Closing the gap: Optimizing Guidance and Control Networks through Neural ODEs

Sebastien Origer[1] and Dario Izzo[1]

[1]*Advanced Concepts Team, European Space Research and Technology Centre (ESTEC), Noordwijk, The Netherlands.*

***Abstract* - We improve the accuracy of Guidance & Control Networks (G&CNETs), trained to represent the optimal control policies of a time-optimal transfer and a mass-optimal landing, respectively. In both cases we leverage the dynamics of the spacecraft, described by Ordinary Differential Equations which incorporate a neural network on their right-hand side (Neural ODEs). Since the neural dynamics is differentiable, the ODEs sensitivities to the network parameters can be computed using the variational equations, thereby allowing to update the G&CNET parameters based on the observed dynamics. We start with a straightforward regression task, training the G&CNETs on datasets of optimal trajectories using behavioural cloning. These networks are then refined using the Neural ODE sensitivities by minimizing the error between the final states and the target states. We demonstrate that for the orbital transfer, the final error to the target can be reduced by 99% on a single trajectory and by 70% on a batch of 500 trajectories. For the landing problem the reduction in error is around $98-99\%$ (position) and $40-44\%$ (velocity). This step significantly enhances the accuracy of G&CNETs, which instills greater confidence in their reliability for operational use. We also compare our results to the popular Dataset Aggregation method (DaGGER) and allude to the strengths and weaknesses of both methods.**

I. Introduction

Guidance and Control Networks (G&CNETs) represent an emerging technique that holds promise for on-board autonomy and the seamless integration of optimality principles into spacecraft and space agents [1, 2, 3, 4, 5, 6, 7]. They serve as an alternative to model predictive control schemes (MPC) [8], capitalizing on the numerous improvements and advances arising from neural network-based research. Both Reinforcement Learning (RL) and Behavioural Cloning (BC) have already demonstrated successful implementations in training G&CNETs for both space and drone-related tasks [9]. However no matter which learning paradigm one chooses, residual approximation errors lead to orbit injection errors that must be corrected at the cost of extra on-board propellant. Therefore, after training a G&CNET, it is important to further reduce the final mismatch between the targeted final orbit injection conditions and the ones achieved by the on-board neural guidance and control. This paper considers neural models to represent the optimal control policy for a time-optimal interplanetary transfer targeting a generic low-thrust Earth rendezvous starting from the asteroid belt [7] and a mass-optimal landing on the asteroid Psyche. We chose these two optimal control problems such that our study covers different timescales and problems of varying difficulty. The transfer is a complex low-thrust problem which lasts years. In contrast, the landing problem requires the G&CNET to learn a discontinuous function representing a bang-bang control profile and lasts only minutes. In our work we use the term Neural ODEs, popularized in [10], to describe Ordinary Differential Equations which have an artificial neural network on their right-hand side. We exploit the fact that, for fixed initial conditions, the solution to such a system depends only on the network parameters. We thus proceed to study the use of Ordinary Differential Equations (ODEs) sensitivities to the network parameters. Since our Neural ODEs are differentiable, the variational equations, or equivalently Pontryagin's adjoint method, enable us to compute efficiently thousands of ODE sensitivities (state transition matrix). These partial derivatives are used to inform a local search into the highly dimensional network parameters space, aligning with the recent trend of Neural ODEs. We use a simple gradient descent algorithm to update the G&CNET parameters such as to minimize the mismatch between the final states and the target states.

II. Methods

A. Optimal control problems

Time-optimal interplanetary transfer

We consider the same time optimal, constant acceleration rendezvous with a body in a perfectly circular orbit

of radius R as in [7]. Let $\mathcal{F} = \left[\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}\right]$ be a rotating frame with angular velocity $\mathbf{\Omega} = \sqrt{\frac{\mu}{R^3}}\hat{\mathbf{k}}$. In this way, the target body position $R\hat{\mathbf{i}}$ remains stationary in $\mathcal{F}$. The dynamics is described by the following ordinary differential equations:

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \dot{v}_x = -\frac{\mu}{r^3}x + 2\Omega v_y + \Omega^2 x + \Gamma i_x \\ \dot{v}_y = -\frac{\mu}{r^3}y - 2\Omega v_x + \Omega^2 y + \Gamma i_y \\ \dot{v}_z = -\frac{\mu}{r^3}z + \Gamma i_z \end{cases} \tag{1}$$

The state vector $\mathbf{x}_T$ (subscript $\square_T$ to indicate the "transfer" optimal control problem) contains the position $\mathbf{r} = [x, y, z]$ and velocity $\mathbf{v} = [v_x, v_y, v_z]$ which are both defined in the rotating frame $\mathcal{F}$. Note that $r = \sqrt{x^2 + y^2 + z^2}$ and μ is the gravitational constant of the Sun. The system is controlled by the thrust direction described by the unit vector $\hat{\mathbf{t}} = [i_x, i_y, i_z]$, generating an acceleration of magnitude Γ. The optimal control problem boils down to finding the optimal time-of-flight t_f and a (piece-wise continuous) function for $\hat{\mathbf{t}}(t)$, where $t \in [t_0, t_f]$, such that, under the dynamics described by Eq.1, the state is steered from any initial state $\mathbf{r}_0$, $\mathbf{v}_0$ to the desired target state $\mathbf{r}_t = R\hat{\mathbf{i}}$, $\mathbf{v}_t = \mathbf{0}$. We are thus minimizing the following cost function: $J = t_f - t_0 = \int_{t_0}^{t_f} dt$ [7]. Let's solve this problem using Pontryagin's Maximum Principle [11], taking into account some useful tips from [12]. Let $\mathcal{H}$ be the Hamiltonian:

$$\mathcal{H}(\mathbf{r}, \mathbf{v}, \boldsymbol{\lambda}_r, \boldsymbol{\lambda}_\mathbf{v}, \hat{\mathbf{t}}) = \boldsymbol{\lambda}_\mathbf{r} \cdot \mathbf{v} + \boldsymbol{\lambda}_\mathbf{v} \cdot \left(-\frac{\mu}{r^3}\mathbf{r} - 2\boldsymbol{\omega} \times \mathbf{v} - \boldsymbol{\omega} \times \boldsymbol{\omega} \times \mathbf{r} + \Gamma\hat{\mathbf{t}} \right) + \lambda_J \tag{2}$$

where $\boldsymbol{\lambda}_\mathbf{r}$ and $\boldsymbol{\lambda}_\mathbf{v}$ are the co-sates functions and λ_J is an additional constant coefficient used to multiple our cost function $J = \lambda_J(t_f - t_0)$. This additional constant increases numerical stability and offers an additional degree of freedom when performing the Backward Generation of Optimal Examples (BGOE) in Sec.II.B [7]. For a trajectory to be optimal the classical necessary condition tells us that thrust direction $\hat{\mathbf{t}}^*$ needs to minimize the Hamiltonian, hence:

$$\hat{\mathbf{t}}^* = -\frac{\boldsymbol{\lambda}_\mathbf{v}}{|\boldsymbol{\lambda}_\mathbf{v}|} \tag{3}$$

The augmented system of equations is then obtained by taking the derivatives of the Hamiltonian with respect to $\dot{\mathbf{x}}_T = \frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}$ and $\dot{\boldsymbol{\lambda}} = -\frac{\partial \mathcal{H}}{\partial \mathbf{x}_T}$:

$$\begin{cases} \dot{\mathbf{r}} = \mathbf{v} \\ \dot{\mathbf{v}} = -\frac{\mu}{r^3}\mathbf{r} - 2\boldsymbol{\omega} \times \mathbf{v} - \boldsymbol{\omega} \times \boldsymbol{\omega} \times \mathbf{r} - \Gamma\frac{\boldsymbol{\lambda}_\mathbf{v}}{|\boldsymbol{\lambda}_\mathbf{v}|} \\ \dot{\boldsymbol{\lambda}}_\mathbf{r} = \mu\left(\frac{\boldsymbol{\lambda}_\mathbf{v}}{r^3} - 3(\boldsymbol{\lambda}_\mathbf{v} \cdot \mathbf{r})\frac{\mathbf{r}}{r^5}\right) - \boldsymbol{\omega} \times \boldsymbol{\omega} \times \boldsymbol{\lambda}_\mathbf{v} \\ \dot{\boldsymbol{\lambda}}_\mathbf{v} = -\boldsymbol{\lambda}_\mathbf{r} + 2\boldsymbol{\omega} \times \boldsymbol{\lambda}_\mathbf{v} \end{cases} \tag{4}$$

Since we consider this to be a free time problem, a trajectory also need to fulfill the $\mathcal{H}|_{t=t_f} = 0$ condition in order to be optimal. Let's find one solution, which we'll refer to as the "nominal trajectory" for this problem in the rest of the paper. We introduce a shooting function to solve the Two Points Boundary Value Problem (TPBVP):

$$\phi(\boldsymbol{\lambda}_{\mathbf{r}_0}, \boldsymbol{\lambda}_{\mathbf{v}_0}, \lambda_J, t_f) = \{\mathbf{r}_f - \mathbf{r}_t, \mathbf{v}_f - \mathbf{v}_t, \mathcal{H}_f, ||\boldsymbol{\lambda}|| - 1\} \tag{5}$$

where $\boldsymbol{\lambda}_{\mathbf{r}_0}, \boldsymbol{\lambda}_{\mathbf{v}_0}$ are the initial co-states values and t_f is the time-of-flight. The final conditions $\mathbf{r}_f$, $\mathbf{v}_f$, and $\mathcal{H}_f$ are computed by propagating Eq.4 from the initial conditions until t_f. We find a root of ϕ using the sequential quadratic programming solver SNOPT [13]. The constraint on the magnitude of the initial co-states $||\boldsymbol{\lambda}|| - 1$ is not strictly necessary, we use it here as it improves numerical stability. As described in [7], the existence of multiple roots for Eq.5 corresponds to the presence of local minima. While not rigorous, we circumvent this problem by solving this problem using different initial guesses for the numerical solver, thereby increasing our confidence that our solution corresponds to the optimal strategy. The nominal trajectory for this optimal control problem has a time-of-flight of $t_f^* = 4.62$ years, see Fig. 1. All values related to this problem are listed in App.A.

Mass-optimal landing on asteroid

We also consider a mass-optimal landing on the asteroid Psyche. Let $\mathcal{R} = \left[\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}\right]$ be a rotating frame with angular velocity $\boldsymbol{\omega}\hat{\mathbf{k}}$ such that the asteroid remains stationary in $\mathcal{R}$. The dynamics is described by the following ordinary differential equations:

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \dot{v}_x = -\frac{\mu}{r^3}x + 2\omega v_y + \omega^2 x + u\frac{c_1}{m}i_x \\ \dot{v}_y = -\frac{\mu}{r^3}y - 2\omega v_x + \omega^2 y + u\frac{c_1}{m}i_y \\ \dot{v}_z = -\frac{\mu}{r^3}z + u\frac{c_1}{m}i_z \\ \dot{m} = -u\frac{c_1}{I_{sp}g_0} \end{cases} \tag{6}$$

The state vector $\mathbf{x}_L$ (subscript $\square_L$ to indicate the "landing" optimal control problem) contains the position $\mathbf{r} = [x, y, z]$, velocity $\mathbf{v} = [v_x, v_y, v_z]$ and mass m. The position $\mathbf{r}$ and velocity $\mathbf{v}$ are both defined in the rotating frame $\mathcal{R}$. Note that $r = \sqrt{x^2 + y^2 + z^2}$. The system is

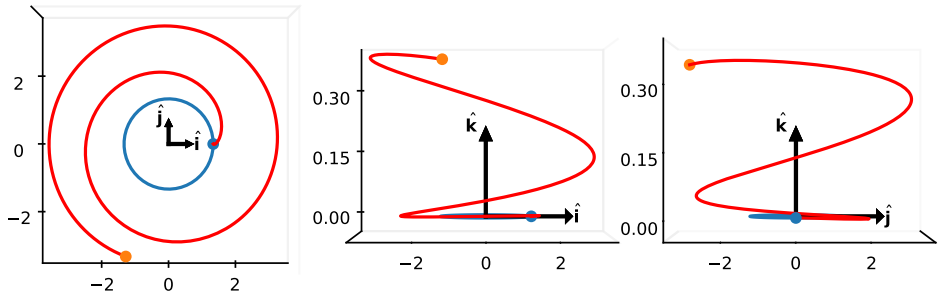

Fig. 1: Interplanetary transfer shown in rotating frame $\mathcal{F}$. Axis unit is AU.

controlled by the thrust direction described by the unit vector $\hat{\mathbf{t}} = [i_x, i_y, i_z]$ and the throttle $u \in [0, 1]$. This is a free-time optimal control problem for which we need to find the controls $u(t)$ and $\hat{\mathbf{t}}(t)$, where $t \in [t_0, t_f]$, such that, under the dynamics described by Eq.6, the state is steered from any initial state $\mathbf{r}_0$, $\mathbf{v}_0$, m_0 to the desired target state $\mathbf{r}_t$, $\mathbf{v}_t$ and final mass m_f (which is left free). To avoid having to immediately solve the mass-optimal control problem we follow the steps laid out in [2, 14] and introduce the following cost function to minimize:

$$J(u(t), t_f) = \int_0^{t_f} \{u - \epsilon \log [u(1-u)]\} \, \mathrm{d}t \tag{7}$$

where the continuation parameter ϵ and the logarithmic barrier allow us to smooth out the problem and keep $u \in [0, 1]$ in the desired bounds. The mass-optimal problem corresponds to $\lim_{\epsilon \to 0} \quad J(u(t), t_f) = (m_0 - m_f) \cdot \frac{c_1}{I_{sp} g_0}$ (substitute Eq.6 in Eq.7) and is very difficult to solve without a good initial guess. We bypass this issue by first solving the problem with $\epsilon = 1$ and use this solution as an initial guess for a slightly smaller ϵ, repeating this cycle until we reach $\epsilon < 10^{-6}$. Let's find the necessary conditions for optimality using Pontryagin's Maximum Principle [11]. We define the Hamiltonian:

$$\begin{aligned} \mathcal{H}(\mathbf{r}, \mathbf{v}, m, \boldsymbol{\lambda}_\mathbf{r}, \boldsymbol{\lambda}_\mathbf{v}, \lambda_m, u, \hat{\mathbf{t}}) = \boldsymbol{\lambda}_\mathbf{r} \cdot \mathbf{v} + \\ \boldsymbol{\lambda}_\mathbf{v} \cdot \left(-\frac{\mu}{r^3}\mathbf{r} - 2\boldsymbol{\omega} \times \mathbf{v} - \boldsymbol{\omega} \times \boldsymbol{\omega} \times \mathbf{r} + u\frac{c_1}{m}\hat{\mathbf{t}} \right) \\ + \lambda_m \left(-u\frac{c_1}{I_{sp} g_0} \right) + u - \epsilon \cdot \log [u(1-u)] \end{aligned} \tag{8}$$

where $\boldsymbol{\lambda}_\mathbf{r}$, $\boldsymbol{\lambda}_\mathbf{v}$ and λ_m are the co-sates functions. Note that we drop the dependence on time for brevity. The optimal thrust direction $\hat{\mathbf{t}}^*$ and u^* both need to minimize the Hamiltonian, hence:

$$\hat{\mathbf{t}}^* = -\frac{\boldsymbol{\lambda}_\mathbf{v}}{|\boldsymbol{\lambda}_\mathbf{v}|} \quad , \quad u^* = \frac{2\epsilon}{2\epsilon + SF + \sqrt{4\epsilon^2 + SF^2}} \tag{9}$$

where SF is a switching function whose zero-crossings correspond to switches between minimal ($u = 0$) and maximal throttle ($u = 1$):

$$SF = \boldsymbol{\lambda}_\mathbf{v} \frac{c_1}{m} \hat{\mathbf{t}}^* - \lambda_m \cdot \frac{c_1}{I_{sp} g_0} + 1 \tag{10}$$

The augmented system of equations is again obtained by taking the derivatives of the Hamiltonian with respect to $\dot{\mathbf{x}}_L = \frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}$ and $\dot{\boldsymbol{\lambda}} = -\frac{\partial \mathcal{H}}{\partial \mathbf{x}_L}$:

$$\begin{cases} \dot{\mathbf{r}} = \mathbf{v} \\ \dot{\mathbf{v}} = -\frac{\mu}{r^3}\mathbf{r} - 2\boldsymbol{\omega} \times \mathbf{v} - \boldsymbol{\omega} \times \boldsymbol{\omega} \times \mathbf{r} - u^* \frac{c_1}{m} \frac{\boldsymbol{\lambda}_\mathbf{v}}{|\boldsymbol{\lambda}_\mathbf{v}|} \\ \dot{m} = -u^* \frac{c_1}{I_{sp} g_0} \\ \dot{\boldsymbol{\lambda}}_\mathbf{r} = \mu \left(\frac{\boldsymbol{\lambda}_\mathbf{v}}{r^3} - 3(\boldsymbol{\lambda}_\mathbf{v} \cdot \mathbf{r}) \frac{\mathbf{r}}{r^5} \right) - \boldsymbol{\omega} \times \boldsymbol{\omega} \times \boldsymbol{\lambda}_\mathbf{v} \\ \dot{\boldsymbol{\lambda}}_\mathbf{v} = -\boldsymbol{\lambda}_\mathbf{r} + 2\boldsymbol{\omega} \times \boldsymbol{\lambda}_\mathbf{v} \\ \dot{\lambda}_m = -\frac{c_1 u^*}{m^2} \boldsymbol{\lambda}_\mathbf{v} \cdot \frac{\boldsymbol{\lambda}_\mathbf{v}}{|\boldsymbol{\lambda}_\mathbf{v}|} \end{cases} \tag{11}$$

Since this is free time problem, we need to add the condition $\mathcal{H}|_{t=t_f} = 0$ and to leave the final mass m_f free we need the transversality condition $\lambda_m|_{t=t_f} = 0$. We introduce a shooting function to solve the TPBVP:

$$\phi(\boldsymbol{\lambda}_{\mathbf{r}_0}, \boldsymbol{\lambda}_{\mathbf{v}_0}, \lambda_{m_0}, t_f) = \left\{ \mathbf{r}_f - \mathbf{r}_t, \mathbf{v}_f - \mathbf{v}_t, \mathcal{H}_f, \lambda_{m_f} \right\} \tag{12}$$

where $\boldsymbol{\lambda}_{\mathbf{r}_0}, \boldsymbol{\lambda}_{\mathbf{v}_0}, \lambda_{m_0}$ are the initial co-states values and t_f is the time-of-flight. The final conditions $\mathbf{r}_f$, $\mathbf{v}_f$, $\mathcal{H}_f$ and λ_{m_f} are computed by propagating Eq.4 from the initial conditions until t_f. As explained for the time-optimal transfer, we solve this TPBVP with multiple restarts (different initial guesses for the root solver) such as to increase our confidence that our solution is the optimal landing strategy. The nominal trajectory for this optimal control problem has a time-of-flight of $t_f^* = 38$ min, see Fig.2. All values related to this problem are listed in App.A.

B. Behavioural cloning

We train two separate neural models to represent to optimal policies for each problem as a function of the spacecraft state $\mathbf{x}$. The resulting neural state feedback

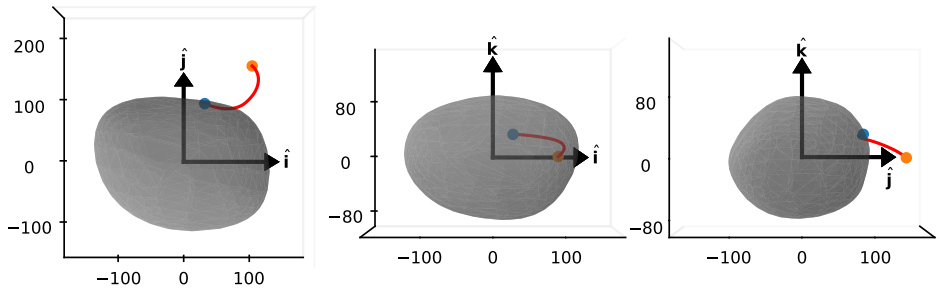

Fig. 2: Landing on Psyche shown in rotating frame $\mathcal{R}$. Axis unit is km.

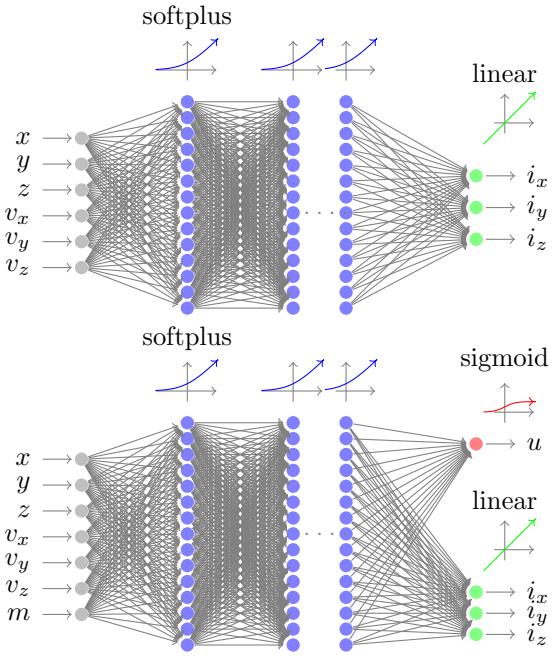

Fig. 3: G&CNET architectures: transfer (top) and landing (bottom). Adapted from [2].

is called a G&CNET: $\mathcal{N}_{\text{transfer}}(\mathbf{x}_T) = \hat{\mathbf{t}}^*$ for the transfer and $\mathcal{N}_{\text{landing}}(\mathbf{x}_L) = [u^*, \hat{\mathbf{t}}^*]$ for the landing. These simple feedforward neural networks can then be used in Eq.1 and Eq.6 respectively to simulate the spacecraft dynamics. The network architectures are shown in Fig.3, for both problems we use 3 hidden layers, each with 128 neurons. We use softplus activation functions for the hidden layers, allowing us to obtain a continuous and differentiable representation of the optimal controls. To avoid saturation issues during training we use linear output activation functions, except for the throttle in the landing problem where we use a sigmoid activation function to keep u bounded between $[0, 1]$.

Training datasets

We generate training datasets for both optimal control problems by leveraging a data augmentation technique called the "Backward Generation of Optimal Examples" (BGOE) [2, 7]. This technique exploits the fact that any solution to the augmented systems of equations (Eq.4 for the transfer and Eq.11 for the landing problem) which satisfies the necessary conditions for optimality is a local optimal trajectory which can be used to learn from. The basic premise of the BGOE is that once a nominal solution is found, one can perturb the final co-states of the augmented system by some carefully crafted vector $\boldsymbol{\Delta}$:

$$\boldsymbol{\lambda}_f^+ = \boldsymbol{\lambda}_f + \boldsymbol{\lambda}_f \cdot \boldsymbol{\Delta} \tag{13}$$

where $\boldsymbol{\Delta}$ needs to be chosen such that the necessary conditions for optimality are still satisfied. In the case of the transfer, each element in $\boldsymbol{\Delta}$ is a number uniformly sampled in $\mathcal{U}(-\delta, \delta)$ except for λ_J which we use to satisfy the free time condition $\mathcal{H}_f = 0$. For the landing problem we also sample each element in $\mathcal{U}(-\delta, \delta)$ except for $\lambda_{m_f} = 0$ which we leave unchanged (free final mass transversality condition) and we use the final mass m_f to satisfy the free time condition $\mathcal{H}_f = 0$ using a root solver and the final mass m_f of the nominal trajectory as initial guess. $\boldsymbol{\lambda}_f^+$ can then be used to back-propagate the augmented system of equation in time. For small perturbations δ, this will result in a new optimal trajectory with the same final states as the nominal trajectory (except for the final mass m_f which is left free in the landing problem) and different initial conditions. While we cannot directly chose the initial conditions, the BGOE allows us to generate hundred thousand optimal trajectories at a fraction of the computational cost one would incur if one had to solve each TPBVP individually [2, 7].

For the transfer we use 400,000 optimal trajectories, each of which is sampled in 100 points equally distance in time, resulting in 40,000,000 optimal state-action pairs to learn from. We show a portion of this dataset in Fig.4. In our experiments, we found that decreasing the corre-

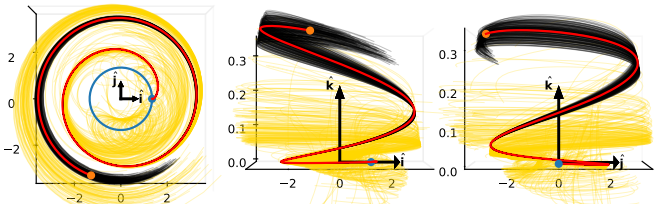

Fig. 4: Bundle of 400 optimal trajectories. Co-state perturbation δ: 1‰ (black), 8% (gold). Interplanetary transfer shown in rotating frame $\mathcal{F}$. Axis unit is AU.

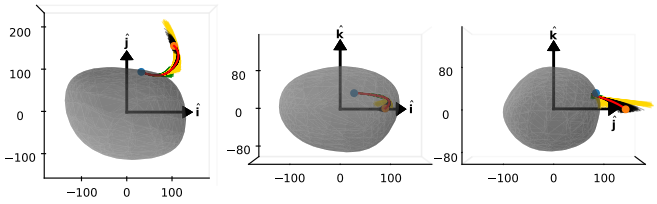

Fig. 5: Bundle of 2,000 optimal trajectories. Co-state perturbation δ: 1‰ (black), 8‰ (gold), 2% (green). Landing on Psyche shown in rotating frame $\mathcal{R}$. Axis unit is km.

lation with the nominal trajectory is crucial to successfully train the G&CNET. We do this by randomly sampling the back-propagation time $a(1+c)t^*_{f_{nom}}$ where $t^*_{f_{nom}}$ is the optimal time-of-flight of the nominal trajectory, $a = 1$ and $c \in [0, 0.07]$. We create two bundles of trajectories, one with a small perturbation size $\delta = 1$‰, hence it closely follows the nominal trajectory, and one with a much larger perturbation size $\delta = 8\%$. As explained in [7], while these trajectories are far away from the nominal trajectory, they were crucial for successful training. We also generate a separate dataset of 1,000 trajectories with $\delta = 1$‰ which will be used to in Subsec.III.

For the landing we use 300,000 optimal trajectories, each of which is sampled in 100 points equally distance in time, resulting in 30,000,000 optimal state-action pairs to learn from, see a portion of the dataset in Fig.5. We use $c \in [0, 0.05]$ and create three bundles of different perturbation size and only back-propagated until a fraction of $t^*_{f_{nom}}$: $\delta = 1$‰ ($a = 1$), $\delta = 8$‰ ($a = 0.8$) and $\delta = 2\%$ ($a = 0.5$). We also generate a separate dataset of 1,000 trajectories with $\delta = 0.5$‰ ($a = 1$) for Subsec.III.

Central to our work is the *heyoka* Python library [15], which we use for all numerical propagation. This Taylor based method allows us to quickly integrate our equations with a numerical tolerance set to machine level, i.e. 10^{-16}. In addition, as we will see in Sec.II.C, the library also allows for a seamless integration of feed forward neural networks in our ODEs and automatic differentiation.

Training procedure

The generated datasets are split up into 80% training data and 20% validation data. We train the G&CNET for the transfer problem over $p = 500$ epochs using an initial learning rate of $\alpha = 5{\cdot}10^{-5}$ with the Adam optimizer [16] and no weight decay. We also use a scheduler which decreases the learning rate by a factor of 90% whenever the loss does not improve over $p = 10$ epochs when evaluated on the validation dataset. The loss function is computed using the cosine similarity of the estimated thrust direction $\hat{\mathbf{t}}_{nn}$ and the ground truth $\hat{\mathbf{t}}^*$, thereby allowing the network to solely focus on the direction and ignoring the norm:

$$\text{cosine\_similarity}\left(\hat{\mathbf{t}}_{nn}, \hat{\mathbf{t}}^*\right) = \frac{\hat{\mathbf{t}}^* \cdot \hat{\mathbf{t}}_{nn}}{|\hat{\mathbf{t}}^*||\hat{\mathbf{t}}_{nn}|} \tag{14}$$

$$\mathcal{L}_{transfer}\left(\hat{\mathbf{t}}_{nn}, \hat{\mathbf{t}}^*\right) = 1 - \text{cosine\_similarity}\left(\hat{\mathbf{t}}_{nn}, \hat{\mathbf{t}}^*\right) \tag{15}$$

We report the loss over the epochs during training and the error in thrust direction (represented by the cosine similarity) over one trajectory in the validation dataset in Fig.6. Notice how the final part of the transfer (last 0.5 year) is usually where the largest errors occur, likely due to a lack of training data in the region of space corresponding to the final part of the transfer.

For the landing problem we use exactly the same training setup except that the total amount of epochs is now $p = 400$ and the loss function contains an additional term to penalize the Mean Squared Error (MSE) between the estimated throttle u_{nn} and the ground truth u^*:

$$\begin{aligned}\mathcal{L}_{landing}\left(u_{nn}, u^*, \hat{\mathbf{t}}_{nn}, \hat{\mathbf{t}}^*\right) = \text{MSE}(u_{nn}, u^*) + 1 \\ -\text{cosine\_similarity}\left(\hat{\mathbf{t}}_{nn}, \hat{\mathbf{t}}^*\right)\end{aligned} \tag{16}$$

Fig.6 shows the loss during training, the estimated throttle versus the ground truth and the cosine similarity between the estimated thrust direction and the ground truth over one trajectory in the validation dataset.

C. Neural ODEs

We use the term Neural Ordinary Differential Equations (Neural ODEs), as popularized in [17], to denote Ordinary Differential Equations which contain an artificial neural network on their right hand side. To illustrate how these can be used to improve the performance of G&CNETs let us a consider the generic system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{x}))$, whose solution $\mathbf{x}(t; \mathbf{x_0}, \boldsymbol{\theta})$ depends explicitly on the initial conditions $\mathbf{x_0}$ and the network parameters $\boldsymbol{\theta}$. Contrary to the behavioural cloning approach, which aims to minimize the approximation error of the optimal control (see loss functions in Eq.15 and Eq.16), imagine that we could update our neural network

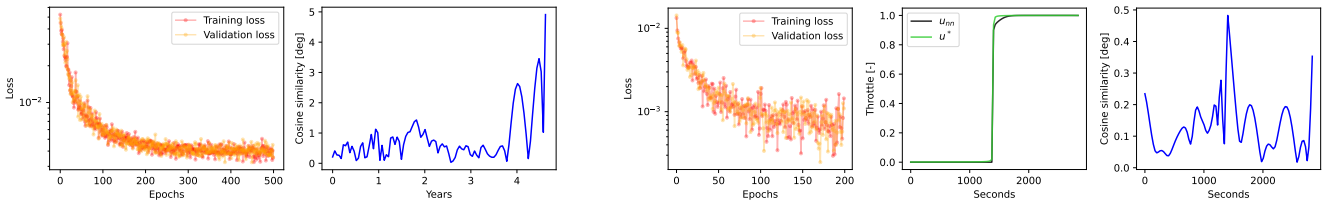

Fig. 6: Loss during training and cosine similarity between the estimated thrust direction and the ground truth over one optimal trajectory in the validation dataset. Transfer (left) and landing (right). For the landing we also show the estimated throttle versus ground truth.

Algorithm 1 Neural ODE fix algorithm.

1: Generate training dataset D_{BC} ▷ Optimal trajectories $[\mathbf{x}^*, \mathbf{u}^*, t^*]$
2: Train G&CNET $\mathcal{N}_{\boldsymbol{\theta}}$ on D_{BC} ▷ Behavioural Cloning
3: Generate training dataset D_T ▷ Optimal trajectories $[\mathbf{x}^*, \mathbf{u}^*, t^*]$
4: Generate validation dataset D_V ▷ Optimal trajectories $[\mathbf{x}^*, \mathbf{u}^*, t^*]$
5: **for** $i = 1$ to N **do**
6: Propagate dynamics $\dot{\mathbf{x}}$ and ODE sensitivities $\frac{d}{dt}\left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}}\right)$ from $\mathbf{x}^*_{0,i}$ until t^*_i in D_T
7: Compute gradients: $\frac{\partial \mathcal{L}_N}{\partial \boldsymbol{\theta}} = \frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}} \frac{\partial \mathcal{L}_N}{\partial \mathbf{x}}$
8: Find optimal learning rate α ▷ Line search or Adam Optimizer
9: Perform gradient descent step: $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \cdot \frac{\partial \mathcal{L}_N}{\partial \boldsymbol{\theta}}$
10: **Return** Best $\mathcal{N}_{\boldsymbol{\theta}}$ on validation dataset D_V

parameters $\boldsymbol{\theta}$ such as to decrease some loss $\mathcal{L}(\mathbf{x}(t; \mathbf{x_0}, \boldsymbol{\theta}))$, which instead depends on the current solution of the system. Let us consider a new loss function $\mathcal{L}_N$ which aims to directly minimize the error between the target state $\mathbf{x}_{target}$ and the current solution of our system when evaluated from some initial conditions $\mathbf{x_0}$ until the corresponding optimal time-of-flight $t = t^*$:

$$\mathcal{L}_N = (\mathbf{x}_{target} - \mathbf{x}(t^*))^2 \tag{17}$$

The gradients of the loss with respect to the network parameters can be rewritten as:

$$\frac{\partial \mathcal{L}_N}{\partial \boldsymbol{\theta}} = \frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}} \frac{\partial \mathcal{L}_N}{\partial \mathbf{x}} \tag{18}$$

One can easily find $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$ analytically, hence we only need to compute the ODE sensitivities $\frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}}$ via the variational equations (or the Pontryagin's adjoint method):

$$\frac{d}{dt}\left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}}\right) = \nabla_{\mathbf{x}} \dot{\mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}} + \frac{\partial \dot{\mathbf{x}}}{\partial \boldsymbol{\theta}} \tag{19}$$

The *heyoka* library [15] allows to compute these derivatives very easily, see tutorial[1]. Finally, a gradient descent step can be used to update the neural network weights:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \cdot \frac{\partial \mathcal{L}_N}{\partial \boldsymbol{\theta}} \tag{20}$$

[1]Tutorial heyoka: `https://bluescarni.github.io/heyoka.py/notebooks/NeuralODEs.html`

The training pipeline is provided as pseudo-code in Algorithm 1, let's walk through each step. First we follow the same steps as in the behavioural cloning pipeline: we generate a training dataset D_{BC} (Line 1), and we train the G&CNET $\mathcal{N}_{\boldsymbol{\theta}}$ on D_{BC} (Line 2). We also generate two separate datasets, which will be use to train (D_T) and validate (D_V) the Neural ODE fix algorithm (Lines 3 and 4). Until now, we always only used the optimal state-action pairs $[\mathbf{x}^*, \mathbf{u}^*]$ from our generated datasets. However solving optimal trajectories also gives the optimal time-of-flight t^*, recall that t_f is part of the decision vector in both shooting functions Eq.5 and Eq.12. This is crucial here as it allows us to evaluate the following system:

$$\begin{cases} \dot{\mathbf{x}} = f(\mathbf{x}, \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{x})) \\ \frac{d}{dt}\left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}}\right) = \nabla_{\mathbf{x}} \dot{\mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}} + \frac{\partial \dot{\mathbf{x}}}{\partial \boldsymbol{\theta}} \end{cases} \tag{21}$$

from $\mathbf{x}^*_{0,i}$ until $t = t^*_i$ in D_T (Line 6). Since $\mathcal{N}_{\boldsymbol{\theta}}$ approximates the optimal control policy, errors will accumulate over the trajectory, resulting in a non-zero error to the target state at $t = t^*$ which can be captured by the loss $\mathcal{L}_N$ in Eq.17. The ODE sensitivities to the network parameters $\boldsymbol{\theta}$ can now be used to compute the gradients of $\mathcal{L}_N$ with respect to these parameters (Line 7). In our experiments we found that using a line search (for instance using *scipy.optimize.minimize_scalar*) or the Adam Optimizer [16] to find the optimal learning rate α for the gra-

dient descent step helps considerably to stabilize training (Lines 8 and 9). Finally, we return the policy which performs best on the validation dataset D_V (Line 10). Note that in our experiments we distinguish between looping over a single trajectory or batches of trajectories in D_T (Line 5).

D. DAGGER

We compare our approach to the popular DAGGER (Dataset Aggregation) algorithm [18]. The idea behind this technique is to let the neural network explore the environment and query an expert (in this case solve the corresponding TPBVP) to obtain the optimal policy, thereby gradually collecting optimal state-action pairs from the states that the network is likely to visit. This technique addresses the issue that the distribution of the initial training data used in behavioural cloning rarely covers the state-space encountered by the network perfectly. Since no new methods or equations need to be introduced, let's run through a concrete example by following the steps laid out in the pseudo-code Algorithm 2. Just like for the Neural ODE fix algorithm we first follow the behavioural cloning pipeline by generating a training dataset D_{BC} (Line 1) and training the G&CNET $\mathcal{N}_{\boldsymbol{\theta}}$ on D_{BC} (Line 2). Here we also generate two separate datasets, which will be used to train (D_T) and validate (D_V) the DAGGER algorithm (Lines 3 and 4). We then propagate the dynamics $\dot{\mathbf{x}} = f(\mathbf{x}, \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{x}))$ from initial conditions $\mathbf{x}^*_{0,i}$ until the corresponding optimal time-of-flight t^*_i (Line 6). Note that compared to Algorithm 1, here it is not as important to know the optimal time-of-flight. The resulting trajectory is then sampled into T steps (Line 7) and we solve all the TPBVPs starting from the sampled states $\mathbf{x}_j$ until the target state (Line 9). Now we can evaluate how well the network approximates the optimal control policy $\mathbf{u}^*$ at state $\mathbf{x}_j$ and add the corresponding optimal trajectories to a new training dataset D_{DG} when the approximation error surpasses some user-defined threshold (Lines 10, 11 and 12). Finally the network is trained using Behavioural Cloning on both the old dataset D_{BC} and the new dataset D_{DG} which contains new state-action pairs that are likely to be encountered when deployed and which the network struggles to approximate accurately. In order to prevent catastrophic forgetting, we found that it is necessary to consider a slightly modified loss function when training $\mathcal{N}_{\boldsymbol{\theta}}$ in Line 13:

$$\mathcal{L} = \mathcal{L}_{D_{BC}} + 0.1 \cdot \mathcal{L}_{D_{DG}} \tag{22}$$

where $\mathcal{L}_{D_{BC}}$ and $\mathcal{L}_{D_{DG}}$ are Eq.15 (transfer) or Eq.16 (landing) when evaluated on the optimal state-action pairs of D_{BC} and D_{DG}. This new loss function allows us to weigh the contribution of each dataset differently which resulted in more accurate networks when evaluating these on the validation dataset D_V (Line 14).

III. Results & Discussion

We show the final position and velocity errors on the training dataset D_T and validation dataset D_V (500 trajectories each) for both optimal control problems in Fig.7 and Fig.8. In all cases, the G&CNETs trained solely with behavioural cloning can be improved considerably. The Neural ODE fix reduces the final mean position and velocity errors by 70% for the transfer and by 98% and 40% for the landing. DaGGER reduces the final mean position and velocity errors by 14% and 28% for the transfer and by 22% and 15% for the landing. We also refined G&CNETs on a single trajectory using the Neural ODE fix. In the case of the interplanetary transfer we improved the final position error from $1,241,662$ km to 2991 km (99% reduction) and final velocity error from $9 \cdot 10^{-2}$ km/s to $5 \cdot 10^{-4}$ km/s (99% reduction). For the asteroid landing problem we improved the final position error from 452 m to 5.4 m (99% reduction) and final velocity error from 0.9 m/s to 0.5 m/s (44% reduction). We found that the main advantages of DaGGER are twofold. First, it allows us to collect states, with their corresponding optimal controls, which are likely to be encountered by the G&CNET, thereby forming a diverse dataset. Second, the computational effort required scales well with the size of the neural network, in contrast to the Neural ODE fix which is better suited for small networks due to the large amount of ODE sensitivities which need to be computed during each iteration. The downsides of DaGGER are that one constantly needs to query an expert, in our case this involves solving TPBVPs with a shooting method. Hence, one constantly runs the risk of injecting suboptimal trajectories (local minima) in the training dataset. In addition, for difficult problems such as the mass-optimal landing where a continuation (homotopy) approach is required to even find a solution, the DaGGER approach suffers from long convergence times or sometimes no convergence at all. Finally, the loss function and training hyper-parameters need to be carefully chosen for DaGGER such as to avoid catastrophic forgetting. The main advantages of the Neural ODE fix are that it allows us to learn from the dynamics and it is possible to give some final state errors more weight than others, thereby correcting what is more important to the user. Finally, in our experiments the Neural ODE fix did not cause catastrophic forgetting.

Algorithm 2 DAGGER algorithm, adapted from [18].

Generate training dataset D_{BC} ▷ Optimal trajectories $[\mathbf{x}^*, \mathbf{u}^*, t^*]$
Train G&CNET $\mathcal{N}_{\boldsymbol{\theta}}$ on D_{BC} ▷ Behavioural Cloning
Generate training dataset D_T ▷ Optimal trajectories $[\mathbf{x}^*, \mathbf{u}^*, t^*]$
Generate validation dataset D_V ▷ Optimal trajectories $[\mathbf{x}^*, \mathbf{u}^*, t^*]$
for $i = 1$ to N **do**
    Propagate $\dot{\mathbf{x}} = f(\mathbf{x}, \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{x}))$ from $\mathbf{x}^*_{0,i}$ until t^*_i in D_T ▷ Eq.1 or Eq.6
    Sample trajectory in T steps
    for $j = 1$ to T **do**
        Solve TPBVP from $\mathbf{x}_j$ with Pontryagin $D_j = \{(\mathbf{x}^*, \mathbf{u}^*)\}$ ▷ Eq.5 or Eq.12
        Compute loss $\mathcal{L}_j = \mathcal{L}(\mathcal{N}_{\boldsymbol{\theta}}(\mathbf{x}_j), \mathbf{u}^*)$ ▷ Eq.15 or Eq.16
        if $\mathcal{L}_j >$ threshold **then**
            Aggregate datasets: $D_{DG} \leftarrow D_{DG} \cup D_j$
    Train G&CNET $\mathcal{N}_{\boldsymbol{\theta}}$ on D_{DG} and D_{BC} ▷ Behavioural Cloning
Return Best $\mathcal{N}_{\boldsymbol{\theta}}$ on validation dataset D_V

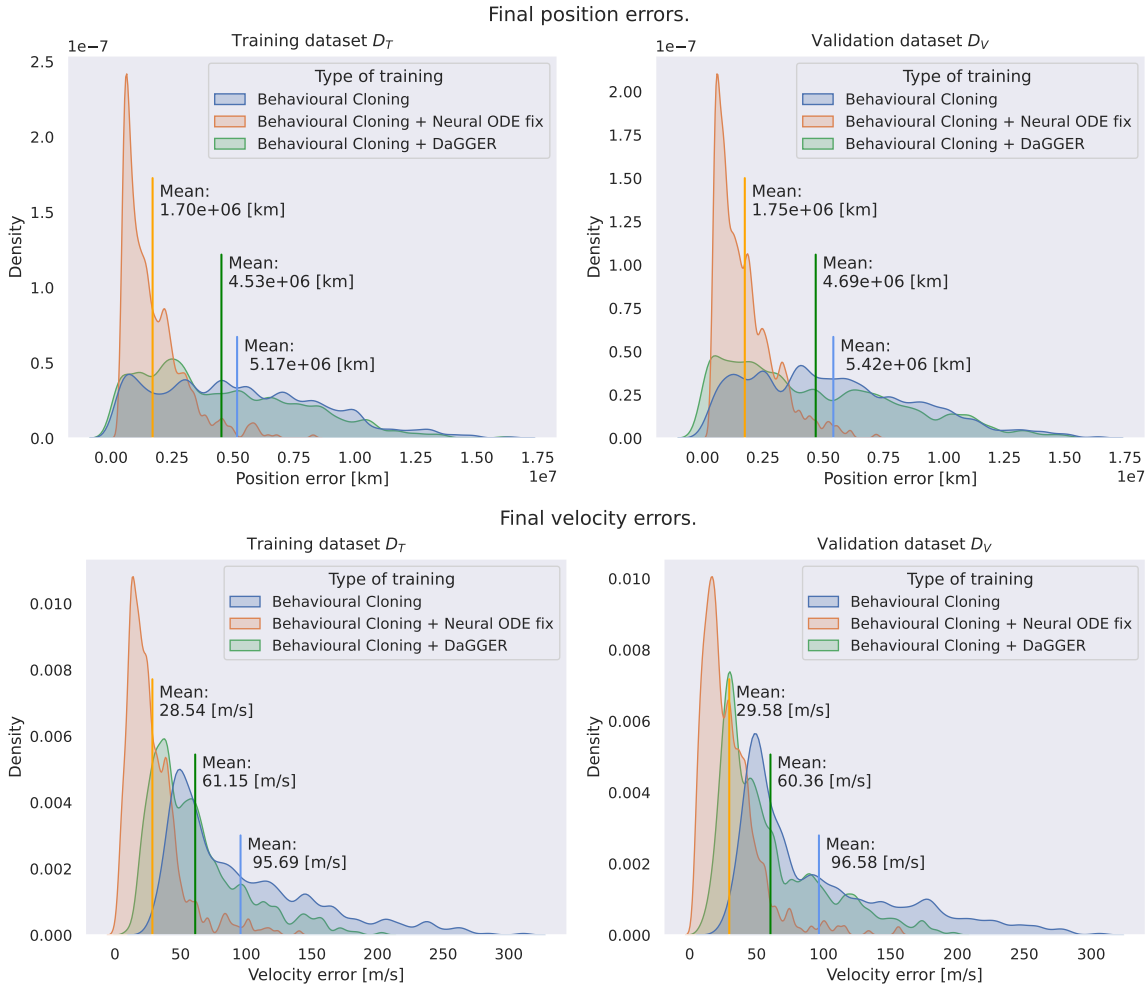

Fig. 7: Position and velocity errors over training and testing dataset of initial G&CNET and refined G&CNETs for interplanetary transfer problem.

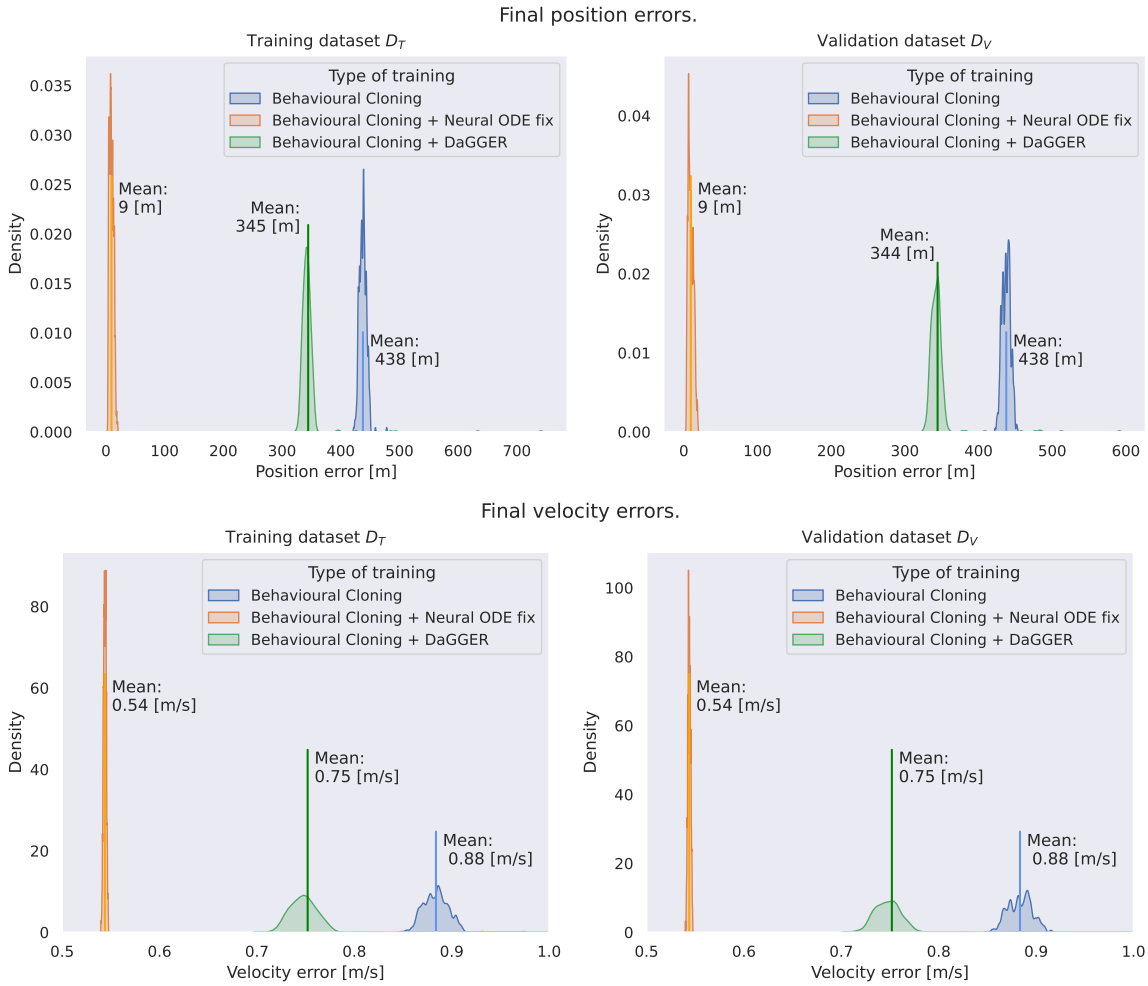

Fig. 8: Position and velocity errors over training and testing dataset of initial G&CNET and refined G&CNETs for asteroid landing problem.

IV. Conclusion

The use of Neural ODEs to improve the accuracy of Guidance & Control Networks has been studied. Both a time-optimal interplanetary transfer and a mass-optimal asteroid landing are considered. In all cases, we find that the final position and velocity errors to the target can be substantially reduced, both for a single trajectory and for a batch of 500 different trajectories. In the case of the interplanetary transfer the final position and velocity errors were reduced by 99% for a single trajectory and the mean position and velocity errors were reduced by 70% for a batch of trajectories. In the case of the asteroid landing the final position and velocity errors were reduced by 99% and 44% for a single trajectory and by the mean position and velocity errors were reduced by 98% and 40% for a batch of trajectories. We were not able to reduce the final errors as drastically with the popular DaGGER approach. Nevertheless we acknowledge the strength and weaknesses of both approaches, most notably the fact that the Neural ODE fix is only suited for small networks, due to the computational burden associated with computing the ODE sensitivities at each iteration.

Appendix

A. Optimal control problems values

References

[1] Sánchez-Sánchez, C. and Izzo, D. "Real-time optimal control via deep neural networks: study on landing problems." Journal of Guidance, Control, and Dynamics, Vol. 41, No. 5, pp. 1122–1135, 2018.

[2] Izzo, D. and Öztürk, E. "Real-time guidance for low-thrust transfers using deep neural networks." Jour-

Tab. 1: Initial conditions, final conditions and constants used for optimal control problems.

Problem	Initial conditions		Final conditions		Frame	Constants	
Interplanetary transfer	x_0	-1.1874388 AU	x_f (R)	1.3 AU	$\mathcal{F}$	Γ	0.1 mm/s^2
	y_0	-3.0578396 AU	y_f	0 AU		μ	(Sun) m^3/s^2
	z_0	0.3569406 AU	z_f	0 AU			
	v_{x_0}	-48.17 km/s	v_{x_f}	0 km/s			
	v_{y_0}	18.30 km/s	v_{y_f}	0 km/s			
	v_{z_0}	0.64 km/s	v_{z_f}	0 km/s			
Asteroid landing	x_0	100 km	x_f	30.27 km	$\mathcal{R}$	I_{sp}	600 s
	y_0	150 km	y_f	90.33 km		g_0	9.8 m/s^2
	z_0	0 km	z_f	32.09 km		c_1	80 N
	v_{x_0}	0.025 km/s	v_{x_f}	0 km/s		μ	1530348199 m^3/s^2
	v_{y_0}	-0.025 km/s	v_{y_f}	0 km/s		ω	0.00041596 rad/s
	v_{z_0}	0.02 km/s	v_{z_f}	0 km/s			
	m_0	600 kg	m_f	left free	-		

nal of Guidance, Control, and Dynamics, Vol. 44, No. 2, pp. 315–327, 2021.

[3] Li, H., Baoyin, H., and Topputo, F. "Neural networks in time-optimal low-thrust interplanetary transfers." IEEE Access, Vol. 7, pp. 156413–156419, 2019.

[4] Federici, L., Benedikter, B., and Zavoli, A. "Deep Learning Techniques for Autonomous Spacecraft Guidance During Proximity Operations." Journal of Spacecraft and Rockets, pp. 1–12, 2021.

[5] Hovell, K. and Ulrich, S. "On Deep Reinforcement Learning for Spacecraft Guidance." "AIAA Scitech 2020 Forum," p. 1600. 2020.

[6] Cheng, L., Wang, Z., Jiang, F., and Zhou, C. "Real-time optimal control for spacecraft orbit transfer via multiscale deep neural networks." IEEE Transactions on Aerospace and Electronic Systems, Vol. 55, No. 5, pp. 2436–2450, 2018.

[7] Izzo, D. and Origer, S. "Neural representation of a time optimal, constant acceleration rendezvous." Acta Astronautica, Vol. 204, pp. 510–517, 2023. ISSN 0094-5765. doi:https://doi.org/10.1016/j.actaastro.2022.08.045.

[8] Eren, U., Prach, A., Koçer, B. B., Raković, S. V., Kayacan, E., and Açıkmeşe, B. "Model predictive control in aerospace systems: Current state and opportunities." Journal of Guidance, Control, and Dynamics, Vol. 40, No. 7, pp. 1541–1566, 2017.

[9] Izzo, D., Blazquez, E., Ferede, R., Origer, S., Wagter, C. D., and de Croon, G. C. H. E. "Optimality Principles in Spacecraft Neural Guidance and Control.", 2023.

[10] Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. "Neural ordinary differential equations." Advances in neural information processing systems, Vol. 31, 2018.

[11] Pontryagin, L. Mathematical Theory of Optimal Processes. CRC Press, 1987.

[12] Jiang, F., Baoyin, H., and Li, J. "Practical techniques for low-thrust trajectory optimization with homotopic approach." Journal of guidance, control, and dynamics, Vol. 35, No. 1, pp. 245–258, 2012.

[13] Gill, P. E., Murray, W., and Saunders, M. A. "SNOPT: An SQP algorithm for large-scale constrained optimization." SIAM review, Vol. 47, No. 1, pp. 99–131, 2005.

[14] Bertrand, R. and Epenoy, R. "New smoothing techniques for solving bang–bang optimal control problems–numerical results and statistical interpretation." Optimal Control Applications and Methods, Vol. 23, No. 4, pp. 171–197, 2002. doi:doi.org/10.1002/oca.709.

[15] Biscani, F. and Izzo, D. "Revisiting high-order Taylor methods for astrodynamics and celestial mechanics." Monthly Notices of the Royal Astronomical Society, Vol. 504, No. 2, pp. 2614–2628, 2021.

[16] Kingma, D. P. and Ba, J. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980, 2014.

[17] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. "Neural Ordinary Differential Equations." S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, "Advances in Neural Information Processing Systems," Vol. 31. Curran Associates, Inc., 2018.

[18] Ross, S., Gordon, G., and Bagnell, D. "A reduction of imitation learning and structured prediction to no-regret online learning." "Proceedings of the fourteenth international conference on artificial intelligence and statistics," pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.