Journal of Guidance, Control, and Dynamics, Volume 47, Issue 5, May 2024, pp. 945-963
DOI:10.2514/1.G007234

Online Corrections to Neural Policy Guidance for Pinpoint Powered Descent

Namhoon Cho*, Hyo-Sang Shin†, and Antonios Tsourdos‡
Cranfield University, Cranfield, MK43 0AL, Bedfordshire, United Kingdom

Davide Amato§
Imperial College London, South Kensington Campus, SW7 2AZ, London, United Kingdom

This study presents incremental correction methods for refining neural network parameters or control functions entering into a continuous-time dynamic system to achieve improved solution accuracy in satisfying the interim point constraints placed on the performance output variables. The proposed approach is to linearise the dynamics around the baseline values of its arguments, and then to solve for the corrective input required to transfer the perturbed trajectory to precisely known or desired values at specific time points, i.e., the interim points. Depending on the type of decision variables to adjust, parameter correction and control function correction methods are developed. These incremental correction methods can be utilised as a means to compensate for the prediction errors of pre-trained neural networks in real-time applications where high accuracy of the prediction of dynamical systems at prescribed time points is imperative. In this regard, the online update approach can be useful for enhancing overall targeting accuracy of finite-horizon control subject to point constraints using a neural policy. Numerical example demonstrates the effectiveness of the proposed approach in an application to a powered descent problem at Mars.

I. Introduction

State-of-the-art guidance algorithms for entry, descent, and landing (EDL) rely on either a *reference tracking* or a *predictor-corrector* approach. In the reference tracking approach, a reference trajectory is generated offline and stored on the vehicle guidance system. Because of process noise, measurement noise, and control errors, the actual trajectory will deviate from the reference. The guidance system is tasked with assigning appropriate control inputs to bring the vehicle back onto the reference trajectory until some assigned target state. The reference trajectory approach has a rich

*Research Fellow, Centre for Autonomous and Cyber-Physical Systems, School of Aerospace, Transport and Manufacturing, n.cho@cranfield.ac.uk, AIAA Member

†Professor of Guidance, Navigation and Control, Centre for Autonomous and Cyber-Physical Systems, School of Aerospace, Transport and Manufacturing, h.shin@cranfield.ac.uk

‡Professor, Centre for Autonomous and Cyber-Physical Systems, School of Aerospace, Transport and Manufacturing, a.tsourdos@cranfield.ac.uk

§Lecturer in Spacecraft Engineering, Department of Aeronautics, Faculty of Engineering, d.amato@imperial.ac.uk

Published by AIAA. This is the Author Accepted Manuscript issued with: Creative Commons Attribution Non-Commercial License (CC:BY:NC 4.0).
The final published version (version of record) is available online at DOI:10.2514/1.G007234. Please refer to any applicable publisher terms of use.

aerospace heritage, having been used for the design of the Apollo [1] and Shuttle [2] entry and descent guidance. The Apollo powered descent guidance law consists of a polynomial thrust acceleration profile, which has been recently extended and generalised to fractional polynomials [3].

In the predictor-corrector approach, no reference trajectory is necessary. Instead, the onboard guidance system predicts the trajectory from the current state estimate to a prescribed final condition. Again because of noise and control errors, the final state will differ from the target. The guidance system then computes a control input such that the error between the final and target states is minimized. Predictor-corrector approaches include the Universal Powered Guidance (UPG) [4] algorithm, Fully Numerical Predictor-corrector Entry Guidance (FNPEG) [5], and Guidance for Fuel Optimal Large Diverts (G-FOLD) algorithms [6].

Although reference trajectory guidance is well-understood and computationally efficient for onboard implementation, it suffers from a series of drawbacks. The discrepancy between the dynamical model used in the generation of the reference trajectory and the actual dynamics experienced by the vehicle will result in a trajectory that is always sub-optimal. If deviations from the reference trajectory are sufficiently large, the available control authority may not be sufficient to recover the reference and could result in exceeding critical constraints. In practice, the GN&C system and actuators are over-designed in order to decrease the risk of failure of the mission, impacting EDL performance metrics such as landed payload mass and accuracy.

Advances in onboard processing have recently made feasible the onboard implementation of predictor-corrector guidance algorithms; notably, these are the current state-of-the-art in EDL architectures for crewed missions [7, 8]. By recomputing a nominal controlled trajectory at each guidance cycle, predictor-correctors compute are more robust to initial dispersions and uncertainty, both dynamical and navigational [9]. Margins on the GN&C system can be consequently reduced while gaining accuracy and landed payload mass. However, the dynamical model used for the prediction step is simplified in order to lower the computational cost, which leads to further sub-optimality of the solution. Most importantly in the context of this work, implementing constraints in either approach is cumbersome and must be done through ad-hoc analytical manipulations depending on the type of constraint and method. In those cases in which the guidance law is differentiable, this is generally because it has been derived under (often strict) simplified assumptions.

Neural policy guidance (NPG) has recently emerged as a novel optimal guidance paradigm [10]. The development of NPG has been motivated by the expressivity and the representation learning capability of NNs, which have been applied to the identification and control of dynamical systems. In neural policy guidance, the functional form of the control law is prescribed as a deep neural network (NN) mapping the vehicle state to the control input, and is a function of a set of parameters. This neural policy is plugged into the equations of motion following the universal differential equations approach [11]. The optimal control problem is then recast as finding the set of neural network parameters generating a control law that minimizes a cost function expressing soft constraints on a final state and propellant or

control effort; that is to say *training* the network. The OCP is solved offline, and generates a reference control policy that is evaluated online by the GN&C system.

The neural policy guidance approach combines several advantages over existing paradigms. Like in direct transcription methods for the generation of reference trajectories [12], the dynamical model used for training can be arbitrarily complex, as the training is performed offline and is not constrained by limited onboard computational resources. In NPG, the constraints expressing the dynamics are automatically satisfied by embedding the NN policy within the equations of motion. The NPG approach is robust, as the NN policy is a function of the state rather than being prescribed as a tracked profile like in a reference trajectory approach. The evaluation of the NN policy can be performed efficiently, as it consists of a sequence of affine transformations (with relatively small matrix sizes) and elementary nonlinear functions. In addition, the NN policy is an analytical function and can be differentiated easily through automatic differentiation tools. This is relevant for stochastic guidance approaches in which the sensitivity of the policy to state deviations and navigation errors is an important consideration.

The class of machine learning models integrating NNs with differential equations provides an essential modelling paradigm for optimal control using a NN policy. Neural differential equations refers to the combination of NNs with differential equations [13]. Neural differential equations can be classified depending on the type of differential equations such as Neural Ordinary Differential Equations (ODEs) [14], Neural Stochastic Differential Equations (SDEs) [15, 16], and Neural Controlled Differential Equations (CDEs) [17, 18]. Provided that continuous-time dynamic systems are usually described by Ordinary Differential Equations (ODEs), the model for the closed-loop system controlled with a NN policy is an combination of NNs and ODEs. Therefore, optimal control using a NN policy needs to deal with certain form of Neural ODEs.

Recent advances in the theory and tools for Neural ODEs have led to the improvements in the methodology to solve an optimal control problem using NN policy. The seminal work in [14] introduced Neural ODEs as a continuous-depth model which replaces a discrete sequence of hidden layers in NNs with ODE solvers. The original motivation for inventing Neural ODEs model was to parametrise the derivative of the hidden state within a NN as an analogue of the residual network architecture for deep learning tasks. With the aid of adaptive time-stepping ODE solvers, Neural ODEs demonstrated benefits including constant memory cost, input-dependent evaluation strategy selection, and explicit tradeoff between numerical precision and speed through tolerance control. The main applications of Neural ODEs are time-series modelling [17, 19, 20], continuous normalising flows [14], and modelling and control of physical systems [21–23].

Various modifications have been proposed to overcome the limitations of Neural ODEs as presented in [14] such as the class of representable functions, training speed, and stability. To relax learned representations from preservation of the input space topology, Augmented Neural ODEs presented in [24] lifts the input dimension of Neural ODEs by zero appending. The capability of Augmented Neural ODEs to learn simpler flows led to better expressivity, enhanced

stability, and improved computational efficiency. Liquid NN which has a specific structure for parameterisation of ODEs resembling non-spiking neuron dynamics also provides a more expressive yet scalable alternative to conventional Neural ODEs [20, 25]. To accelerate training speed, the shortcomings of the naive adjoint method have been addressed in [26] by introducing an interpolation-based method for efficient approximation of gradients, in [27] by developing a trajectory checkpointing strategy for more accurate gradient estimation, and in [28] by using appropriate seminorms as stepping criteria to reduce the number of unnecessary rejections. To enforce stability of Neural ODEs, control-theoretical structures have been introduced in Stable Neural Flows [29] through the structure of gradient flow on an energy functional parameterised with NN and in LyaNet [30] by using a Lyapunov loss that penalises violation of an exponential stability condition.

Despite the progresses made in the machine learning domain for continuous-depth models, the widely-used methodology for optimal control purpose based on Neural ODEs has shortcomings in problems where a high accuracy of closed-loop state solution in satisfying given constraints at certain time points is desired. Training of Neural ODEs typically leverages unconstrained optimisation of NN parameters. The process basically follows gradient descent to minimise a cost functional that depends on the state solution of ODEs, regardless of the detailed procedure for computing the gradients [10, 31]. The first-order methods for unconstrained optimisation such as ADAM [32] and NADAM [33] are widely used. In this setup, one way to train a NN policy for satisfying interim point constraints is to add terms penalising constraint violation in the objective function. However, this methodology often produces a policy that results in limited constraint satisfaction accuracy even if the weights multiplied to the penalty terms are very large. This is mainly because a soft constrained formulation allows increase in the penalty terms as a compromise to improve the overall objective function. Also, very large weights can cause numerical instabilities that may lead to degraded accuracy of gradients or even failure of training process due to overflow. For these reasons, the typical training methodology for Neural ODEs have limitations in enforcing interim point constraints in the state solution.

The aforementioned limitations render the current Neural ODEs training pipeline inappropriate particularly for aerospace applications. The problem of concern includes the guidance problems for pinpoint landing in planetary landers and target interception in missiles. Given the knowledge of the system dynamics, the possibility to directly optimise a feedback policy and the straightforwardness of gradient-descent-based design tools motivate the approach using Neural ODEs to tackle optimal guidance problems with waypoint and goal point constraints. If successful, direct policy optimisation can be simpler and more efficient in terms of development effort and in performance than trajectory optimisation followed by tracking control [34, 35] or continuous online trajectory optimisation [36–40]. It can reduce the optimality gap due to the necessity of command conversion and the restrictiveness of modelling approximations such as the inefficiency of classical explicit guidance laws based on linear quadratic regulator under the actual environments [1, 3, 41]. However, the limited targetting accuracy issue may prevent further exploration of this approach. It is necessary to overcome the limitation of soft constrained formulation for training Neural ODEs to realise the usefulness of direct

policy optimisation.

With this background, this work aims to develop methods to enhance the utility of NN policy optimisation in continuous-time control problems with interim point constraints. In principle, constrained optimisation can be performed naively for training Neural ODEs subject to constraints. However, it will add complexity to the training process in finding feasible solutions since the number of decision variables is usually large and the decision variables affect the state variables only indirectly. Online retraining of a NN policy is also unrealistic not only due to the prohibitively large amount of computation but also because of the unreliability of nonconvex optimisation done in an on-the-fly manner. Instead, the proposed approach is to apply incremental correction methods as an add-on second stage to improve constraint satisfaction accuracy while retaining the current Neural ODEs methodology for training a baseline policy. This approach takes inspirations from the philosophy of neighbouring optimal control and gradient methods for trajectory optimisation [42–44]. The proposed approach consists in local linearisation of the system dynamics around the baseline input values followed by the design of an augmentation for the input to enforce state constraints. The correction algorithms are developed in both the NN parameter space and the control function space. The online correction of parameter or control function in addition to the offline training of NN baseline policy will realise a hybrid offline-online framework combining the benefits of two paradigms: i) the versatility of direct policy optimisation, and ii) the robustness against uncertainties provided by feedback through online correction. In summary, the proposed methods can be used as a post-processing step to achieve the state constraints without changing the existing methodology for training Neural ODEs. Note that the present study does not attempt to entirely replace the existing methods that are used for constrained trajectory optimisation in many aerospace applications.

The rest of the paper is organised as follows. Section II provides a brief overview of the proposed approach consisting of baseline policy training followed by incremental correction. Section III presents the derivation of the incremental correction methods; i) parameter correction method in Sec. III.A, and ii) control function correction method in Sec. III.B. Section IV demonstrates the efficacy of the incremental correction approach through illustrative examples. Concluding remarks are summarised in Sec. V.

II. Overview of the Neural Optimal Control Problem

This study considers the problem of designing a NN policy for finite-horizon control of a known continuous-time nonlinear dynamic system subject to the interim point constraints imposed on the performance output which is linear in

the state. Mathematical formulation of the problem can be written as follows:

$$\begin{aligned} \underset{\mathbf{u}(t)}{\text{minimise}} \quad & J = T\left(t_f, \mathbf{x}\left(t_f\right)\right) + \int_{t_0}^{t_f} S\left(\tau, \mathbf{x}\left(\tau\right), \mathbf{u}\left(\tau\right)\right) d\tau \\ \text{subject to} \quad & \dot{\mathbf{x}}\left(t\right) = \mathbf{f}\left(t, \mathbf{x}\left(t\right), \mathbf{u}\left(t\right)\right), \quad \mathbf{x}\left(t_0\right) = \mathbf{x}_0 \\ & \mathbf{z}\left(t_i\right) = \mathbf{H}\mathbf{x}\left(t_i\right) = \mathbf{z}_i \qquad (i = 1, \cdots, N) \end{aligned} \tag{1}$$

where t, $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{z}$ denote the time, the state, the control input, and the performance output, respectively; T, S, $\mathbf{f}$, and $\mathbf{H}$ denote the terminal cost, the stage cost, the known ODE function, and the constant output matrix, respectively; t_0, t_i for $i = 1, \cdots, N$, and t_f denote the initial time, the set of interim points, and the final time, respectively. Note that $t_N = t_f$ in the usual setup. The overdot notation stands for the time-derivative of a quantity. A NN policy denoted by $\boldsymbol{\pi}$ refers to a function that gives the control input at each instance t as

$$\mathbf{u}\left(t\right) = \boldsymbol{\pi}\left(t, \mathbf{x}\left(t\right), \boldsymbol{\theta}\right) \tag{2}$$

where $\boldsymbol{\theta}$ denotes the vector of NN parameters. Applying constrained parameter optimisation methods for finding the NN policy parameters cannot straightforwardly solve the problem, since the constraints are imposed on the solution of the dynamic system rather than on the NN parameters that are embedded inside the differential equations. Also, random initialisation of the large number of NN parameters is likely to pose difficulties in convergence to a feasible solution with constrained trajectory optimisation methods which require initial guess for the control input. As a workaround, a widely-used methodology is to penalise constraint violations through the definition of objective function and exploit the tools for training Neural ODEs to solve the following problem

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{minimise}} \quad & J_{aug} = T\left(t_f, \mathbf{x}\left(t_f\right)\right) + \int_{t_0}^{t_f} S\left(\tau, \mathbf{x}\left(\tau\right), \mathbf{u}\left(\tau\right)\right) d\tau + P\left(\left\{\mathbf{H}\mathbf{x}\left(t_i\right) - \mathbf{z}_i\right\}_{i=1}^{N}\right) + R\left(\boldsymbol{\theta}\right) \\ \text{subject to} \quad & \dot{\mathbf{x}}\left(t\right) = \mathbf{f}\left(t, \mathbf{x}\left(t\right), \mathbf{u}\left(t\right)\right), \quad \mathbf{x}\left(t_0\right) = \mathbf{x}_0 \\ & \mathbf{u}\left(t\right) = \boldsymbol{\pi}\left(t, \mathbf{x}\left(t\right), \boldsymbol{\theta}\right) \end{aligned} \tag{3}$$

where P and R denote the terms introduced for penalisation and regularisation. However, it was observed in numerical experiments that the achieved targeting accuracy tends to leave room for improvement even when the weights given to the penalty terms are very large.

This study presents a two-stage approach to improve the accuracy of satisfying the constraints while circumventing the difficulties in considering hard constraints in NN policy training. Figure 1 shows the schematic diagram of the proposed two-stage approach. Stage 1 is to train a baseline NN policy $\boldsymbol{\pi}$ which is to find the solution $\boldsymbol{\theta}^*$ for the soft constrained problem given in Eq. (3). Stage 2 is to find the incremental correction in the NN parameter or the control

input (which will be denoted by $\tilde{\boldsymbol{\theta}}$ or $\tilde{\mathbf{u}}$ in Sec. III, respectively) for the closed-loop dynamics linearised around the baseline trajectory to satisfy the interim point constraints.

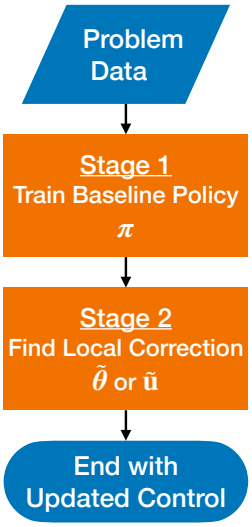


Fig. 1 Overview of Proposed Approach

In this study, the continuous-time policy gradient method developed in [10] is utilised for Stage 1. Readers are referred to [10] for the details of the adjoint sensitivity technique and the `Julia`-based package named `ContinuousTimePolicyGradients.jl` [45]. The following section presents two different methods for Stage 2.

III. Incremental Correction Methods

Given the initial state, trajectories of the closed-loop system controlled by a NN policy can be modified to satisfy performance output constraints by adding specifically designed correction to either the NN parameter or the control input. Section III.A presents the parameter correction method that finds $\tilde{\boldsymbol{\theta}}$, and Sec. III.B describes the control function correction method that finds $\tilde{\mathbf{u}}(t)$.

A. Parameter Correction

Consider the continuous-time system dynamics given by

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}_\theta(t, \mathbf{x}(t), \boldsymbol{\theta}), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{z}(t) &= \mathbf{H}\mathbf{x}(t) \end{aligned} \tag{4}$$

where t, $\mathbf{x} \in \mathbb{R}^{n\times 1}$, $\mathbf{z} \in \mathbb{R}^{p\times 1}$, and $\boldsymbol{\theta} \in \mathbb{R}^{l\times 1}$ denote the time, the state, the performance output, and the parameter vector, respectively. In Eq. (4), $\mathbf{f}_\theta$ represents the ODE function with $\boldsymbol{\theta}$ as its decision variable, which is defined by

$$\mathbf{f}_\theta(t, \mathbf{x}, \boldsymbol{\theta}) := \mathbf{f}(t, \mathbf{x}, \boldsymbol{\pi}(t, \mathbf{x}, \boldsymbol{\theta})) \tag{5}$$

where $\mathbf{f}$ is the ODE function that appears in Eq. (1). The performance output is defined as a linear combination of the state variables that needs to satisfy given constraints.

Suppose that the NN is already trained by minimising an objective function and the optimised parameter vector $\boldsymbol{\theta}^*$ is given as the result of Stage 1. Let $\mathbf{x}^*(t)$ denote the solution predicted by integrating the ODE model with $\boldsymbol{\theta}^*$ and a given initial condition $\mathbf{x}_0^*$. That is,

$$\begin{aligned} \dot{\mathbf{x}}^*(t) &= \mathbf{f}_\theta\left(t, \mathbf{x}^*(t), \boldsymbol{\theta}^*\right), \quad \mathbf{x}^*(t_0) = \mathbf{x}_0^* \\ \mathbf{z}^*(t) &= \mathbf{H}\mathbf{x}^*(t) \end{aligned} \tag{6}$$

for $\forall t \in \left[t_0, t_f\right]$. Consider a small perturbation in both the state and the parameter that can be expressed as

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{x}^*(t) + \tilde{\mathbf{x}}(t) \\ \boldsymbol{\theta} &= \boldsymbol{\theta}^* + \tilde{\boldsymbol{\theta}} \end{aligned} \tag{7}$$

where the tilde notation refers to the perturbed terms. Linearising Eq. (4) around the baseline prediction $\mathbf{x}^*(t)$ and the baseline parameter vector $\boldsymbol{\theta}^*$ yields

$$\begin{aligned} \dot{\tilde{\mathbf{x}}}(t) &\approx \left.\frac{\partial \mathbf{f}_\theta}{\partial \mathbf{x}}\right|_{\mathbf{x}^*(t), \boldsymbol{\theta}^*} \tilde{\mathbf{x}}(t) + \left.\frac{\partial \mathbf{f}_\theta}{\partial \boldsymbol{\theta}}\right|_{\mathbf{x}^*(t), \boldsymbol{\theta}^*} \tilde{\boldsymbol{\theta}} \\ &:= \mathbf{A}_\theta(t)\, \tilde{\mathbf{x}}(t) + \mathbf{B}_\theta(t)\, \tilde{\boldsymbol{\theta}} \end{aligned} \tag{8}$$

where the Jacobian matrices $\mathbf{A}_\theta(t)$ and $\mathbf{B}_\theta(t)$ are defined accordingly. The perturbed dynamic system in Eq. (8) is generally a linear time-varying (LTV) system whose solution can be written as

$$\tilde{\mathbf{x}}(t) = \boldsymbol{\Phi}(t, t_0)\, \tilde{\mathbf{x}}(t_0) + \int_{t_0}^{t} \boldsymbol{\Phi}(t, \tau)\, \mathbf{B}_\theta(\tau)\, \tilde{\boldsymbol{\theta}} d\tau \tag{9}$$

where $\boldsymbol{\Phi}(t_2, t_1)$ is the state transition matrix for transfer from t_1 to t_2 defined by

$$\dot{\boldsymbol{\Phi}}(t, t_0) = \mathbf{A}_\theta(t)\, \boldsymbol{\Phi}(t, t_0), \quad \boldsymbol{\Phi}(t_0, t_0) = \mathbf{I} \tag{10}$$

Since $\boldsymbol{\theta}$ is a constant perturbation of parameters, Eq. (9) can be rewritten as

$$\tilde{\mathbf{x}}(t) = \boldsymbol{\Phi}(t, t_0)\, \tilde{\mathbf{x}}(t_0) + \mathbf{M}(t)\, \tilde{\boldsymbol{\theta}} \tag{11}$$

with $\mathbf{M}(t) := \int_{t_0}^{t} \boldsymbol{\Phi}(t, \tau)\, \mathbf{B}_\theta(\tau)\, d\tau$. With the knowledge of $\mathbf{A}_\theta(t)$ and $\mathbf{B}_\theta(t)$, the matrix $\mathbf{M}(t)$ can be evaluated without explicitly computing the state transition function but instead by integrating the following differential equation

once from t_0 to t_f:

$$\dot{\mathbf{M}}(t) = \mathbf{A}_\theta(t)\,\mathbf{M}(t) + \mathbf{B}_\theta(t), \quad \mathbf{M}(t_0) = \mathbf{0} \tag{12}$$

It is obvious from Eq. (11) that the state perturbation at a time point t is related linearly with the parameter perturbation $\tilde{\boldsymbol{\theta}}$. Also note from Eq. (11) that the matrix $\mathbf{M}(t)$ is the first-order sensitivity of the state solution with respect to the perturbation in parameters $\frac{\partial \mathbf{x}(t)}{\partial \boldsymbol{\theta}}$ at time t.

Let the interim point constraints imposed on the performance output variables at some time points t_i be given by

$$\mathbf{z}(t_i) = \mathbf{z}_i \qquad (i = 1, \cdots, N). \tag{13}$$

The baseline trajectory does not generally satisfy the interim point constraints. The main concept of the proposed approach is to find a parameter correction $\tilde{\boldsymbol{\theta}}$ such that the updated parameter $\boldsymbol{\theta}$ will bring the updated prediction $\mathbf{z}(t)$ on $\mathbf{z}_i$ at each t_i. In other words, $\tilde{\mathbf{z}}(t_i) = \mathbf{H}\tilde{\mathbf{x}}(t_i) = \mathbf{z}_i - \mathbf{z}^*(t_i)$ should be satisfied by the design of $\tilde{\boldsymbol{\theta}}$. In view of Eq. (11), the required correction can simply be obtained by concatenation of constraint relations followed by the Moore-Penrose generalised inverse, i.e.,

$$\tilde{\boldsymbol{\theta}} = \mathbf{L}^\dagger (\mathbf{Z}_h - \mathbf{F}_h) \tag{14}$$

where

$$\begin{aligned} \mathbf{L} &:= \mathbf{H}\begin{bmatrix} \mathbf{M}(t_1) & \cdots & \mathbf{M}(t_N) \end{bmatrix} \\ \mathbf{Z}_h &:= \begin{bmatrix} \mathbf{z}_1 - \mathbf{z}^*(t_1) & \cdots & \mathbf{z}_N - \mathbf{z}^*(t_N) \end{bmatrix} \\ \mathbf{F}_h &:= \mathbf{H}\begin{bmatrix} \boldsymbol{\Phi}(t_1, t_0)\,\tilde{\mathbf{x}}(t_0) & \cdots & \boldsymbol{\Phi}(t_N, t_0)\,\tilde{\mathbf{x}}(t_0) \end{bmatrix} \end{aligned} \tag{15}$$

and † refers to the pseudoinverse. By using parameter correction in Stage 2, the updated control input can be obtained as

$$\mathbf{u}(t) = \boldsymbol{\pi}\left(t, \mathbf{x}(t), \boldsymbol{\theta}^* + \tilde{\boldsymbol{\theta}}\right) \tag{16}$$

Remark 1. In many cases of employing a deep NN, usually l far exceeds n. Thus, an underdetermined system of linear equations needs to be solved for the parameter correction. In this regard, it is sensible to utilise the Moore-Penrose generalised inverse as it produces the minimum-Frobenius-norm solution.

Remark 2. In modern scientific computing environments, the Jacobians of the ODE function involving a NN can be computed by using symbolic or automatic differentiation tools. One should determine which of the available local sensitivity analysis methods suits the purpose depending on the application. Also, the Jacobian array $\mathbf{L}$ in Eq. (14) is a large fat matrix since the number of NN parameters is usually much greater than the number of control inputs in physical systems. Its pseudoinverse computation might show considerable dependence on the relative tolerance setting.

Therefore, the tolerance should be selected appropriately to improve constraint satisfaction accuracy reliably.

Remark 3. Parameter correction can also be performed by exploiting local forward sensitivity analysis, which directly gives the Jacobian of state solution with respect to parameters along time [46]. In modern differentiable programming environment, automatic differentiation through an ODE solver can simply be employed for the purpose of obtaining the sensitivity matrix $\left.\frac{\partial \mathbf{x}^*(t_i)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^*}$ instead of directly constructing the dynamics Jacobian matrices $\mathbf{A}_\theta(t)$ and $\mathbf{B}_\theta(t)$ then solving Eq. (12) for $\mathbf{M}(t_i)$.

B. Control Function Correction

Consider the continuous-time system dynamics given by

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{u}(t) &= \boldsymbol{\pi}(t, \mathbf{x}(t), \boldsymbol{\theta}) \\ \mathbf{z}(t) &= \mathbf{H}\mathbf{x}(t) \end{aligned} \tag{17}$$

where t, $\mathbf{x} \in \mathbb{R}^{n\times 1}$, $\mathbf{u} \in \mathbb{R}^{m\times 1}$, $\mathbf{z} \in \mathbb{R}^{p\times 1}$, and $\boldsymbol{\theta} \in \mathbb{R}^{l\times 1}$ denote the time, the state, the control input, the performance output, and the parameter, respectively. In Eq. (17), $\mathbf{f}$ represents the ODE function with $\mathbf{u}$ as its decision variable, which appears in Eq. (1).

One may consider perturbing $\mathbf{f}$ to find a control function correction instead of performing parameter correction through the linearisation of the parameter-embedded form ODE function $\mathbf{f}_\theta$. Suppose that the optimised parameter vector $\boldsymbol{\theta}^*$ is given a priori as the result of NN training in Stage 1 based on unconstrained optimisation. Let $\mathbf{x}^*(t)$ and $\mathbf{u}^*(t)$ denote the baseline state and control input, respectively, that are obtained by forward propagation of Eq. (17) with given $\boldsymbol{\theta}^*$ and $\mathbf{x}_0^*$. Then, the following relation holds

$$\begin{aligned} \dot{\mathbf{x}}^*(t) &= \mathbf{f}(t, \mathbf{x}^*(t), \mathbf{u}^*(t)), \quad \mathbf{x}^*(t_0) = \mathbf{x}_0^* \\ \mathbf{u}^*(t) &= \boldsymbol{\pi}(t, \mathbf{x}^*(t), \boldsymbol{\theta}^*) \\ \mathbf{z}^*(t) &= \mathbf{H}\mathbf{x}^*(t) \end{aligned} \tag{18}$$

for $\forall t \in \left[t_0, t_f\right]$. Now, a small perturbation in both the state and the control input defined by

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{x}^*(t) + \tilde{\mathbf{x}}(t) \\ \mathbf{u}(t) &= \mathbf{u}^*(t) + \tilde{\mathbf{u}}(t) \end{aligned} \tag{19}$$

leads to the linearisation of Eq. (17) around the baseline state $\mathbf{x}^*(t)$ and the baseline control input $\mathbf{u}^*(t)$ as

$$\begin{aligned}\dot{\tilde{\mathbf{x}}}(t) &\approxeq \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}^*(t),\mathbf{u}^*(t)} \tilde{\mathbf{x}}(t) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{\mathbf{x}^*(t),\mathbf{u}^*(t)} \tilde{\mathbf{u}}(t) \\ &:= \mathbf{A}_u(t)\,\tilde{\mathbf{x}}(t) + \mathbf{B}_u(t)\,\tilde{\mathbf{u}}(t)\end{aligned} \tag{20}$$

The Jacobian matrices $\mathbf{A}_u(t)$ and $\mathbf{B}_u(t)$ are defined appropriately according to Eq. (20). The solution of the perturbed dynamic system in Eq. (20) is related to the control function correction as

$$\tilde{\mathbf{x}}(t) = \mathbf{\Phi}(t, t_0)\,\tilde{\mathbf{x}}(t_0) + \int_{t_0}^{t} \mathbf{\Phi}(t, \tau)\,\mathbf{B}_u(\tau)\,\tilde{\mathbf{u}}(\tau)\,d\tau \tag{21}$$

where $\mathbf{\Phi}(t_2, t_1)$ represents the state transition matrix associated with $\mathbf{A}_u(t)$.

The proposed approach is to find a minimal amount of control function correction to satisfy the interim point constraints by solving the following function space optimisation problem:

$$\begin{aligned}&\text{minimise} && \tilde{J} = \frac{1}{2}\int_{t_0}^{t_f} \tilde{\mathbf{u}}^T(\tau)\,\mathbf{R}(\tau)\,\tilde{\mathbf{u}}(\tau)\,d\tau \\ &\text{subject to} && \tilde{\mathbf{z}}(t_i) = \mathbf{H}\mathbf{\Phi}(t_i, t_0)\,\tilde{\mathbf{x}}(t_0) + \mathbf{H}\int_{t_0}^{t_i} \mathbf{\Phi}(t_i, \tau)\,\mathbf{B}_u(\tau)\,\tilde{\mathbf{u}}(\tau)\,d\tau \\ &&& \quad = \mathbf{z}_i - \mathbf{z}^*(t_i) \qquad (i = 1, \cdots, N)\end{aligned} \tag{22}$$

where $\mathbf{R}(t) = \mathbf{R}^T(t) \succ 0$ is a weighting function. The interim point indices are sorted in the order of increasing time, i.e., $t_0 \le t_1 < \cdots < t_N \le t_f$. The problem can be solved by using various linear control approaches. Applying the method of constraint-coupling Lagrange multipliers, one can define the augmented cost function as

$$\begin{aligned}\tilde{J}_a := &\frac{1}{2}\int_{t_0}^{t_f} \tilde{\mathbf{u}}^T(\tau)\,\mathbf{R}(\tau)\,\tilde{\mathbf{u}}(\tau)\,d\tau \\ &+ \sum_{i=1}^{N} \boldsymbol{\mu}_i{}^T \left(\mathbf{z}_i - \mathbf{z}^*(t_i) - \mathbf{H}\mathbf{\Phi}(t_i, t_0)\,\tilde{\mathbf{x}}(t_0) - \mathbf{H}\int_{t_0}^{t_i} \mathbf{\Phi}(t_i, \tau)\,\mathbf{B}_u(\tau)\,\tilde{\mathbf{u}}(\tau)\,d\tau\right)\end{aligned} \tag{23}$$

where $\boldsymbol{\mu}_i \in \mathbb{R}^{p\times 1}$ represents the Lagrange multiplier vector associated with the i-th performance output constraint. The upper limit of the integral for each interim point constraint can be lifted up to t_f by rewriting Eq. (23) as

$$\begin{aligned}\tilde{J}_a := &\sum_{i=1}^{N} \boldsymbol{\mu}_i{}^T \left(\mathbf{z}_i - \mathbf{z}^*(t_i) - \mathbf{H}\mathbf{\Phi}(t_i, t_0)\,\tilde{\mathbf{x}}(t_0)\right) \\ &+ \int_{t_0}^{t_f} \left[\frac{1}{2}\tilde{\mathbf{u}}^T(\tau)\,\mathbf{R}(\tau)\,\tilde{\mathbf{u}}(\tau) - \sum_{i=1}^{N} \boldsymbol{\mu}_i{}^T 1(\tau \le t_i)\,\mathbf{H}\mathbf{\Phi}(t_i, \tau)\,\mathbf{B}_u(\tau)\,\tilde{\mathbf{u}}(\tau)\right] d\tau\end{aligned} \tag{24}$$

with the activator function defined by

$$1\left(t \leq t_i\right) := \begin{cases} 1 & \text{if } t \leq t_i \\ 0 & \text{if } t > t_i \end{cases} \tag{25}$$

The first variation of the augmented cost function $\delta \tilde{J}_a$ vanishes at the optimal solution. This leads to the set of equations describing the first-order necessary condition for optimality. More specifically, the first necessary condition requires $\frac{\partial\left(\text{integrand of } \tilde{J}_a\right)}{\partial(\tilde{\mathbf{u}}(\tau))} = \mathbf{0}$ to hold, which results in

$$\begin{aligned} \tilde{\mathbf{u}}\left(\tau\right) &= \mathbf{R}^{-1}\left(\tau\right) {\mathbf{B}_u}^T\left(\tau\right) \sum_{i=1}^{N} \mathbf{\Phi}^T\left(t_i, \tau\right) \mathbf{H}^T 1\left(\tau \leq t_i\right) \boldsymbol{\mu}_i \\ &= \mathbf{R}^{-1}\left(\tau\right) {\mathbf{B}_u}^T\left(\tau\right) \mathbf{E}\left(\tau\right) \bar{\boldsymbol{\mu}} \end{aligned} \tag{26}$$

where

$$\mathbf{E}\left(\tau\right) := \left[\mathbf{\Phi}^T\left(t_1, \tau\right) \mathbf{H}^T 1\left(\tau \leq t_1\right) \quad \cdots \quad \mathbf{\Phi}^T\left(t_N, \tau\right) \mathbf{H}^T 1\left(\tau \leq t_N\right)\right] \tag{27}$$

$$\bar{\boldsymbol{\mu}} := \left[{\boldsymbol{\mu}_1}^T \quad \cdots \quad {\boldsymbol{\mu}_N}^T\right]^T \tag{28}$$

As the second necessary condition, $\frac{\partial \tilde{J}_a}{\partial \boldsymbol{\mu}_i} = \mathbf{0}$ should be satisfied $\forall\, i = 1, \cdots N$, which can be rewritten as

$$\mathbf{H} \int_{t_0}^{t_f} 1\left(\tau \leq t_i\right) \mathbf{\Phi}\left(t_i, \tau\right) \mathbf{B}_u\left(\tau\right) \tilde{\mathbf{u}}\left(\tau\right) d\tau = \mathbf{z}_i - \mathbf{z}^*\left(t_i\right) - \mathbf{H}\mathbf{\Phi}\left(t_i, t_0\right) \tilde{\mathbf{x}}\left(t_0\right) \tag{29}$$

for $i = 1, \cdots, N$. Substituting Eq. (26) into Eq. (29) gives

$$\begin{aligned} &\mathbf{z}_i - \mathbf{z}^*\left(t_i\right) - \mathbf{H}\mathbf{\Phi}\left(t_i, t_0\right) \tilde{\mathbf{x}}\left(t_0\right) \\ &= \int_{t_0}^{t_f} 1\left(\tau \leq t_i\right) \mathbf{H}\mathbf{\Phi}\left(t_i, \tau\right) \mathbf{B}_u\left(\tau\right) \mathbf{R}^{-1}\left(\tau\right) {\mathbf{B}_u}^T\left(\tau\right) \sum_{j=1}^{N} \mathbf{\Phi}^T\left(t_j, \tau\right) \mathbf{H}^T 1\left(\tau \leq t_j\right) \boldsymbol{\mu}_j d\tau \\ &= \sum_{j=1}^{N} \int_{t_0}^{t_f} 1\left(\tau \leq t_i\right) 1\left(\tau \leq t_j\right) \mathbf{H}\mathbf{\Phi}\left(t_i, \tau\right) \mathbf{B}_u\left(\tau\right) \mathbf{R}^{-1}\left(\tau\right) {\mathbf{B}_u}^T\left(\tau\right) \mathbf{\Phi}^T\left(t_j, \tau\right) \mathbf{H}^T d\tau \boldsymbol{\mu}_j \\ &:= \sum_{j=1}^{N} \mathbf{\Psi}_{ij} \boldsymbol{\mu}_j \end{aligned} \tag{30}$$

where

$$\mathbf{\Psi}_{ij} := \mathbf{H} \int_{t_0}^{\min\left(t_i, t_j\right)} \mathbf{\Phi}\left(t_i, \tau\right) \mathbf{B}_u\left(\tau\right) \mathbf{R}^{-1}\left(\tau\right) {\mathbf{B}_u}^T\left(\tau\right) \mathbf{\Phi}^T\left(t_j, \tau\right) d\tau \mathbf{H}^T \tag{31}$$

Note that the following property of the product of activator functions is used in the derivation of Eq. (31).

$$\int_{t_0}^{t_f} 1\left(t \leq t_i\right) 1\left(t \leq t_j\right) f\left(t\right) dt = \int_{t_0}^{\min\left(t_i, t_j\right)} f\left(t\right) dt \tag{32}$$

Although the coefficient matrix $\mathbf{\Psi}_{ij}$ is expressed as a definite integral in Eq. (31), it is more convenient to evaluate $\mathbf{\Psi}_{ij}$ through propagation of associated differential equation instead of directly computing the integral using quadrature methods. The integrand contains $\mathbf{\Phi}\left(t_i, \tau\right)$ which is already a result of propagation, and therefore, Eq. (31) can be rewritten as

$$\mathbf{\Psi}_{ij} = \mathbf{H}\mathbf{U}\left(t_i\right)\mathbf{N}\left(\min\left(t_i, t_j\right)\right)\mathbf{U}^T\left(t_j\right)\mathbf{H}^T \tag{33}$$

with the fundamental solution matrix $\mathbf{U}\left(t\right)$ and the auxiliary matrix $\mathbf{N}\left(t\right)$ defined according to

$$\dot{\mathbf{U}}\left(t\right) = \mathbf{A}_u\left(t\right)\mathbf{U}\left(t\right), \qquad \mathbf{U}\left(t_0\right) = \mathbf{I} \tag{34}$$

$$\dot{\mathbf{N}}\left(t\right) = \mathbf{U}^{-1}\left(t\right)\mathbf{B}_u\left(t\right)\mathbf{R}^{-1}\left(t\right)\mathbf{B}_u{}^T\left(t\right)\mathbf{U}^{-T}\left(t\right), \qquad \mathbf{N}\left(t_0\right) = \mathbf{0} \tag{35}$$

Indeed, Eq. (33) can be evaluated for each combination of i and j through a single forward sweep of the differential equations in Eqs. (34) and (35). Therefore, the vertical concatenation of the relation in Eq. (30) followed by the matrix inverse yields the constraint-coupling multipliers as

$$\bar{\boldsymbol{\mu}} = \bar{\mathbf{\Psi}}^{-1}\left(\mathbf{Z}_v - \mathbf{F}_v\right) \tag{36}$$

with

$$\bar{\mathbf{\Psi}} := \begin{bmatrix} \mathbf{\Psi}_{11} & \mathbf{\Psi}_{12} & \cdots & \mathbf{\Psi}_{1N} \\ \mathbf{\Psi}_{21} & \mathbf{\Psi}_{22} & \cdots & \mathbf{\Psi}_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{\Psi}_{N1} & \mathbf{\Psi}_{N2} & \cdots & \mathbf{\Psi}_{NN} \end{bmatrix}, \quad \mathbf{Z}_v := \begin{bmatrix} \mathbf{z}_1 - \mathbf{z}^*\left(t_1\right) \\ \cdots \\ \mathbf{z}_N - \mathbf{z}^*\left(t_N\right) \end{bmatrix}, \quad \mathbf{F}_v := \begin{bmatrix} \mathbf{H}\mathbf{\Phi}\left(t_1, t_0\right)\tilde{\mathbf{x}}\left(t_0\right) \\ \cdots \\ \mathbf{H}\mathbf{\Phi}\left(t_N, t_0\right)\tilde{\mathbf{x}}\left(t_0\right) \end{bmatrix} \tag{37}$$

Finally, substituting Eq. (36) back into Eq. (26) yields the optimal control function correction as follows:

$$\tilde{\mathbf{u}}\left(t\right) = \mathbf{R}^{-1}\left(t\right)\mathbf{B}_u{}^T\left(t\right)\mathbf{E}\left(t\right)\bar{\mathbf{\Psi}}^{-1}\left(\mathbf{Z}_v - \mathbf{F}_v\right) \tag{38}$$

By using control function correction in Stage 2, the updated control input can be obtained as

$$\mathbf{u}\left(t\right) = \boldsymbol{\pi}\left(t, \mathbf{x}\left(t\right), \boldsymbol{\theta}^*\right) + \tilde{\mathbf{u}}\left(t\right) \tag{39}$$

Remark 4. If only a single interim point constraint is given for the final time t_f, the control function correction reduces to

$$\tilde{\mathbf{u}}\left(t\right) = \mathbf{R}^{-1}\left(t\right)\mathbf{B}_u{}^T\left(t\right)\mathbf{\Phi}^T\left(t_f, t\right)\mathbf{H}^T\mathbf{\Psi}^{-1}\left(\mathbf{z}_f - \mathbf{z}^*\left(t_f\right) - \mathbf{H}\mathbf{\Phi}\left(t_f, t_0\right)\tilde{\mathbf{x}}\left(t_0\right)\right) \tag{40}$$

where

$$\boldsymbol{\Psi} = \mathbf{H} \int_{t_0}^{t_f} \boldsymbol{\Phi}\left(t_f, \tau\right) \mathbf{B}_u\left(\tau\right) \mathbf{R}^{-1}\left(\tau\right) \mathbf{B}_u{}^T\left(\tau\right) \boldsymbol{\Phi}^T\left(t_f, \tau\right) d\tau \mathbf{H}^T \tag{41}$$

which is the usual linear quadratic regulator solution for the terminal control problem.

Remark 5. The proposed incremental correction method can be slightly modified to solve problems with the interim point constraints imposed on the output $\mathbf{z}\left(t\right) = \mathbf{h}\left(\mathbf{x}\left(t\right)\right)$ for a nonlinear function $\mathbf{h}\left(\cdot\right)$. This can be done by computing the influence functions, i.e., sensitivity functions, associated with the output variables. In the special case where only a single constraint is imposed on the output at the final time, the procedure to determine control function correction becomes identical to the generalised model predictive static programming technique [47].

Remark 6. The control function correction method can suffer from numerical instabilities in propagation of Eqs. (34) and (35) to obtain matrices $\mathbf{U}\left(t\right)$ and $\mathbf{N}\left(t\right)$, respectively, when the ODE solver employs too tight relative and absolute tolerance setting. Although the choice of weighting function $\mathbf{R}\left(\tau\right)$ itself is not a source of computational challenges, appropriate selection that can bring improvements in the constraint targeting accuracy as expected might be challenging when coupled with the aforementioned numerical instabilities.

Remark 7. Sections III.A and III.B do not necessarily assume $\mathbf{x}_0 = \mathbf{x}_0^*$, thus the term $\tilde{\mathbf{x}}\left(t_0\right)$ appears in the command equations. The initial state perturbation term realises tracking error feedback when t_0 and $\mathbf{x}_0$ in the expressions are viewed as the current time and state, respectively, at each instance. Therefore, the control laws can be implemented in a feedback form based on continuous re-initialisation at the expense of increased online computation burden.

IV. Application to Pinpoint Powered Descent Guidance

This section presents examples to demonstrate the effectiveness of the proposed correction methods in a practical application. The problem of powered descent guidance for pinpoint landing on Mars is considered for illustration as the landing accuracy is of interest in this problem [48].

A. Problem Description

The following sections will provide the elements that are necessary to completely describe the optimisation problem for NN policy design and incremental correction; i) system model (Sec. IV.A.1), ii) policy (Sec. IV.A.2), iii) constraints (Sec. IV.A.3), iv) objective function (Sec. IV.A.4).

1. System Model

Consider the vehicle motion with respect to a Mars-centred, Mars-fixed coordinate system. The spacecraft dynamics can be expressed as

$$
\begin{aligned}
\dot{\mathbf{r}} &= \mathbf{v} \\
\dot{\mathbf{v}} &= -\frac{\mu}{r^3}\mathbf{r} + \frac{\mathbf{F}_L + \mathbf{F}_D + \mathbf{F}_T}{m} - 2\boldsymbol{\Omega} \times \mathbf{v} - \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \mathbf{r}) \\
\dot{m} &= -\frac{T}{I_{sp} g}
\end{aligned}
\tag{42}
$$

where $\mathbf{r}$ and $\mathbf{v}$ denote the position and velocity vectors with respect to the centre of Mars, respectively, $r := \|\mathbf{r}\|$ denotes the radial distance, $\boldsymbol{\Omega} := \begin{bmatrix} 0 & 0 & \Omega \end{bmatrix}^T$ denotes the angular velocity for Mars rotation, m denotes the vehicle mass. The lift, drag, and thrust forces are represented by $\mathbf{F}_L$, $\mathbf{F}_D$, and $\mathbf{F}_T$ in Eq. (42), respectively, with $T := \|\mathbf{F}_T\|$, I_{sp}, and g representing the thrust magnitude, specific impulse, and gravitational acceleration at the Earth surface, respectively.

Consider the wind axes and the force direction angles that are defined according to Fig. 2. The three-dimensional force vectors can be described as

$$
\begin{aligned}
\mathbf{F}_L &= L \left(\cos\sigma_L \frac{\mathbf{v}_a \times \mathbf{r}}{\|\mathbf{v}_a \times \mathbf{r}\|} \times \hat{\mathbf{v}}_a + \sin\sigma_L \frac{\mathbf{v}_a \times \mathbf{r}}{\|\mathbf{v}_a \times \mathbf{r}\|} \right) \\
\mathbf{F}_D &= -D\hat{\mathbf{v}}_a \\
\mathbf{F}_T &= T \left(\cos\eta_T \cos\sigma_T \frac{\mathbf{v}_a \times \mathbf{r}}{\|\mathbf{v}_a \times \mathbf{r}\|} \times \hat{\mathbf{v}}_a + \cos\eta_T \sin\sigma_T \frac{\mathbf{v}_a \times \mathbf{r}}{\|\mathbf{v}_a \times \mathbf{r}\|} + \sin\eta_T \hat{\mathbf{v}}_a \right)
\end{aligned}
\tag{43}
$$

where $\mathbf{v}_a := \mathbf{v} - \mathbf{v}_w$ denotes the relative velocity with respect to the surrounding air flow of $\mathbf{v}_w$, while the hat notation $\hat{(\cdot)}$ refers to the unit vector in the direction of a quantity. Also, σ_T and η_T are the azimuth and elevation angles for the thrust, respectively, and σ_L is the bank angle.

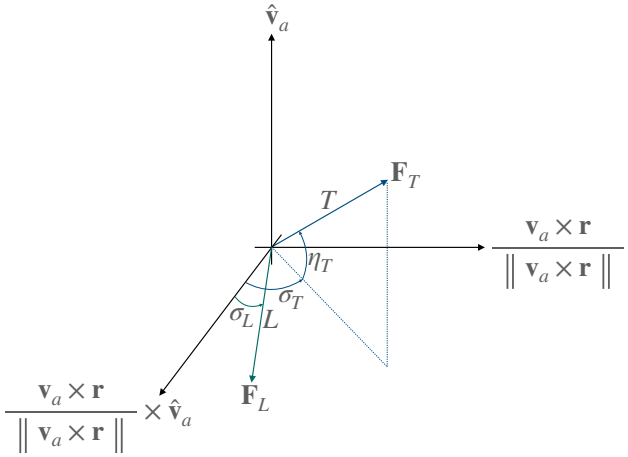


Fig. 2 Lift and Thrust Force Resolved in Wind Axes

The magnitudes of the lift and drag can be expressed as

$$
\begin{aligned}
L &= \frac{1}{2}\rho\|\mathbf{v}_a\|^2 C_L S = R_{L/D} D \\
D &= \tfrac{1}{2}\rho\|\mathbf{v}_a\|^2 C_D S = \tfrac{m}{2\beta}\rho\|\mathbf{v}_a\|^2
\end{aligned}
\qquad \text{where} \quad \rho(h) = \rho_0 \exp\left(-\frac{h}{H}\right)
\tag{44}
$$

where $R_{L/D}$ is the lift-to-drag ratio, $\beta(t) = \frac{m(t)}{C_D S}$ is the ballistic coefficient of the vehicle, and ρ is the atmospheric density that is approximately modelled as a decreasing exponential function of the altitude h measured above the surface.

The thrust model is given by

$$T = f_{\text{sw}}\left(m, m_{\text{dry}}, m_{\text{sw}}\right) T_{\max} \delta_T \tag{45}$$

where f_{sw} is a mass-dependent activation function that takes value in $[0, 1]$, $T_{\max}$ denotes the maximum thrust magnitude, and δ_T denotes the throttle command that scales between $[0, 1]$. The activation function is introduced to model the impossibility to produce thrust in the absence of fuel. A smooth activation function defined by

$$f_{\text{sw}}\left(m, m_{\text{dry}}, m_{\text{sw}}\right) = \frac{1 - \cos\left(\frac{\min\left(\max\left(m - m_{\text{dry}}, 0\right), m_{\text{sw}}\right)}{m_{\text{sw}}} \pi\right)}{2} \tag{46}$$

is introduced instead of the discrete switching function to keep the gradient computation required in backpropagation through the ODE dynamics well-defined. In Eq. (46), m_{dry} denotes the dry mass, and $m_{\text{sw}} > 0$ denotes the design parameter which determines the steepness of the activation function. Figure 3 shows an example plot of the thrust activation function in Eq. (46) with $m_{\text{dry}} = 51\,600\text{kg}$ and $m_{\text{sw}} = 1\text{kg}$.

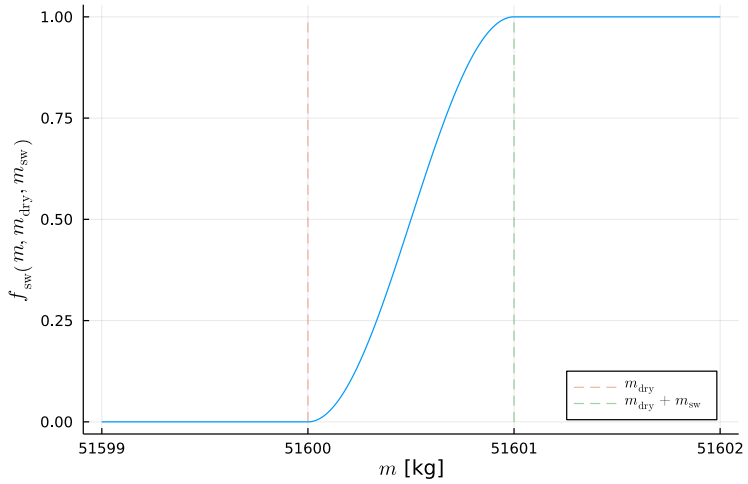


Fig. 3 Illustrative Example of Thrust Activation Function

2. Policy

The policy is defined to be a fully-connected feedforward NN which takes the normalised position and velocity errors as inputs and generates the throttle command and thrust direction angles as its outputs.

$$\begin{bmatrix} \delta_T & \sigma_T & \eta_T \end{bmatrix} = \boldsymbol{\pi}\left(\begin{bmatrix} \frac{\mathbf{r} - \mathbf{r}_{fd}}{s_0} & \frac{\mathbf{v} - \mathbf{v}_{fd}}{V_0} \end{bmatrix}, \boldsymbol{\theta}\right) \tag{47}$$

The activation function for the output layer is chosen specifically to confine its range to a bounded interval (y_{lb}, y_{ub}) as

$$f_{\text{scale}}(u) = \frac{y_{ub} - y_{lb}}{1 + \exp(-u)} + y_{lb} \tag{48}$$

This architectural choice is made to scale the policy network outputs so that the magnitude and the direction angles of the thrust vector naturally satisfy the physical limits given by

$$\begin{aligned} \frac{T_{\min}}{T_{\max}} &\leq \delta_T \leq 1 \\ \eta_{\min} &\leq \eta_T \leq \eta_{\max} \\ \sigma_{\min} &\leq \sigma_T \leq \sigma_{\max} \end{aligned} \tag{49}$$

where $T_{\min}$ is the minimum thrust magnitude; $\eta_{\min}$, $\eta_{\max}$, $\sigma_{\min}$, and $\sigma_{\max}$ are the thrust pointing limits.

3. Constraints

In this example, it is assumed for simplicity that both the vehicle position and velocity initially lie on the XZ-plane and the desired final position and velocity are also on the same plane as shown in Fig. 4. In Fig. 4, R_M, θ, γ, and $s := R_M \cos^{-1}\left(\hat{\mathbf{r}} \cdot \hat{\mathbf{r}}_{f_d}\right)$ denote the planet radius, latitude, flight path angle measured in local vertical/horizontal frame, and ground track distance to the target position. The subscripts 0 and f_d refer to the quantities pertaining to the initial and desired final conditions, respectively. The initial and final conditions are given by

$$\mathbf{r}_0 = (R_M + h_0)\begin{bmatrix} \cos\theta_0 \\ 0 \\ \sin\theta_0 \end{bmatrix}, \quad \mathbf{v}_0 = V_0\begin{bmatrix} -\sin(\theta_0 - \gamma_0) \\ 0 \\ \cos(\theta_0 - \gamma_0) \end{bmatrix}, \quad \mathbf{r}_{f_d} = R_M\begin{bmatrix} \cos\theta_{f_d} \\ 0 \\ \sin\theta_{f_d} \end{bmatrix}, \quad \mathbf{v}_{f_d} = -V_{f_d}\hat{\mathbf{r}}_{f_d} \tag{50}$$

where $V := \|\mathbf{v}\|$ denotes the ground speed. Note that one can specify either θ_0 or s_0 to prescribe the initial position since $s_0 = R_M\left(\theta_{f_d} - \theta_0\right)$. Also note that the performance output constraints at the final time consist of the desired final position and velocity conditions, leaving the final mass unconstrained.

4. Objective Function

A powered descent guidance problem can be formulated as a finite-horizon optimal control problem. The goal is to find a NN policy that achieves the desired final position and velocity at the given final time while minimising the fuel consumption. This goal is encoded into the objective function considered for baseline policy optimisation in Stage 1 as

$$J_{aug} = k_{r_f}\frac{\left\|\mathbf{r}\left(t_f\right) - \mathbf{r}_{f_d}\right\|^2}{{s_0}^2} + k_{v_f}\frac{\left\|\mathbf{v}\left(t_f\right) - \mathbf{v}_{f_d}\right\|^2}{{V_0}^2} + k_T\int_{t_0}^{t_f}\delta_T(\tau)\,d\tau + k_\theta\|\boldsymbol{\theta}\|^2 \tag{51}$$

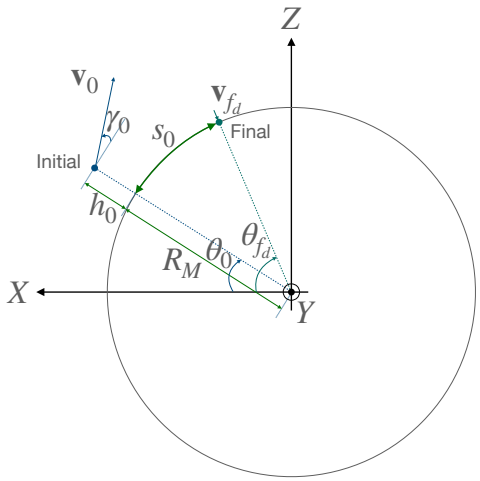


Fig. 4 Mars Landing Guidance Problem Geometry (The circle represents the planetary surface.)

where k_{r_f}, k_{v_f}, k_T, and k_θ are the positive constant weights. The L_2-norm of the NN parameters is included for regularisation. The proposed two-stage approach considers the final position and velocity constraints as hard constraints in Stage 2 for computing the required corrections, i.e., $\tilde{\boldsymbol{\theta}}$ or $\tilde{\mathbf{u}}(t)$. Although Stage 1 relies on a soft constrained formulation, strong penalisation of the errors with large k_{r_f} and k_{v_f} can promote tighter satisfaction of the final constraints at the end of Stage 1.

B. Simulation Setup

Two experiments are performed. The simulation scenarios mainly differ in the initial condition used for testing:

- *Case 1:* single identical initial condition for both baseline policy training and testing.
- *Case 2:* multiple initial conditions for testing that are perturbed from the initial condition for baseline policy training.

Case 1 in Sec. IV.C.2 aims to test the effectiveness of two incremental correction methods in improving the actual accuracy in satisfying the performance output constraint at the final time. The secondary purpose of Case 1 is to demonstrate the full process of NN policy design using the proposed two-stage approach specifically for the powered descent application. The performance of the NN policy is evaluated with various random seeds used for NN weight initialisation considering fixed initial and final conditions.

Case 2 in Sec. IV.C.3 aims to show the utility of incremental correction in dealing with the adverse effect of the inital position dispersion in landing accuracy. In practice, imperfect handover from the entry guidance phase and uncertainties such as environmental disturbances cause a dispersion in the initial condition for the powered descent phase. As a simple model for the dispersion, the closed-loop system is propagated for a set of initial positions obtained along the circle of radius 100m which lies on the plane perpendicular to the initial velocity and is centred at the nominal

initial position. The performance of the baseline policy alone, the policies updated with parameter correction and control function correction are compared with each other. The NN training configuration and simulation parameters are identical across both cases, except for the initial position.

The network architecture for the baseline policy and the optimiser setup used for its training are summarised in Table 1. Each iteration of the policy optimisation process involves simulation of the closed-loop trajectories, in which an individual trajectory simulation is stopped if $t \geq 43$s or if $\left\|\mathbf{r} - \mathbf{r}_{f_d}\right\| \leq 100$m and $\mathbf{v} \cdot \left(\mathbf{r} - \mathbf{r}_{f_d}\right) \geq 0$ at the same time. As briefly mentioned in Sec. II, this study utilises the `Julia` package `ContinuousTimePolicyGradients.jl` which is developed in [10] and available at [45] for baseline policy training. The simulation environment utilises `Zygote.jl` developed in [49] for source-to-source backward mode automatic differentiation.

Table 1 Configuration for Baseline Policy Training (Stage 1)

Description	Details	Value
NN architecture	Number of nodes	[6, 10, 3, 3]†
	Type of activation function	[tanh, tanh, linear, f_{scale}]†
	Initialisation algorithm	Uniform Xavier
Optimiser for 1st pass	Algorithm	ADAM
	Learning rate	10^{-2}
	Decay rate for 1st momentum estimate	9×10^{-1}
	Decay rate for 2nd momentum estimate	9.99×10^{-1}
	Maximum number of iterations	10^3
Optimiser for 2nd pass	Algorithm	BFGS
	Linesearch method	HagerZhang
	Initial step size	10^{-4}
	Maximum number of iterations	10^2
Weights of cost terms	k_{r_f}	10^6
	k_{v_f}	10^5
	k_T	1
	k_θ	10^{-6}
ODE solver	Algorithm	Tsit5
	Relative tolerance	10^{-4}
	Absolute tolerance	10^{-8}

† From input layer (left) to output layer (right), f_{scale} defined in Eq. (48)

Incremental correction for computing either $\tilde{\boldsymbol{\theta}}$ or $\tilde{\mathbf{u}}(t)$ is applied only once at the initial time to clearly compare the two different correction methods (see Remark 7 for the note on continuous re-initialisation). A relative tolerance of 0.005 is used for computing the pseudoinverse in the parameter correction method.

The model data for the dynamical system found in [41, 50], including the Martian environment and the vehicle initial conditions, are considered for the illustrative example. The initial and final conditions defining the mission scenario, the parameters for vehicle dynamics model and policy design, and the parameters for environmental physics model are

described in Tables 2-4, respectively.

Table 2 Simulation Parameters: Initial and Final Conditions

Parameter	Value	Unit
h_0	2480	m
V_0	505	m/s
γ_0	0	deg
s_0	11500	m
m_0	62000	kg
V_{f_d}	2.5	m/s
θ_{f_d}	45	deg
t_f	43	s

Table 3 Simulation Parameters: Vehicle Dynamics and Policy

Parameter	Value	Unit
m_{dry}	51600	kg
m_{sw}	1	kg
I_{sp}	360	s
β_0	379	kg/m^2
$R_{L/D}$	0.54	-
σ_L	0	deg
$T_{\max}$	8×10^5	N
$T_{\min}$	$0.2T_{\max}$	N
$\eta_{\max}$	90	deg
$\eta_{\min}$	-90	deg
$\mathbf{R}(t)$	diag (10, 1, 1)	-

Table 4 Simulation Parameters: Environmental Physics

Parameter	Value	Unit
$\mathbf{v}_w$	$\mathbf{0}$	m/s
μ	4.282837×10^{13}	m^3/s^2
R_M	3389.5×10^3	m
$\boldsymbol{\Omega}$	$2\pi/1.025957$	rad/d
ρ_0	0.0263	kg/m^3
H	10153.6	m
g	9.805	m/s^2

C. Simulation Results

The following sections present the results of numerical experiments. Section IV.C.1 first briefly discusses the performance characteristics of the proposed method. Sections IV.C.2 and IV.C.3 provide detailed results for Case 1 and Case 2, respectively, and Sec. IV.C.4 compares the computation time required for each incremental correction method.

1. Summary of the Results

This section discusses the main observations before proceeding to the detailed results of each simulation case. Overall, different incremental correction algorithms employed in Stage 2 lead to considerable differences in the satisfaction of constraints on the performance output. Although the trend understandably depends on design parameters being used in each method, the numerical experiments in Cases 1 and 2 together suggest that the parameter correction method tends to result in more consistent trajectory and thus reliable in satisfying the performance output constraints. Computational efficiency of the incremental correction algorithms substantially depends on the way how the gradients are computed. The parameter correction method has advantage over the control function correction method in terms of wall-clock computation time only when the sensitivity matrix is obtained by forward propagation of the associated augmented ODE. The results of Case 2 show that the proposed two-stage approach with parameter correction can effectively compensate for small perturbation from the condition considered for baseline policy training.

The parameter correction method can be seen as deliberate induction of overfitting of NN parameters to satisfy the performance output constraints at given time points. In this perspective, the improved landing position accuracy and precision with the parameter correction method can be viewed as an advantage of having an overparametrised function as the control law.

The results also show potential limitations of the parameter correction method. The final position accuracy is improved in many cases, however, it is degraded in rare cases where the baseline policy achieve high accuracy before applying correction. The final velocity error shows slight increase with respect to the baseline in many cases. Also, the problem size depends on the number of policy parameters due to the necessity of computing the sensitivity matrix and its pseudoinverse. Further works should address these limitations to take full advantage of the correction method.

2. Case 1. Single Matched Initial Condition

Stage 1. Baseline Policy Training with Various t_f and Fixed $\left(k_{r_f}, k_{v_f}\right)$ The available thrust magnitude of the spacecraft is limited, leading to limitations on the region reachable within a fixed flight time. Baseline policy training for Stage 1 is performed with various values of t_f ranging from 38s to 44s and the cost weights provided in Table 1 to determine a physically feasible solution. Table 5 shows the final position and velocity errors, and the final mass. Figures 5-7 show three-dimensional trajectory, position error history $e_r\left(t\right) := \left\|\mathbf{r}\left(t\right) - \mathbf{r}_{f_d}\right\|$ with velocity error history $e_v\left(t\right) := \left\|\mathbf{v}\left(t\right) - \mathbf{v}_{f_d}\right\|$, and baseline control history $\mathbf{u}_{base}\left(t\right)$, respectively, obtained with the baseline policies for each

t_f. Figure 5 shows that the vehicle approaches close to target without unnecessarily consuming much control effort in the horizontal direction with the baseline policies for each t_f. The position and the velocity errors that are defined by the norm of the vector difference from their respective desired final value tend to zero as shown in Fig. 6. The time histories of the thrust magnitude and direction angles shown in Fig. 7 describe a continuous trend dependent upon t_f. In all cases, the optimised policy generates maximal reverse thrust as the vehicle approaches the end of flight, which can be seen from throttle command δ_T close to 1 and thrust elevation angle η_T close to -90 deg at the final time. The final position error is the least at $t_f = 43$s as shown in Table 5. Therefore, $t_f = 43$s is chosen for testing Stage 2 methods.

Table 5 Final Errors and Mass with Baseline Policy - $\left(k_{r_f}, k_{v_f}\right) = (10^6, 10^5)$

t_f [s]	e_{r_f} [m]	e_{v_f} [m/s]	m_f [kg]
38	9.928844	15.411199	52958.348
39	9.209262	3.0781	52737.383
40	16.326994	1.2886723	52793.54
41	5.5097303	1.801128	52774.832
42	3.419008	2.3275573	53207.31
43	2.6560874	1.0882493	53713.812
44	5.1203003	1.2965195	54007.098

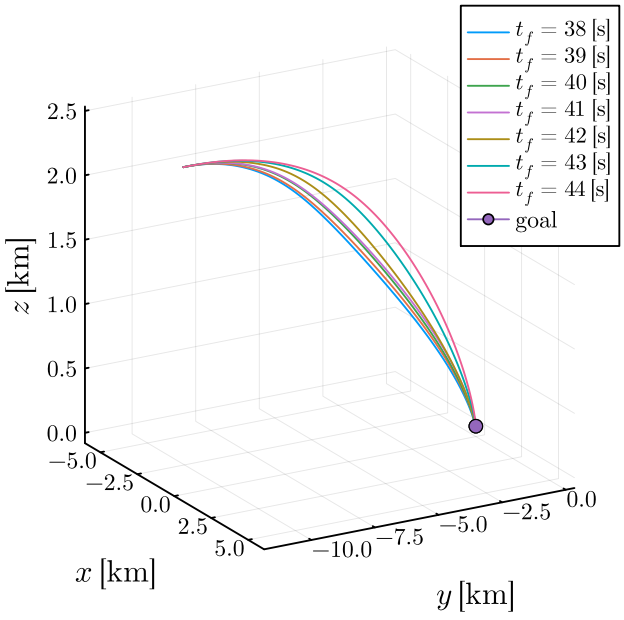


Fig. 5 Three-Dimensional Trajectory with Baseline Policy

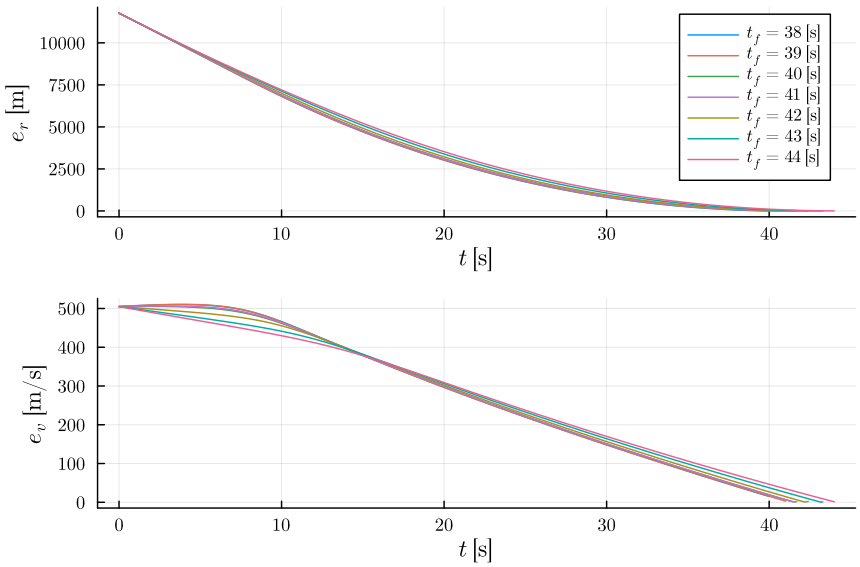


Fig. 6 Error History with Baseline Policy

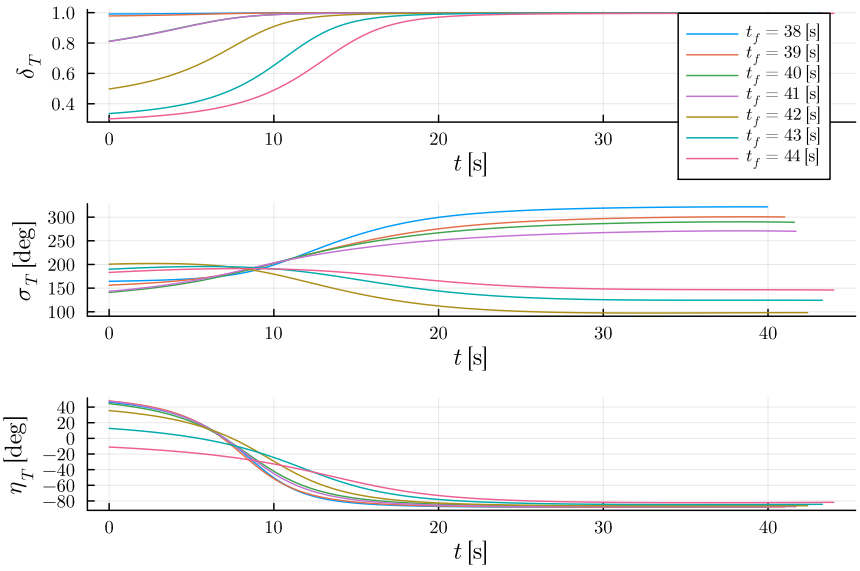


Fig. 7 Control History with Baseline Policy

Remark 8. The computational issues observed during the trials in Stage 1 require attention to facilitate practical implementation. First, when single-precision floating point data type is used for real numbers to reduce memory burden, the quantities which involve values in the order of planet radius R_M in calculation should be avoided from being included in the policy representation. The lack of precision due to low resolution manifests itself in the policy training as high numerical sensitivity in the gradient computation and in the variable-step integration of the ODEs. Second, parametrising the policy with respect to the groundtrack distance s defined by $R_M \cos^{-1}\left(\hat{\mathbf{r}} \cdot \hat{\mathbf{r}}_{f_d}\right)$ may trigger faulty

numerical behaviours during training due to the inaccuracy of arccosine function in some scientific computing systems. Third, the input and output variables of the NN would better be of a similar order of magnitude to avoid the gradient computed through the NN being sensitive only to a certain variable. This is the reason for introducing normalisation in the inputs of the NN and choosing throttle command δ_T instead of thrust magnitude T as an output in Eq. (47).

Stage 2. Incremental Correction with Fixed t_f and Various Random Seeds The result of baseline policy optimisation in Stage 1 substantially depends on initialisation of the NN parameters, which is done by random sampling. Here, the entire process of the proposed two-stage approach is repeated with different random seeds while fixing t_f = 43s. Both the parameter and the control function correction methods are tested to demonstrate their characteristics.

Figure 8 shows final errors and mass for different random seeds. Figures 9-12 show position error history, velocity error history, baseline control history, and control update history $\tilde{\mathbf{u}}(t) := \mathbf{u}(t) - \mathbf{u}_{base}(t)$, respectively, for a fixed random seed. The results shown are obtained i) with no correction, ii) with parameter correction, and iii) with control function correction. For clear comparison of correction methods with respect to the baseline performance, each of the lower panels of Figs. 9, 10 and Fig. 12 show the pointwise difference for each quantity. Figure 8 indicates that the two incremental correction methods exhibit substantially different characteristics in their performance in terms of the capability to reduce final position and/or velocity errors. The final errors are neither completely nullified nor always reduced by performing policy correction once at the initial time. According to Fig. 8 and additional numerical experiments, the parameter correction method yields significant reduction in the final position error e_{r_f} while showing ineffectiveness in reducing the final velocity error e_{v_f}. Also, the final position error is increased in a rare case where the baseline policy already achieved high accuracy. On the other hand, the control function correction method provides improved final velocity targeting accuracy with increased fuel expenditure, however, the final position accuracy is degraded even in comparison to the case with baseline policy alone. Also, a noticeable rapid change in the control input is often observed around the end of flight as shown in the history of σ_T in Fig. 12.

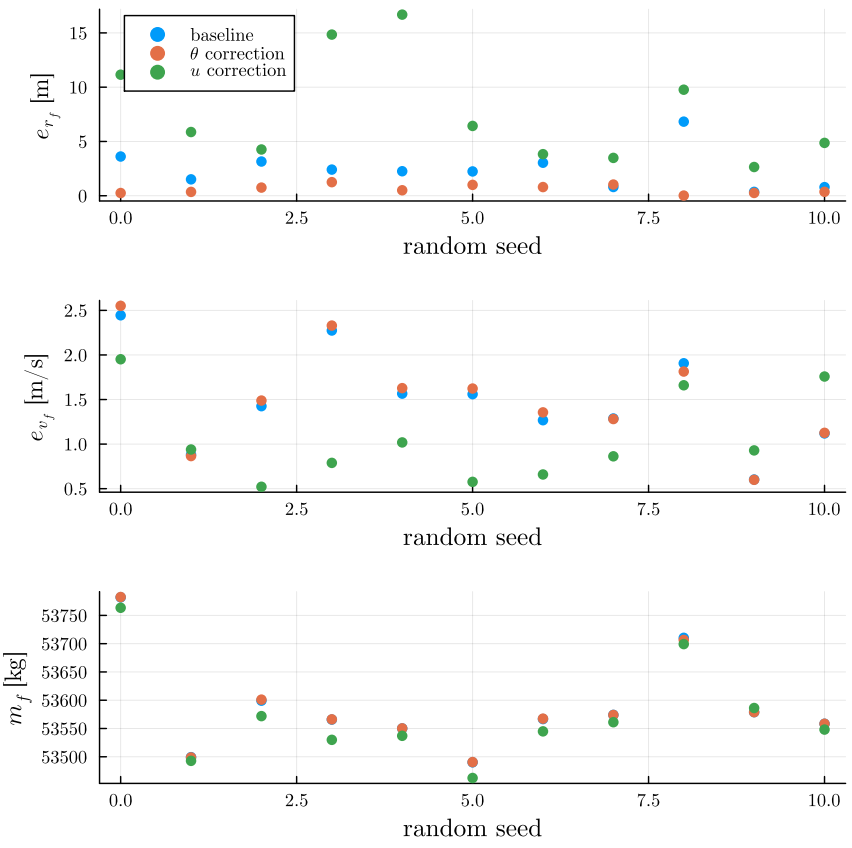


Fig. 8 Final Errors and Mass with Incremental Correction (Case 1)

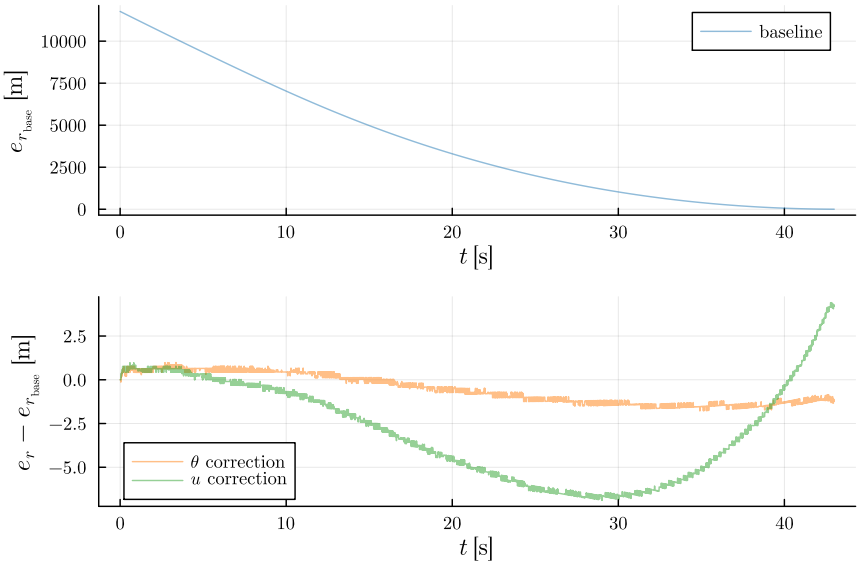


Fig. 9 Position Error History with Incremental Correction - Single Random Seed (Case 1)

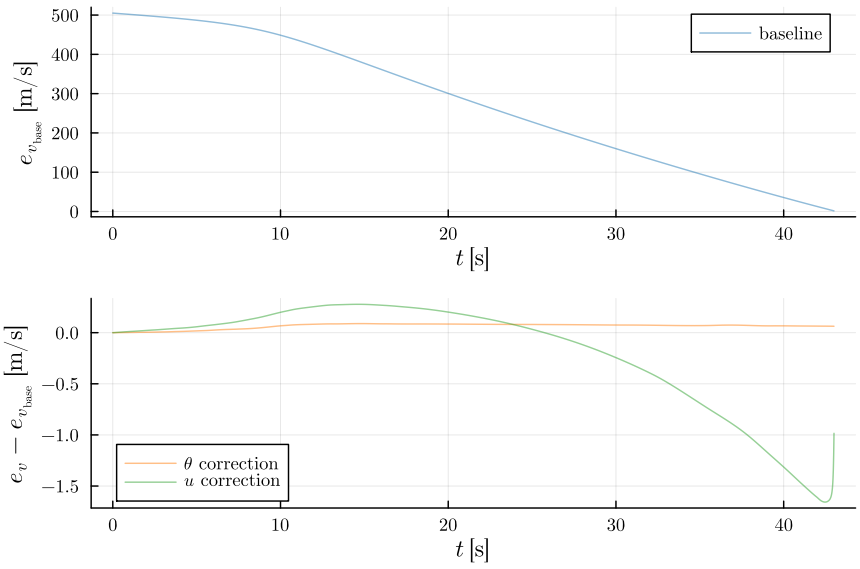


Fig. 10 Velocity Error History with Incremental Correction - Single Random Seed (Case 1)

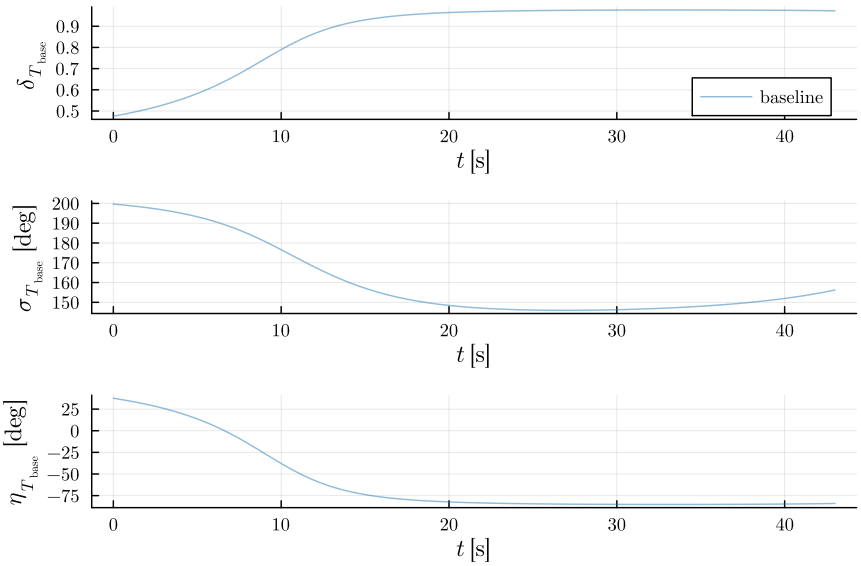


Fig. 11 Baseline Control History with Incremental Correction - Single Random Seed (Case 1)

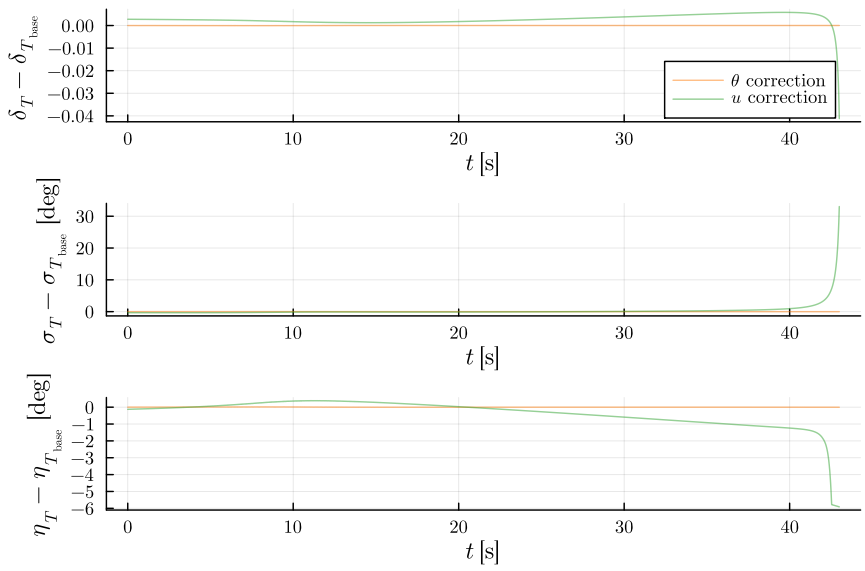


Fig. 12 Control Update History with Incremental Correction - Single Random Seed (Case 1)

3. Case 2. Dispersion in Initial Conditions

Stage 1. Baseline Policy Training The initial and final conditions considered in Case 1 are taken as the nominal conditions for Case 2, and the baseline policy trained in Case 1 for $t_f = 43$s is re-used in Case 2.

Stage 2. Incremental Correction with Fixed t_f and Various Random Seeds Similar to Case 1, both correction techniques are applied to update the baseline policies that are trained by using different random seeds for NN parameter initialisation. The initial position perturbation is modelled deterministically as

$$\bar{\mathbf{r}}_0 = \mathbf{r}_0 + \bar{r}\left(\cos\alpha \frac{\mathbf{v}_0 \times \mathbf{r}_0}{\|\mathbf{v}_0 \times \mathbf{r}_0\|} \times \hat{\mathbf{v}}_0 + \sin\alpha \frac{\mathbf{v}_0 \times \mathbf{r}_0}{\|\mathbf{v}_0 \times \mathbf{r}_0\|}\right) \tag{52}$$

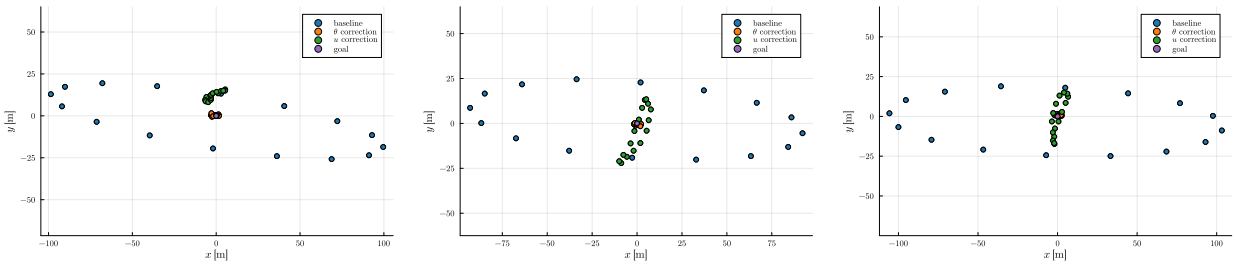
where α is the angle introduced to parametrise the circle of radius $\bar{r} = 100$m centred at $\mathbf{r}_0$ on the plane perpendicular to $\hat{\mathbf{v}}_0$. 16 equally spaced points are obtained by considering a range of α values in $[0, 2\pi]$ with the interval of $\pi/8$ rad.

Each subfigure in Fig. 13 shows the final position projected on the horizontal plane attached to the desired landing position for all combinations of initial condition and incremental correction method, with a fixed baseline policy. In Fig. 13, the clusters of points are scattered around the desired landing position with different size and shape depending on the correction method. The size of the dispersion in terms of the area contained within the convex hull over the trace of final positions is the least with the parameter correction method regardless of the random seed. The centroid of the dispersion resulting from parameter correction is also well-aligned with the goal position for all random seeds. The control function correction method could also reduce the size of the landing point cluster as compared to the case with no correction, however, the irregularly-shaped trace is not always centred around the desired landing position.

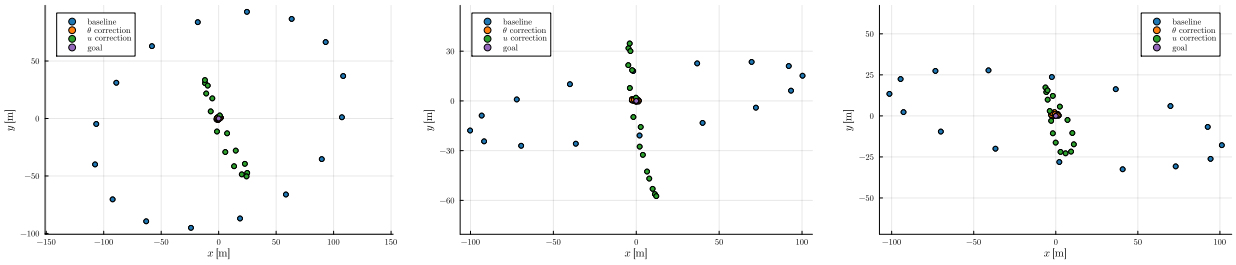
Likewise, Fig. 14 shows the final velocity error represented in the local horizontal and vertical coordinate system with respect to the desired landing position for all combination of baseline policy, initial condition, and incremental correction method. Here, the final velocity error is defined by the difference between the achieved and the desired final velocity vectors, i.e., $\mathbf{v}\left(t_f\right) - \mathbf{v}_{f_d}$. The trend observed in the final velocity error with control function correction is inconsistent across different random seeds and tends to show high variance depending on initial position in each random seed. The irregular tendency manifests as large magnitude of final velocity error in some cases. As for the parameter correction method, the improvement in constraint satisfaction accuracy is relatively limited for the final velocity as compared to that for the final position. However, the distribution of final velocity error resulting from parameter correction does not involve serious outliers that may pose high risk at the final time.

To facilitate more quantitative analysis of errors, Fig. 15 shows the final position and velocity errors, and final mass. In Fig. 15, the dot and the error bar represent the mean and standard deviation ($\pm 1\sigma$) over the range of initial conditions for each random seed, respectively. The plot is consistent with the observation from Fig. 13. The accuracy and precision of landing position represented by the mean and the standard deviation of e_{r_f}, respectively, are improved most significantly by the parameter correction method. The performance benefit of the control function correction method in final position is not entirely clear as it might result in larger standard deviation in e_{r_f}. The final velocity error reduction is not apparent with the parameter correction as in Case 1.

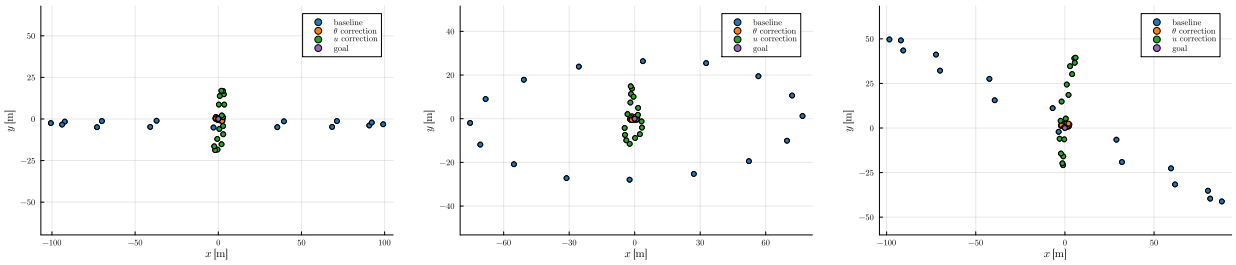
Figures 16-19 show position error history, velocity error history, baseline control history, and control update history $\tilde{\mathbf{u}}\left(t\right) := \mathbf{u}\left(t\right) - \mathbf{u}_{base}\left(t\right)$, respectively, with different Stage 2 correction strategies for a fixed random seed. For clear comparison of correction methods with respect to the baseline performance, each of the lower panels of Figs. 16, 17, along with Fig. 19 show the pointwise difference between each quantity of interest and its baseline counterpart. In the presence of initial position perturbation, the trajectories obtained with each method form a family of solutions that continuously depend on the initial condition, meaning that the actual trajectories do not involve any disruptive irregularities due to the incremental correction. The convergence of position and velocity errors over time follows the similar general trend at the large scale with or without incremental correction as it can be seen from Figs. 16 and 17. Figure 18 confirms that the baseline policy applied to each perturbed initial condition generated smooth control input histories in a small region around the baseline control history for the nominal initial condition shown in Fig. 11. However, with the control function correction method, a rapid change in σ_T near the final time appears in Fig. 19 for all initial conditions, with a similar pattern as observed in Fig. 12 for Case 1. This undesirable behaviour might be attributed to the discrepancy between the actual trajectory and the baseline trajectory that is predicted at the instance of computing the corrective input and taken as the reference for dynamics linearisation. The discrepancy is usually the largest at the final time as it accumulates over time, leading to the amplification of corrective input $\tilde{\mathbf{u}}\left(t\right)$ as $t \to t_f$. Another reason lies in the essential difference between the two correction methods, which differ in the choice of decision variables. The control function correction method not only enforces constraint satisfaction, but also minimises the



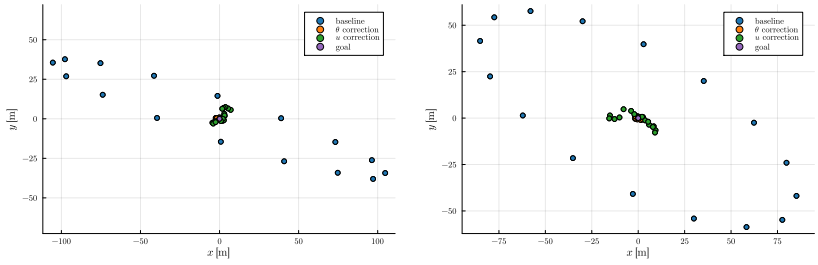
(a) Random Seed 0 **(b) Random Seed** 1 **(c) Random Seed** 2



(d) Random Seed 3 **(e) Random Seed** 4 **(f) Random Seed** 5

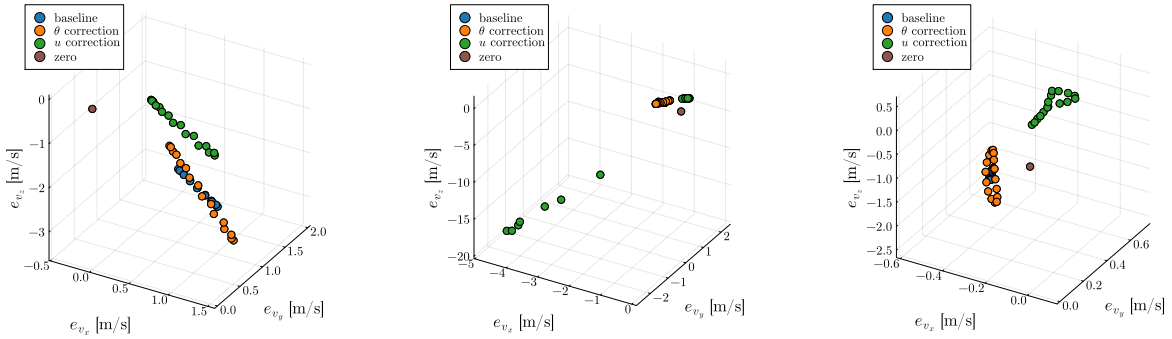


(g) Random Seed 6 **(h) Random Seed** 7 **(i) Random Seed** 8

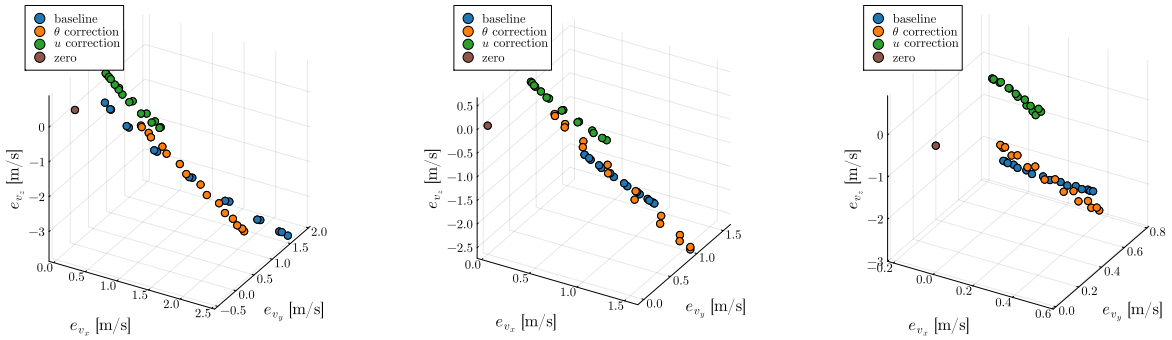


(j) Random Seed 9 **(k) Random Seed** 10

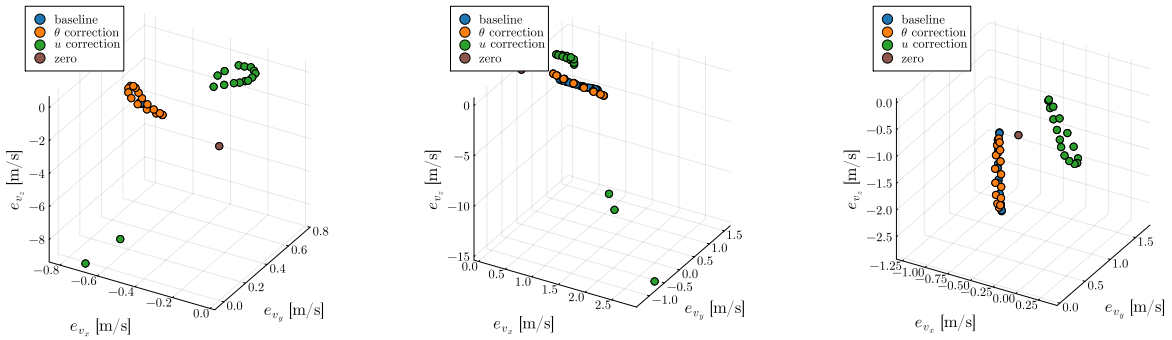
Fig. 13 Final Position on Horizontal Plane (Case 2)



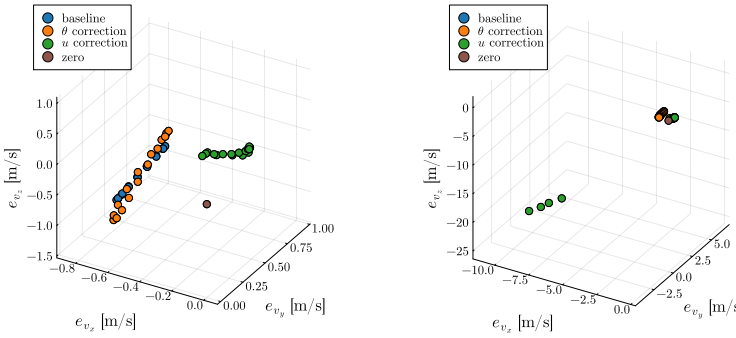
(a) Random Seed 0 **(b) Random Seed** 1 **(c) Random Seed** 2



(d) Random Seed 3 **(e) Random Seed** 4 **(f) Random Seed** 5



(g) Random Seed 6 **(h) Random Seed** 7 **(i) Random Seed** 8



(j) Random Seed 9 **(k) Random Seed** 10

Fig. 14 Final Velocity Error (Case 2)

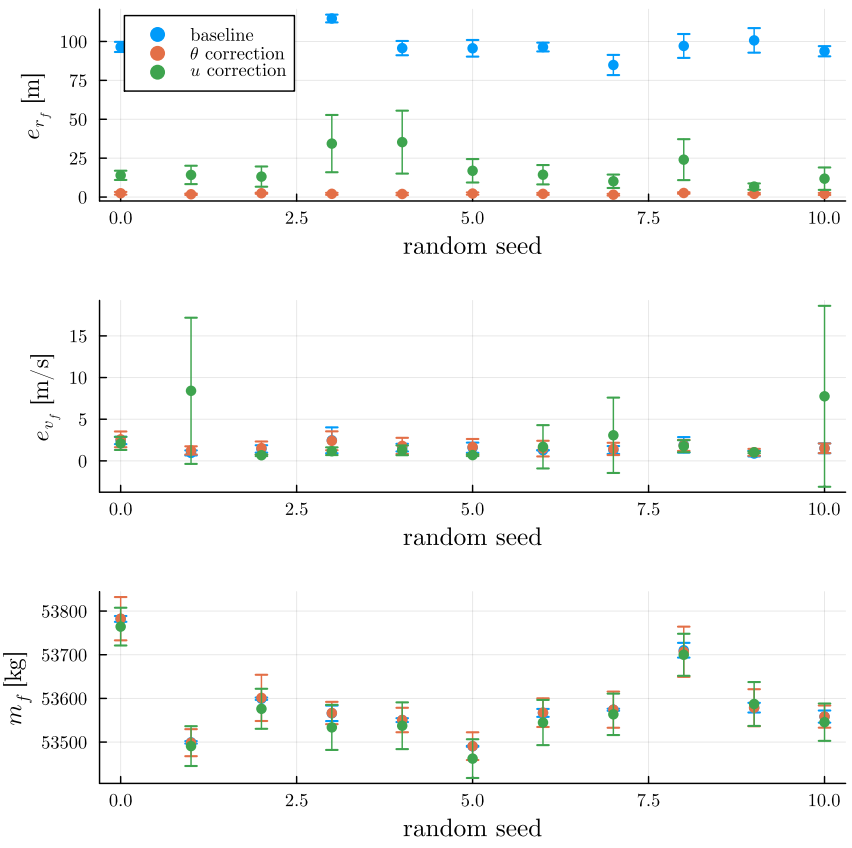


Fig. 15 Final Errors and Mass with Incremental Correction (Case 2)

weighted $\mathcal{L}_2$-norm of the corrective input $\tilde{\mathbf{u}}(t)$. The parameter correction method also solves for the minimum l_2-norm solution $\tilde{\boldsymbol{\theta}}$ for the linear system of equations, but it does not necessarily translate into minimal amount of change at the level of control input $\mathbf{u}$ with respect to the signal norm.

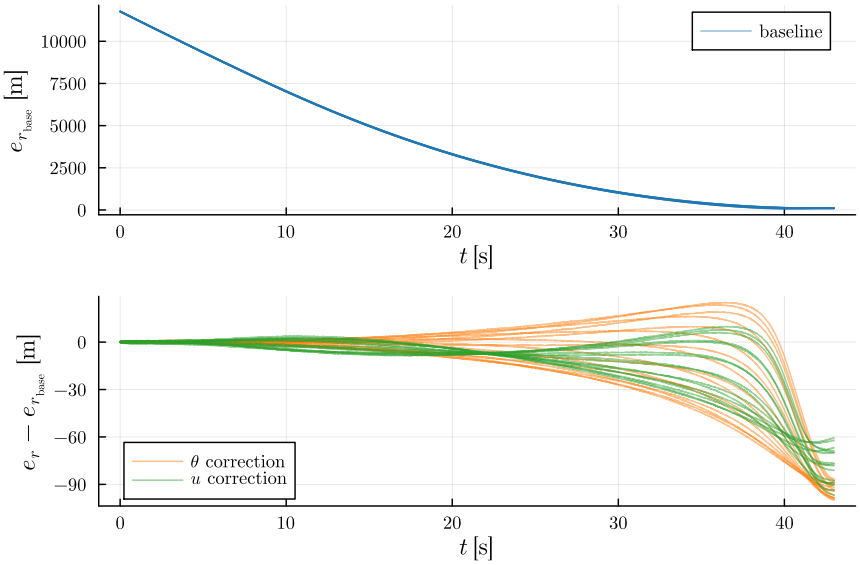


Fig. 16 Position Error History with Incremental Correction - Single Random Seed (Case 2)

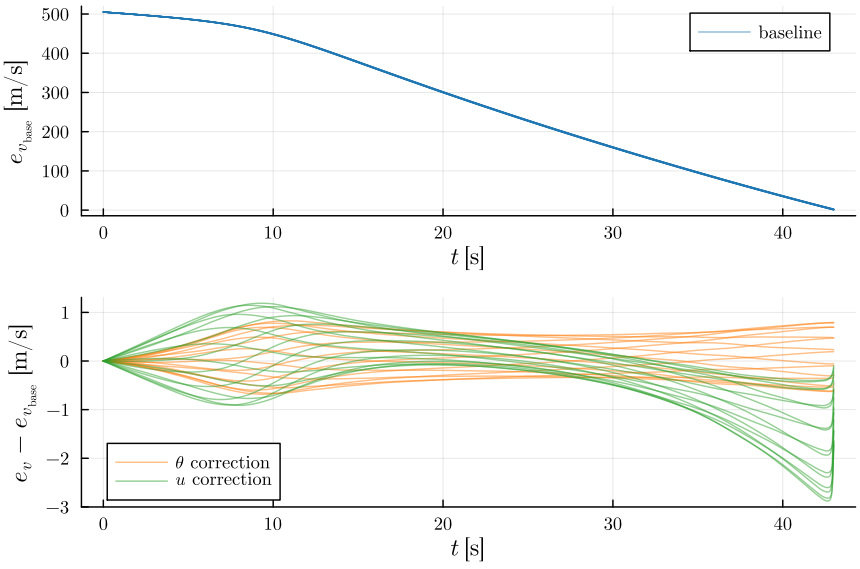


Fig. 17 Velocity Error History with Incremental Correction - Single Random Seed (Case 2)

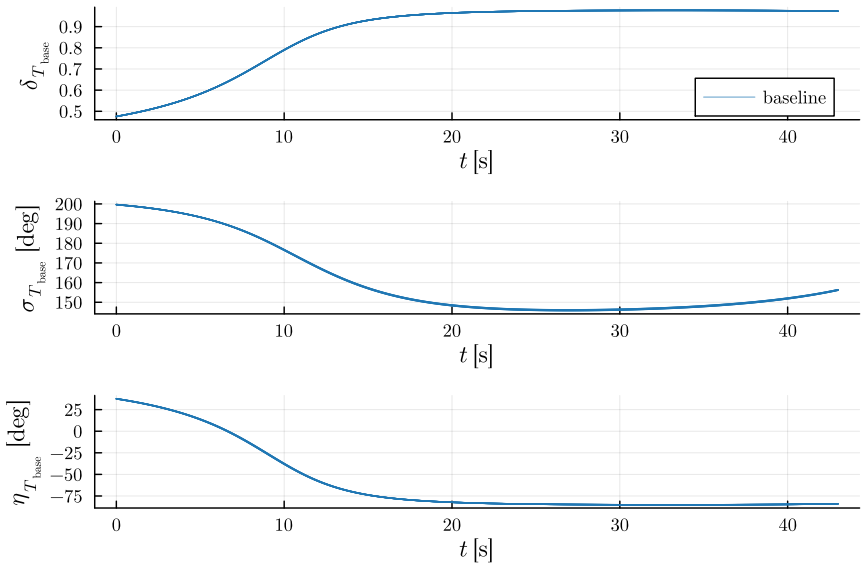


Fig. 18 Baseline Control History with Incremental Correction - Single Random Seed (Case 2)

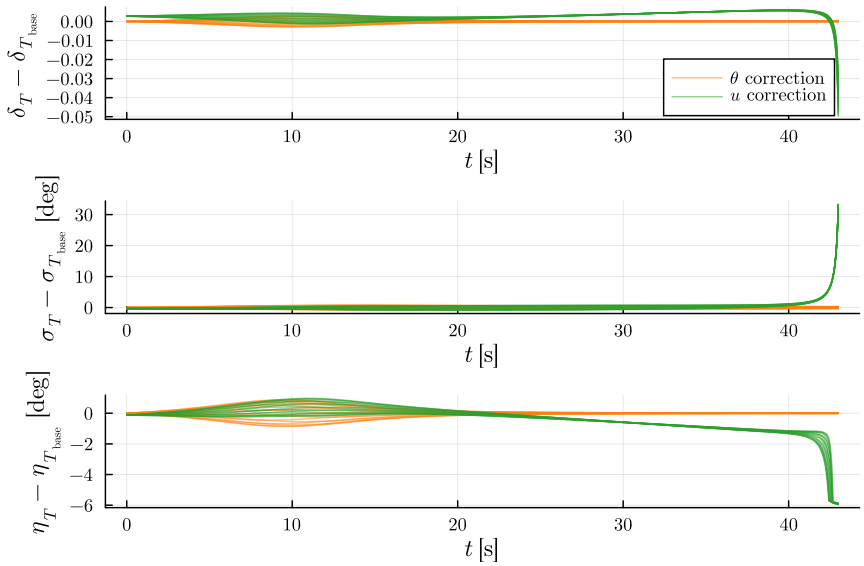


Fig. 19 Control Update History with Incremental Correction - Single Random Seed (Case 2)

4. Computation Time Comparison

Table 6 gives the average simulation time for each combination of random seed and correction method as measured on a Macbook Pro 15-inch 2017 with 2.8 GHz Quad-Core Intel Core i7 CPU and 16GB 2133 MHZ LPDDR3 RAM. Two different methods to get the required sensitivity matrix $\left.\frac{\partial \mathbf{x}^*(t_f)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^*}$ are tested for the parameter correction method; i) forward propagation of the sensitivity matrix by solving an augmented ODE ($\boldsymbol{\theta}$ - prop), and ii) automatic differentiation of the ODE solver ($\boldsymbol{\theta}$ - AD). The control function correction method is implemented with forward propagation of the fundamental solution matrix $\mathbf{U}(t)$ and the auxiliary matrix $\mathbf{N}(t)$ ($\mathbf{u}$ - prop) instead of quadrature-based integral computation. In Case 2, an ensemble ODE problem was constructed for each random seed and correction method to simulate the same dynamics for 16 different initial conditions through multithreading on CPU with 8 threads consisting of 4 physical and 4 virtual cores. The average simulation time is obtained by repeating the single ODE simulation in Case 1 or the ensemble ODE simulation in Case 2 10 times and then taking the average of elapsed times to reduce noise. In Table 6, the simulation time for the baseline case does not include the time consumed in baseline policy training. The values in Table 6 include the time spent due to multithreading initialisation and data transfer, memory allocation, correction computation done at the beginning of simulation, simulation execution, and garbage collection, except the compilation time spent at the first run of each method because of the Just-In-Time compilation behaviour of `Julia`. Therefore, the data provided here should be regarded as an indicator of relative computational burden. More detailed profiling along with code optimisation will be needed to assess the actual performance on a real-time computer.

Table 6 Average Wall-Clock Time for Simulation [in Seconds]

Seed	Case 1[†]			Case 2[†]		
	baseline	$\boldsymbol{\theta}$ - prop	$\mathbf{u}$ - prop	baseline	$\boldsymbol{\theta}$ - prop	$\mathbf{u}$ - prop
0	0.199	1.037	2.332	1.109	5.638	10.670
1	0.238	1.074	2.293	1.031	5.015	9.660
2	0.227	1.082	2.323	1.044	5.024	9.625
3	0.228	1.057	2.292	1.024	5.033	9.397
4	0.239	1.188	2.315	1.038	5.702	9.847
5	0.229	1.087	2.362	1.035	5.127	9.791
6	0.233	1.034	2.387	1.048	4.880	9.334
7	0.230	1.128	2.158	1.040	7.906	9.694
8	0.201	1.057	2.173	0.970	4.447	9.188
9	0.196	1.174	2.198	0.948	4.920	9.054
10	0.198	1.080	2.324	0.957	4.666	9.065

[†] elapsed time for single propagation, [‡] elapsed time for multi-threaded ensemble of 16 propagations

The main cause of the increased simulation time for both correction methods appears to be the Jacobian calculation using automatic differentiation (See Remarks 2 and 3). In $\boldsymbol{\theta}$ - prop, the augmented ODE is constructed using automatic differentiation of the given dynamics function and it is propagated. In $\boldsymbol{\theta}$ - AD, the computational graph for sensitivity

calculation is generated using automatic differentiation through the ODE solver and it is evaluated. In **u** - prop, automatic differentiation is called at two different points; i) once when the correction is triggered to obtain linearised system matrices $\mathbf{A}_u(t)$ and $\mathbf{B}_u(t)$ for $\forall t \in \left[t_0, t_f\right]$ around the *stored* baseline state and input that are required to precompute the matrix $\bar{\mathbf{\Psi}}$ in Eq. (38), and ii) at each instance to linearise the system dynamics around the *measured* current state and the *stored* baseline input to obtain $\mathbf{B}_u(t)$ in Eq. (38).

The computational load in terms of average wall-clock time for simulation is shown to be lighter for $\boldsymbol{\theta}$ - prop than **u** - prop. $\boldsymbol{\theta}$ - AD turned out to be substantially more computationally expensive than $\boldsymbol{\theta}$ - prop while the two different implementations give almost identical trajectories. The average elapsed time for $\boldsymbol{\theta}$ - AD was about 4 times that of $\boldsymbol{\theta}$ - prop. The computation time for **u** - prop might be reduced with a more efficient way to compute the linearised system matrices. In both $\boldsymbol{\theta}$ - prop and $\boldsymbol{\theta}$ - AD, the problem size depends on the length l of parameter $\boldsymbol{\theta}$ as the sensitivity matrix is of dimension $n \times l$. Therefore, choosing a large l will increase the time required to compute the sensitivity matrix and its pseudoinverse. Although a baseline policy does not have to be a very large NN, careful selection of the network size and an efficient software for Jacobian calculation will be necessary.

V. Conclusion

This study presents refinement methods for improving the accuracy of satisfying performance output constraints at given time points in continuous-time dynamic systems in which the policy is provided by a neural network. Provided a baseline neural network policy constructed a priori, the incremental correction can be performed either at the level of neural network parameters or at the level of control input variables, so that the updated controller can enforce point constraints on performance output. The effectiveness of the proposed two-stage approach consisting of baseline policy optimisation followed by incremental correction was illustrated on a Mars landing guidance problem in the powered descent phase. The two types of incremental correction methods exhibited different performance characteristics, demanding a comparative study before deciding which method will be more appropriate for implementation depending on the application. For the Mars landing example addressed in this study, the parameter correction method showed a relative advantage in computational reliability and landing position targeting accuracy. As long as the required sensitivity matrices can be computed accurately and efficiently, the parameter correction method can be useful as a plug-and-play extension for improving the constraint satisfaction accuracy to any baseline neural network policy trained by using off-the-shelf package relying on unconstrained optimisation.

References

[1] Klumpp, A. R., "Apollo Lunar Descent Guidance," *Automatica*, Vol. 10, No. 2, 1974, pp. 133–146. doi:10.1016/0005-1098(74)90019-3.

[2] Harpold, J. C., and Gavert, D. E., "Space Shuttle Entry Guidance Performance Results," *Journal of Guidance, Control, and*

Dynamics, Vol. 6, No. 6, 1983, pp. 442–447. doi:10.2514/3.8523.

[3] Lu, P., "Theory of Fractional-Polynomial Powered Descent Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 3, 2020, pp. 398–409. doi:10.2514/1.G004556.

[4] Lu, P., "Propellant-Optimal Powered Descent Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 4, 2018, pp. 813–826. doi:10.2514/1.G003243.

[5] Lu, P., Brunner, C. W., Stachowiak, S. J., Mendeck, G., Tigges, M. A., and Cerimele, C. J., "Verification of a Fully Numerical Entry Guidance Algorithm," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2017, pp. 230–247. doi:10.2514/1.G000327.

[6] Açikmeşe, B., and Ploen, S. R., "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 5, 2007, pp. 1353–1366. doi:10.2514/1.27553.

[7] Lugo, R. A., Powell, R., and Dwyer-Cianciolo, A. M., "Overview of a Generalized Numerical Predictor-Corrector Targeting Guidance with Application to Human-Scale Mars Entry, Descent, and Landing," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, 2020. doi:10.2514/6.2020-0846.

[8] Lugo, R. A., Dwyer-Cianciolo, A. M., Dutta, S., Williams, R. A., Pensado, A., and Chen, P.-T., "Integrated Precision Landing Performance Results for a Human-Scale Mars Landing System," *AIAA Scitech 2022 Forum*, American Institute of Aeronautics and Astronautics, 2022. doi:10.2514/6.2022-0607.

[9] Cianciolo, A. D., and Powell, R. W., "Entry, Descent, and Landing Guidance and Control Approaches to Satisfy Mars Human Mission Landing Criteria," *AAS/AIAA Space Flight Mechanics Meeting*, San Antonio, TX, USA, 2017.

[10] Cho, N., Shin, H.-S., and Tsourdos, A., "Optimisation of Structured Neural Controller Based on Continuous-Time Policy Gradient," arXiv:2201.06262, 2022. https://arxiv.org/abs/2201.06262.

[11] Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., and Edelman, A., "Universal Differential Equations for Scientific Machine Learning," arXiv:2001.04385, 2021. https://arxiv.org/abs/2001.04385.

[12] Sagliano, M., "Pseudospectral Convex Optimization for Powered Descent and Landing," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 2, 2018, pp. 320–334. doi:10.2514/1.G002818.

[13] Kidger, P., "On Neural Differential Equations," Ph.D. thesis, Mathematical Institute, University of Oxford, 2021. http://ora.ox.ac.uk/objects/uuid:af32d844-df84-4fdc-824d-44bebc3d7aa9.

[14] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K., "Neural Ordinary Differential Equations," *32nd Conference on Neural Information Processing Systems*, Montreal, Canada, 2018. https://proceedings.neurips.cc/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html.

[15] Tzen, B., and Raginsky, M., "Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit," , 2019. https://arxiv.org/abs/1905.09883.

[16] Kidger, P., Foster, J., Li, X., and Lyons, T. J., "Neural SDEs as Infinite-Dimensional GANs," *38th International Conference on Machine Learning*, Virtual, 2021. https://proceedings.mlr.press/v139/kidger21b.html.

[17] Kidger, P., Morrill, J., Foster, J., and Lyons, T., "Neural Controlled Differential Equations for Irregular Time Series," *34th Conference on Neural Information Processing Systems*, Virtual, 2020. https://proceedings.neurips.cc/paper/2020/hash/4a5876b450b45371f6cfe5047ac8cd45-Abstract.html.

[18] Morrill, J., Kidger, P., Yang, L., and Lyons, T., "On the Choice of Interpolation Scheme for Neural CDEs," *Transactions on Machine Learning Research*, 2022. https://openreview.net/forum?id=caRBFhxXIG.

[19] Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. K., "Latent Ordinary Differential Equations for Irregularly-Sampled Time Series," *33rd Conference on Neural Information Processing Systems*, Vancouver, Canada, 2019. https://proceedings.neurips.cc/paper/2019/hash/42a6845a557bef704ad8ac9cb4461d43-Abstract.html.

[20] Hasani, R., Lechner, M., Amini, A., Liebenwein, L., Ray, A., Tschaikowski, M., Teschl, G., and Rus, D. R., "Closed-Form Continuous-Time Neural Networks," *Nature Machine Intelligence*, Vol. 4, 2022, pp. 992–1003. doi:10.1038/s42256-022-00556-7.

[21] Du, J., Futoma, J., and Doshi-Velez, F., "Model-based Reinforcement Learning for Semi-Markov Decision Processes with Neural ODEs," *34th Conference on Neural Information Processing Systems*, Virtual, 2020. https://proceedings.neurips.cc/paper/2020/hash/e562cd9c0768d5464b64cf61da7fc6bb-Abstract.html.

[22] Yildiz, C., Heinonen, M., and Lähdesmäki, H., "Continuous-time Model-based Reinforcement Learning," *38th International Conference on Machine Learning*, Virtual, 2021. https://proceedings.mlr.press/v139/yildiz21a.html.

[23] Massaroli, S., Poli, M., Califano, F., Park, J., Yamashita, A., and Asama, H., "Optimal Energy Shaping via Neural Approximators," *SIAM Journal on Applied Dynamical Systems*, Vol. 21, No. 3, 2022, pp. 2126–2147. doi:10.1137/21M1414279.

[24] Dupont, E., Doucet, A., and Teh, Y. W., "Augmented Neural ODEs," *33rd Conference on Neural Information Processing Systems*, Vancouver, Canada, 2019. https://proceedings.neurips.cc/paper/2019/hash/21be9a4bd4f81549a9d1d241981cec3c-Abstract.html.

[25] Hasani, R., Lechner, M., Amini, A., Rus, D., and Grosu, R., "Liquid Time-constant Networks," *35th AAAI Conference on Artificial Intelligence*, Virtual, 2021. doi:10.1609/aaai.v35i9.16936.

[26] Daulbaev, T., Katrutsa, A., Markeeva, L., Gusak, J., Cichocki, A., and Oseledets, I., "Interpolation Technique to Speed Up Gradients Propagation in Neural ODEs," *34th Conference on Neural Information Processing Systems*, Virtual, 2020. https://proceedings.neurips.cc/paper/2020/hash/c24c65259d90ed4a19ab37b6fd6fe716-Abstract.html.

[27] Zhuang, J., Dvornek, N., Li, X., Tatikonda, S., Papademetris, X., and Duncan, J., "Adaptive Checkpoint Adjoint Method for Gradient Estimation in Neural ODE," *37th International Conference on Machine Learning*, Virtual, 2020. https://proceedings.mlr.press/v119/zhuang20a.html.

[28] Kidger, P., Chen, R. T. Q., and Lyons, T. J., ""Hey, that's not an ODE": Faster ODE Adjoints via Seminorms," *38th International Conference on Machine Learning*, Virtual, 2021. https://proceedings.mlr.press/v139/kidger21a.html.

[29] Massaroli, S., Poli, M., Bin, M., Park, J., Yamashita, A., and Asama, H., "Stable Neural Flows," , 2020. https://arxiv.org/abs/2003.08063.

[30] Rodriguez, I. D. J., Ames, A., and Yue, Y., "LyaNet: A Lyapunov Framework for Training Neural ODEs," *39th International Conference on Machine Learning*, Baltimore, MD, USA, 2022. https://proceedings.mlr.press/v162/rodriguez22a.html.

[31] Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., and Dixit, V., "DiffEqFlux.jl - A Julia Library for Neural Differential Equations," arXiv:1902.02376, 2019. https://arxiv.org/abs/1902.02376.

[32] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations*, San Diego, CA, USA, 2015. https://arxiv.org/abs/1412.6980.

[33] Dozat, T., "Incorporating Nesterov Momentum into ADAM," *4th International Conference on Learning Representations*, San Juan, Puerto Rico, 2016. https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ.

[34] Shen, H., Seywald, H., and Powell, R. W., "Desensitizing the Minimum-Fuel Powered Descent For Mars Pinpoint Landing," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 1, 2010, pp. 108–115. doi:10.2514/1.44649.

[35] Martin, M. S., Mendeck, G. F., Brugarolas, P. B., Singh, G., Serricchio, F., Lee, S. W., Wong, E. C., and Essmiller, J. C., "In-flight experience of the Mars Science Laboratory Guidance, Navigation, and Control system for Entry, Descent, and Landing," *CEAS Space Journal*, Vol. 7, No. 2, 2015, pp. 119–142. doi:10.1007/s12567-015-0091-3.

[36] Açikmeşe, B., Behçet, and Blackmore, L., "Lossless Convexification of a Class of Optimal Control Problems with Non-convex Control Constraints," *Automatica*, Vol. 47, No. 2, 2011, pp. 341–347. doi:10.1016/j.automatica.2010.10.037.

[37] Açikmeşe, B., Behçet, Carson, J. M., and Blackmore, L., "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Transactions on Control Systems Technology*, Vol. 21, No. 6, 2013, pp. 2104–2113. doi:10.1109/TCST.2012.2237346.

[38] Scharf, D. P., Açikmeşe, B., Dueri, D., Benito, J., and Casoliva, J., "Implementation and Experimental Demonstration of Onboard Powered-Descent Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2017, pp. 213–229. doi:10.2514/1.G000399.

[39] Malyuta, D., Reynolds, T. P., Szmuk, M., Lew, T., Bonalli, R., Pavone, M., and Açikmeşe, B., Behçet, "Convex Optimization for Trajectory Generation: A Tutorial on Generating Dynamically Feasible Trajectories Reliably and Efficiently," *IEEE Control Systems Magazine*, Vol. 42, No. 5, 2022, pp. 40–113. doi:10.1109/MCS.2022.3187542.

[40] Lu, P., and Callan, R., "Propellant-Optimal Powered Descent Guidance Revisited," *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 2, 2023, pp. 215–230. doi:10.2514/1.G007214.

[41] Lu, P., "Augmented Apollo Powered Descent Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 3, 2019, pp. 447–457. doi:10.2514/1.G004048.

[42] Bryson, A. E., and Denham, W. F., "A Steepest-Ascent Method for Solving Optimum Programming Problems," *Journal of Applied Mechanics*, Vol. 29, No. 2, 1962, pp. 247–257. doi:10.1115/1.3640537.

[43] Bryson, A. E., Denham, W. F., and Dreyfus, S. E., "Optimal Programming Problems with Inequality Constraints I: Necessary Conditions for Extremal Solutions," *AIAA Journal*, Vol. 1, No. 11, 1963, pp. 2544–2550. doi:10.2514/3.2107.

[44] Denham, W. F., and Bryson, A. E., "Optimal Programming Problems with Inequality Constraints II: Solution by Steepest-Ascent," *AIAA Journal*, Vol. 2, No. 1, 1964, pp. 25–34. doi:10.2514/3.2209.

[45] Cho, N., "`ContinuousTimePolicyGradients.jl`," GitHub Repository, August 2022. https://github.com/nhcho91/ContinuousTimePolicyGradients.jl, ver 0.1.4.

[46] Ma, Y., Dixit, V., Innes, M., Guo, X., and Rackauckas, C., "A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions," arXiv:1812.01892, 2018. https://arxiv.org/abs/1812.01892.

[47] Maity, A., Oza, H. B., and Padhi, R., "Generalized Model Predictive Static Programming and Angle-Constrained Guidance of Air-to-Ground Missiles," *Journal of Guidance, Control, and Dynamics*, Vol. 37, No. 6, 2014, pp. 1897–1913. doi:10.2514/1.G000038.

[48] McMahon, J. W., Amato, D., Kuettel, D., and Grace, M. J., "Stochastic Predictor-Corrector Guidance," *AIAA SciTech Forum*, San Diego, CA, USA, 2022. doi:10.2514/6.2022-1771.

[49] Innes, M. J., "Don't Unroll Adjoint: Differentiating SSA-Form Programs," arXiv:1810.07951, 2018. https://arxiv.org/abs/1810.07951.

[50] Amato, D., and McMahon, J. W., "Deep Learning Method for Martian Atmosphere Reconstruction," *Journal of Aerospace Information Systems*, Vol. 18, No. 10, 2021, pp. 728–738. doi:10.2514/1.I010922.



https://dspace.org/

Online corrections to neural policy guidance for pinpoint powered descent

Cho, Namhoon; Shin, Hyo-Sang; Tsourdos, Antonios; Amato, Davide

2024-05-01
American Institute of Aeronautics and Astronautics (AIAA)
https://doi.org/10.2514/1.G007234
Cho N, Shin HS, Tsourdos A, Amato D. (2024) Online corrections to neural policy guidance for pinpoint powered descent. Journal of Guidance, Control, and Dynamics, Volume 47, Issue 5, May 2024, pp. 945-963

Downloaded from a DSpace repository. https://dspace.org/