Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Physics-informed Deep Neural Operators for xGEO Angles-only Initial Orbit Determination

Komal Gupta
University of Arizona
Andrea D'Ambrosio
University of South Florida
Roberto Furfaro
robertof@arizona.edu

University of Arizona

Research Article

Keywords: Cislunar SSA, Orbit Determination, IOD, Neural Operators

Posted Date: November 23rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11253943/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Physics-informed Deep Neural Operators for xGEO Angles-only Initial Orbit Determination

Komal Gupta[1], Andrea D'Ambrosio[2], Roberto Furfaro[1*]

[1]Department of Systems and Industrial Engineering, The University of Arizona, 1127 E. James E. Rogers Way, Tucson, 85721, Arizona, USA.
[2]Department of Mechanical and Aerospace Engineering, University of South Florida, 4202 E. Fowler Avenue, Tampa, 33620, Florida, USA.

*Corresponding author(s). E-mail(s): robertof@arizona.edu;
Contributing authors: komalgupta@arizona.edu;
dambrosio@arizona.edu;

Abstract

Orbit determination in cislunar space is one of the biggest challenges for Space Situational Awareness beyond the geostationary orbit due to the large distances and chaotic dynamics involved. This paper presents a novel Initial Orbit Determination (IOD) algorithm based on deep neural operator, or DeepONet, that is capable of providing an accurate IOD solution from angle-only observations of cislunar objects. We leverage the ability of Physics-Informed DeepONets to learn an operator that represents the mapping from observations to states directly from the data. The dataset is simulated using a perturbed two-body dynamical framework and the trained model is compared to Gauss's IOD method in terms of its ability to provide an accurate initial reference trajectory for the Batch Least Squares algorithm. We find that the DeepONet provides comparable or better initialization for Batch Least Squares algorithm, especially in highly perturbed environments where Gauss's method fails to converge, such as in the region close to the Moon.

Keywords: Cislunar SSA, Orbit Determination, IOD, Neural Operators

1 Introduction

Orbit determination (OD) is one of the most important problems in astrodynamics. The knowledge of the orbit of a space object is an important first step for many problems within the field of Space Situational Awareness (SSA), such as tracking, attitude determination and catalog maintenance. Over the last few decades, the proliferation of resident space objects around the Earth has put pressure on SSA operations [1]. As a result, new methods are in demand which can scale with the increasing number of objects. Recently, there has been a growing interest within the SSA community in cislunar or xGEO space, which is the region extending from the GEO belt to the orbit of the Moon, the where the Earth's gravity is still the dominant force, but with significant perturbations due to the Moon, Sun and the outer planets. Several missions such as ARTEMIS [2] are planned in this region, highlighting the need for accurate OD methods in the complex dynamical environment of the xGEO region.

While the development of new systems such as radar and GPS over the past century have drastically improved the accuracy of orbit determination for objects in the vicinity of the Earth (such as those in low-earth or geostationary orbits), optical observations coupled with angle-only methods remain the most widely used techniques for determining the orbit of objects farther away from the Earth. These methods can broadly be classified into two categories: Initial Orbit Determination (IOD) methods such as Gauss' method, Gooding's method and Double-R iteration [3], which rely on a small number of observations and the geometry of the orbit to estimate an orbit; and statistical orbit determination methods [4], which use a larger set of measurements along with estimation algorithms, such as Batch Least Squares (BLS) or Kalman filters, to refine an initial estimate of the state. Typically, these methods are used in conjunction, with the IOD method providing a reference trajectory which is then refined iteratively by the statistical OD algorithms. However, the success of the OD process is limited in cislunar space, as the presence of perturbations leads to significant non-linearities and deviations from the Keplerian assumption made by most IOD methods, and the large distances involved reduce the quality of the measurements. For cooperative spacecrafts in cislunar space, several methods for OD are available or have been proposed. Batch processing of range and range-rate measurements obtained from NASA Deep Space Network have been used for missions such as ARTEMIS [2] and the James Webb Space Telescope [5]. Recently, several methods for orbit determination using space-based detectors have been proposed for the cislunar region, such as using inter-satellite communication links [6] and intercepted radio frequency signals [7]. However, these methods rely on the spacecraft actively communicating with the detector and cannot be used for non-cooperative objects, defunct satellites and space debris, in which case optical observations are the only way to determine the orbit.

In recent years, several new methods which tackle the non-linearity of the OD problem using Machine Learning have been proposed in the literature. Hartikainen et. al. [8] use Latent Force Models [9] to represent the non-linear dynamical model, which can be combined with Kalman Filtering approach to efficiently predict the orbit. Another approach that is slowly gaining popularity is based on Physics-informed Neural Networks (PINNs) [10], which incorporate the dynamical model directly into the training of the neural network in order to constrain the output to respect physical

laws. Badura et. al. [11] propose using transfer learning to provide a warm start for the training of PINNs for OD, in order to make the process both faster and more stable. Furthermore, Scorsoglio et. al. [12] use a PINN trained via Extreme Learning Machine (ELM) [13] to perform fast and accurate OD by randomly assigning fixed input weights to a single layer feedforward neural network, and only learning its output weights during training.

In this paper we present a novel IOD method for cislunar objects based on deep neural operator (or DeepONet). DeepONet was first introduced by Lu et.al. [14] as a neural network architecture capable of learning non-linear operators from data, and have been successfully used in various applications including learning the solution operator for PDEs [15], uncertainty quantification [16], and real-time simulations of complex physical systems [17]. Physics-informed DeepONet (or PI-DeepONet) [15] is a variant of the DeepONet that incorporates physics loss in order to enforce physical constraints on the operator learned by the model. Our approach is a variant of the PI-DeepONet architecture and consists of multiple branch networks and an observation loss in addition to the physics and regression losses. This allows our model to effectively learn an "operator" that maps from the space of observations to the space of states, making sure at the same time that the physics behind the data is satisfied. We use a restricted perturbed two-body dynamical model to generate simulated trajectories and corresponding angular observations in the form of topocentric right ascension and declination. We compare our method to two commonly used IOD methods - Gauss's IOD method and Double-r iteration - in terms of its ability to provide a good initial reference trajectory for the Batch Least Squares (BLS) algorithm. In summary, the main contributions of this work are as follows: 1) to the best of the authors' knowledge, neural operators have been exploited for the first time in the literature as a feasible method for IOD in the xGEO environment, by establishing the conceptual methodology and architecture; 2) the proposed neural operator-based methodology provides IOD results that are comparable or more accurate than both Gauss's IOD method and Double-r iteration; 3) BLS OD algorithm performance has been enhanced using DeepONets as an IOD method in perturbed environments when compared to Gauss and Double-r IOD methods; and 4) once trained, DeepONet IOD method is significantly computationally faster than both Gauss's and Double-r IOD method, as it does not require any iterations to provide the solution.

This manuscript is organized as follows. Section 2 provides background on the dynamical and observation models, and reviews the OD methods of Gauss, Double-r iteration, and Batch Least Squares. Section 3 explains the vanilla DeepONet architecture, as well as our proposed variant. Section 4 covers the data generation pipeline, followed by an explanation of the training process including the losses and metrics in section 5. Section 6 presents the results and discusses the key findings of this work, and finally section 7 provides the concluding remarks.

2 Background

2.1 Dynamical Model

In order to accurately represent the dynamics of cislunar space, we employ a perturbed two-body model written in a J2000 reference frame to generate our dataset of simulated trajectories. J2000 is an inertial reference frame centered on the Earth with the x-axis aligned with the vernal equinox, the z-axis along the Earth's rotation axis, and the y-axis to complete the right-handed frame. The system of ordinary differential equations (ODEs) that govern the motion of the satellite in this framework can be written in terms of its Cartesian position and velocity vectors as:

$$\begin{cases} \dot{\mathbf{r}} = \mathbf{v} \\ \dot{\mathbf{v}} = -\frac{\mu_e}{r^3}\mathbf{r} + \mathbf{a}_p \end{cases} \tag{1}$$

where $\mathbf{r}$ and $\mathbf{v}$ are the position and velocity vectors of the satellite in the aforementioned reference frame, $\mu_e = 398600.1$ km^3/s^2 is the Earth's gravitational parameter, and $\mathbf{a}_p$ represents the perturbing accelerations. In this work, we consider the third-body perturbation due to the Moon and the Sun. In this case, $\mathbf{a}_p$ can be written as

$$\mathbf{a}_p = \sum_i \mu_{3B_i} \left(\frac{\mathbf{r}_{sat-3B_i}}{r^3_{sat-3B_i}} - \frac{\mathbf{r}_{e-3B_i}}{r^3_{e-3B_i}} \right) \tag{2}$$

where $i \in \{Sun, Moon\}$, $\mathbf{r}_{A-B}$ represents the position vector from body A to body B and r_{A-B} represents its magnitude, the distance between bodies A and B.

2.2 Observation model

In addition to the dynamical model, we need an observation model that describes the relationship between the states and observations. In this work, we use the topocentric right ascension (RA) and declination (Dec) to describe the observables as these are typically used to provide optical telescope observations. We compute the topocentric right ascension (α) and declination (δ) for each of the simulated trajectories as

$$\begin{cases} \alpha = \arctan\left(\frac{y-y_o}{x-x_o}\right) \\ \delta = \arcsin\left(\frac{z-z_o}{\sqrt{(x-x_o)^2+(y-y_o)^2+(z-z_o)^2}}\right) \end{cases} \tag{3}$$

where $\mathbf{r}_o(t) = [x_o, y_o, z_o]$ represents the position vector of the observer in the J2000 frame.

2.3 Review of OD methods

The following section provides background on the OD algorithms used in this paper, namely Gauss's method, Double-r iteration, and Batch Least Squares. Gauss's method and Double-r iteration are angles-only IOD methods used as a benchmark to compare the performance of DeepONet as an IOD method. Subsequently, we compare the three methods based on their ability to initialize the Batch Least Squares algorithm.

2.3.1 Gauss's IOD method

Gauss's method of initial orbit determination is one of the oldest and most well-documented IOD methods in literature [18]. The method uses three observations of an object from a given observer and leverages the geometry of the object's path relative to the observer to determine the corresponding inertial position and velocity vectors. The object is assumed to be in a Keplerian orbit under purely two-body dynamics, such that the three position vectors all lie in the same plane. Given three angular measurements $(\alpha_1, \delta_1), (\alpha_2, \delta_2), \text{and } (\alpha_3, \delta_3)$ at times $t_1, t_2, \text{and } t_3$, the line-of-sight (LOS) vectors can be computed as

$$\hat{\boldsymbol{\rho}}_i = \cos\,\delta_i\,\cos\,\alpha_i\hat{\mathbf{I}} + \cos\,\delta_i\,\sin\,\alpha_i\hat{\mathbf{J}} + \sin\,\delta_i\hat{\mathbf{K}} \qquad \text{for } i = 1,2,3 \tag{4}$$

where $\hat{\mathbf{I}}$, $\hat{\mathbf{J}}$ and $\hat{\mathbf{K}}$ are the unit vectors along the x, y and z axes respectively in the J2000 frame. The unknown inertial position vectors of the object ($\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$) are then expressed in terms of the positions of the observer at the three time instants ($\mathbf{R}_1$, $\mathbf{R}_2$, and $\mathbf{R}_3$) and the LOS vectors ($\hat{\boldsymbol{\rho}}_1$, $\hat{\boldsymbol{\rho}}_2$ and $\hat{\boldsymbol{\rho}}_3$) as

$$\mathbf{r}_1 = \mathbf{R}_1 + \rho_1\hat{\boldsymbol{\rho}}_1 \tag{5}$$
$$\mathbf{r}_2 = \mathbf{R}_2 + \rho_2\hat{\boldsymbol{\rho}}_2 \tag{6}$$
$$\mathbf{r}_3 = \mathbf{R}_3 + \rho_3\hat{\boldsymbol{\rho}}_3 \tag{7}$$

where (ρ_1, ρ_2 and ρ_3) are the unknown slant ranges at the three observation times. Using Lagrange coefficients and the constraint on planarity of the three position vectors, the OD process is reduced to finding the state $[\mathbf{r}_2,\ \mathbf{v}_2]$ of the object at the intermediate time t_2. The magnitude of $\mathbf{r}_2$ is determined by solving an 8^{th}-order polynomial, following which the other unknowns can be determined via vector geometry. We skip the formal mathematical proofs for brevity as they have been covered extensively in literature [18]. Gauss's method achieves best performance when the observations are separated by 10° or less [3], while it exhibits singularities when the line-of-sight vectors lie in the plane on the orbit.

2.3.2 Double-r iteration

Double-r iteration is another commonly used IOD method that uses angle-only observations to determine an orbit [19]. The method starts by guessing the magnitudes of the object's position vectors vectors at two separate times (r_1 and r_2), and initializes the orbit by computing $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$. Interested readers are referred to Escobal [19] for formal mathematical passages. These values are then used to compute the time intervals between ($\mathbf{r}_3$, $\mathbf{r}_2$) and ($\mathbf{r}_1$, $\mathbf{r}_2$) as a function of r_1 and r_2 using Kepler's equation:

$$M_i = E_i - e\sin E_i \qquad i = 1,2,3 \tag{8}$$

where e is the eccentricity of the orbit, and E and M are the eccentric anomaly and mean anomaly, respectively. Finally, a differential correction scheme is used to match the computed time intervals ($\bar{\tau}_1$, $\bar{\tau}_3$) with the known time intervals between the

observations (τ_1, τ_3), by minimizing the following functions:

$$F_1 \equiv \tau_1 - \bar{\tau}_1$$
$$F_2 \equiv \tau_3 - \bar{\tau}_3$$

where,

$$\tau_1 = t_1 - t_2 \qquad \tau_3 = t_3 - t_2$$
$$\bar{\tau}_1 = \left(\frac{M_1 - M_2}{n}\right) \qquad \bar{\tau}_3 = \left(\frac{M_3 - M_2}{n}\right)$$

where M_i is the mean anomaly at time t_i. Double-r iteration is more efficient than Gauss's method for observations spread over larger arcs; however, the initial guess of the position magnitudes r_1 and r_2 has to be accurate enough for it to converge [3].

2.3.3 Batch Least Squares

Batch Least Squares (BLS) algorithm is one of the most popular methods for orbit determination. The algorithm starts with a reference trajectory, and estimates the state deviation vector using a given set of observations. The key step in this formulation is linearization: since the dynamical model (Eq. 1) and the observation model (Eq. 3) are both non-linear, they must be linearized around the reference trajectory.

Given a set of non-linear dynamical equations and observation model

$$\dot{\mathbf{X}} = F(\mathbf{X}(t), t) \qquad \mathbf{X}(t_k) \equiv \mathbf{X}_k \tag{9}$$
$$\mathbf{Y}_i = G(\mathbf{X}_i, t_i) + \nu_i \qquad i = 1, \ldots, p \tag{10}$$

where $\mathbf{X}_k$ represents the n-dimensional state vector at time t_k, $\mathbf{Y}_i$ is the m-dimensional set of observations, p is the number of observations, and k and i are indices over states and observations respectively. $F(\cdot)$ and $G(\cdot)$ are non-linear functions representing the dynamical model (Eq. 1) and observation model (Eq. 3) respectively, both of which can be linearized about the reference trajectory and written in the form of matrices as follows

$$A(t) = \left[\frac{\partial F(t)}{\partial \mathbf{X}(t)}\right]^* \qquad \tilde{H}_i = \left[\frac{\partial G}{\partial \mathbf{X}}\right]^*_i \tag{11}$$

where the superscript (*) denotes that the expression is evaluated on the reference trajectory. Using $\bar{\mathbf{x}}_0$ and $\bar{P}_0$ as the *a priori* estimate of initial state deviation and covariance respectively, the least squares solution for the initial state deviation estimate can be written as

$$\hat{\mathbf{x}}_0 = (H^T R^{-1} H + \bar{P}_0^{-1})^{-1} (H^T R^{-1} \mathbf{y} + \bar{P}_0^{-1} \bar{\mathbf{x}}_0) \tag{12}$$

where

$$H = \begin{bmatrix} \tilde{H}_1 \Phi(t_1, t_0) \\ \vdots \\ \tilde{H}_p \Phi(t_p, t_0) \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} \mathbf{Y}_1 - G(\mathbf{X}_1^*, t_1) \\ \vdots \\ \mathbf{Y}_p - G(\mathbf{X}_p^*, t_p) \end{bmatrix} \tag{13}$$

and R is the observation noise covariance matrix. The algorithm is iterated until the norm of the state deviation estimate $||\hat{\boldsymbol{x}}_0||$ reaches below a pre-specified tolerance level ϵ, at which point the method is said to have converged [4].

3 DeepONet

3.1 Vanilla DeepONet

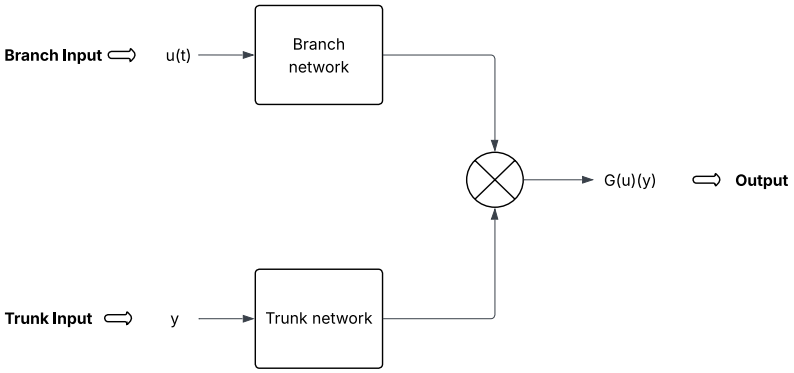


Fig. 1: Vanilla DeepONet Architecture

DeepONets are based on the universal approximation theorem for operators, which states that a neural network can be used to approximate any non-linear operator i.e., a mapping from a set of functions to another set of functions [20]. This is an extension of the universal approximation theorem which states that a neural network with arbitrary width and depth can approximate any continuous function to an arbitrary degree of accuracy [21, 22], and has been used to justify the use of neural networks to model complex relationships found in various real world tasks. The DeepONet architecture consists of two sub-networks: a network called *branch* which takes as input a set of functions evaluated at fixed points called "sensor locations", and another network called *trunk* which accepts a set of points to evaluate the output functions. The outputs of the branch and trunk networks are then combined to get the final output - a set of output functions evaluated at locations specified by the trunk input. The strength of this architecture lies in the fact that it incorporates a strong inductive bias towards learning operators: the branch network learns the operator from a set of input functions, while the trunk network learns to evaluate the operator at the given "collocation points", thereby learning a function to function mapping, i.e., an

operator. The only constraint required by the original framework is for all the input functions to be evaluated at the same sensor locations, while no constraint is imposed on the evaluation points (input to the trunk network). Figure 1 shows a schematic of the vanilla DeepONet architecture.

3.2 Proposed architecture

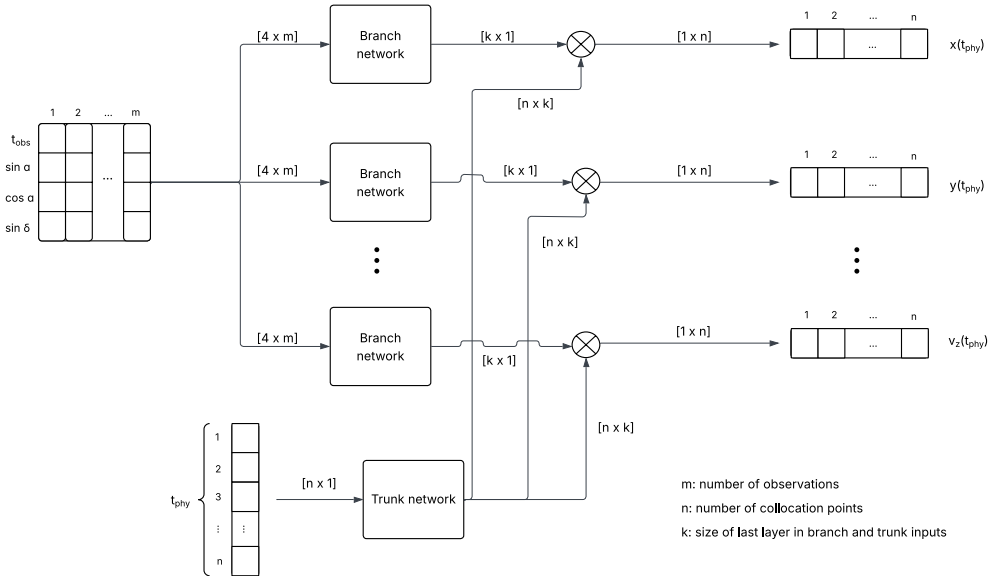
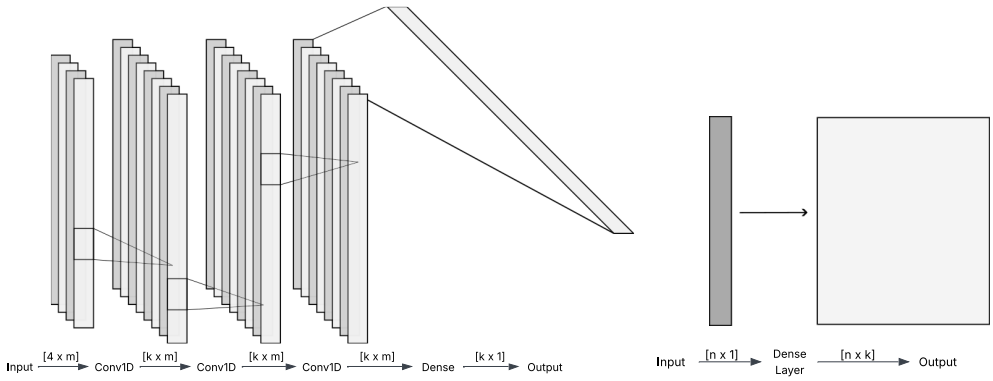


Fig. 2: Architecture of the proposed DeepONet

DeepONet is a high-level architecture that uses two separate networks to learn a function to function mapping. The proposed architecture is a variant of the DeepONet especially designed to learn a map from a set of observations to a set of states in order to perform initial orbit determination. The input to the branch network is the timeseries of observations, which is a vector of length m with four features - $[t_{obs},\ sin(\alpha),\ cos(\alpha),\ sin(\delta)]$ - where m is the number of sensor locations and is equal to the number of observations. Here we use the sine and cosine of the angles α and δ rather than the angles themselves to avoid singularities as proposed by Scorsoglio et.al. [12]. The input to the trunk is a vector (of length n) of time instants t_{phy} where we want to compute the states. The output is the 6-dimensional state vector $[x(t),\ y(t),\ z(t),\ v_x(t),\ v_y(t),\ v_z(t)]$ computed at each time instant t_{phy}. Since in our case we are learning six components of the state, we use six different branch networks each with the same input and architecture. Note that even though each branch network learns a separate set of weights, the six state components are still coupled as the entire architecture is trained together with a physics-informed loss function. Each branch network is composed of three 1-D convolutional layers, each with 256 output



(a) (b)

Fig. 3: Architecture of the (a) Branch and (b) Trunk networks

channels, ReLU activation functions and 0.1 dropout layers in between. These are followed by a linear layer that reduces the temporal dimension from m to 1. The trunk network is simply a linear layer that converts every timestep into a vector of 256 features. Finally, each output is obtained by multiplying the outputs of the branch and trunk networks as given by the following tensor contraction operation:

$$o_n = \sum_k b_k t_{nk} \tag{14}$$

where b_k and t_{nk} are components of the outputs of branch and trunk networks respectively.

Table 1: Parameters of the proposed architecture

Parameter	**Value**
Number of sensor locations (m)	35
Number of layers in each branch network	3 Convolutional + 1 Dense
Number of output channels in each branch layer (k)	256
Dropout (only after 1D convolution layers)	0.1
Activation function	ReLU
Number of evaluation points (n)	35
Number of layers in trunk network	1 Dense
Number of output features of trunk layer (k)	256

The proposed architecture is shown in Figure 2 along with the shapes of the input and output tensors for every operation, also summarized in Table 1. Note that since the model is trained using mini-batch gradient descent algorithm, all tensors have an additional batch dimension as the first dimension, which is not shown in the diagram for clarity. The architectures of the branch and trunk networks are shown in Figure 3.

4 Dataset Preparation

The dataset for this work comprises of 10000 open trajectories simulated in the xGEO regime. The initial state for each sample is chosen by randomly and uniformly sampling a set of osculating Keplerian orbital elements (a, e, i, Ω, ω and ν) and converting them into position and velocity vectors in J2000 frame to get the initial state, which are then propagated using the perturbed two-body dynamical model as described in Eq. 1. Each initial state is propagated for 12 hours starting from a fixed epoch time (2025-05-05 03:05:09 UTC), and the state vector is recorded every 15 minutes. Finally, for every state vector we compute the corresponding observables (α and δ) according to the observation model described by Eq. 3. In order to simulate real observations, we add Gaussian noise with zero mean and standard deviation of 0.5 arcsecond to the simulated measurements. The observer is assumed to be located at the Biosphere 2 observatory near Tucson, Arizona (USA) at (32.5764° N, 249.1517° E). Out of the total propagation time of 12 hours, only the first 8.5 hours correspond to the Sun being more than 10° below the horizon at the observer's location, therefore, considering the visibility of the satellite from the observer, only the states and observations at these times are used to build the dataset. Note that for the purpose of this study, we consider a relatively simple case of a single epoch time and a single observer with observations recorded at equally spaced times, compared to real-world OD problems which often involve more than one epoch times, variable track lengths and sampling times. However, we use this simple case to highlight the feasibility of using DeepONets to perform IOD by learning a mapping from observations to states, and so we defer the generality analysis to future works.

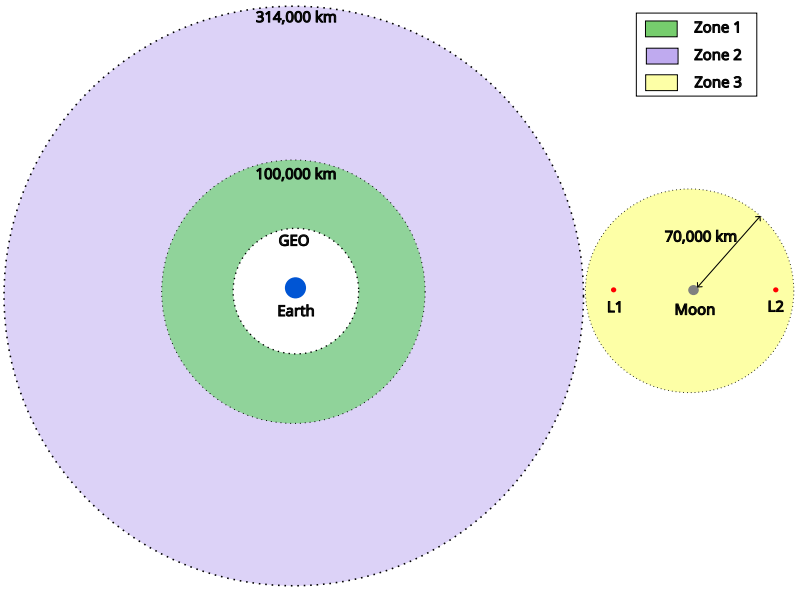


Fig. 4: xGEO zones

Through trial and error we found that a single DeepONet model was not able to capture the complex and varying dynamics of the entire xGEO region. We therefore divide the region into three zones and create three different datasets in order to train a separate model for each zone. The first zone extends from the GEO belt distance (42,164 km) to 100,000 km from the Earth's center, and contains the objects in GEO and the graveyard orbits beyond. The objects in this zone mostly follow two-body dynamics, and perturbations due to the Sun and the Moon may be negligible over shorter timescales (as considered in this paper). The second zone (also centered on the Earth) extends from 100,000 km to 314,000 km from the Earth's center, and contains most transfer orbits to the Moon. In this region, the perturbations due to the Sun and Moon become more significant than Zone 1 as we get farther from the Earth, though the perturbations are still not noticeable over the timescales considered in this paper. The third zone is centered on the Moon and extends from 2,000 km to 70,000 km from the Moon's center, and contains orbits around the Moon along with the L_1 and L_2 Lagrange points. This region is both the farthest from the Earth as well as the most perturbed, and therefore is the most challenging case of the three. The three zones are shown in Figure 4.

Table 2: Common parameters used to generate the data for each zone

Parameter	**Value**
Epoch time	2025-05-05 03:05:09 UTC
Observer location	32.5764° N, 249.1517° E
Time of flight	8.5 hours
Δt	15 minutes
Earth's Gravitational Parameter (μ_{earth})	398600.4418 km^3/s^2
Earth's equatorial radius (R_e)	6378.1 km
Moon's Gravitational Parameter (μ_{moon})	4902.8006 km^3/s^2
Moon's equatorial radius (R_m)	1737.4 km
Sun's Gravitational Parameter (μ_{sun})	1.3271 $\times 10^{11}$ km^3/s^2

Table 3: Zone-specific parameters used to generate the data for each zone

Parameter	**Zone 1**[1]	**Zone 2**[1]	**Zone 3**[2]
Semi-major axis (a)	42,614 km to 100,000 km	100,000 km to 314,000 km	2,000 km to 70,000 km
Eccentricity (e)	0 to 0.75	0 to 0.75	0 to 0.75
Inclination (i)	0° to 30°	0° to 30°	0° to 90°
RAAN (Ω)	0° to 360°	0° to 360°	0° to 360°
Argument of Perigee (ω)	0° to 360°	0° to 360°	0° to 360°
True anomaly (ν)	0° to 360°	0° to 360°	0° to 360°

[1]Orbital elements are relative to Earth.

[2]Orbital elements are relative to Moon.

In addition to the distance, we also constrain the inclination and eccentricity of the simulated trajectories in each zone to values that are more likely to be seen in practice. These values along with other constants used to generate the dataset are shown in Tables 2 and 3.

5 Model Training and Evaluation

This section details the training and evaluation of the models. Each dataset is normalized prior to training, a standard technique which helps neural networks converge faster and achieve better generalization. The models are then trained in a supervised learning framework, using mini-batch gradient descent algorithm with the Adam optimizer and a compound loss function which incorporates regression, physics and observation losses. Following training, the models' performance over the test dataset is evaluated using two metrics designed for this problem, as described below.

5.1 Normalization

In general, neural networks achieve faster convergence when the inputs and outputs are in similar (and smaller) ranges, as it lets the model focus on the patterns in the data rather than its scale. For this work, we normalize both the time and output states which are on very different scales otherwise. The time vector is always normalized between $[0, 1]$, such that $t = 0$ corresponds to epoch time and $t = 1$ corresponds to epoch time + 8.5 hours. The output states are normalized using the minimum and maximum values of the position and velocity components, i.e.,

$$min_p = -\max\{|x|, |y|, |z|\} \qquad max_p = \max\{|x|, |y|, |z|\} \tag{15}$$

$$min_v = -\max\{|v_x|, |v_y|, |v_z|\} \qquad max_v = \max\{|v_x|, |v_y|, |v_z|\} \tag{16}$$

5.2 Loss functions

The loss function used to train the model is a weighted sum of three separate loss functions: 1) Regression Loss, 2) Physics Loss, and 3) Observation loss, and can be written as

$$L_{tot} = \lambda_1 L_{reg} + \lambda_2 L_{phy} + \lambda_3 L_{obs} \tag{17}$$

where L_{reg}, L_{phy}, and L_{obs} are the three components of the total loss function, and λ_1, λ_2 and λ_3 are their relative weights.

5.2.1 Regression loss

The regression loss is simply the mean-squared error (MSE) between the true and predicted outputs of the model. The loss can be written as

$$L_{reg} = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{x}_i - \tilde{\mathbf{x}}_i||^2 \tag{18}$$

where $\mathbf{x}_i = [x_i \ y_i \ z_i \ v_{x_i} \ v_{y_i} \ v_{z_i}]$ is the true state at time t_i, whereas $\tilde{\mathbf{x}}_i = [\tilde{x}_i \ \tilde{y}_i \ \tilde{z}_i \ \tilde{v}_{x_i} \ \tilde{v}_{y_i} \ \tilde{v}_{z_i}]$ is the DeepONet predicted state.

5.2.2 Physics loss

The physics loss is the sum of residuals of each ODE in the coupled system, computed as

$$L_{phy} = L_x + L_y + L_z + L_{v_x} + L_{v_y} + L_{v_z} \tag{19}$$

where

$$L_x = \frac{1}{N}\sum_{i=1}^{N} L_\beta\left(\frac{d\tilde{x}_i}{dt} - \tilde{v}_{x_i}\right)$$

$$L_y = \frac{1}{N}\sum_{i=1}^{N} L_\beta\left(\frac{d\tilde{y}_i}{dt} - \tilde{v}_{y_i}\right)$$

$$L_z = \frac{1}{N}\sum_{i=1}^{N} L_\beta\left(\frac{d\tilde{z}_i}{dt} - \tilde{v}_{z_i}\right)$$

$$L_{v_x} = \frac{1}{N}\sum_{i=1}^{N} L_\beta\left(\frac{d^2\tilde{x}_i}{dt^2} + \frac{\mu_e\tilde{x}_i}{\tilde{r}_i^3} - a_{p\tilde{x}_i}\right)$$

$$L_{v_y} = \frac{1}{N}\sum_{i=1}^{N} L_\beta\left(\frac{d^2\tilde{y}_i}{dt^2} + \frac{\mu_e\tilde{y}_i}{\tilde{r}_i^3} - a_{p\tilde{y}_i}\right)$$

$$L_{v_z} = \frac{1}{N}\sum_{i=1}^{N} L_\beta\left(\frac{d^2\tilde{z}_i}{dt^2} + \frac{\mu_e\tilde{z}_i}{\tilde{r}_i^3} - a_{p\tilde{z}_i}\right)$$

where $\tilde{r}_i = \sqrt{\tilde{x}_i^2 + \tilde{y}_i^2 + \tilde{z}_i^2}$, and a_{px_i}, a_{py_i}, and a_{pz_i} are x, y, and z components respectively of the perturbing acceleration vector in the inertial frame at time t_i. Here $L_\beta(a)$ refers to Huber Loss, which is defined by the piecewise function

$$L_\beta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| < \beta \\ \beta \cdot (|a| - \frac{\beta}{2}) & \text{otherwise} \end{cases} \tag{20}$$

We choose Huber loss (with $\beta = 1.0$) in place of simple mean squared error to compute the physics loss since it is less sensitive to outliers, and leads to faster convergence in the early epochs of training. The derivatives are computed with respect to the trunk input using Autograd, Pytorch's automatic differentiation engine. Note that the derivatives computed via automatic differentiation are the derivatives of scaled outputs with respect to scaled time, and must be unscaled before computing the physics loss. The unscaled derivatives (denoted by prime (')) can be written using chain rule as follows:

$$\frac{dx'}{dt'} = \frac{dx}{dt} \cdot \frac{dt}{dt'} \cdot \frac{dx'}{dx} \tag{21}$$

$$\frac{d^2x'}{dt'^2} = \frac{d^2x}{dt^2} \cdot \left(\frac{dt}{dt'}\right)^2 \cdot \left(\frac{dx'}{dx}\right)^2 \tag{22}$$

where $\frac{dx}{dt}$ and $\frac{d^2x}{dt^2}$ are computed via Autograd. In addition, the derivative of the scaled variables with respect to unscaled variables can be computed as

$$x = \frac{2(x' - min_p)}{max_p - min_p} - 1 \qquad \frac{dx}{dx'} = \frac{2}{max_p - min_p}$$
$$t = \frac{t' - t_{min}}{t_{max} - t_{min}} \qquad \frac{dt}{dt'} = \frac{1}{t_{max} - t_{min}}$$

5.2.3 Observation loss

The observation loss is the sum of residuals of the observables, i.e., $[sin(\alpha),\ cos(\alpha),\ sin(\delta)]$, computed as

$$L_{obs} = L_{sra} + L_{cra} + L_{sdec} \tag{23}$$

where

$$L_{sra} = \frac{1}{M}\sum_{i=1}^{M} L_\beta(\sin(\alpha_i) - \sin(\tilde{\alpha}_i))$$
$$L_{cra} = \frac{1}{M}\sum_{i=1}^{M} L_\beta(\cos(\alpha_i) - \cos(\tilde{\alpha}_i))$$
$$L_{sdec} = \frac{1}{M}\sum_{i=1}^{M} L_\beta(\sin(\delta_i) - \sin(\tilde{\delta}_i))$$

Here, $\beta = 1.0$, (α, δ) are the input observables, $(\tilde{\alpha}, \tilde{\delta})$ are the observables computed using the output of the model, and M is the number of observations.

5.3 Metrics

This subsection describes the metrics used to evaluate the DeepONet's prediction over the test dataset. For a set of input observations (α_i, δ_i), the predicted observables (α_i^*, δ_i^*) can be computed using the predicted trajectory and the observation model of Eq. 3, where i is the index corresponding to the time of the observation. The index where the norm of right ascension and declination residuals is minimum is given as

$$\kappa = \arg\min_i \sqrt{(\alpha_i - \alpha_i^*)^2 + (\delta_i - \delta_i^*)^2} \tag{24}$$

The metrics are then defined as follows:

1. **The minimum norm of observation residual:** measures the error between the true and predicted line-of-sight direction to the object. It is defined as the minimum

of the norm of angle residuals computed at all observation points, i.e.,

$$\gamma = \sqrt{(\alpha_\kappa - \alpha_\kappa^*)^2 + (\delta_\kappa - \delta_\kappa^*)^2} \tag{25}$$

2. **Percent error at the point with minimum observation residual norm:** measures the error between true and predicted state in the direction of the line-of-sight as a fraction of the true distance from the Earth. It is defined using the Euclidean distance (d) between the true and predicted trajectories at the point where the observation residual is minimum, i.e.,

$$d = \sqrt{(x_\kappa - x_\kappa^*)^2 + (y_\kappa - y_\kappa^*)^2 + (z_\kappa - z_\kappa^*)^2} \tag{26}$$

$$\% \text{ position error} = \frac{d}{\sqrt{x_\kappa^2 + y_\kappa^2 + z_\kappa^2}} \cdot 100 \tag{27}$$

where $(x_\kappa, y_\kappa, z_\kappa)$ is the position on the true trajectory corresponding to the observation $(\alpha_\kappa, \delta_\kappa)$, while $(x_\kappa^*, y_\kappa^*, z_\kappa^*)$ is the position on the DeepONet's predicted trajectory corresponding to the observation $(\alpha_\kappa^*, \delta_\kappa^*)$. Figure 5 shows a visualization of the metrics described above.

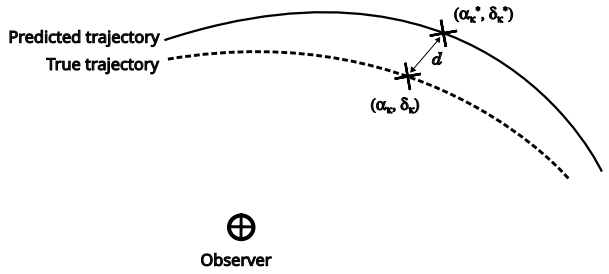


Fig. 5: Visualization of metrics used to evaluate the performance of the DeepONet

6 Results and Discussion

In this section, we present the results of our proposed model on the simulated dataset for each of the three zones. Most importantly, we compare the performance of our proposed model to Gauss's method and Double-r iteration for IOD, and compare the results of Batch Least Squares with all three initializations. We report the performance of each IOD method in terms of percentage of samples for which the Batch Least Squares OD algorithm converges to a "good" solution. Here, a "good" solution is when the distance between true and estimated trajectories at initial time is less than a threshold value.

$$||\mathbf{r}_0 - \hat{\mathbf{r}}_0|| < \Delta \tag{28}$$

As the distances involved are extremely large and vary greatly between the three zones, we present our results in canonical units in order to simplify the analysis and put the

Table 4: Hyperparameters

Parameter	Value
Size of dataset	10000
Epochs	5000
Optimizer	Adam
Learning rate	0.001
Batch size	64
Regression loss weight (λ_1)	1
Physics Loss weight (λ_2)	1
Observation Loss weight (λ_3)	1

values in perspective. For reference, 1 canonical distance unit or DU is equal to 6378.1 km, i.e., the radius of the Earth. The threshold for Zone 1 and 2 was set at 0.1 DU, or about 638 km, whereas the threshold for Zone 3 was set at 0.25 DU, or about 1595 km. Note that since Zone 3 is both the farthest zone (and therefore has the smallest observed arcs) as well as the most perturbed one due to its proximity to the Moon, we choose a higher threshold for it in order to keep the accuracy at a similar level with the other zones. The hyperparameters common to all tests are listed in Table 4, and were chosen heuristically. A model was trained on each dataset for 5000 epochs, and the loss on the validation dataset was monitored in order to find and save the best weights of the model for each case. The training time for each model was about 3.25 hours on a desktop machine equipped with Intel® Core™ i9-14900K processor, 64 GB RAM, and a single NVIDIA GeForce RTX 4070 GPU.

Table 5: Average inference times per sample and per dataset for each zone (excluding samples which do not converge)

Inference time over test dataset	Method	Zone 1	Zone 2	Zone 3
Per sample	Gauss's method	140 ms	211 ms	558 ms
	Double-r iteration	581 ms	816 ms	595 ms
	DeepONet	**0.14 ms**	**0.14 ms**	**0.14 ms**
Entire dataset	Gauss's method	203 s	310 s	710 s
	Double-r iteration	760 s	1160 s	600 s
	DeepONet	**0.210 s**	**0.210 s**	**0.210 s**

Table 5 shows the comparison between the inference time for the DeepONet IOD method and the computational time for the classical IOD methods, computed per sample and over the entire dataset for every sample in the test set (except the ones for which the classical IOD methods do not converge). For the DeepONet, once the training is complete, the inference is extremely fast, taking an average of about 0.14 ms per sample in each test dataset and 210 ms for the entire test dataset for each zone. In comparison, both Gauss's method and Double-r iteration requires significantly longer computational times, as can be seen from Table 5. Note that finding the compute time for classical methods is much more complicated as they requires several iterations to

converge (depending on the trajectory itself), so we report the average times for the samples in our test datasets. This highlights one of the main advantages of DeepONet over Gauss and Double-r methods for IOD, that is its ability to provide a reasonable IOD solution always in a single function call and in a fraction of the time taken by either of the classical methods.

6.1 Zone 1

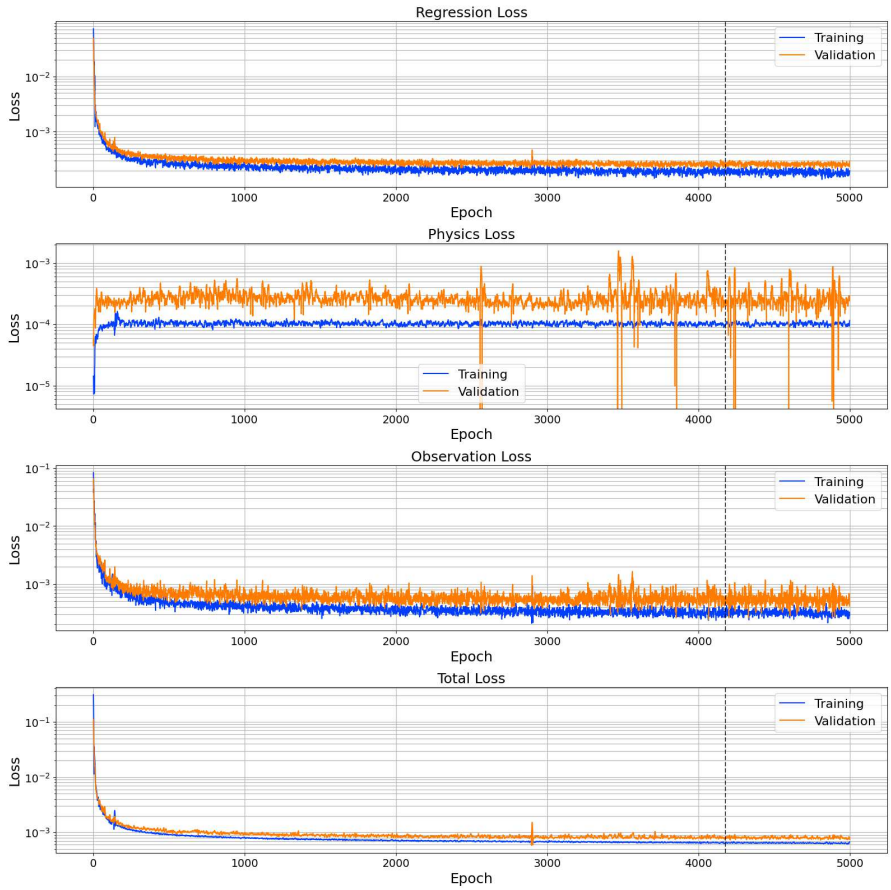
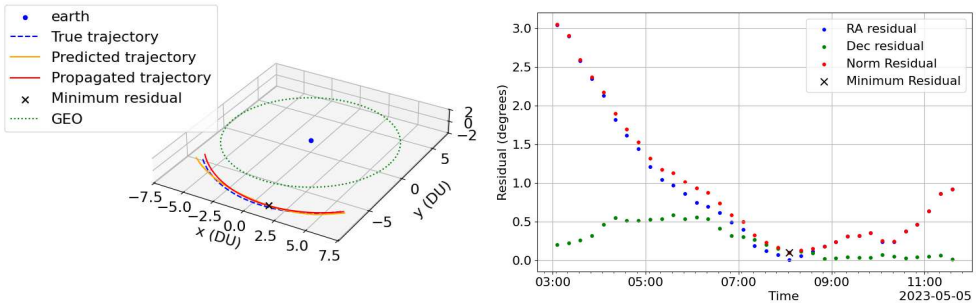


Fig. 6: Loss functions on training and validation subsets of Zone 1 dataset

Zone 1 is the nearest zone to the Earth, where the dominant effect is the Earth's gravity resulting in mostly Keplerian orbits. For this zone, 8.5 hours of observations imply that roughly a third of the trajectory is observed, which is long enough for the model to learn and predict a reasonably good IOD solution. Figure 6 shows the progression of the regression, physics, observation and total loss functions respectively over 5000 training epochs for both training and validation subsets. From the figure

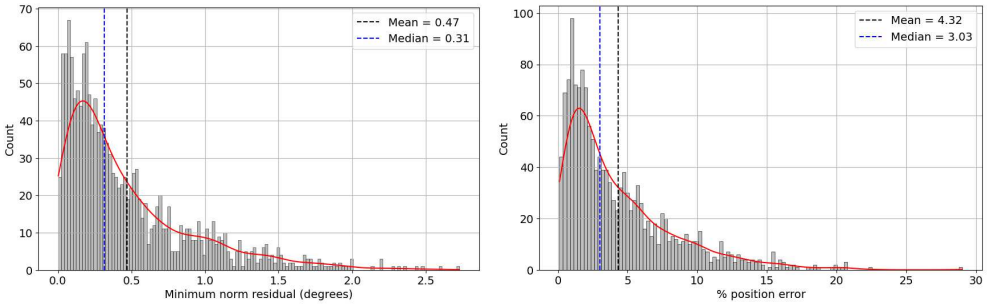
it can be seen that all losses except physics loss decrease significantly over the first 1000 epochs on both subsets, following which the decline slows down. Curiously, the physics loss increases in the early epochs before stagnating for the rest of the training process. Our analysis of the model's predictions after the first few epochs shows that this occurs because, while these predictions represent a physically feasible trajectory (hence the low physics loss), they do not match the observations and are far from the true trajectory in space (hence the high regression and observation losses). As the training progresses and the model learns to correctly map the observations to states, the regression and observation losses decrease and the physics loss settles on different but still low value. The dashed line in Figure 6 marks the point in training where the total validation loss reaches its minimum value of 6.92×10^{-4} (epoch 4180) and the model weights at this epoch are saved as the best model.



(a) True and predicted trajectory

(b) Observation residuals

Fig. 7: Sample trajectory and DeepONet's prediction from Zone 1 dataset



(a) Minimum observation residuals

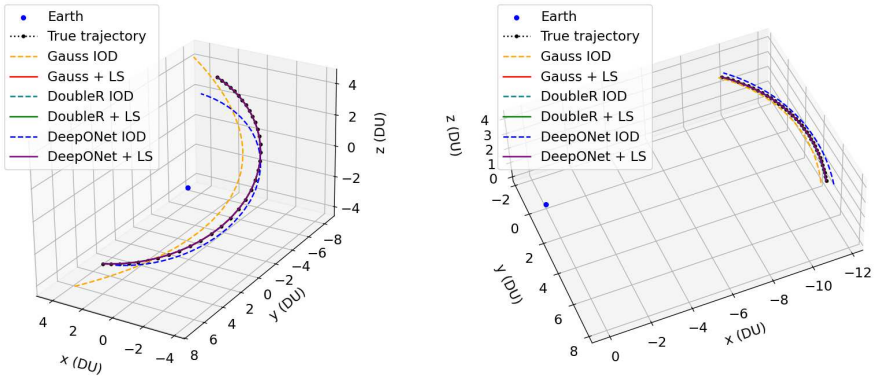
(b) % Position error at point of minimum observation residual

Fig. 8: Distribution of metrics for Zone 1 dataset

Table 6: Comparison of DeepONet with classical IOD methods when used in conjunction with Batch Least Squares algorithm

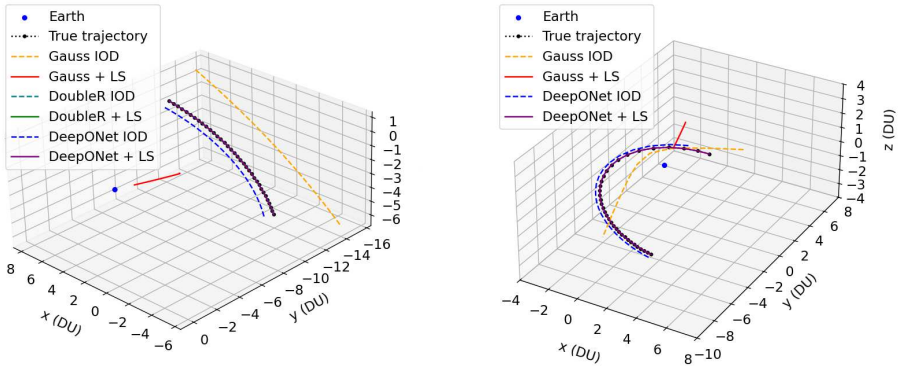
Method	Zone 1	Zone 2	Zone 3
Gauss + BLS	80.1 %	78.4 %	48.1 %
Double-r + BLS	86.1 %	80.5 %	21.1 %
DeepONet + BLS	**92.4 %**	**86.6 %**	**61.5 %**

The trained model is evaluated on the test subset of the dataset which comprises 1500 trajectories. A sample trajectory and the DeepONet's prediction, along with the corresponding observation residuals is shown in Figure 7. The predicted state corresponding to the point of minimum observation residual is propagated backward in time to generate a smooth reference trajectory which is used to initialize Batch Least Squares. The distribution of the aforementioned metrics over the test dataset are shown in Figure 8. From the histogram on the left, it can be seen that the norm residuals are distributed with a mean of 0.47 degrees, and is under 1 degrees for most of the samples, which indicates that the model learns to accurately predict the line-of-sight direction to the object. Furthermore, the histogram on the right shows the error in position at the point with minimum observation residual, which is important as a large error here leads to a reference trajectory that is far from the true trajectory. For a majority of samples, this is less than 10% of the object's distance from the observer, implying that the reference trajectory is close to the real trajectory in space for most cases in the test dataset.



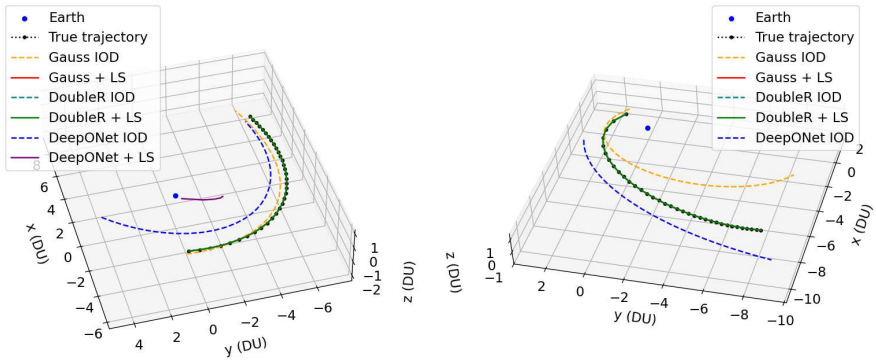
(a) (b)

Fig. 9: Zone 1 examples where all IOD methods provide a good initialization



(a) (b)

Fig. 10: Zone 1 examples where one or all of classical IOD methods do not provide a good initialization



(a) (b)

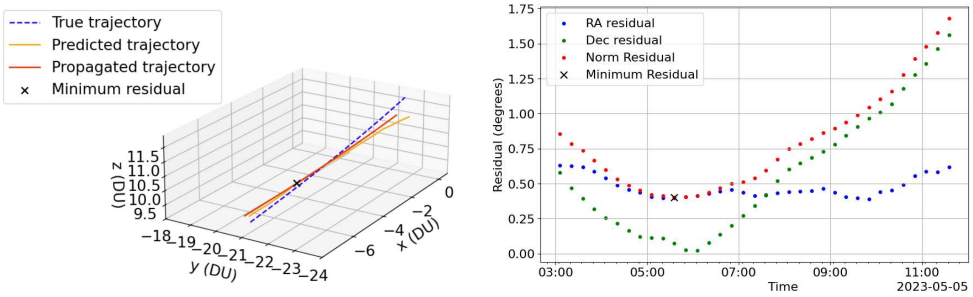
Fig. 11: Zone 1 examples where DeepONet IOD does not provide a good initialization

When used as an IOD for Batch Least Squares algorithm, the DeepONet outperforms both classical IOD methods over the test dataset. Using Gauss and Double-r methods to initialize BLS algorithm provides a good solution for 80.01% and 86.06% of the test dataset respectively, whereas when using DeepONet the number rises to

92.4%. These results are summarized in Table 6. Figures 9, 10 and 11 show some of the samples from the test dataset along with the solution predicted by the three IOD methods, as well as the solution of BLS with all three initializations (except when the filter diverges as in the last example). Also shown is the final OD solution using the BLS algorithm initialized with reference trajectory obtained from the three IOD methods. Since the trajectories in this region mostly follow Keplerian dynamics, all three IOD methods provide good initial reference trajectories for most samples as shown in Figure 9. However, Gauss's IOD method does not provide a good initialization for BLS in cases where the line-of-sight vectors are too far from each other, or when they are not on the same plane due to the noise added to the data, and in these cases the BLS filter converges to a solution far from the true trajectory, as seen in Figure 10. Meanwhile, Double-r iteration can handle observations that are far apart (as for the sample on the left in Figure 10, Double-r IOD and Double-r + LS overlap with true trajectory), but still fails in some cases due to the noise in the data (as for the sample on the right in Figure 10, Double-r iteration does not converge). The DeepONet model does not have these limitations, thus outperforming the classical IOD methods in providing a better initialization for the BLS algorithm. However, the DeepONet model can sometimes fail to predict a good IOD solution in which case the Batch LS algorithm either converges onto a trajectory far from the true trajectory or diverges entirely, as seen in Figure 11 (left and right samples respectively).

6.2 Zone 2

Zone 2 is similar to Zone 1 in that the trajectories are still mostly Keplerian, and the perturbations due to the Sun and the Moon are not significant enough in comparison to the Earth's gravity. However, the main difference from Zone 1 is the objects' distances from the Earth, which are up to 3 times larger than in the previous zone, resulting in much smaller observed arcs in the same time window of 8.5 hours. The loss functions for this zone are shown in Figure 13. Similar to the previous case, the losses decrease rapidly for the first 1000 epochs and then their descent slows down. The best model with minimum total validation loss of 5.68×10^{-4} is achieved at epoch 4116 (represented by the dashed line).



(a) True and predicted trajectory

(b) Observation residuals

Fig. 12: Sample trajectory and DeepONet's prediction from Zone 2 dataset

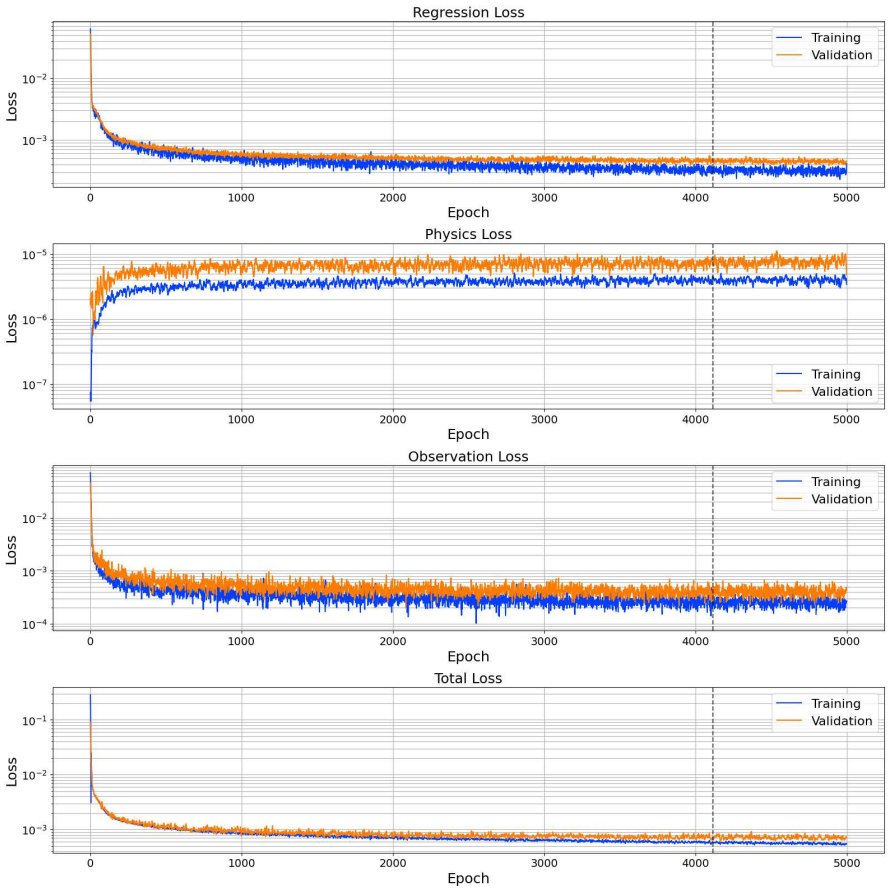
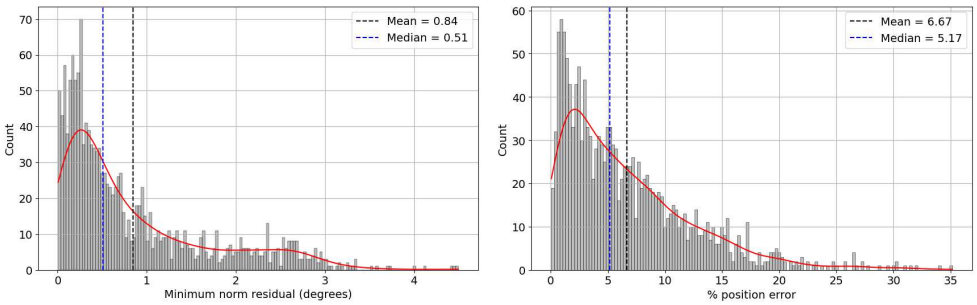


Fig. 13: Loss functions on training and validation subsets of Zone 2 dataset

A sample trajectory from the test dataset is shown in Figure 12 along with the DeepONet's prediction. The corresponding observation residuals show the minimum residual point that was used to generate the reference trajectory. The metrics are computed on the test dataset and are distributed as shown in Figure 14. Once again the mean of the norm residuals is lower than 1 degree, even though the distribution has a longer tail. At the same time, the distribution of position error at the point with minimum observation residual also shows a slightly higher mean and is wider than the previous case. This can be attributed to the fact that the objects are much further away and the model sees much less of the orbit over the chosen time frame of 8.5 hours, leading to the prediction being less accurate.

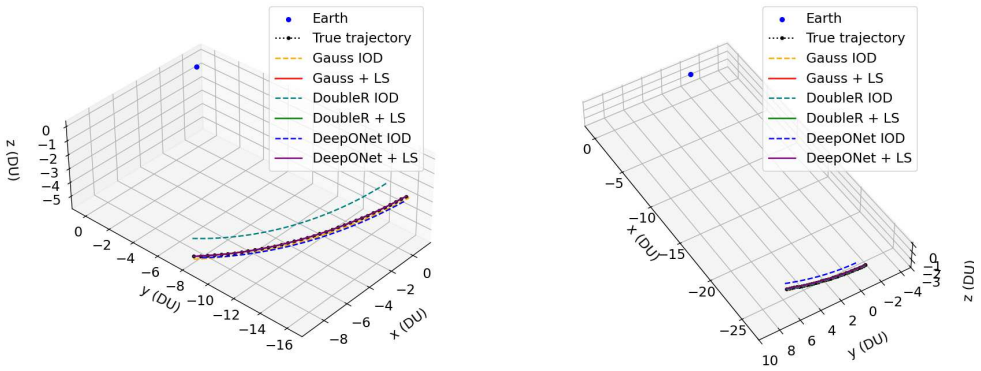


(a) Minimum observation residuals

(b) % Position error at point of minimum observation residual

Fig. 14: Distribution of metrics for Zone 2 dataset

However, when used as an IOD in conjunction with BLS, the DeepONet still outperforms both classical IOD method over the test dataset. The DeepONet + BLS provides a good solution for 86.6% of the samples, while Gauss + BLS and Double-r + BLS converge to a good solution for 78.4% and 80.5% of the samples respectively. Figures 15, 16 and 17 show some of the samples from the test dataset along with

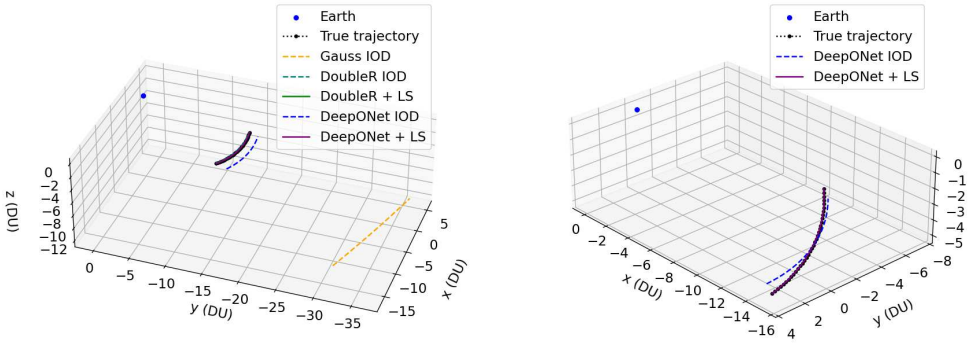


(a)

(b)

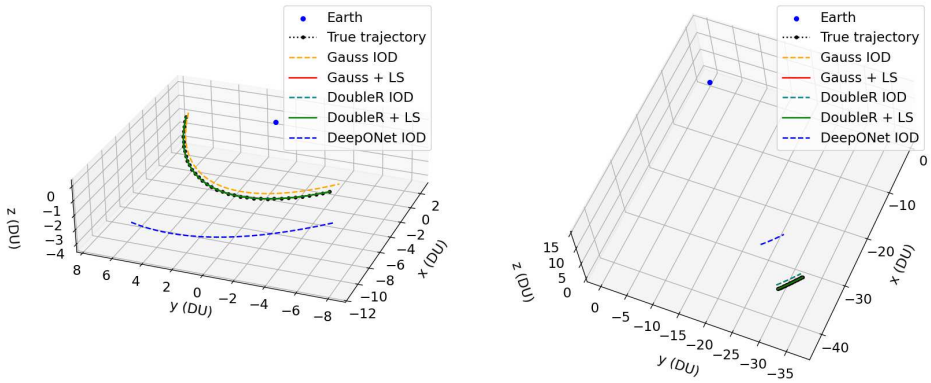
Fig. 15: Zone 2 examples where all IOD methods provide a good initialization

the IOD solution provided by the three methods, as well as the OD solution when initialized with the three methods. Figure 15 shows two typical examples for which all methods provide a good IOD solution. Meanwhile, Figure 16 shows some samples



(a) (b)

Fig. 16: Zone 2 examples where one or all classical IOD methods do not provide a good initialization



(a) (b)

Fig. 17: Zone 2 examples where DeepONet IOD does not provide a good initialization

for which the classical IOD solutions are far from the true trajectory (Gauss's IOD method fails for the sample on the left and both Gauss and Double-r iteration fail for the sample on the right), in which case the BLS algorithm diverges completely, but the DeepONet still provides a good solution in these cases. Finally, Figure 17 shows

two cases for which the DeepONet predicts the trajectory to be far from the true trajectory. We note that the DeepONet mostly predicts the correct line-of-sight direction even when the predicted solution is far from the true solution and the BLS filter diverges, as seen in the examples in Figure 17.

6.3 Zone 3

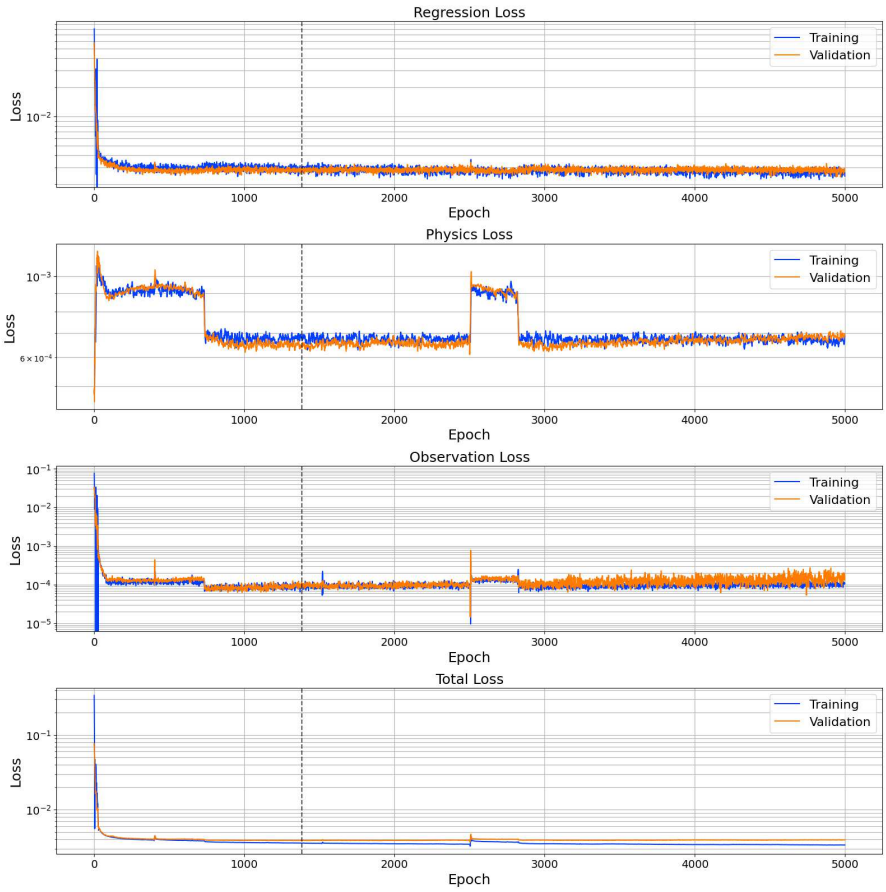
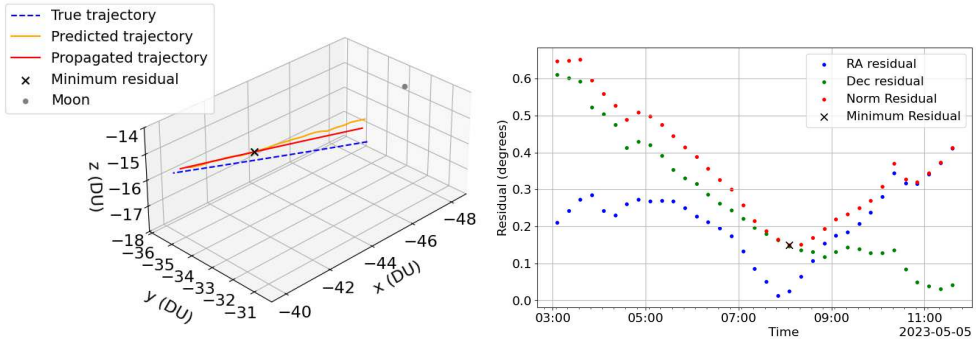


Fig. 18: Loss functions on training and validation subsets of Zone 3 dataset

Zone 3 is the most challenging of the three cases due to its proximity to the Moon. In addition to the very short length of observed arc, the trajectories are perturbed significantly by both the Sun and the Moon, which makes it much harder for the model to learn the dynamics from observations. The loss functions during training are shown in Figure 18. As in the previous two cases, the regression and observation losses decline rapidly over the first 500 epochs before stagnating. However, unlike before, the

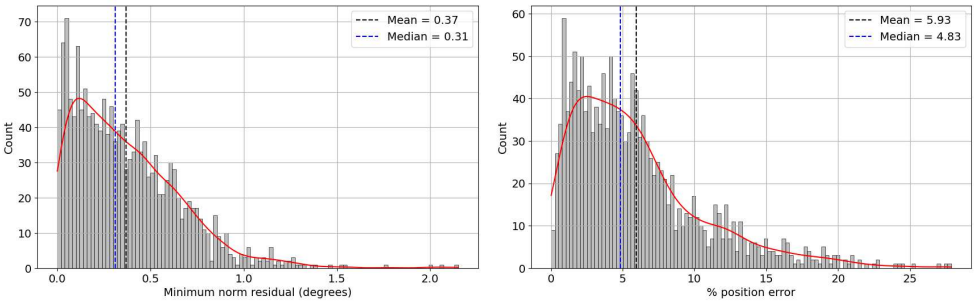
physics loss reaches close to its minimum value after 1000 epochs. The total validation loss achieves a minimum of 3.82×10^{-3} at epoch 1383, denoted by the dashed line.



(a) True and predicted trajectory

(b) Observation residuals

Fig. 19: Sample trajectory and DeepONet's prediction from Zone 3 dataset

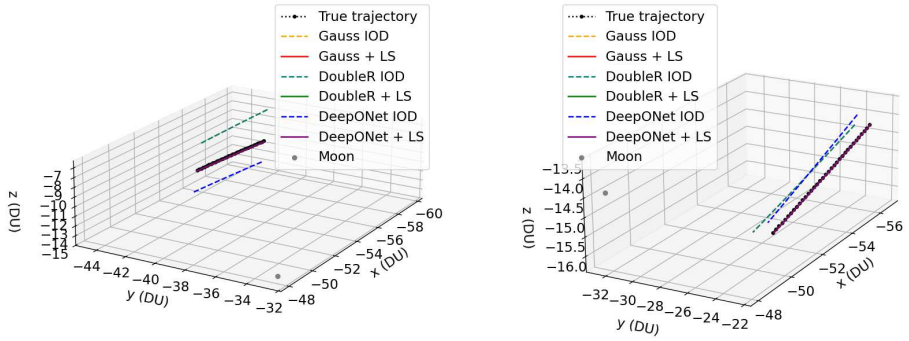


(a) Minimum observation residuals

(b) % Position error at point of minimum observation residual

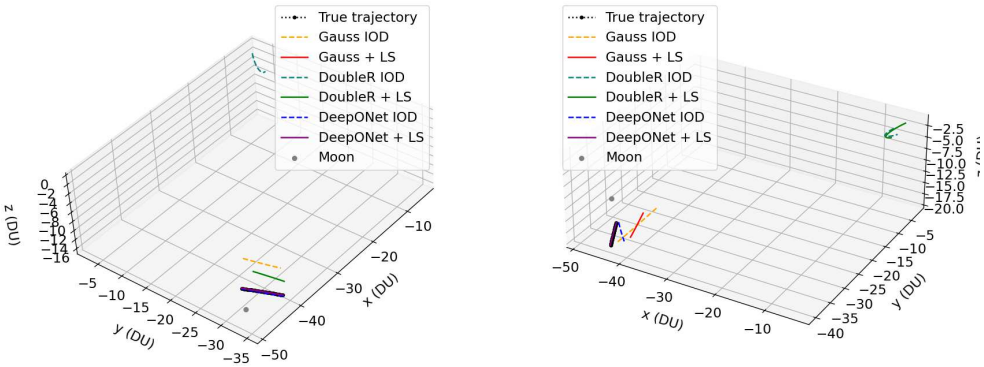
Fig. 20: Distribution of metrics for Zone 3 dataset

Figure 19 shows a sample from the test subset of the zone 3 dataset. We see that the predicted trajectory is not as close to the true trajectory as it was in the previous regions, even though the observation residuals remain small. This is because predicting the range is difficult by observing only the angles, since multiple trajectories can be mapped to the same set of observations. The distribution of metrics over the test dataset are shown in Figure 20. We see that despite the strong perturbations due to the Sun and the Moon, the minimum norm of observation residuals are lower than 1 degree for a majority of the samples. In addition, the position error at the point of minimum observation residual is also under 10% for most of the dataset.



(a) (b)

Fig. 21: Zone 3 examples where both IOD methods provide a good initialization

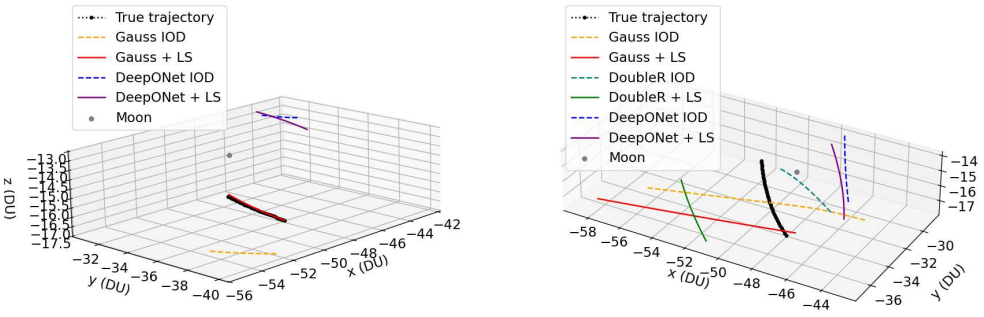


(a) (b)

Fig. 22: Zone 3 examples where one or both classical IOD methods do not provide a good initialization

When used in conjunction with BLS, the DeepONet once again outperforms both classical methods of IOD by providing a good initialization for 61.5% of the samples, compared to 48.1% when using Gauss's IOD method and 21.1% when using Double-r

iteration, on the same dataset. Some samples from this dataset are shown in Figs. 21, 22 and 23. Notably, the performance of all three IOD methods is much lower than in the previous zones, which can be attributed to both the strongly perturbed nature of the environment as well as the distance of the trajectories from the Earth resulting in very short observed arcs. This is a problem when using classical IOD methods as they rely on the assumption that the trajectories are Keplerian, and do not account for any perturbing effects, in which case the IOD solution can be quite far from the true solution as seen in Figure 22. Meanwhile, the DeepONet learns directly from data and accounts for the perturbations in the physics loss, and thus it performs better in this region.



(a) (b)

Fig. 23: Zone 3 examples where DeepONet IOD does not provide a good initialization

Figure 23 shows some examples where the DeepONet model's prediction is far from the true trajectory in the region close to the Moon (for the sample on the left only Gauss's method provides a reasonable initialization, whereas for the sample on the right none of the three IOD methods do so). Note that in some cases as shown in Figure 23, where the trajectory is on the far side of the Moon, the DeepONet predicts the IOD solution to be on the opposite (near) side, even though both the true trajectory and DeepONet predicted trajectory have similar line-of-sight directions, in which case the BLS algorithm does not converge onto the true solution. Nevertheless, for most samples, even though the length of the observed arc in this region is very small, it is long enough for the model to correctly predict the line-of-sight direction and the approximate location of the trajectory even in regions close to the Moon. This highlights the main advantage of the DeepONet over classical methods: its capacity to learn the dynamics of the region and map from observations to state in order to provide a reasonable IOD solution even in the presence of perturbations.

7 Conclusions

In this paper we presented a neural operator based architecture capable of performing Initial Orbit Determination in the xGEO regime using angles-only observation data. We identified three separate zones within the xGEO region based on the distance from the Earth and the dominant perturbing effects, and generate a dataset for each zone using a perturbed two-body dynamical model with third-body effects due to the Sun and the Moon. For each zone, we trained a separate model using a compound loss function in order to ensure that the model learns the correct mapping from observations to states while being constrained by the dynamics of the region. Finally, we compared the performance of our models with classical IOD methods when used to initialize Batch Least Squares algorithm in each zone. We found that in each zone, the DeepONet provides a better initialization for the BLS algorithm than both Gauss and Double-r methods for $5-10\%$ of the dataset. This is especially true for Zone 3, where the classical methods often fail to converge due to the highly non-Keplerian dynamics of the region closest to the Moon, while the DeepONet still provides a solution that is close enough to the real trajectory to serve as a good initialization for BLS. This ability to provide a solution even in the presence of perturbations is the main advantage of DeepONet-based IOD method over classical IOD methods.

Future work in this direction could involve considering more complex dynamical and observational models, as well as longer observation durations, such as over a period of days or weeks. Different architectures for branch and trunk networks could also be explored, for example, using recurrent layers or transformers, instead of the convolutional layers employed in this work. Lastly, the potential applications of neural operators for orbit determination with other data modalities, such as range/range-rate, could be considered.

References

[1] ESA Space Debris Office: Annual space environment report. Technical report, European Space Agency (2025). Accessed on November 14, 2025. https://www.sdo.esoc.esa.int/environment_report/Space_Environment_Report_latest.pdf

[2] Creech, S., Guidi, J., Elburn, D.: Artemis: An overview of nasa's activities to return humans to the moon. In: 2022 IEEE Aerospace Conference (AERO), pp. 1–7 (2022). https://doi.org/10.1109/AERO53065.2022.9843277

[3] Vallado, D.A., McClain, W.D.: Fundamentals of Astrodynamics and Applications. Microcosm Press, Hawthorne, CA (2013). https://books.google.com/books?id=8OjfnQEACAAJ

[4] Schutz, B., Tapley, B., Born, G.H.: Statistical Orbit Determination. Academic Press, Burlington, MA (2004). https://books.google.com/books?id=Ct3qN1VCHewC

[5] Small, J., Stoker-Spirt, E., Kaushik, A., Yu, C., Logan, J., Nicholson, A., Yu, W.:

Orbit determination for the james webb space telescope during launch and early orbit. In: 2022 AAS/AIAA Astrodynamics Specialist Conference, Paper Number: AAS 22-607, Charlotte, North Carolina, USA (2022). https://ntrs.nasa.gov/api/citations/20220010187/downloads/JWST_OD_Paper_AAS_Charlotte_v4.pdf

[6] Hesar, S.G., Parker, J.S., Leonard, J.M., McGranaghan, R.M., Born, G.H.: Lunar far side surface navigation using Linked Autonomous Interplanetary Satellite Orbit Navigation (LiAISON). Acta Astronautica **117**, 116–129 (2015) https://doi.org/10.1016/j.actaastro.2015.07.027

[7] Crum, S., Gunter, B.C.: Cislunar orbit determination via intercepted communication signals. In: American Institute for Astronau-tics and Aeronautics Ascend Conference, AIAA Paper 2023-4635 (2023). https://doi.org/10.2514/6.2023-4635

[8] Hartikainen, J., Seppänen, M., Särkkä, S.: State-space inference for non-linear latent force models with application to satellite orbit prediction. In: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, vol. 1, pp. 903–910 (2012). https://www.scopus.com/inward/record.uri?eid=2-s2.0-84867116396&partnerID=40&md5=09116a70180a12396ea873c9641b9dc0

[9] Álvarez, M., Luengo, D., Lawrence, N.D.: Latent force models. In: Twelfth International Conference on Artificial Intelligence and Statistics, Florida USA (2009). https://proceedings.mlr.press/v5/alvarez09a.html

[10] Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics **378**, 686–707 (2019) https://doi.org/10.1016/j.jcp.2018.10.045

[11] Badura, G., Velez-Reyes, M., Gunter, B., Valenta, C., Ho, K.: Regularizing Training of Physics Informed Neural Networks (PINNs) for Cislunar Orbit Determination via Transfer Learning. In: Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference (2024). https://amostech.com/TechnicalPapers/2024/Machine-Learning-for-SDA/Badura.pdf

[12] Scorsoglio, A., D'Ambrosio, A., Le Corre, L., Gray, B., Reddy, V., Furfaro, R.: Physics-informed orbit determination for x-geo space situational awareness. Acta Astronautica **238**, 271–285 (2026) https://doi.org/10.1016/j.actaastro.2025.09.002

[13] Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: Theory and applications. Neurocomputing **70**(1), 489–501 (2006) https://doi.org/10.1016/j.neucom.2005.12.126 . Neural Networks

[14] Lu, L., Jin, P., Karniadakis, G.E.: Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. arXiv:1910.03193 (2019) https://doi.org/10.48550/arXiv.1910.03193

[15] Wang, S., Wang, H., Perdikaris, P.: Learning the solution operator of parametric partial differential equations with physics-informed deeponets. Science Advances **7**(40), 8605 (2021) https://doi.org/10.1126/sciadv.abi8605

[16] Ye, K., Zhao, J., Liu, X., Moya, C., Lin, G.: Deeponet based uncertainty quantification for power system dynamics with stochastic loads. In: 2023 IEEE Power & Energy Society General Meeting (PESGM), pp. 1–6 (2023). https://doi.org/10.1109/PESGM52003.2023.10252937

[17] Kontolati, K., Goswami, S., Karniadakis, G., Shields, M.: Learning nonlinear operators in latent spaces for real-time predictions of complex dynamics in physical systems. Nature Communications **15** (2024) https://doi.org/10.1038/s41467-024-49411-w

[18] Curtis, H.D.: Orbital Mechanics for Engineering Students, Third edition edn. Butterworth-Heinemann, Boston, MA (2014). https://doi.org/10.1016/C2011-0-69685-1

[19] Escobal, P.R.: Methods of Orbit Determination. J. Wiley, New York (1965). https://books.google.com/books?id=G_MDAQAAIAAJ

[20] Chen, T., Chen, H.: Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Transactions on Neural Networks **6**(4), 911–917 (1995) https://doi.org/10.1109/72.392253

[21] Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems **2**, 303–314 (1989) https://doi.org/10.1007/BF02551274

[22] Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**, 359–366 (1989) https://doi.org/10.1016/0893-6080(89)90020-8