magnitude for the different test cases covered up until now. This suggests that trajectory complexity is less of a factor in its predictive performance, and that other factors such as data set size and training iterations are more dominant. Conversely, the GRU network seems to be far better in more constant trajectories.

Summarising this segment of the sensitivity study reveals that the machine learning method is capable of optimising high-fidelity controls and trajectories to a reasonably satisfactory extent. However, it often underperforms compared to the MPC-based method when the latter is appropriately configured. Notably, the machine learning approach excels in scenarios requiring anticipatory actions, where it tends to outperform the MPC-based method.

In terms of network performance, the GRU network generally delivers better results in scenarios characterised by infrequent control changes. Under these conditions, both the average prediction error and the variance in prediction at each time step are significantly lower for the GRU network than for the Transformer network. This stability in the GRU's performance can be attributed to its design, which effectively captures dependencies in time-series data with minimal updates, maintaining more stable error profiles in scenarios with less dynamic control demands.

However, the GRU network's prediction accuracy deteriorates over time, particularly in scenarios involving complex control actions, as observed in test case 3. For the Transformer network, errors are more uniformly distributed throughout the sequence, but both the average error and variability are higher. This increased spread in variance for the Transformer network stems from its ability to attend to any part of the input sequence equally, which can introduce greater overall variability in more complex or varied control sequences, leading to less consistent optimisation outcomes.

6.2. Influence of active-learning & retraining

This section delves into the third research question, which seeks to understand how active learning can be used to enhance the predictive accuracy and adaptability of machine learning networks within hybrid space mapping systems. The investigation into active learning is driven by the necessity to reduce the amount of data, while maintaining flexibility. In this context, the following section examines:

- **Predictive Accuracy:** How the number of retraining iterations influences the model's ability to predict new data points accurately, thereby reducing overall prediction errors as more data is incorporated into the training process. This will be measured as the average error between the high-fidelity trajectory and the predicted trajectory for the optimised controls.
- **Adaptability:** The capability of the model to adjust to new or evolving data scenarios, reflecting its robustness and flexibility to handle real-world operational dynamics. Here the average error and variance in error per iteration is key in uncovering how the network adapts as the optimisation progresses.

By comparison, the previous section focused mostly on the actual predicted physical trajectory, and the impact of the networks on how their prediction changes. This section is more about general predictive ability and how it changes over the course of the optimisation process and under varying number of retraining iterations. By exploring these aspects, this section aims to provide insights into the effectiveness of active learning strategies and their practical implications for machine learning networks in complex mapping tasks.

Test case 1

Starting again with the first test case, the networks predictive ability is shown for multiple retraining epochs(iterations):

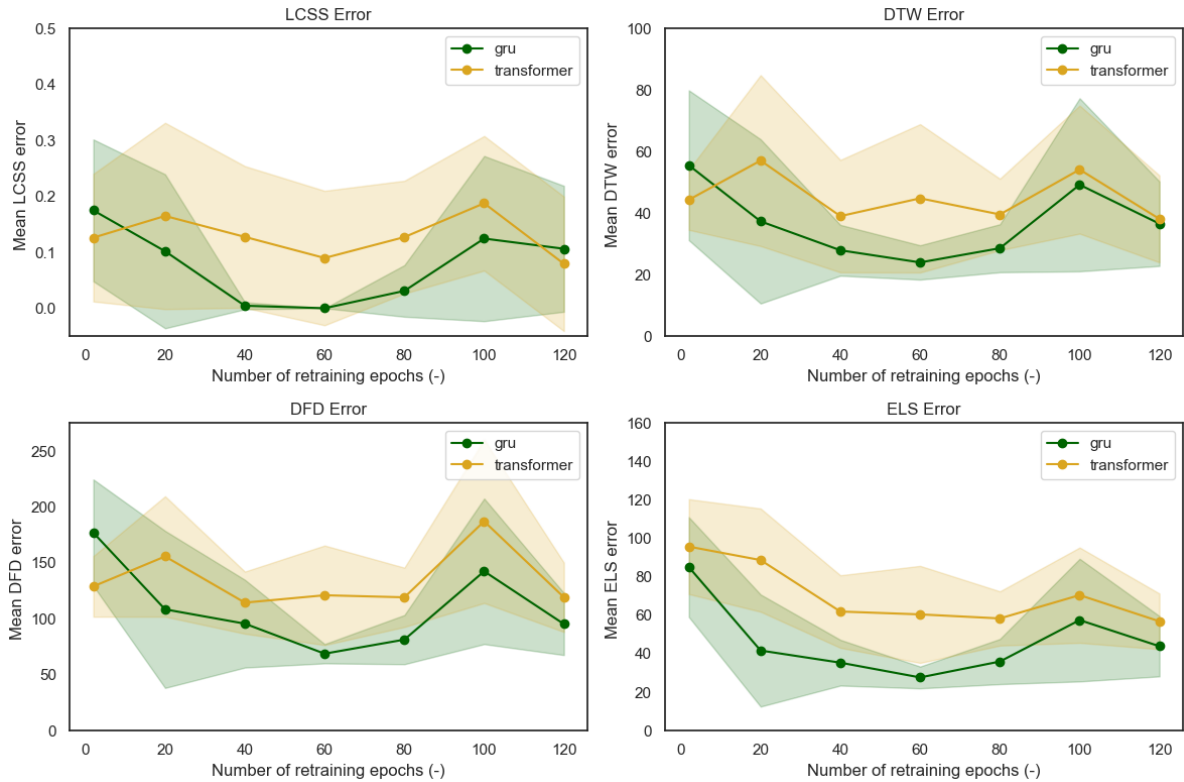


Figure 6.8: Influence of the number of retraining iterations on the predictive performance for the optimised controls for the first test case. The shaded region represent the 95% confidence interval for 9 different random seeds.

In the accompanying graph, the average error across various error metrics is displayed for varying retraining iterations, revealing a consistent trend: GRU networks generally outperform Transformer networks, particularly with a moderate number of retraining iterations. A critical observation from the analysis is the identification of an optimal range around 60 retraining iterations, where prediction accuracy and precision are maximised. This finding is crucial as it illustrates the balance between insufficient training, which results in underfitting, and excessive training, which leads to overfitting and subsequently poorer performance.

The key takeaway from this data is that the number of retraining iterations plays a significant role in the network's predictive performance concerning the optimised controls. This investigation was crucial to ensure that the optimised controls were not the result of random, inaccurate predictions. By demonstrating that the network can achieve consistent and accurate predictions with low variance, it substantiates the reliability of the optimisation results. It confirms that the optimiser effectively integrates the dynamics of the high-fidelity model into its calculations. Conversely, a high prediction error would suggest that the resulting trajectory might have occurred by chance, indicating reliance on an inadequate model approximation rather than a genuine understanding of the high-fidelity model's dynamics. Looking at the adaptive ability of the network next:

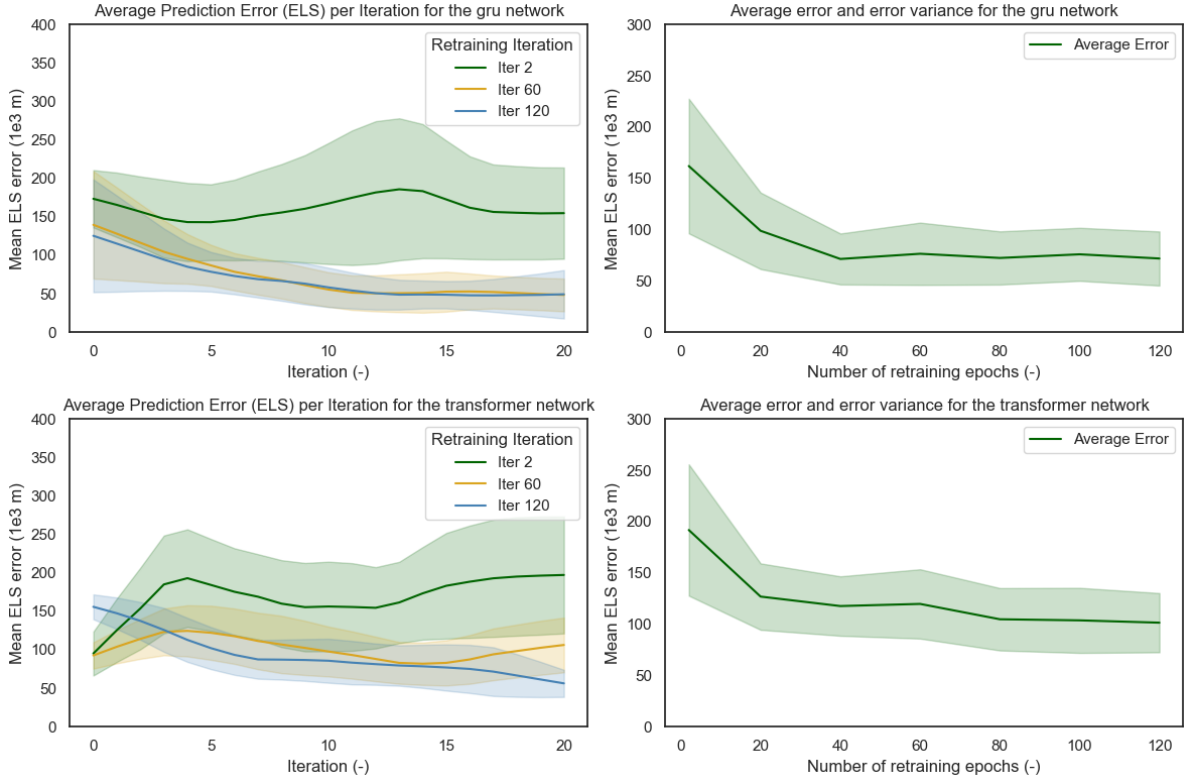


Figure 6.9: Average prediction error of the networks at every iteration for multiple retraining iterations for the first test case. The shaded region represent the 95% confidence interval for 9 different random seeds.

To accurately interpret the graph, it is important to understand its components. The left column displays the average prediction error at each iteration, indicating how effectively the network adapts to new controls generated by the optimiser. Note that here, only the ELS is displayed, as this error type represents the individual error at each time step the best. Initially, the error for $n = 60$ and $n = 120$ declines, suggesting that the network is actively learning and improving its ability to predict the control outcomes. As the error begins to level out after several iterations, it signals a transition from active learning to stabilisation, where the network is no longer improving significantly but continues to adapt and correct itself in response to new controls.

The right column presents the average error and variance over the whole optimisation process for different retraining iterations. This highlights the overall effect of the number of retraining iterations and acts as a clarification of the left graph. An initial decrease in error affirms that increasing the number of retraining iterations enhances predictive accuracy and precision. However, extending retraining beyond a certain point appears to yield diminishing returns on predictive performance.

The combination of data from Figure 6.8 and Figure 6.9 suggests that for this case, the optimum results are obtained with 60 retraining iterations. This number represents a balance between the frequency of retraining per optimisation step and the total number of steps taken until the optimiser achieves the lowest error value. It is crucial to acknowledge that the optimisation landscape is dynamic, continuously evolving as the network updates. This evolving landscape poses unique challenges, particularly in ensuring consistent optimisation decisions. The use of a steepest gradient descent approach, which bases its decisions solely on the current state of the optimisation landscape, helps navigate these challenges by adapting to the immediate conditions without basing its predictions on outdated or inaccurate data.

Test case 4

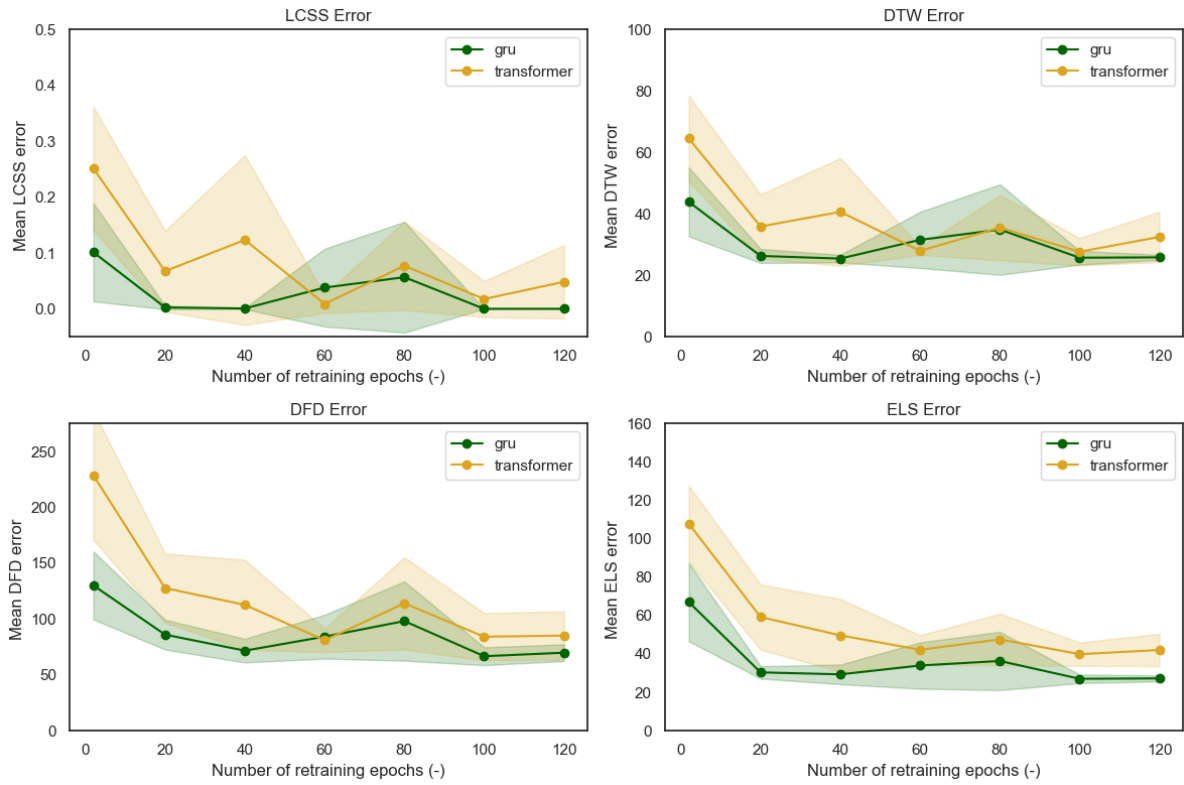


Figure 6.10: Influence of the number of retraining iterations on the predictive performance for the optimised controls for the fourth test case. The shaded region represent the 95% confidence interval for 9 different random seeds.

The final test case highlight some of the complexity involving the decision on how many retraining iterations to use. While initially the graph looks similar in shape compared to Figure 6.8, the average error and variance decreases again, implying that multiple configurations are possible. As the displayed results are an average of multiple runs, this rules out any one-off results. All in all, the results above again highlight the out performance of GRU models in terms of average prediction and variance.

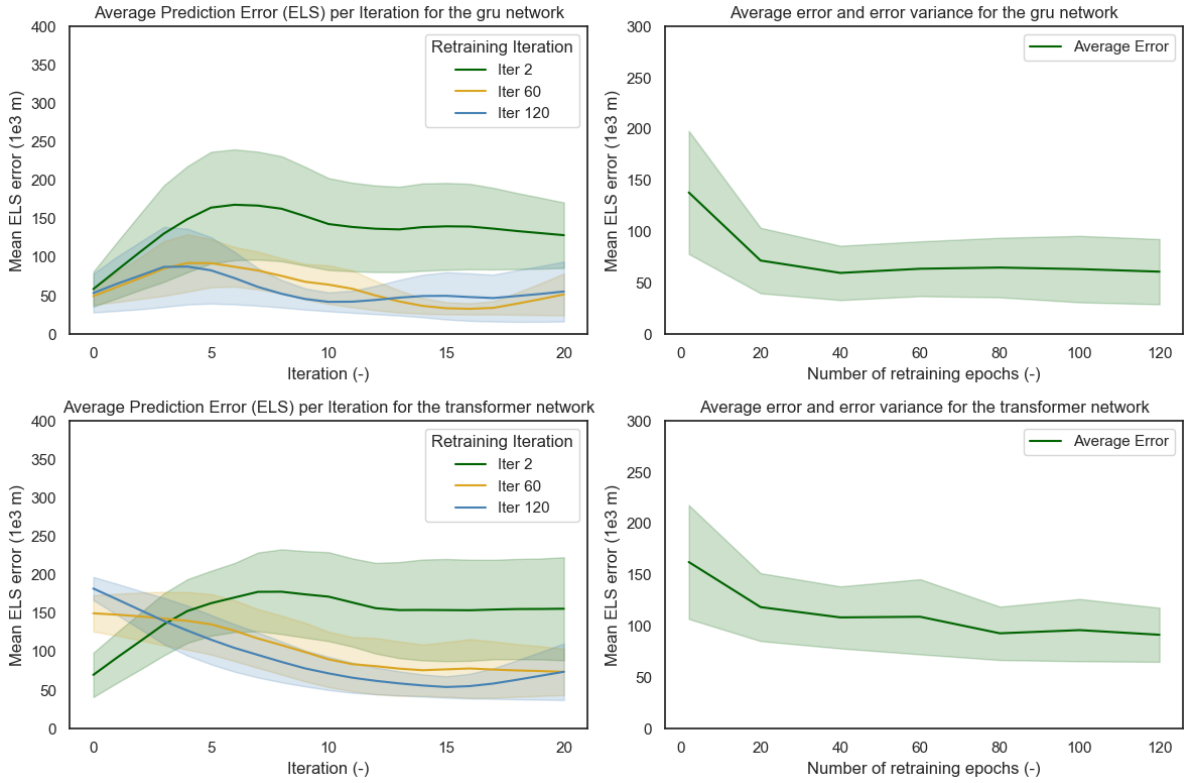


Figure 6.11: Average prediction error of the networks at every iteration for multiple retraining iterations for the fourth test case. The shaded region represent the 95% confidence interval for 9 different random seeds.

Looking at the prediction error evolution over the course of the optimisation process it is clear that retraining is absolutely essential in achieving consistent performance. Recalling the initial high-fidelity state (Figure 3.5), it can be seen that the initial state is quite different from the reference, most importantly they are out of phase near the end of the control sequence. This requires a significant change in controls to compensate, which falls far outside of the initial sample region the model was initially trained on. Adapting to this change is crucial, which is why the higher number of retraining iterations manage to score higher than almost no training.

Summarising this section; the results presented underscore the significance of active learning in enhancing the networks' ability to adapt to new data points and control adjustments. Active learning is pivotal, as evidenced by the increase in predictive error and error variance observed when networks operate without ongoing training.

Notably, the optimal performance for both networks is typically achieved with a medium number of retraining iterations, although this is influenced by the specific case and the dynamics of the optimisation process. The GRU networks generally outperform their transformer counterparts, which display a higher sensitivity to the volume of training. This sensitivity in transformers necessitates more extensive training to reach comparable levels of accuracy.

In practice, about 60 retraining iterations tend to yield the best results across various test scenarios. However, this number is not a one-size-fits-all solution as can be seen in Figure D.10; it is dependent upon the specific requirements and the evolving nature of the optimisation process. This highlights the nuanced relationship between training frequency, network architecture, and control optimiser, which collectively dictate the networks' predictive ability.

7

Results Analysis & Discussion

This section will review the primary outcomes of the sensitivity analysis to address the main research question. It begins with a concise summary of the critical findings. Following this, the results will be revisited to address the sub-questions systematically, progressively building towards the overarching research question. The section will conclude with a comprehensive reflection on the integration of space mapping techniques and a discussion on the limitations encountered during this research.

7.1. Results summary

In this study, two space mapping implementations were studied; a MPC based method, and a machine learning based method using active learning. These methods represent two fundamental approaches to mapping trajectories. These are sequential control optimisation of segments, and control optimisation of the whole sequence at once. Naturally, this has large implication on the resulting controls. The results presented in the previous chapters are summarised below:

MPC method

The analysis of the MPC method across four test cases demonstrates that, overall, MPC is well-suited for control optimisation. However, it encounters limitations in scenarios requiring anticipation beyond the set planning horizon, often resulting in trajectory overshoots, particularly in the first and last cases examined. The influence of the number of segments, which impacts the low-fidelity predictor model's accuracy, does not exhibit a consistent relationship with overall mapping performance. However, customising the model's configuration on a case-by-case basis significantly enhances its effectiveness.

The duration of the planning horizon plays a crucial role, having a pronounced impact on mapping performance. Statistical analysis indicates that while shorter planning horizons generally yield favourable results, extended horizons are necessary in scenarios that demand anticipatory adjustments. This adjustment is crucial for accommodating future trajectory demands, improving the predictability and accuracy of the mapped trajectory.

In essence, MPC proves to be a robust mapping tool when finely tuned for specific scenarios. However, it may fall short in situations that require comprehensive manoeuvre anticipation due to its focus on optimising immediate performance rather than the entire manoeuvre. To fully assess the effectiveness of MPC, a combination of qualitative visual analysis and quantitative methods is indispensable. Different error methodologies reveal various facets of the trajectory discrepancies, and only a combined approach that integrates both visual and statistical analyses can provide a complete evaluation of the model's performance.

Machine learning method

The sensitivity analysis performed in chapter 6 highlights the strengths and limitations of the machine learning-based space mapping approach compared to the more traditional MPC-based method. While the machine learning approach, particularly when utilising GRU networks, generally delivers satisfactory results in optimising high-fidelity controls and trajectories, it often falls short of the performance achieved by a well-tuned MPC method. This discrepancy is most apparent in scenarios that do not demand anticipatory actions, where the MPC method's approach of locally tailor solutions are more pronounced.

In scenarios requiring foresight and complex anticipatory actions however, the machine learning method often surpasses the MPC approach. This advantage is attributed to the machine learning method's ability to process entire sequences simultaneously, allowing for a more complete view of the trajectory, which is crucial in anticipating future states.

The GRU network, in particular, excels in more gradually changing environments with infrequent control changes, showcasing lower average prediction errors and reduced variance at each time step. This stability is due to the GRU's design, which uses its recurrency to capture time-dependent dynamics with fewer parameter updates, thereby maintaining a more consistent error profile under stable control conditions.

Conversely, in more dynamic scenarios with complex control actions, such as those presented in test case 3, the GRU network's performance begins to decrease, with a noticeable deterioration in prediction accuracy over time. The Transformer network, while demonstrating a higher overall error and variability, maintains a more uniform error distribution across the sequence and for different use cases. This behaviour stems from the Transformer's architecture, which, through its positional encoding mechanism, considers all parts of the input sequence equally, thereby introducing more variability but potentially enhancing the model's responsiveness to changes in the control sequence.

Overall, both networks perform the best with a medium level of retraining iterations, with around 60 iterations proving to be effective in most cases. However, the exact number of iterations for optimal performance can vary depending on the specific demands and complexity of the optimisation case, as well as how the optimisation process evolves over time.

These findings highlight the importance of model configuration and the inherent trade-offs between different types of network architectures in handling the complexities of trajectory optimisation.

7.2. Answering the research question

The ultimate goal of any analysis is to provide a complete answer to the main research question. This section will systematically address each research question, leading up to the main research question. Starting with the sub question.

Sub-question a

Which type of network is best suited for a hybrid space mapping-machine learning implementation?

This sub-question can be answered by looking at the results from section 6.1 and section 6.2. The general trend across all results is that GRU networks on average are a better fit for the high-fidelity network, and experience lower variance in prediction. This is crucial in reducing the dependency on a lucky random seed for good optimiser results. To add nuance to this answer, transformer networks, while experiencing a slightly higher average and average variance are less impacted by adding complexity to the input sequence. In cases where there is significant control input with a high frequency, transformer networks achieve an error on par with GRU networks. The reason for this difference in performance can partially be attributed to the almost double amount of trainable parameters for the transformer network compared to the GRU network, even if the same amount of layers and nodes are

selected. In more traditional approaches this is compensated by increasing the amount of training data and training iterations, something that is not feasible in this application.

Sub-question b

How does active learning influence the predictive accuracy and adaptability of machine learning networks in hybrid space mapping implementations?

Based on the results from section 6.2, active learning plays a crucial role in ensuring that models are able to adapt to new control sequences generated by the control optimiser. Especially comparing (almost) no retraining to even a medium amount of retraining yields significant improvements in network accuracy and precision over time. Furthermore, active learning is especially influential for transformer networks. This can be explained again by the almost double amount of trainable parameters, which benefit more from adjustments. On the flip side, with low amount of retraining, the GRU networks stand out.

Sub-question c

How does hybrid space mapping-machine learning compare to existing trajectory mapping techniques such as MMSA in terms of computational cost and mapping error

Already briefly mentioned in the results summary above, but reviewing the results in section 5.2 and section 6.1 reveals that the MPC based method generally outperforms the machine-learning method. Both in terms of mapping accuracy, as well in the number of iterations required. However, in situations where anticipation is required such as test case 4, the machine learning approach performed on par with the MPC based method. Furthermore, MPC methods overall achieve a better fit on the short term, whereas the machine learning optimised trajectories follow the general trend of the reference trajectory. Moving on to the main research question:

How does the integration of space mapping and sequence-to-sequence networks facilitate the mapping of optimal control sequences between low and high-fidelity flight mechanic models to minimise trajectory differences in non-linear flight regimes?

The integration of space mapping and sequence-to-sequence neural networks offers a promising yet nuanced approach to mapping optimal control sequences compared to traditional MPC-based methods. While MPC excels in achieving short to medium-term accuracy with fewer iterations, it struggles in scenarios requiring anticipation of future states—conditions where machine learning approaches can provide comparable or superior results.

Active learning is crucial within the optimisation architecture, significantly enhancing the integration of machine learning with space mapping by reducing the initial data requirement. The analysis shows that active learning not only boosts the predictive accuracy of networks but also enables them to adapt continuously to new data generated by control optimisers. This adaptability is particularly advantageous in complex scenarios characterised by frequent changes in control inputs, allowing machine learning models, especially transformer networks, to adjust dynamically and maintain robust performance. Their ability to sustain low error rates and variance under evolving conditions underscores the potential of machine learning-based mapping with minimal datasets.

In conclusion, integrating space mapping with sequence-to-sequence neural networks presents substantial potential to enhance the mapping of optimal control sequences, leveraging machine learning's strengths to manage complex and dynamic control scenarios more effectively than traditional MPC methods. However, this approach's reliance on extremely small datasets introduces vulnerabilities to

random variability between training sessions, impacting the reliability and consistency of the mapping results.

7.3. Discussion

This section evaluates the integration of machine learning with space mapping against established conditions and explores the inherent limitations of the methodologies used. This discussion aims to provide context findings of the previous section with respect to the objective of this study.

7.3.1. Integration conditions

In section 3.2, a set of conditions for defining a successful hybrid mapping algorithm was defined. Reviewing each conditions reveals that on all fronts, except for the flexible adaptation to various scenarios, the current implementation has not succeeded. In this section, these conditions will be briefly discussed, as well as some of the limitations of this research and their implications.

- **Accuracy improvement:** The integrated method, while showing potential in scenarios requiring anticipatory actions, generally under performs in accuracy compared to the MPC method. The MPC's precise tuning of control sequences for immediate future states typically results in more accurate trajectory mapping. It's important to note that this research is structured as a comparative study focusing not on peak performance but on demonstrating differences in approach. In scenarios involving more complex or longer sequences, the disparity in accuracy might diminish as the strengths of the machine learning method become more pronounced.
- **Data efficiency:** Another nuanced area is data efficiency. Due to the design of this study, an extensive analysis of the optimiser or exploration of different optimisers was not conducted. In complex optimisation problems, the architecture of the optimiser can significantly influence convergence rates. Additionally, the dual function of control optimisation and data sampling complicates data efficiency. If an optimiser selects a sub-optimal next point, the machine learning model must still use this data for updates, potentially training the network on less beneficial data. However, every high-fidelity data point represents valuable information, enhancing the model's knowledge base. All in all, for an extra 10 high-fidelity model evaluation required for the initial data set, the accuracy did significantly increase.
- **Robustness:** This aspect is perhaps the most critical challenge in achieving a successful implementation. As discussed in section 4.3, the small dataset size introduces significant challenges related to prediction stability and susceptibility to random variations. The complexities associated with dataset selection, already a hurdle in conventional machine learning, are greatly amplified in this context due to the limited data. Conversely, the MPC method, which solves a control optimisation problem at each segment, also faces challenges. Although these are simpler in nature, they present multiple opportunities for the optimiser to converge to a local minimum. Such occurrences can have prolonged impacts throughout the trajectory, though they are generally easier to mitigate and better understood compared to the challenges of managing a small dataset.
- **Flexibility:** Flexibility is the area where the machine learning implementation notably outperforms the MPC method. While MPC requires a suitable predictor model that can approximate the high-fidelity model for a short duration, machine learning networks can bridge fundamental differences between models, provided there is at least some stable underlying relationship.

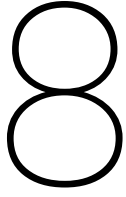
The main takeaway from this section is that while the current implementation may not be satisfactory, there are a lot of nuanced points and complex factors that influence the effectiveness of hybrid space mapping methods. The next section will dive into some discussing some of the limitations encountered in this study

7.3.2. Research limitations

A common limitation in computational studies, particularly those involving machine learning, is the constraint imposed by hardware capabilities. However, in this research, the small size of the datasets ensures that training is exceptionally quick, mitigating typical hardware limitations. This allows the study to focus more on the inherent limitations introduced by the implementation methodology itself rather than external hardware constraints.

The primary limitation of this study stems from the chosen methodology, particularly the small dataset size and the variability it introduces. The use of small datasets, introduces significant challenges in terms of model stability and predictive reliability. This limitation is further compounded by the integration of optimisation and machine learning techniques, which inherently introduces complexity and interdependence between the network performance and the optimisation process. Practically this means that this study is by no means a complete analysis of hybrid space mapping, but merely an initial exploration of the concept.

Ultimately, this implementation represents a series of compromises and nuances. The convergence of machine learning and optimisation within the framework of space mapping presents unique challenges that are not typically encountered when these disciplines operate independently. This integration, while innovative, highlights the need for careful consideration of data quality, model training strategies, and the interpretability of results within such hybrid systems. Further research is needed to address these challenges, potentially through the development of more robust training methodologies or by curating the datasets to enhance the generalisability and reliability of the machine learning models within the space mapping context.



Conclusion & suggestions for future research

In this chapter the main conclusion of this study will be given, followed by some recommendation for future research.

8.1. Conclusion

In this study, the integration of space mapping and machine learning techniques was explored to enhance the mapping of optimal control sequences in aerospace applications. The research focused on comparing traditional MPC methods with a novel machine learning approach utilising active learning and sequence-to-sequence models like GRU and Transformer networks.

A fundamental component in the successful integration of machine learning networks was the implementation of the PKI-D model architecture. This robust framework enabled the networks to act as a corrective layer to the low-fidelity model, enhancing the overall predictive accuracy. The PKI-D architecture leverages the low-fidelity model as an initial approximation, which the machine learning network subsequently refines. By having access to the low-fidelity models output and the control input, the network efficiently adjusts its predictions, ensuring a high degree of accuracy and leveraging the full potential of the available information.

The findings reveal that while MPC generally provides high accuracy and requires fewer iterations in scenarios with less dynamic control demands, its performance diminishes in complex scenarios requiring foresight beyond its planning horizon. In contrast, the machine learning approach, particularly when enhanced by active learning, shows promise in these complex scenarios by adapting effectively to new data and maintaining robust performance.

A key advantage of using machine learning in this context is its flexibility. The machine learning models, demonstrate the capability to handle sequences with varying control dynamics, an area where traditional MPC can be limited. However, the effectiveness of these models heavily relies on the amount of retraining they undergo during the optimisation process, especially the transformer networks. It was observed that there is a sweet-spot range of retraining iterations that balances model adaptability with the risk of overfitting. In general, GRU models appear to be most accurate and precise for the majority of the test cases discussed

The integration of active learning proved crucial in mitigating the challenges posed by small initial datasets. By continuously incorporating new data during the optimisation process, the machine learning models were able to refine their predictions, which allowed them to adapt as the optimisation progressed. This approach aligns well with the principles of space mapping by minimising high-fidelity model evaluations and leveraging low-fidelity models to accelerate the optimisation process.

Nevertheless, the study also highlights significant challenges. The integration of machine learning with space mapping introduces complexity in managing the interplay between model training and optimisation. Additionally, the variability introduced by small datasets and the dependence on initial conditions underscore the need for robust model design and careful setup of learning parameters.

In conclusion, this research demonstrates that while the integration of machine learning with space mapping presents notable advantages in handling complex control sequences, it also brings forth new challenges in model training and data management.

8.2. Suggestions for future research.

This study has touched upon several pathways for further investigation that could significantly enhance the integration of machine learning with complex engineering systems. One promising area is the exploration of machine learning models trained on extremely small, tailored datasets. This research could help refine the efficiency of training processes and improve the adaptability of models to specific applications, minimising the overall data requirements. Additionally, a deeper understanding of which network architectures best approximate physical processes is crucial. A comparative analysis of sequential versus parallel processing models will provide insights into which methodologies not only align more closely with the inherent nature of physical dynamics but also offer enhanced predictive accuracy.

Another critical area of future research involves developing optimiser architectures that effectively balance the dual objectives of optimising control sequences and strategically curating datasets. Initial experiments with Bayesian models have shown promising results, suggesting that further exploration could yield robust optimisation strategies. These strategies could facilitate the seamless integration of machine learning into traditional engineering simulations, offering a new toolkit for tackling complex optimisation problems with high precision and reliability.

References

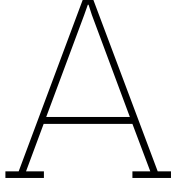
- [1] M. Klöwer, M. R. Allen, D. S. Lee, S. R. Proud, L. Gallagher, and A. Skowron, “Quantifying aviation’s contribution to global warming,” *Environmental Research Letters*, vol. 16, no. 10, p. 104 027, Oct. 2021, ISSN: 1748-9326, DOI: 10.1088/1748-9326/ac286e.
- [2] V. Grewe, A. Gangoli Rao, T. Grönstedt, *et al.*, “Evaluating the climate impact of aviation emission scenarios towards the Paris agreement including COVID-19 effects,” en, *Nature Communications*, vol. 12, no. 1, p. 3841, Jun. 2021, ISSN: 2041-1723, DOI: 10.1038/s41467-021-24091-y.
- [3] H. D. Kim, A. T. Perry, and P. J. Ansell, “A Review of Distributed Electric Propulsion Concepts for Air Vehicle Technology,” en, in *2018 AIAA/IEEE Electric Aircraft Technologies Symposium*, Cincinnati, Ohio: American Institute of Aeronautics and Astronautics, Jul. 2018, ISBN: 978-1-62410-572-2, DOI: 10.2514/6.2018-4998.
- [4] S. Kamo, J. Rosenow, H. Fricke, and M. Soler, “Robust optimization integrating aircraft trajectory and sequence under weather forecast uncertainty,” en, *Transportation Research Part C: Emerging Technologies*, vol. 152, p. 104 187, Jul. 2023, ISSN: 0968090X, DOI: 10.1016/j.trc.2023.104187.
- [5] C. L. Bottasso, A. Croce, D. Leonello, and L. Riviello, “Optimization of Critical Trajectories for Rotorcraft Vehicles,” en, *Journal of the American Helicopter Society*, vol. 50, no. 2, pp. 165–177, Apr. 2005, ISSN: 21616027, DOI: 10.4050/1.3092853.
- [6] S. Hartjes, “An Optimal Control Approach to Helicopter Noise and Emissions Abatement Terminal Procedures,” Ph.D. dissertation, Delft University of Technology, 2015, DOI: 10.4233/UUID:CA4B93AC-6A39-4C89-8699-E0A351E7FE2F.
- [7] V. Boltyanski, R. Gamkrelidze, E. Mishchenko, and L. Pontryagin, “The maximum principle in the theory of optimal processes of control,” en, *IFAC Proceedings Volumes*, vol. 1, no. 1, pp. 464–469, Aug. 1960, ISSN: 14746670, DOI: 10.1016/S1474-6670(17)70089-4.
- [8] O. Hernández-Lerma, L. R. Laura-Guarachi, S. Mendoza-Palacios, and D. González-Sánchez, *An Introduction to Optimal Control Theory: The Dynamic Programming Approach* (Texts in Applied Mathematics), en. Cham: Springer International Publishing, 2023, vol. 76, ISBN: 978-3-031-21138-6 978-3-031-21139-3, DOI: 10.1007/978-3-031-21139-3.
- [9] J. R. R. A. Martins and A. Ning, *Engineering design optimization*, eng. Cambridge: Cambridge University Press, 2022, ISBN: 978-1-108-83341-7, DOI: 10.1017/9781108980647.
- [10] D. Wolpert and W. Macready, “No free lunch theorems for optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr. 1997, ISSN: 1089778X, DOI: 10.1109/4235.585893.
- [11] J. F. Schumann and A. M. Aragón, *A machine learning approach for fighting the curse of dimensionality in global optimization*, arXiv:2110.14985 [cs, math], Nov. 2022, [Online]. Available: <http://arxiv.org/abs/2110.14985> (visited on 11/12/2023).
- [12] M. L. Santoni, E. Raponi, R. De Leone, and C. Doerr, *Comparison of High-Dimensional Bayesian Optimization Algorithms on BBOB*, arXiv:2303.00890 [cs, math, stat], Jul. 2023, [Online]. Available: <http://arxiv.org/abs/2303.00890> (visited on 11/13/2023).

- [13] A. Jain and M. Morari, *Computing the racing line using Bayesian optimization*, arXiv:2002.04794 [cs], Feb. 2020, [Online]. Available: <http://arxiv.org/abs/2002.04794> (visited on 11/13/2023).
- [14] A. De Marco, P. M. D’Onza, and S. Manfredi, “A deep reinforcement learning control approach for high-performance aircraft,” en, *Nonlinear Dynamics*, vol. 111, no. 18, pp. 17 037–17 077, Sep. 2023, ISSN: 0924-090X, 1573-269X, DOI: 10.1007/s11071-023-08725-y.
- [15] J. G.-H. Carretero, F. J. S. Nieto, and R. R. Cordon, “Aircraft trajectory simulator using a three degrees of freedom aircraft point mass model,” en, in *Proceedings of the 3rd International Conference on Application and Theory of Automation in Command and Control Systems*, Naples Italy: ACM, May 2013, pp. 114–117, ISBN: 978-1-4503-2249-2, DOI: 10.1145/2494493.2494509.
- [16] J. Sun, J. M. Hoekstra, and J. Ellerbroek, “OpenAP: An Open-Source Aircraft Performance Model for Air Transportation Studies and Simulations,” en, *Aerospace*, vol. 7, no. 8, p. 104, Jul. 2020, ISSN: 2226-4310, DOI: 10.3390/aerospace7080104.
- [17] M. Babl and J. Engelbrecht, “Automatic Deep Stall Recovery using Optimal Trajectory Planning,” en, *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 508–15 515, 2020, ISSN: 24058963, DOI: 10.1016/j.ifacol.2020.12.2377.
- [18] S. Koziel, J. Bandler, and K. Madsen, “A Space-Mapping Framework for Engineering Optimization—Theory and Implementation,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 54, no. 10, pp. 3721–3730, Oct. 2006, ISSN: 0018-9480, DOI: 10.1109/TMTT.2006.882894.
- [19] T. P. Scholcz, A. H. V. Zuijlen, and H. Bijl, “A MULTI-MODEL INCREMENTAL ADAPTIVE STRATEGY TO ACCELERATE PARTITIONED FLUID-STRUCTURE ALGORITHMS USING SPACE-MAPPING,” en, 2011, Publisher: Unpublished, DOI: 10.13140/2.1.2222.5608.
- [20] J. Bandler, Q. Cheng, S. Dakroury, *et al.*, “Space Mapping: The State of the Art,” en, *IEEE Transactions on Microwave Theory and Techniques*, vol. 52, no. 1, pp. 337–361, Jan. 2004, ISSN: 0018-9480, DOI: 10.1109/TMTT.2003.820904.
- [21] C. L. Bottasso, C.-S. Chang, A. Croce, D. Leonello, and L. Riviello, “Adaptive planning and tracking of trajectories for the simulation of maneuvers with multibody models,” en, *Computer Methods in Applied Mechanics and Engineering*, vol. 195, no. 50-51, pp. 7052–7072, Oct. 2006, ISSN: 00457825, DOI: 10.1016/j.cma.2005.03.011.
- [22] M. Simsek, Q. J. Zhang, H. Kabir, Y. Cao, and N. S. Sengor, “The recent developments in microwave design,” en, *International Journal of Mathematical Modelling and Numerical Optimization*, vol. 2, no. 2, p. 213, 2011, ISSN: 2040-3607, 2040-3615, DOI: 10.1504/IJMMNO.2011.039429.
- [23] N. McCullum, *Deep Learning Neural Networks Explained in Plain English*, Jun. 2020, [Online]. Available: <https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/>.
- [24] M. M. Taye, “Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions,” en, *Computers*, vol. 12, no. 5, p. 91, Apr. 2023, ISSN: 2073-431X, DOI: 10.3390/computers12050091.
- [25] H. Abdel-Jaber, D. Devassy, A. Al Salam, L. Hidaytallah, and M. EL-Amir, “A Review of Deep Learning Algorithms and Their Applications in Healthcare,” en, *Algorithms*, vol. 15, no. 2, p. 71, Feb. 2022, ISSN: 1999-4893, DOI: 10.3390/a15020071.
- [26] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A Critical Review of Recurrent Neural Networks for Sequence Learning*, arXiv:1506.00019 [cs], Oct. 2015, [Online]. Available: <http://arxiv.org/abs/1506.00019> (visited on 11/15/2023).

- [27] S. Hiriyannaiah, A. Srinivas, G. K. Shetty, S. G.M., and K. Srinivasa, “A computationally intelligent agent for detecting fake news using generative adversarial networks,” en, in *Hybrid Computational Intelligence*, Elsevier, 2020, pp. 69–96, ISBN: 978-0-12-818699-2, DOI: 10.1016/B978-0-12-818699-2.00004-4.
- [28] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to Sequence Learning with Neural Networks*, arXiv:1409.3215 [cs], Dec. 2014, [Online]. Available: <http://arxiv.org/abs/1409.3215> (visited on 11/16/2023).
- [29] Chaitanya Bharathi Institute of Technology(Autonomous), K. M.Tarwani, and S. Edem, “Survey on Recurrent Neural Network in Natural Language Processing,” *International Journal of Engineering Trends and Technology*, vol. 48, no. 6, pp. 301–304, Jun. 2017, ISSN: 22315381, DOI: 10.14445/22315381/IJETT-V48P253.
- [30] C. Yun, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar, *Are Transformers universal approximators of sequence-to-sequence functions?* arXiv:1912.10077 [cs, stat], Feb. 2020, [Online]. Available: <http://arxiv.org/abs/1912.10077> (visited on 11/06/2023).
- [31] S. Bhattamishra, A. Patel, and N. Goyal, *On the Computational Power of Transformers and its Implications in Sequence Modeling*, arXiv:2006.09286 [cs, stat], Oct. 2020, [Online]. Available: <http://arxiv.org/abs/2006.09286> (visited on 11/16/2023).
- [32] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training Recurrent Neural Networks*, arXiv:1211.5063 [cs], Feb. 2013, [Online]. Available: <http://arxiv.org/abs/1211.5063> (visited on 11/21/2023).
- [33] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, *Recent Advances in Recurrent Neural Networks*, arXiv:1801.01078 [cs], Feb. 2018, [Online]. Available: <http://arxiv.org/abs/1801.01078> (visited on 11/21/2023).
- [34] I. R. Jenkins, L. O. Gee, A. Knauss, H. Yin, and J. Schroeder, “Accident Scenario Generation with Recurrent Neural Networks,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI: IEEE, Nov. 2018, pp. 3340–3345, ISBN: 978-1-72810-321-1 978-1-72810-323-5, DOI: 10.1109/ITSC.2018.8569661.
- [35] R. C. Staudemeyer and E. R. Morris, *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*, arXiv:1909.09586 [cs], Sep. 2019, [Online]. Available: <http://arxiv.org/abs/1909.09586> (visited on 11/21/2023).
- [36] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, *Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN*, arXiv:1803.04831 [cs], May 2018, [Online]. Available: <http://arxiv.org/abs/1803.04831> (visited on 11/22/2023).
- [37] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi, *Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture*, arXiv:1802.06338 [cs], Oct. 2018, [Online]. Available: <http://arxiv.org/abs/1802.06338> (visited on 10/20/2023).
- [38] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention Is All You Need,” 2017, Publisher: arXiv Version Number: 7, DOI: 10.48550/ARXIV.1706.03762.
- [39] S. Ahmed, I. E. Nielsen, A. Tripathi, S. Siddiqui, R. P. Ramachandran, and G. Rasool, “Transformers in Time-Series Analysis: A Tutorial,” en, *Circuits, Systems, and Signal Processing*, vol. 42, no. 12, pp. 7433–7466, Dec. 2023, ISSN: 0278-081X, 1531-5878, DOI: 10.1007/s00034-023-02454-8.
- [40] K. Babić, S. Martinčić-Ipšić, and A. Meštrović, “Survey of Neural Text Representation Models,” en, *Information*, vol. 11, no. 11, p. 511, Oct. 2020, ISSN: 2078-2489, DOI: 10.3390/info1110511.

- [41] H. Xie, Z. Qin, G. Y. Li, and B.-H. Juang, “Deep Learning Enabled Semantic Communication Systems,” *IEEE Transactions on Signal Processing*, vol. 69, pp. 2663–2675, 2021, arXiv:2006.10685 [eess], ISSN: 1053-587X, 1941-0476, DOI: 10.1109/TSP.2021.3071210.
- [42] R. Sathya and A. Abraham, “Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification,” en, *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, no. 2, 2013, ISSN: 21654069, 21654050, DOI: 10.14569/IJARAI.2013.020206.
- [43] P. Ren, Y. Xiao, X. Chang, *et al.*, *A Survey of Deep Active Learning*, arXiv:2009.00236 [cs, stat], Dec. 2021, [Online]. Available: <http://arxiv.org/abs/2009.00236> (visited on 07/02/2024).
- [44] Y. Gal, R. Islam, and Z. Ghahramani, *Deep Bayesian Active Learning with Image Data*, arXiv:1703.02910 [cs, stat], Mar. 2017, [Online]. Available: <http://arxiv.org/abs/1703.02910> (visited on 07/02/2024).
- [45] F. B. Smith, A. Foster, and T. Rainforth, *Making Better Use of Unlabelled Data in Bayesian Active Learning*, arXiv:2404.17249 [cs, stat], Apr. 2024, [Online]. Available: <http://arxiv.org/abs/2404.17249> (visited on 07/02/2024).
- [46] L. Zhao, Y. Zuo, T. Li, and C. L. P. Chen, “Application of an Encoder–Decoder Model with Attention Mechanism for Trajectory Prediction Based on AIS Data: Case Studies from the Yangtze River of China and the Eastern Coast of the U.S,” en, *Journal of Marine Science and Engineering*, vol. 11, no. 8, p. 1530, Jul. 2023, ISSN: 2077-1312, DOI: 10.3390/jmse11081530.
- [47] Y. Pang, N. Xu, and Y. Liu, “Aircraft Trajectory Prediction using LSTM Neural Network with Embedded Convolutional Layer,” *Annual Conference of the PHM Society*, vol. 11, no. 1, Sep. 2019, ISSN: 2325-0178, 2325-0178, DOI: 10.36001/phmconf.2019.v11i1.849.
- [48] K. Kutsuzawa, S. Sakaino, and T. Tsuji, “Sequence-to-Sequence Model for Trajectory Planning of Nonprehensile Manipulation Including Contact Model,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3606–3613, Oct. 2018, ISSN: 2377-3766, 2377-3774, DOI: 10.1109/LRA.2018.2854958.
- [49] T. Guffanti, D. Gammelli, S. D’Amico, and M. Pavone, *Transformers for Trajectory Optimization with Application to Spacecraft Rendezvous*, arXiv:2310.13831 [cs], Dec. 2023, [Online]. Available: <http://arxiv.org/abs/2310.13831> (visited on 12/12/2023).
- [50] H. B. Braiek and F. Khomh, *Machine Learning Robustness: A Primer*, arXiv:2404.00897 [cs], May 2024, [Online]. Available: <http://arxiv.org/abs/2404.00897> (visited on 06/30/2024).
- [51] A. Alwosheel, S. Van Cranenburgh, and C. G. Chorus, “Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis,” en, *Journal of Choice Modelling*, vol. 28, pp. 167–182, Sep. 2018, ISSN: 17555345, DOI: 10.1016/j.jocm.2018.07.002.
- [52] K. Jordan, *On the Variance of Neural Network Training with respect to Test Sets and Distributions*, arXiv:2304.01910 [cs], Jun. 2024, [Online]. Available: <http://arxiv.org/abs/2304.01910> (visited on 06/30/2024).
- [53] R. Shelton and M. Madden, “Six degree-of-freedom (6-DOF) Flight Simulation Check-cases,” NASA Engineering and Safety Center, Tech. Rep., Apr. 2015, [Online]. Available: <https://nescacademy.nasa.gov/flightsim> (visited on 11/29/2023).
- [54] J. S. Berndt, *Open Source Flight Dynamics Model in C++*, [Online]. Available: <https://jsbsim.sourceforge.net/JSBSimFlyer.pdf> (visited on 11/29/2023).
- [55] K. Toohey and M. Duckham, “Trajectory similarity measures,” en, *SIGSPATIAL Special*, vol. 7, no. 1, pp. 43–50, May 2015, ISSN: 1946-7729, DOI: 10.1145/2782759.2782767.

-
- [56] Y. Tao, A. Both, R. I. Silveira, *et al.*, “A comparative analysis of trajectory similarity measures,” en, *GIScience & Remote Sensing*, vol. 58, no. 5, pp. 643–669, Jul. 2021, ISSN: 1548-1603, 1943-7226, DOI: 10.1080/15481603.2021.1908927.



Low-Fidelity Flight Mechanics Models

In this section of the appendix, a full overview of the low- and high-fidelity flight mechanics model is given. This model is based on the airbus A320, with the parameters sourced from JSBSim for consistency with the high-fidelity model:

Table A.1: Overview of aircraft parameters

| Parameter | Value | unit |
|----------------------------------|--------|------------|
| Lift Curve slope (CL_α) | 2π | rad^{-1} |
| Wing area (S) | 124 | m^2 |
| Maximum thrust (T_{max}) | 240 | kN |
| Maximum weight (m) | 63956 | kg |
| zero lift drag (C_{d0}) | 0.018 | - |
| Linearised drag polar (K) | 0.039 | - |

Starting of with the low-fidelity model, it is crucial that it is easy to evaluate as that is one of the key defining traits of a low-fidelity model. Presented earlier in subsection 2.2.2, the flight mechanics model from [15] presents an ideal candidate as low-fidelity model. Adapted for a 2D case this becomes:

$$\frac{d}{dt} \begin{bmatrix} x \\ z \\ V \\ \gamma \end{bmatrix} = \begin{bmatrix} V \cos(\gamma) \\ V \sin(\gamma) \\ \frac{g}{W} [(T \cos(\theta - \gamma) - D) - W \sin(\gamma)] \\ \frac{g}{WV} [(L + T \sin(\theta - \gamma) - D) - W \cos(\gamma)] \end{bmatrix} \quad (A.1)$$

$$T = T_{max} * \delta_T \quad (A.2)$$

$$L = CL_\alpha \alpha q S \quad (A.3)$$

$$D = C_{d0} + K * C_L^2 \quad (A.4)$$

Note that for this flight mechanics model, change in mass is neglected as the trajectories in question are relatively short and fuel consumption is negligible. Another point of importance are the control inputs of these models. The model above directly sets the geometric pitch angle (θ) and thrust setting thereby using them as inputs. On the other hand, JSBSim which functions as the high-fidelity model uses elevator deflection (δ_e) and thrust setting as input. This modelling difference requires a mapping from δ_e to θ . For this mapping two approaches are considered; an open loop and closed loop mapping approach.

A.1. Open loop

For the open loop implementation, elevator deflection is linked to θ through the use of input dampening. Here, the angle of attack is added as an extra state and implemented as follows:

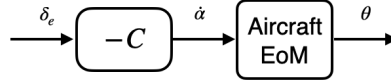


Figure A.1: Block diagram of a closed loop controller. C represents a tuneable gain factor.

$$\frac{d\alpha}{dt} = -C * \delta_e \quad (\text{A.5})$$

During each step, the angle of attack and flight path angle are updated using Equation A.5 and Equation A.1 which then allows θ to be calculated. While not a perfect representation of the actual longitudinal dynamics, this approach captures the general longitudinal dynamics. A problem with this approach is that it is, among other parameters, very sensitive to gain factor C . Elevator deflection primarily regulates the rate of pitch change, meaning slight adjustments in C can dramatically alter the resultant angle of attack and geometric pitch angle. Additionally, minor variations in control settings at a constant gain can lead to significantly divergent trajectories. Such sensitivity poses challenges for machine learning models, which ideally require that small changes in controls correspond to proportional changes in trajectories to facilitate simpler learning patterns.

Conversely, sensitivity to model parameters and gain factors are desirable when adjusting the model to locally approximate the high-fidelity model, such as in model predictive control. Here, the low-fidelity model serves as a local predictor, where the focus is on short-term accuracy rather than long-term stability. The tendency for the model's behaviour to diverge quickly is less problematic due to the shorter simulation time frames involved.

A.2. Closed loop

To improve model stability, a closed loop proportional controller can be introduced to regulate the diverging behaviour of direct rate-of-change control:

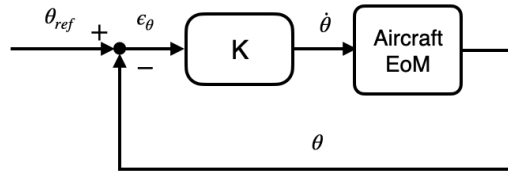


Figure A.2: Block diagram of a closed loop controller. K represents a tuneable gain factor.

$$\theta_{ref} = -C * \delta_e \quad (\text{A.6})$$

The block diagram and Equation A.6 illustrates how a reference geometric pitch angle set by δ_e is incorporated in the aircraft equations of motion and forward simulation. The coupling of δ_e to θ_{ref} provides a basic mapping of elevator deflection to pitch angle, which is crucial for ensuring that both the low- and high-fidelity models operate with the same set of inputs. It is also important to note that the low-fidelity model in this framework is not designed to mimic high-fidelity models with exact precision. Rather, its primary role is to provide a simplified and computationally efficient alternative to the high-fidelity model. The value of these models lies in their adaptability to various scenarios and

applications. In an open-loop configuration, the model serves well for short-term approximations, while in a closed-loop setup, it offers a reliable initial approximation for more extended periods. In summary, these models are employed either as tools for local approximation in the open-loop configuration or as a basis for initial predictions in closed-loop scenarios.

A.3. High-fidelity model

In this study JSBSim is used as the high-fidelity model. JSBSim is a well established flight dynamics simulation tool which has garnered widespread recognition in academic research [53]. Due to its efficiency, extensive support, and numerous capabilities, JSBSim is able to simulate a wide array of aircraft types. Essentially, it is a 6-DoF non-linear flight dynamics simulator that allows for customisable aircraft models [54] [14]. Another reason is that JSBSim represents a significant step in model complexity compared to point mass models, while still not being taxing to run.

In this project, the JSBSim model was limited to 2-D motion only, excluding any lateral or rotational movements such as yaw and roll. This was done by setting any lateral velocity and rotational movement component to zero during each step of the simulation.

Aerodynamic and thrust modelling

Aerodynamic modelling in JSBSim is based on lookup tables, which catalogues aerodynamic coefficients under various flight conditions and configurations, including different flap settings. These lookup tables are important because they encapsulate complex aerodynamic relationships within a straightforward tabular format. Using interpolation techniques, the simulator can accurately derive values for conditions that lie between defined points in the table. This method allows for modelling aerodynamic forces with a high degree of fidelity, accommodating non-linear variations that are often observed in real-world flight dynamics.

Similarly, thrust modelling in JSBSim is handled through a robust approach that utilises both lookup tables and mathematical models. This is particularly relevant for jet engines, where performance depends on throttle settings, Mach number, and the ambient temperature. The integration of lookup tables into the standard equations of motion models enables the simulation to determine engine thrust output, mirroring the actual engine performance.

B

Error Determination Methods

In the realm of flight mechanics and control systems, accurately estimating trajectory errors is crucial for effective operation. The complexity of flight dynamics and the influence of unpredictable environmental conditions necessitate robust methods to assess the deviation of an actual flight path from its planned trajectory. Two trajectories might not exactly overlap, but if they are similar in nature this might also constitute as a "good" mapping. Other factors such as sampling irregularity and different sequence lengths make direct comparison more difficult [55]. This section of the appendix outlines several computational techniques that have been developed to quantify trajectory discrepancies effectively.

Lock-Step Euclidean distance (LSED)

This method is perhaps the most straightforward approach to trajectory comparison. Here the distance between points on different trajectories is calculated for each time step. Given as an equation this is:

$$EU(A, B) = \sqrt{\sum_{i=1}^n dist_2^2(a_i, b_i)} \quad (B.1)$$

Given a trajectory A and B with length n, this gives the total error between two paths. Note that here the two trajectories must be of the same length. Factors like sampling irregularity appearing in a sequence cannot be accounted for in this cases.

Discrete Fréchet Distance (DFD) & Dynamic Time warping (DTW)

These approaches are based on the principle of furthest distance between any point in the trajectory set. Intuitively this can be explained by the analogy of a person walking a dog with a leash [55]. Throughout the walk, the dog may vary its speed, halt, or change direction, resulting in a trajectory that differs from that of the person. However, the trajectories remain approximately similar, influenced by the leash's length. Both Dynamic Time Warping (DTW) and Dynamic Furthest Distance (DFD) methodologies quantify this similarity by metaphorically representing the 'leash length' required to align both paths, thus providing an indication of their comparative similarity.

The difference in DFD and DTW lie in their respective path scoring approach. The process begins by constructing a table that calculates the distance between each point on one trajectory and every other point on the counterpart trajectory. A cost path is defined as the path it takes to go from the start of one sequence diagonally to the end of the other sequence. DTW takes the sum of the lowest cost path, where cost refers to the distance between points. DFD only considers the maximum value in this matrix, emphasising the greatest deviation between the two trajectories.

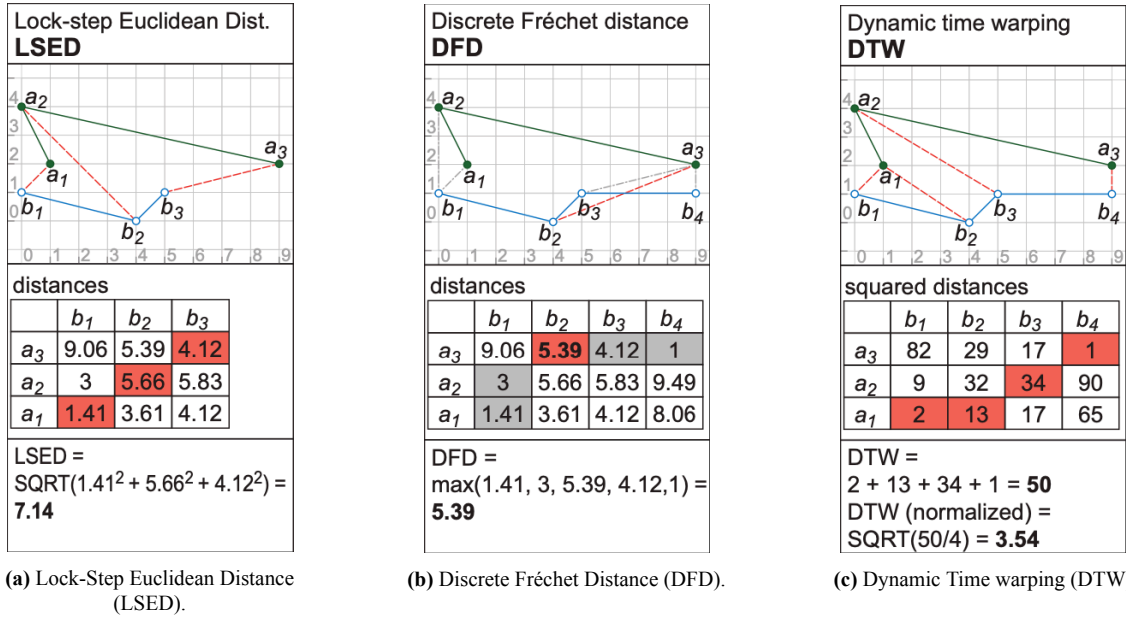


Figure B.1: Overview of three trajectory similarity scoring methods. Note that DFD and DTW are able to handle varying sequence lengths whereas LSED cannot. Distances computed in distance table are based on euclidean distances. This approach can be also be applied to compare trajectories of low and high-fidelity flight mechanics models. Adapted from [56].

As can be seen in Figure B.1, the scoring method has significant influence on how trajectory deviations are tolerated. In turn, this also has a significant impact on how an optimiser based on these trajectory differences might behave. This consideration is particularly relevant in the realm of optimal control trajectories, where accommodating certain levels of variance might be beneficial. For instance, specific control actions such as adjusting thrust levels, are not instantaneous and can introduce a degree of trajectory lag. Employing methods like DFD or DTW can render these variances less punishing, thereby being more agreeable during optimisation.

Longest Common Sub-Sequence (LCSS)

Another common approach to comparing two sequences is the Longest Common Sub-Sequence (LCSS) method. Where the previous methods measure distances, LCSS quantifies the similarity between sequences through a score. This score reflects the number of similar points within the sequences while traversing the sequences monotonically. Originally intended for comparing two sequences of letters which are discrete entities, a threshold value for spatial differences ϵ can be used to adapt this method for flight trajectories. To allow for temporal differences, a different threshold value δ is used. These must be chosen with care and are dependent on the situation. In case the expected deviations are small, a high threshold can cause all the deviations to be acceptable which in turn leads to an incorrect high score. The following pseudo-code describes how the LCSS score is formed [55]:

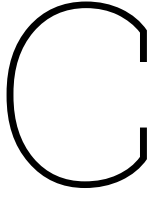
Algorithm 4 LCSS with Spatial and Temporal Thresholds

```

1: procedure LCSS( $A, B, \epsilon, \delta$ )
2:    $m \leftarrow$  length of  $A$ 
3:    $n \leftarrow$  length of  $B$ 
4:   Initialize matrix  $dp$  of size  $(m + 1) \times (n + 1)$  to all zeros
5:   for  $i \leftarrow 1$  to  $m$  do
6:     for  $j \leftarrow 1$  to  $n$  do
7:       if  $\sqrt{\sum_{k=1}^d (A[i - 1][k] - B[j - 1][k])^2} \leq \epsilon$  and  $|i - j| \leq \delta$  then
8:          $dp[i][j] \leftarrow dp[i - 1][j - 1] + 1$ 
9:       else
10:         $dp[i][j] \leftarrow \max(dp[i - 1][j], dp[i][j - 1])$ 
11:      end if
12:    end for
13:  end for
14:  return  $dp[m][n] / \min(m, n)$ 
15: end procedure

```

Line 7 in the code above is the main point of importance. This is where the spatial and temporal threshold is incorporated into the score. If the computed distance between point i in trajectory A and point j in trajectory B is below threshold ϵ , and within a certain number of indexes δ assuming an equal point sampling rate, this combination of points is added to the total similarity score. By iterating through both sequences, the total score is built up. A big advantage with LCSS scores is that they are always normalised between 0 and 1 due to the final step in the algorithm. This makes them highly suitable for application in optimisation problems.



MPC method supporting results

This section of the appendix is dedicated to presenting supporting test cases that complement the main MPC results discussed in chapter 6

C.1. Base implementation

Test case 1

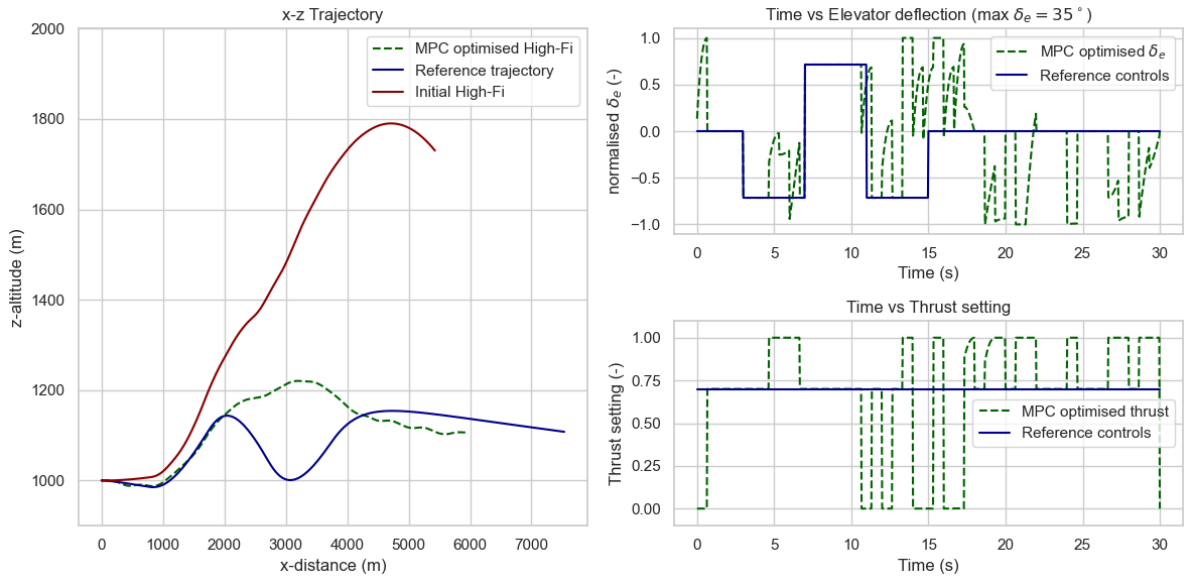


Figure C.1: MPC optimised high-fidelity trajectory for test case 1. Furthermore, $n = 45$ segments were used, and a planning window of $T = 3s$ and a manoeuvre duration of 30 seconds.

Table C.1: Error comparison between the initial and MPC optimised high-fidelity trajectory error compared to the reference trajectory for the first test case. For the LCSS error, a spatial threshold of 100 meter was used, and a temporal threshold of 30 steps were used, which is equivalent to 1 second.

| Error Type | Initial error | Optimised error | Percentage decrease w.r.t initial |
|------------------------------|---------------|-----------------|-----------------------------------|
| DTW error (1e3 m) | 606.0 | 189.9 | 68.7% |
| DFD (leash length) error (m) | 2195.8 | 1570.7 | 28.5% |
| LCSS (-) | 0.74 | 0.65 | 12.4% |
| ELS (1e3) | 751.4 | 511.1 | 32.0% |

An initial observation from Figure C.1 shows that although the MPC method initially tracks the reference trajectory well, it soon diverges. This divergence could be attributed to the optimiser's failure to adequately anticipate subsequent trajectory changes, resulting in an overshoot that cannot be corrected swiftly enough. This underscores the importance of considering the entire manoeuvre, particularly in trajectories that exhibit large variations. Furthermore, Table C.1 indicates that despite the optimiser's inability to accurately follow the reference trajectory throughout, it successfully reduced the error compared to the initial trajectory. This highlights the method's partial effectiveness, even when facing challenges with trajectory prediction.

Test case 2

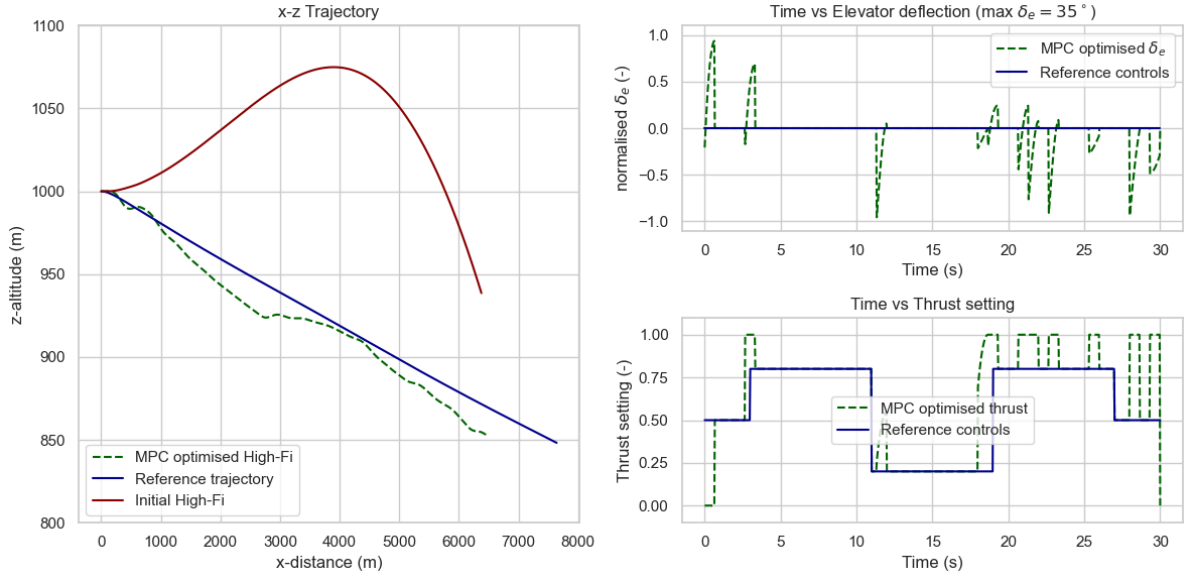


Figure C.2: MPC optimised high-fidelity trajectory for test case 2. Furthermore, $n = 45$ segments were used, and a planning window of $T = 3$ seconds and a manoeuvre duration of 30 seconds.

Table C.2: Error comparison between the initial and MPC optimised high-fidelity trajectory error compared to the reference trajectory for test case 2

| Error Type | Initial error | Optimised error | Percentage decrease w.r.t initial |
|------------------------------|---------------|-----------------|-----------------------------------|
| DTW error (1e3 m) | 182.5 | 87.5 | 52.1% |
| DFD (leash length) error (m) | 1265.7 | 1178.3 | 6.9% |
| LCSS (-) | 0.67 | 0.53 | 21.1% |
| ELS (1e3) | 422.2 | 379.1 | 10.2% |

For the second test case, the optimiser performed much better compared to the first test case. While, adhering more closely to the intended trajectory, some oscillations around the reference trajectory were noticeable early on, as indicated by two spikes in elevator control actuation near the beginning. This overcompensation early in the sequence has ramifications throughout the remainder of the sequence, highlighting a potential risk associated with the MPC approach. Since the segments are processed sequentially, a poorly optimised control segment affects every subsequent segment, emphasising the need for careful selection of the control optimiser to avoid getting trapped in sub-optimal local minima.

Despite these challenges, the optimiser significantly enhanced the trajectory fit, as evidenced in Table C.2. It is important to note, however, that the discrepancy between different error estimation methods persists. The LCSS error, for instance, decreased by a smaller margin than one might infer

from a visual inspection of Figure C.2, underscoring the complexity of accurately assessing trajectory alignment solely through numerical metrics.

C.2. Number of segments analysis

Test case 2

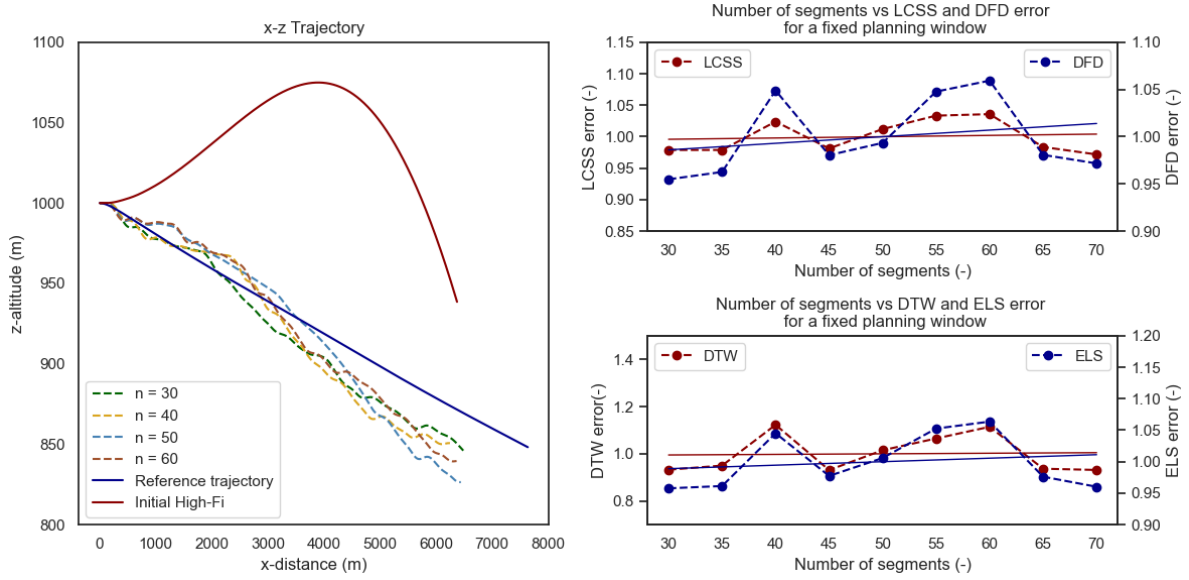


Figure C.3: Effect of varying the number of segments for a fixed planning time of 3 seconds.

Table C.3: Comparison between the different plot regression lines statistical values for the relation between number of segments and error for test case 2

| Error Type | Average value | R^2 | p-value |
|------------------------------|---------------|-------|---------|
| DTW error (1e3 m) | 74.1 | 0.002 | 0.917 |
| DFD error (leash length) (m) | 1056.47 | 0.055 | 0.542 |
| LCSS (-) | 0.50 | 0.012 | 0.782 |
| ELS (1e3 m) | 318.7 | 0.031 | 0.650 |

Judging from Figure C.3, the number of steering segments, and consequently the size of each segment, appears to have little to no impact on the overall trajectory and associated error. This observation is further corroborated by Table C.3, which details the average error and statistical parameters of the trend line. For all error values, the R^2 value remains below 0.05, and the p-value is notably high (above 0.5). These statistics suggest that the error is not significantly influenced by the number of segments, indicating that the observed consistency is unlikely to be attributable to fortunate sampling or random variations in model output.

Test case 4

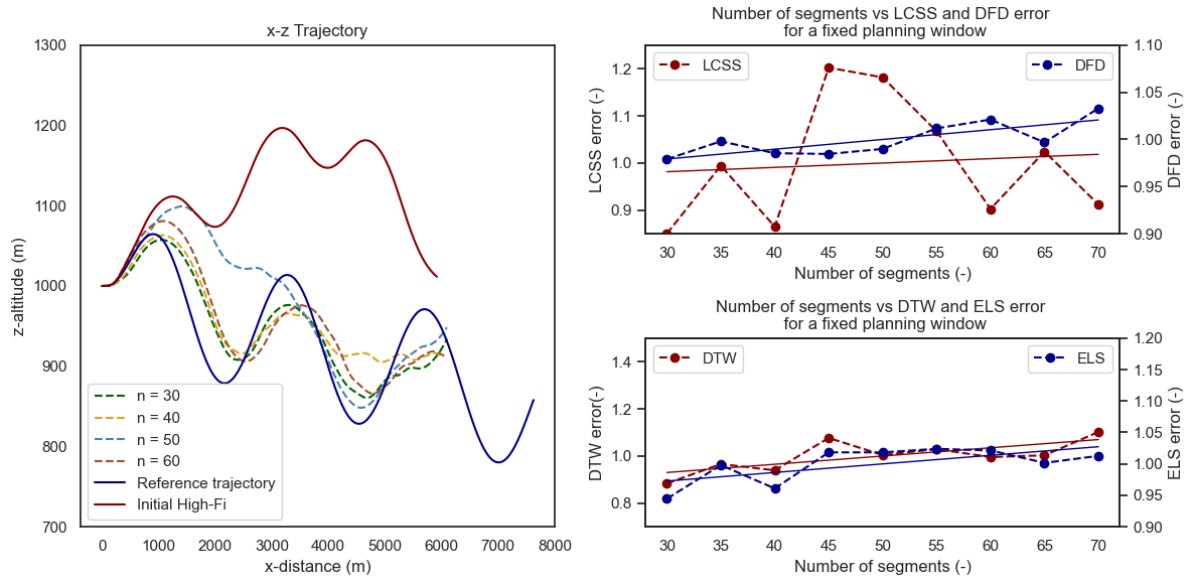


Figure C.4: Effect of varying the number of segments for a fixed planning time of 3 seconds. for the third case.

Table C.4: Comparison between the different plot regression lines statistical values for the relation between number of segments and error for test case 4.

| Error Type | Average value | R^2 | p-value |
|------------------------------|---------------|-------|---------|
| DTW error (1e3 m) | 140.7 | 0.516 | 0.029 |
| DFD error (leash length) (m) | 1362.8 | 0.602 | 0.014 |
| LCSS (-) | 0.64 | 0.009 | 0.805 |
| ELS (1e3 m) | 409.3 | 0.433 | 0.054 |

Looking the low R^2 and relatively high p-values again suggest a weak relation between the number of segments and mapping performance. Referencing Figure C.4 reveals that indeed the trajectories for the different number of segments are all very similar, with the exception of $n=50$ segments as an outlier. This again support the notion that the number of segments is a problem specific and requires case specific tuning.

C.3. Planning horizon analysis

Test case 1

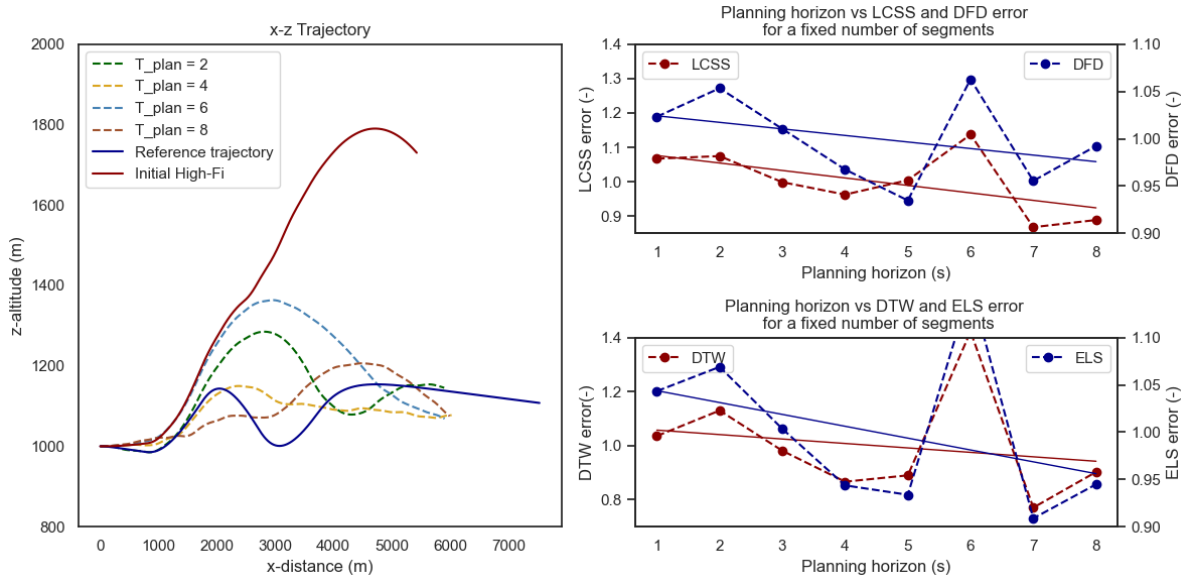


Figure C.5: Effect of varying planning horizon duration for a fixed number of 45 segments and a manoeuvre duration of 30 seconds.

Table C.5: Comparison between the different plot regression lines statistical values for the relation between planning horizon and error for test case 1

| Error Type | Average value | R^2 | p-value |
|------------------------------|---------------|-------|---------|
| DTW error (1e3 m) | 160.7 | 0.039 | 0.637 |
| DFD error (leash length) (m) | 1370.5 | 0.136 | 0.369 |
| LCSS (-) | 0.627 | 0.330 | 0.137 |
| ELS (1e3 m) | 419.4 | 0.135 | 0.371 |

Looking at the graph above, there seems to be a limit to the planning window duration for which the optimiser is able to generate good results. Visually this can be seen in the left graph where the trajectory for $T = 4$ yields better results compared to $T = 2$. Increasing the planning horizon even further yields visibly worse results for both $T = 6$ and $T = 8$ seconds. Regarding the $T = 8$ trajectory and its lower score, this can be attributed to the fact that it is more or less "in-line" with the reference trajectory but without actually fitting well to the reference. As the errors are skewed to the horizontal displacement as the low-fidelity reference reaches much further due to modelling discrepancies, this is prioritised by the optimiser. Effectively this means that there is a sweet spot where short-term vertical displacement is balanced with horizontal displacement which yields a satisfactory fit.

Another complicating factor in this approach is the fact that an optimisation problem has to be run for every segment, increasing the potential for sub-optimal results. Furthermore, in case the control optimisation for a segment is not able to converge, this has consequences for every subsequent sequence.

Test case 2

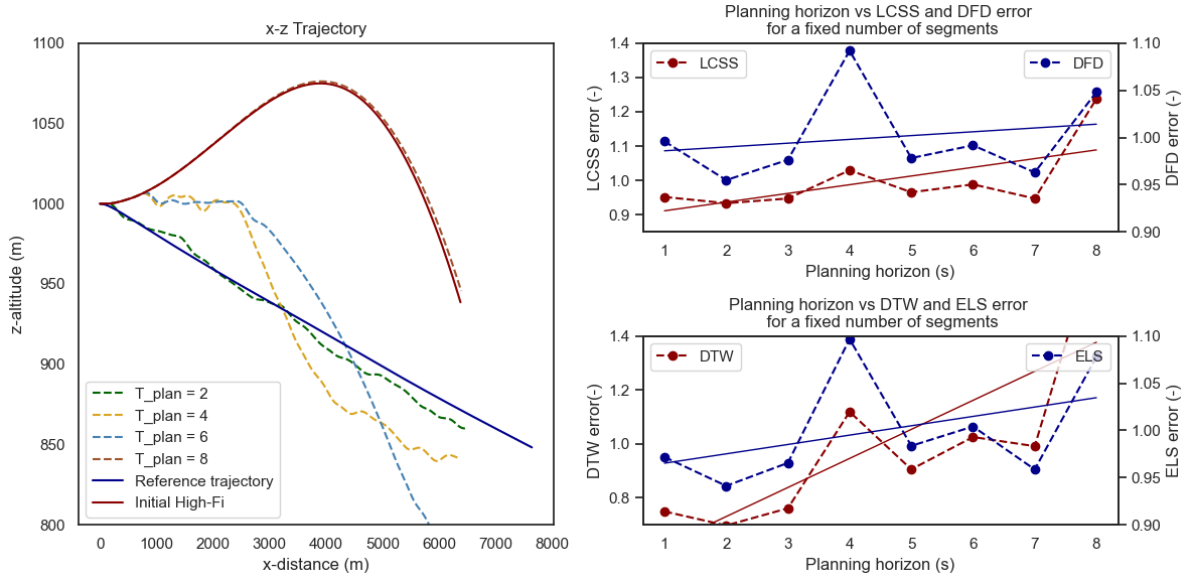


Figure C.6: Effect of varying planning horizon duration for a fixed number of 60 segments and a manoeuvre duration of 30 seconds.

Table C.6: Comparison between the different plot regression lines statistical values for the relation between planning horizon and error for test case 1

| Error Type | Average value | R^2 | p-value |
|------------------------|---------------|-------|---------|
| DTW (1e3 m) | 90.6 | 0.612 | 0.022 |
| DFD (leash length) (m) | 1061.0 | 0.045 | 0.616 |
| LCSS (-) | 0.520 | 0.380 | 0.104 |
| ELS (1e3 m) | 322.6 | 0.181 | 0.294 |

In the second test case, observations align closely with the patterns identified in the first reference case. The optimal time horizon for adjustments appears to be around 2-3 seconds; extending beyond this duration significantly deteriorates performance. However, the error metrics provide only a partial view of the outcomes. For instance, despite the trajectory for $T = 8$ seconds showing minimal deviation from the original path, this is attributed to the structure of the optimiser. If the control optimiser fails to enhance the controls, it defaults to the original control set. This inability to optimise the trajectory leads to a modest increase in DFD error by 5% but results in a more pronounced 40% rise in DTW error. Such outcomes underscore the necessity of considering multiple types of error metrics to fully understand performance dynamics. Additionally, the DTW error trend proves significant both in model fit, with a robust R^2 , and in data confidence, indicating its reliability as a performance indicator.

D

Machine Learning method results

This section of the appendix is dedicated to presenting supporting test cases that complement the main machine learning results discussed in chapter 6.

D.1. Network type analysis

Test case 2

Where the first test case investigates the networks' ability to compensate for elevator deflection, the second test case highlights the networks' ability to process variations in thrust

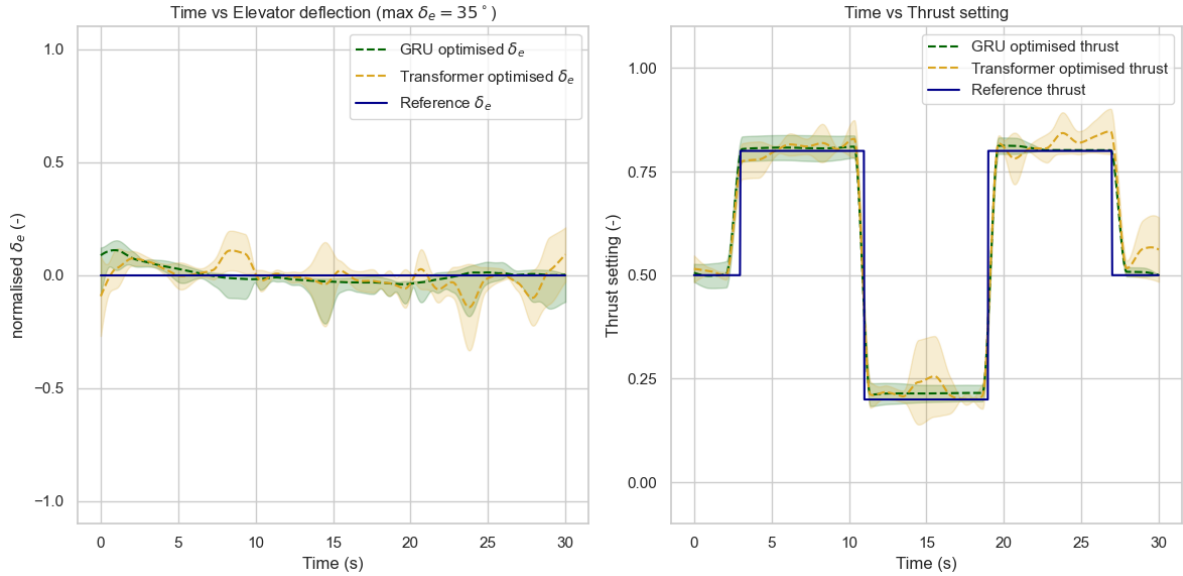


Figure D.1: Machine learning optimised controls for test case 2. The controls optimised using a GRU network have a variance (expressed as a percentage of the maximum control value) of 3.1% and 2.8% for the elevator and thrust setting respectively. For the transformer network the variance is 9.1% and 5.0% respectively.

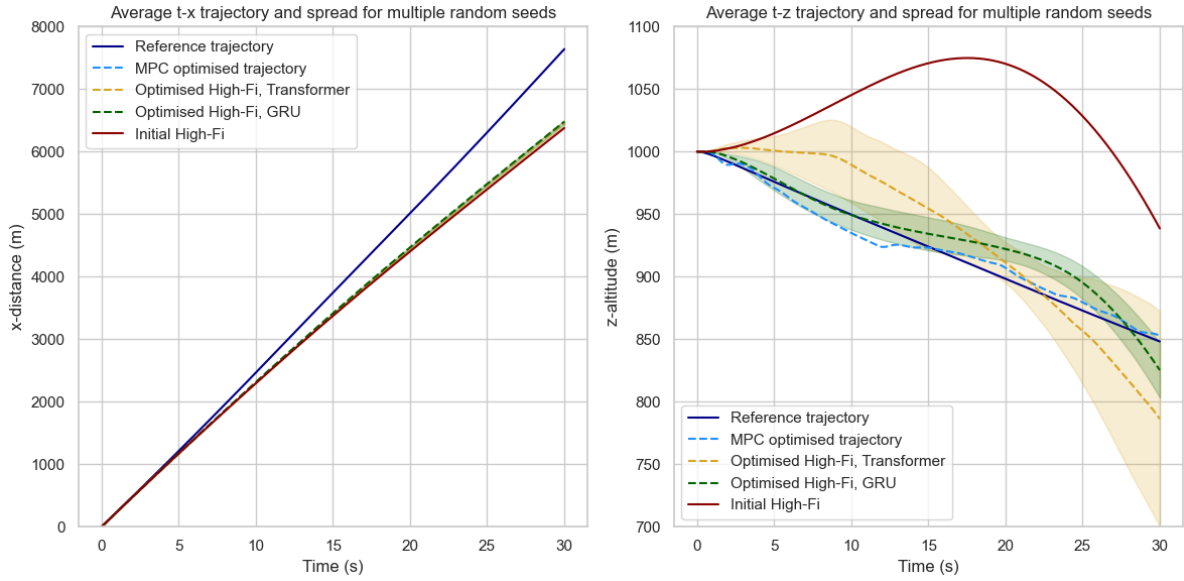


Figure D.2: Machine learning optimised high-fidelity trajectory for test case 2. For the GRU network, the x and z variance is 10.5m and 16.0m respectively. For the transformer network the variance is 24.0m and 42.0m.

The results indicate that while the machine learning-optimised trajectory has difficulty adhering closely to the target trajectory, the MPC method achieves a more accurate tracking. Among the machine learning models, the GRU network consistently outperforms the Transformer network in terms of both optimisation outcomes and result variance. In scenarios involving relatively simple tracking trajectories, the MPC method demonstrates superior performance.

Table D.1: Optimised high-fidelity trajectory error metrics for test case 2.

| (a) GRU network | | | (b) Transformer network | | |
|---------------------------|---------------------|--------------|---------------------------|---------------------|--------------|
| Error Type | Mean error μ | STD σ | Error Type | Mean error μ | STD σ |
| DTW (1e3 m) | 90.4 | 5.9 | DTW (1e3 m) | 118.3 | 25.7 |
| DFD (leash length) (m) | 1160.5 | 21.1 | DFD (leash length) (m) | 1187.0 | 67.5 |
| LCSS (-) | 0.513 | 0.01 | LCSS (-) | 0.565 | 0.082 |
| ELS (m) | 366.6 | 9.2 | ELS (m) | 383.6 | 24.7 |

The performance metrics in the tables clearly show the GRU network's superior performance. Across all error types the GRU network not only achieves lower mean errors but also maintains much tighter variance compared to the Transformer network. This consistency highlights the GRU's robustness and reliability in trajectory prediction under varying conditions.

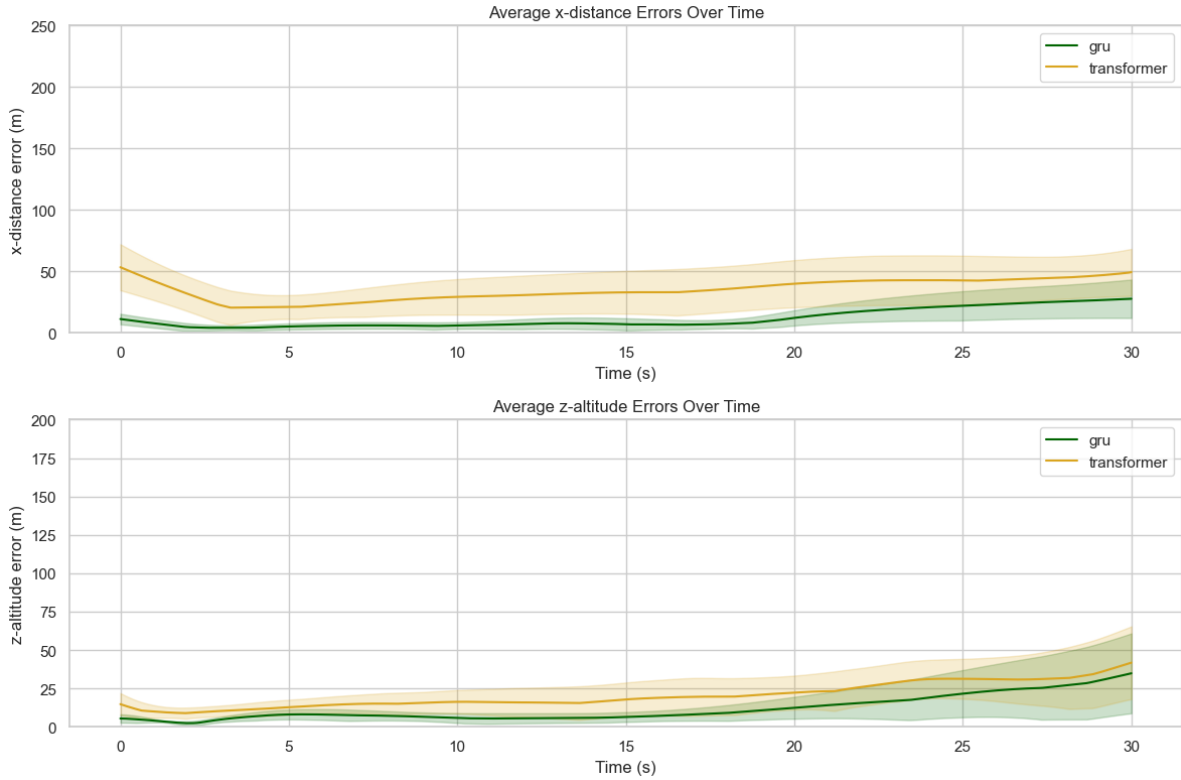


Figure D.3: Average prediction error and error variance of the GRU and transformer network for test case 2. The average GRU prediction variance is 8.8m and 10.9m for x and z dimension respectively. The average transformer network variance is 23.7m and 15.0m.

The dominance of the GRU network is again confirmed by its prediction error and variance. Similar to the first test case, the GRU network, being a recurrent network, experiences a loss of accuracy and precision as the prediction extends further into the future. Additionally, a slight increase in prediction error is observed around each thrust change (around $t = 3$, $t = 11$, and $t = 18$ seconds). This increase is less pronounced compared to the elevator deflection case, suggesting that the networks are able to process acceleration changes more efficiently.

Test case 4

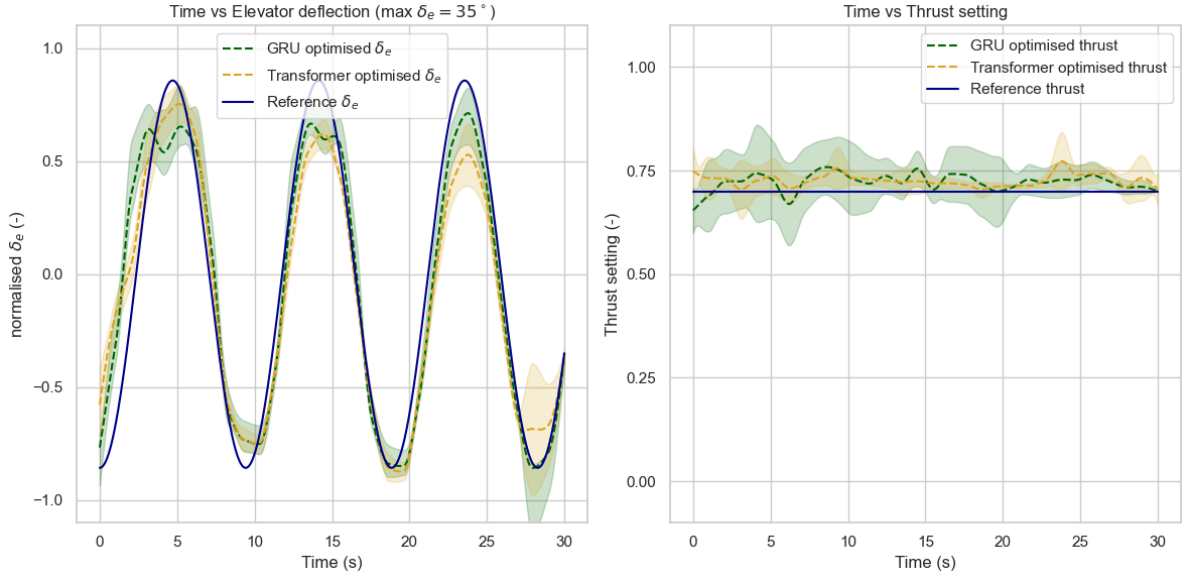


Figure D.4: Machine learning optimised controls for test case 4. The controls optimised using a GRU network have a variance (expressed as a percentage of the maximum control value) of 2.75% and 2.62% for the elevator and thrust setting respectively. For the transformer network the variance is 7.77% and 4.52% respectively.

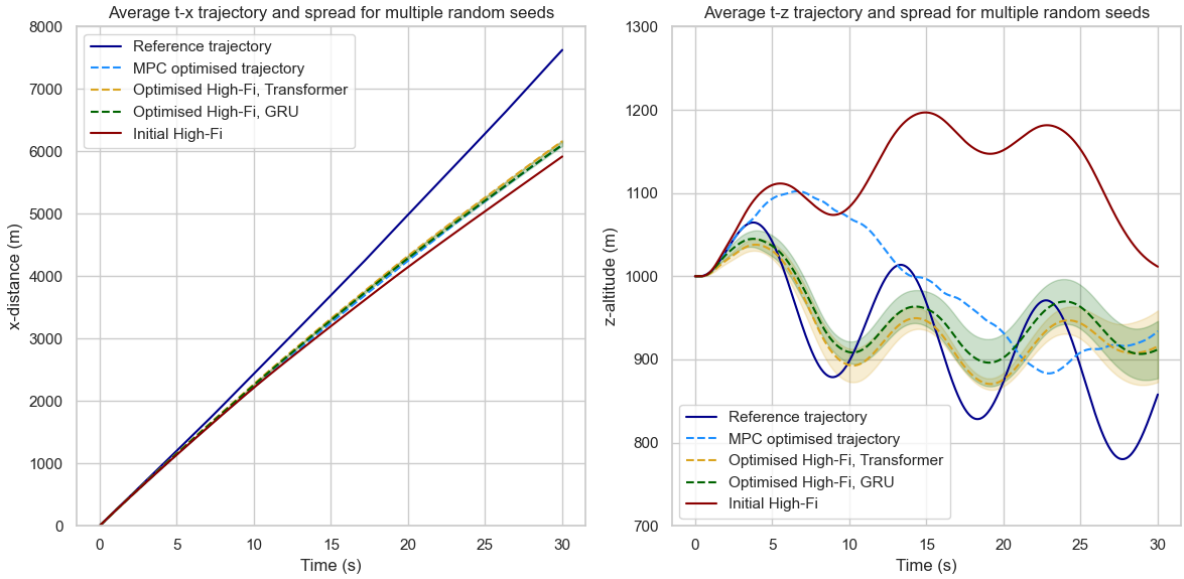


Figure D.5: Machine learning optimised high-fidelity trajectory for test case 4. For the GRU network, the x and z variance is 4.79m and 32.2m respectively. For the transformer network the variance is 12.1m and 17.5m.

In the final test case, the advantage of considering the entire trajectory at once is evident. The control optimiser in both the GRU and transformer network cases generates a trajectory that captures the overall movement of the reference trajectory, albeit at the expense of short-term fitting accuracy. Conversely, the MPC method, which optimises for a shorter planning horizon, initially provides an excellent fit but tends to quickly overshoot the reference trajectory. This demonstrates the trade-offs between different approaches in terms of their ability to balance overall trajectory alignment with immediate accuracy.

Table D.2: Optimised high-fidelity trajectory error metrics for test case 4.

| (a) GRU network | | | (b) Transformer network | | |
|------------------------|---------------------|--------------|-------------------------|---------------------|--------------|
| Error Type | Mean error μ | STD σ | Error Type | Mean error μ | STD σ |
| DTW (1e3 m) | 177.7 | 4.2 | DTW (1e3 m) | 170.5 | 9.7 |
| DFD (leash-length) (m) | 1328.0 | 12.9 | DFD (leash-length) (m) | 1297.1 | 76.3 |
| LCSS (-) | 0.743 | 0.011 | LCSS (-) | 0.743 | 0.014 |
| ELS (m) | 437.2 | 2.9 | ELS (m) | 422.7 | 27.4 |

For network performance, the GRU network consistently edges out the transformer network. This difference is noticeable in the case of the LCSS error, where the GRU network significantly outperforms the transformer. This superiority is visually apparent as the GRU network more closely adheres to the reference trajectory, especially towards the end of the sequence. Overall, the machine learning-based approach demonstrates superior results compared to the MPC method.

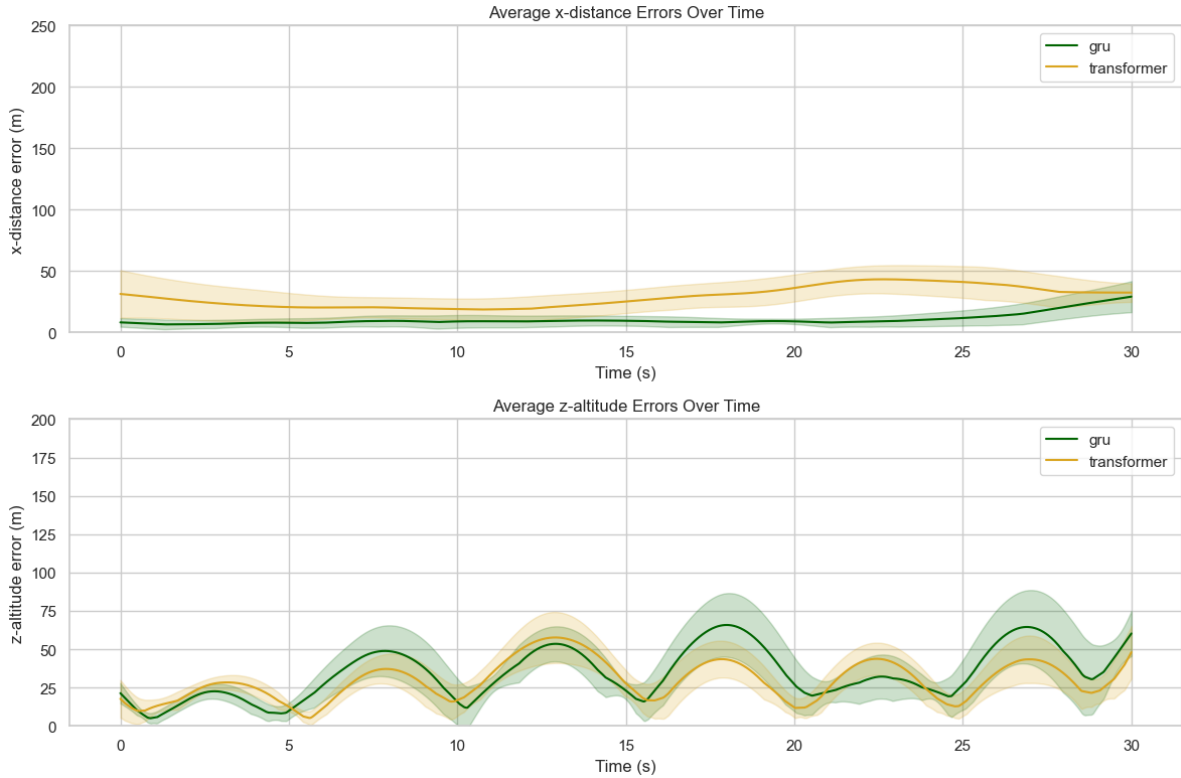


Figure D.6: Average prediction error and error variance of the GRU and transformer network for test case 3. The average GRU prediction variance is 7.1m and 19.2m for x and z dimension respectively. The average transformer network variance is 16.1m and 14.2m.

Examining the network fit of the high-fidelity model reveals that the GRU network consistently outperforms the transformer network in terms of average error and variance. Additionally, the influence of control deflection is evident in the z-deflection error, where spikes in error correspond with peaks in elevator control deflection. However, this behaviour is less pronounced in the x-distance covered when observing thrust deflection.

D.2. Active learning analysis

Test case 2

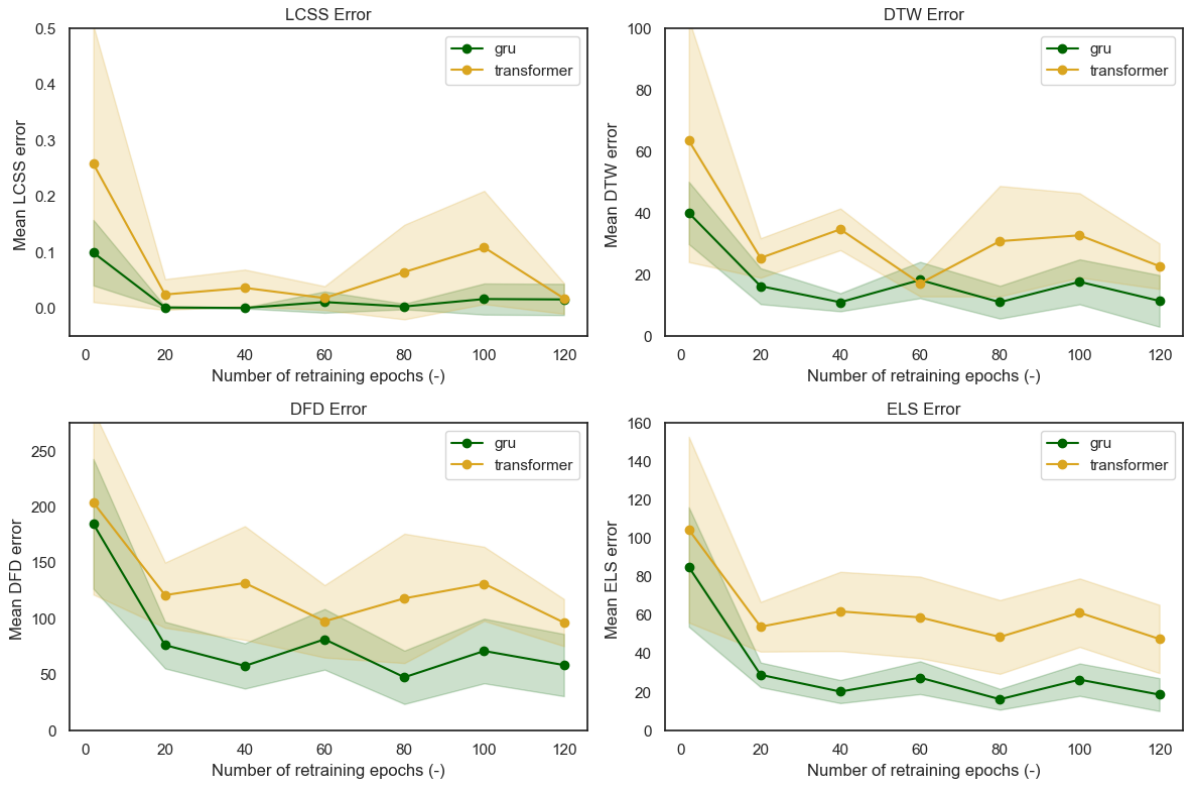


Figure D.7: Influence of the number of retraining iterations on the predictive performance for the optimised controls for the second test case. The shaded region represent the 95% confidence interval for 9 different random seeds.

When comparing the results to the first test case, it is evident that the error is significantly higher at the beginning and decreases towards the end. This observation underscores the importance of retraining and expanding the dataset to enhance prediction reliability. Similarly to previous observations, the GRU network continues to outperform the transformer network in this scenario. Moving on to the average prediction error:

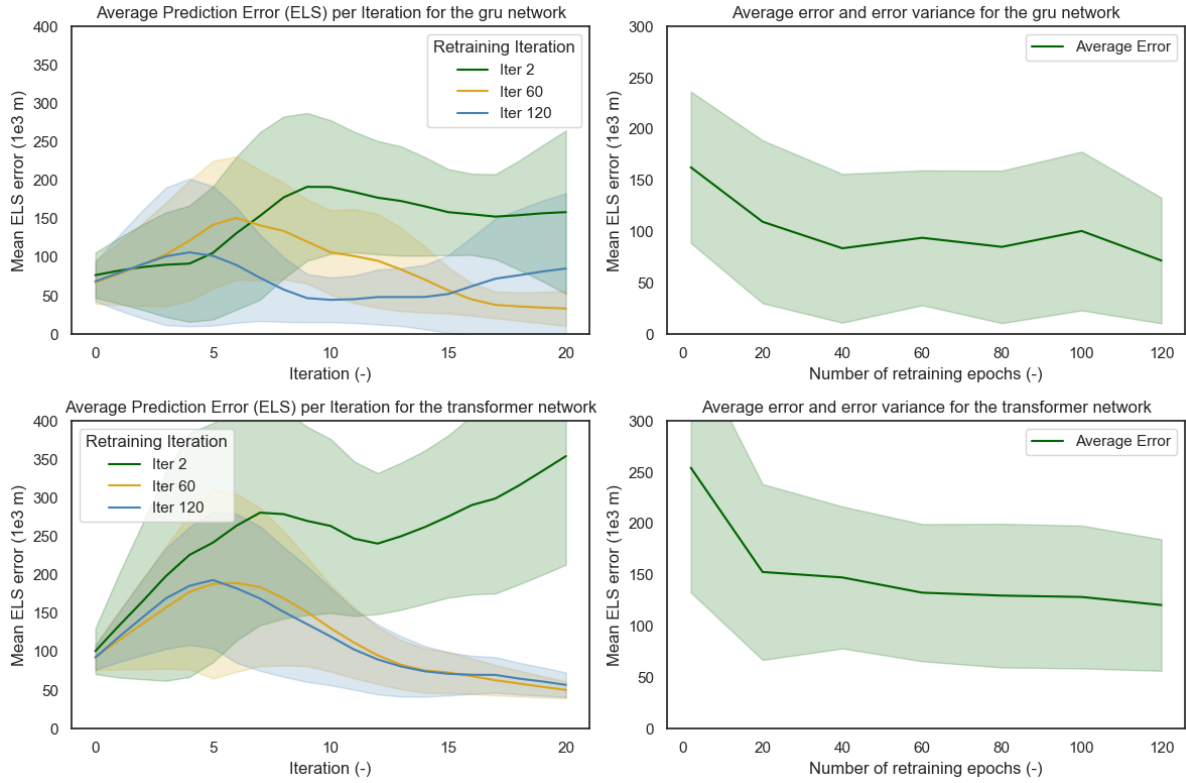


Figure D.8: Average prediction error of the networks at every iteration for multiple retraining iterations for the second test case. The shaded region represent the 95% confidence interval for 9 different random seeds.

The graph reinforces the assertion that an increased amount of training is critical for accurate predictions. Notably, in scenarios with no retraining (highlighted in green in the left column), the prediction error worsens over time, underscoring that continuous learning is essential even in seemingly simpler cases. The right column further emphasizes the need for sufficient training. Surprisingly, the confidence interval is larger than in the first test case, which can be attributed to the initial prediction accuracy of the network.

Test case 3

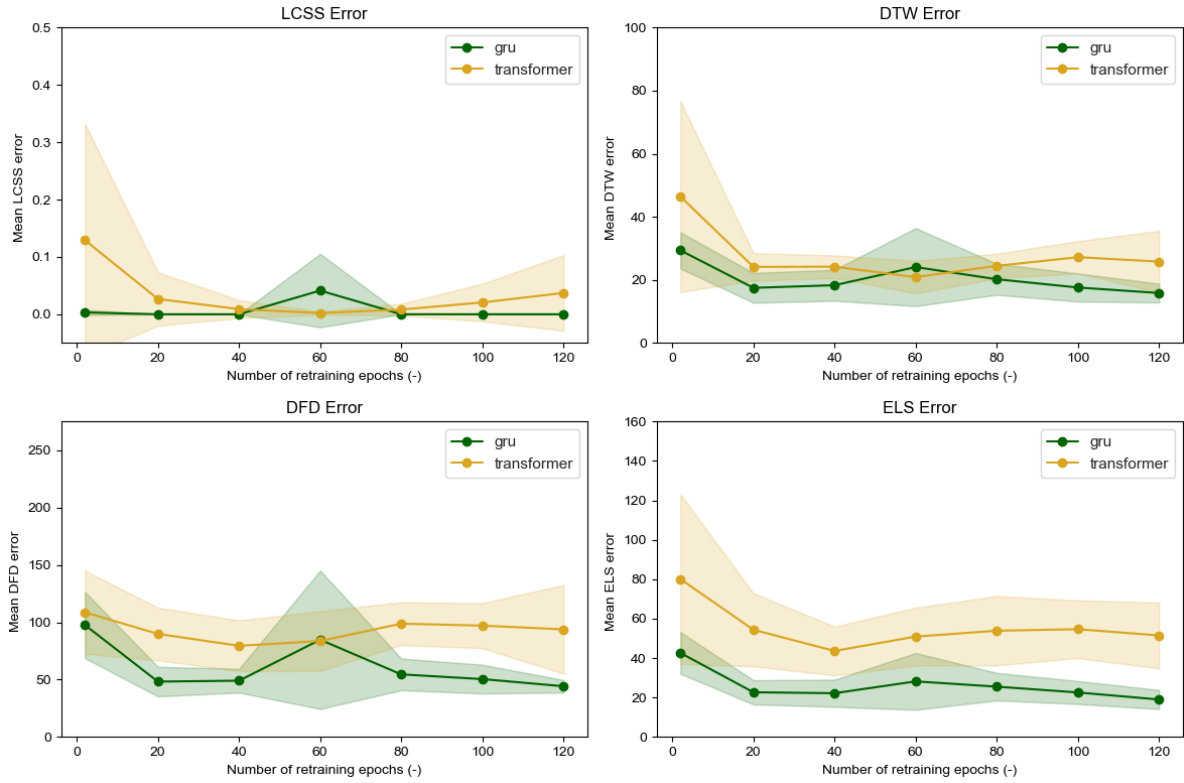


Figure D.9: Influence of the number of retraining iterations on the predictive performance for the optimised controls for the third test case. The shaded region represent the 95% confidence interval for 9 different random seeds.

In the third test case, a more complex elevator actuation is used relative to the first two scenarios. This complexity provides a rigorous test of the networks' predictive capabilities. Under these conditions, the GRU network displays an increase in prediction error, indicating its sensitivity to complex inputs. Conversely, the transformer network maintains a consistent error rate, showcasing its robustness in more demanding scenarios.

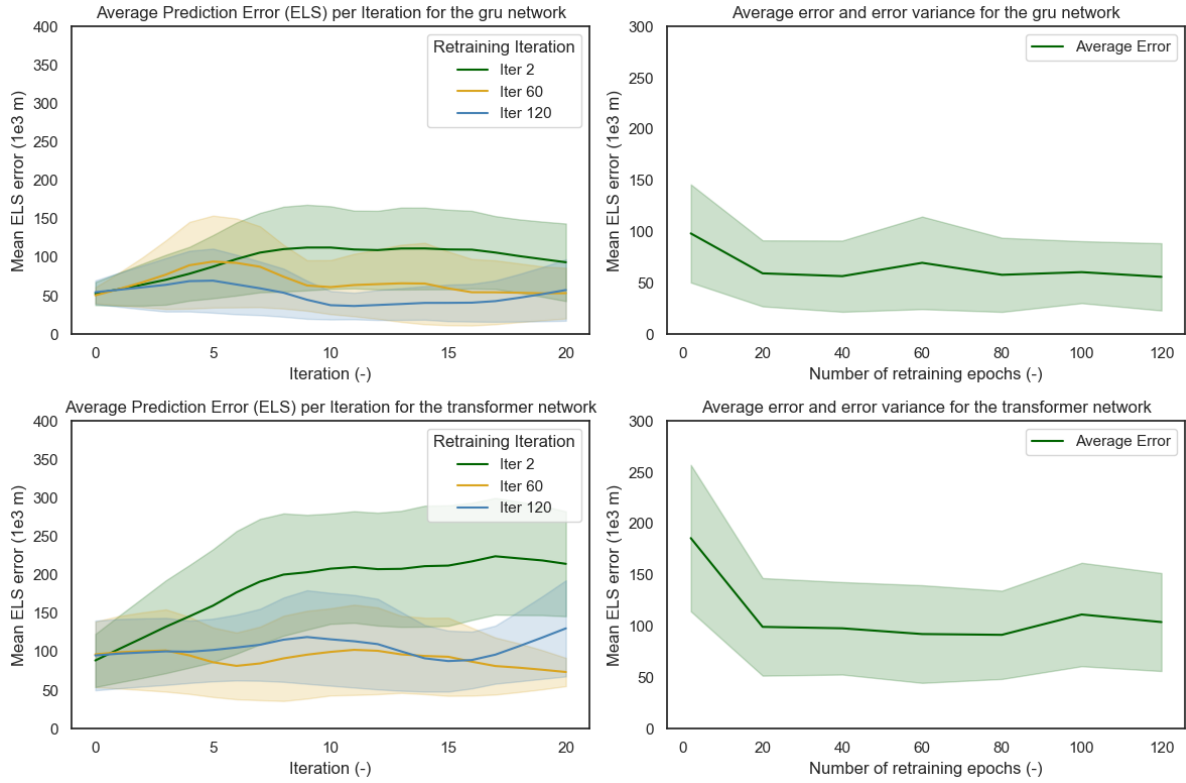


Figure D.10: Average prediction error of the networks at every iteration for multiple retraining iterations for the third test case. The shaded region represent the 95% confidence interval for 9 different random seeds.

Analysis of the prediction error per optimisation step indicates that transformers are particularly responsive to the extent of training they receive. Without significant retraining, transformers exhibit a noticeable increase in error, more so than GRU networks, which have approximately 50% fewer trainable parameters. This difference highlights the critical need for substantial training for transformers to perform optimally.