# Neural Laplace Control for Continuous-time Delayed Systems

**Samuel Holt**
University of Cambridge
sih31@cam.ac.uk

**Alihan Hüyük**
University of Cambridge
ah2075@cam.ac.uk

**Zhaozhi Qian**
University of Cambridge
zq224@maths.cam.ac.uk

**Hao Sun**
University of Cambridge
hs789@cam.ac.uk

**Mihaela van der Schaar**
University of Cambridge & The Alan Turing Institute
mv472@cam.ac.uk

## Abstract

Many real-world offline reinforcement learning (RL) problems involve continuous-time environments with delays. Such environments are characterized by two distinctive features: firstly, the state $x(t)$ is observed at irregular time intervals, and secondly, the current action $a(t)$ only affects the future state $x(t+\tau)$ with an *unknown* delay $\tau > 0$. A prime example of such an environment is satellite control where the communication link between earth and a satellite causes irregular observations and delays. Existing offline RL algorithms have achieved success in environments with irregularly observed states in time or known delays. However, environments involving both irregular observations in time and unknown delays remains an open and challenging problem. To this end, we propose *Neural Laplace Control*, a continuous-time model-based offline RL method that combines a Neural Laplace dynamics model with a model predictive control (MPC) planner—and is able to learn from an offline dataset sampled with irregular time intervals from an environment that has a inherent unknown constant delay. We show experimentally on continuous-time delayed environments it is able to achieve near expert policy performance.

## 1 INTRODUCTION

Online Reinforcement learning methods struggle to be applied to many real-world environments, for example in healthcare, business, and autonomous driving environments. RL methods rely on costly trial-and-error approaches performed either online or in a realistic simulator of the environment—which are not often readily available. In contrast, "offline" model-based reinforcement learning learns the environment dynamics, from a previously collected dataset of state-action trajectories, which is often readily available. It then controls the system to a desired goal using any suitable planning method, such as training a policy (Fujimoto et al., 2018) or using a model predictive controller (MPC) (Williams et al., 2016).

In practice, real-world environments are continuous in time by nature and possess constant delays $\tau$ whereby either actions are not executed instantaneously $\boldsymbol{a}(t+\tau)$, or states are not observed instantaneously $\boldsymbol{x}(t+\tau)$ (we formally define these later, in Section 3). For instance, in healthcare, observing a treatment effect $\boldsymbol{a}(t)$ from giving a patient a medication is not observable instantaneously and is instead *delayed* $\boldsymbol{x}(t+\tau)$, whilst measurements may be measured at *irregular* time intervals, $\boldsymbol{x}(t+\Delta_i)$, $\Delta_i \neq \Delta_j$—as is common where the frequency of observations is indicative of the patient's medical status (Goldberger et al., 2000). Similarly in autonomous driving it can take more than $\tau = 0.4$s for a hydraulic automotive braking system to generate a desired deceleration, therefore accounting for the delayed environment dynamics is *crucial* for correct safe control of the vehicle (Bayan et al., 2010). All together these environment dynamics, can often be described through sets of *delay differential equations* (DDEs) (Lynch & Park, 2017), however are often *unknown*.

Prior work has shown the success of model-based RL to learn from offline datasets consisting separately of either (1) irregularly-sampled data with no environment delays $\Delta_i \neq \Delta_j, \tau = 0$ (Yildiz et al., 2021) with **continuous-time methods**, or (2) regularly-sampled data with environment delays $\Delta_i = \Delta_j, \tau > 0$ (Chen et al., 2021) with **discrete-time delay methods**. However, performing offline model-based RL with both delays $\tau > 0$ and irregularly-sampled data $\Delta_i \neq \Delta_j$ for continuous-time control tasks is a largely understudied problem, yet an important problem setting.

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206. 

The existing two individual approaches are inherently incompatible with each other. On one hand, **continuous-time methods** use a continuous-time model, such as neural *ordinary differential equation* (ODE), to learn from irregularly-sampled observations $\Delta_i \neq \Delta_j, \tau = 0$ (Yildiz et al., 2021; Du et al., 2020). However, ODEs cannot model environments with unknown delays $\tau > 0$ (Holt et al., 2022). Furthermore, neural-ODE based models suffer from poor computational efficiency, when scaling to longer time horizons in a given trajectory (as shown in Section 5.2).

On the other hand, the existing methods for handling delays only considers a discrete-time $\Delta_i = \Delta_j, \tau > 0$ environment. Where these methods assume the delay $\tau$ is either *known* a priori (Firoiu et al., 2018) or is implicitly learned from a discrete buffer of previously executed actions $\bar{\boldsymbol{a}}_{i-1} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}\}$ (or states) (Chen et al., 2021). A simple approach for handling environments with delays $\tau > 0$ is to greatly increase the time interval between actions performed, so as to synchronize the agent's actions with its delayed observations. However, such an approach would lead to a "waiting agent" that is clearly sub optimal in most environments.

Hence, the following two properties are highly desirable for offline model-based RL to perform in more real-world environments.
**(P1) Learn from irregular samples**: able to learn from irregularly-sampled $\Delta_i \neq \Delta_j$ in time offline datasets resulting from a continuous-time environment.
**(P2) Learn delayed dynamics**: can learn the delayed dynamics of the environment, implicitly modeling any *unknown* delays $\tau > 0$.

To fulfill P1 and P2, we propose **Neural Laplace Control** (NLC), a continuous-time model-based RL method. Rather than describing the environment dynamics with a (neural) ODE, it uses Neural Laplace to learn implicit *delay differential equation* dynamics, which simultaneously accounts for unknown delays $\tau > 0$ and continuous-time dynamics $\Delta_i \neq \Delta_j$. This brings two immediate advantages. First, many continuous-time control problems involving delay DEs can easily be represented and solved in the Laplace domain (Schiff, 1999; Åström & Murray, 2021; Yi et al., 2008). Secondly, Neural Laplace Control bypasses the standard numerical ODE solvers and uses an inverse Laplace transform algorithm (Holt et al., 2022) to reconstruct any future state $\boldsymbol{x}(t + \Delta_i)$ of the dynamics model with the same amount of compute. This makes employing more principled planning strategies such as MPC feasible for continuous-time domains over expensive numerical step wise ODE based models.

Specifically, Neural Laplace Control is able to tackle the novel continuous-control problem formulated of having both *states observed at irregular time intervals* $\Delta_i \neq \Delta_j$ and an *unknown fixed delay* $\tau > 0$ *in the environment*. We motivate Neural Laplace Control as a principled approach for this problem and demonstrate the empirical effectiveness in experiments.

**Contributions** Our contributions are two-fold: ① In Section 4, we formulate and motivate the novel Neural Laplace Control method, that can learn a dynamics model that can encode an environments unknown delay differential equation dynamics from irregularly-sampled in time state-action trajectories (P1,P2). ② In section 5.1, we benchmark Neural Laplace Control against the existing continuous-time model-based approaches on standard continuous-time delayed environments. Specifically, we demonstrate that Neural Laplace Control is able to achieve near expert policy performance, significantly achieving a higher episode reward than the other competing continuous-time model-based baseline methods. We also gain insight and understanding of how Neural Laplace Control works in Section 5.2, of how it can correctly extrapolate to longer time horizons for the dynamics model and is computationally more efficient for predicting the next state at a longer time horizon, thereby making model predictive control feasible for longer time horizons for a fixed compute budget. All together, we learn such a model from irregularly-sampled states and actions in time $\Delta_i \neq \Delta_j$ (P1) and environments that possess a delay that is unknown $\tau > 0$ (P2).

A PyTorch (Paszke et al., 2019) implementation of the code is at https://github.com/samholt/NeuralLaplaceControl, and have a broader research group codebase at https://github.com/vanderschaarlab/NeuralLaplaceControl.

## 2 RELATED WORK

In offline reinforcement learning, an agent learns from a fixed replay buffer and is not permitted to interact with the environment (Wu et al., 2019). While both model-free (Kumar et al., 2019; 2020; Fujimoto & Gu, 2021) and model-based (Kidambi et al., 2020; Wang et al., 2021) approaches have been proposed for offline RL, in general, model-based methods have been shown to be more sample efficient than model-free methods (Moerland et al., 2020). The main challenge in model-based RL is known as "extrapolation error" (Fujimoto et al., 2019), whereby the learnt dynamics model inaccuracies compound for a larger number of future predicted time steps. Hence, it is crucial in model-based RL to learn an appropriate dynamics model that is capable of accurately capturing the unique characteristics of an environment. However, despite the fact that many environments operate in continuous-time by nature and contain action or observation delays, almost all of the existing approaches to model-based RL consider dynamics models only suited to the conventional discrete-time $\Delta_i = \Delta_j$ setting with no delays $\tau = 0$. We review here some of the few approaches that go beyond the conventional setting,

Table 1: Comparison with related model-based approaches to RL. **(P1) Learn from irregular samples**—can it learn from an offline dataset sampled at irregular times, $\Delta_i \neq \Delta_j$? **(P2) Learn delayed dynamics**—can it learn environments that contain a delay $\tau > 0$? Neural Laplace Control is the only method that can both learn from irregular samples (P1) as well as learn environments that contain a delay (P2).

| Approach | True Dynamics | Data Available | Reference | Model | (P1) $\Delta_i \neq \Delta_j$ | (P2) $\tau > 0$ |
|---|---|---|---|---|---|---|
| Conventional model-based RL | $\boldsymbol{x}_{t+1} \sim f(\boldsymbol{x}_t, \boldsymbol{a}_t)$ | $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{a}_i)\}_{i=0}^n$ | Williams et al. (2017) | MDP / Neural Network | ✗ | ✗ |
| Discrete-time delay methods | $\boldsymbol{x}_{t+1} \sim f(\boldsymbol{x}_t, \boldsymbol{a}_{t-\tau})$ | $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{a}_i)\}_{i=0}^n$ | Chen et al. (2021) | DA-MDP / RNN | ✗ | ✓ |
| Continuous-time methods | $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{a}(t))$ | $\mathcal{D} = \{(\boldsymbol{x}(t_i), \boldsymbol{a}(t_i))\}_{i=0}^n$ s.t. $\exists i,j : t_{i+1} - t_i \neq t_{j+1} - t_j$ | Yildiz et al. (2021) | Neural ODE | ✓ | ✗ |
| | | | Du et al. (2020) | Latent ODE | ✓ | ✗ |
| **Neural Laplace Control** | $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{a}(t-\tau))$ | $\mathcal{D} = \{(\boldsymbol{x}(t_i), \boldsymbol{a}(t_i))\}_{i=0}^n$ s.t. $\exists i,j : t_{i+1} - t_i \neq t_{j+1} - t_j$ | **(Ours)** | Neural Laplace Control | ✓ | ✓ |

namely (i) *discrete-time delay methods* and (ii) *continuous-time methods*.

**Discrete-time Delay Methods** Modeling environments with either delayed observations $\boldsymbol{x}(t+\tau)$ or delayed actions $\boldsymbol{a}(t+\tau)$ are equivalent in form (Katsikopoulos & Engelbrecht, 2003). Prior work models regular sampled $\Delta_i = \Delta_j$ (discrete time) environments with constant time delays $\tau > 0$, and provides the agent with the current state $\boldsymbol{x}(t)$, and a history of past actions performed in the environment $\bar{\boldsymbol{a}}_{i-1} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}\}$, whereby the history action window is larger than or equal to the observation or action delay in the environment (Walsh et al., 2009; Firoiu et al., 2018; Bouteiller et al., 2020; Liotet et al., 2021; Agarwal & Aggarwal, 2021). Recently, Chen et al. (2021) proposed delay-aware Markov decision processes (MDPs) that are capable of modeling delayed dynamics in discrete-time based on regularly sampled data, with an RNN encoding the history of past actions and the current state. Moreover, Derman et al. (2020) proposes a discrete-time *known* delay method—whereby, Derman et al. (2020)'s App. D.2 benchmarks against a model-free A2C baseline that only uses the current state-action fed into a RNN, that is *unable* to learn the delay.

**Continuous-time Methods** Real world data is often sampled irregularly $\Delta_i \neq \Delta_j$, as such (Yildiz et al., 2021) propose to use Neural ODEs (Chen et al., 2018b) as their continuous-time dynamics model that can model irregularly-sampled environments with no delays $\tau = 0$. Similarly, the work of Du et al. (2020) uses a Latent ODE model when planning policies. Moreover, the work of Seedat et al. (2022) uses a controlled differential equation (Kidger et al., 2020b) to model counterfactual outcomes. However, these existing approaches are limiting, as an ODE-based model by definition cannot handle a delay differential equation, necessitating the need for a model that can learn and model more diverse classes of differential equations. Recent models, of modeling diverse classes of differential equations is made possible with the work of Neural Laplace (Holt et al., 2022) by representing them in the Laplace domain. These Laplace-based models have been shown to be able to model such systems, be more accurate and scale better with increasing time horizons in time complexity. Our approach, namely Neural Laplace Control, essentially extends Neural Laplace to the setting of controlled systems—that is systems that evolve based on an action signal $\boldsymbol{a}(t)$—so that it can be used in planning policies in a RL setting. Furthermore, Bruder & Pham (2007) provides theory for the continuous time specific setting where action is an impulse, rather than a multivariate continuous input and the environment dynamics are a diffusion process.

We summarize the key related work in Table 1 and provide an extended discussion of additional related works, including a review of the benefits of using model-based RL and using model predictive control, which happens to be our preferred strategy for planning policies, in Appendix A.

## 3 PROBLEM FORMULATION

**States & Actions** For a system with *state* space $\mathcal{X} = \mathbb{R}^{d_{\mathcal{X}}}$ and *action* space $\mathcal{A} = \mathbb{R}^{d_{\mathcal{A}}}$, the state at time $t \in \mathbb{R}$ is denoted as $\boldsymbol{x}(t) = [x_1(t), \ldots, x_{d_{\mathcal{X}}}(t)] \in \mathcal{X}$ and the action at time $t \in \mathbb{R}$ is denoted as $\boldsymbol{a}(t) = [a_1(t), \ldots, a_{d_{\mathcal{A}}}(t)] \in \mathcal{A}$. We elaborate that *state trajectory* $\boldsymbol{x} : \mathbb{R} \to \mathcal{X}$ and *action trajectory* $\boldsymbol{a} : \mathbb{R} \to \mathcal{A}$ are both functions of time, where an individual state $\boldsymbol{x}(t) \in \mathcal{X}$ or an individual action $\boldsymbol{a}(t) \in \mathcal{A}$ are points on these trajectories. Given a time interval $\mathcal{I} \subseteq \mathbb{R}$, $\boldsymbol{x}_{\mathcal{I}} \in \mathcal{X}^{\mathcal{I}}$ and $\boldsymbol{a}_{\mathcal{I}} \in \mathcal{A}^{\mathcal{I}}$ we denote the partial state and action trajectories on that interval such that $\boldsymbol{x}_{\mathcal{I}}(t) = \boldsymbol{x}(t)$ and $\boldsymbol{a}_{\mathcal{I}}(t) = \boldsymbol{a}(t)$ for $t \in \mathcal{I}$. Finally, we also note that action values are usually bounded by an actuator's limits hence we also restrict the action space to $\mathcal{A} = [\boldsymbol{a}_{\min}, \boldsymbol{a}_{\max}]$, i.e., a box in Euclidean space.

**Environment Dynamics** Dynamics of the system are described by a non-autonomous non-linear controlled delay *differential equation* with a constant action delay $\tau \in \mathbb{R}_+$:

$$\dot{\boldsymbol{x}}(t) = \frac{d\boldsymbol{x}(t)}{dt} = f(\boldsymbol{x}(t), \boldsymbol{a}(t-\tau)) \tag{1}$$

where function $f : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^{d_{\mathcal{X}}}$ maps the current state and delayed action pair $\boldsymbol{x}(t), \boldsymbol{a}(t-\tau)$ to a state derivative $\dot{\boldsymbol{x}}(t)$. We note that this setting of *Continuous-time Control* (Kwakernaak & Sivan, 1972; Åström & Murray,

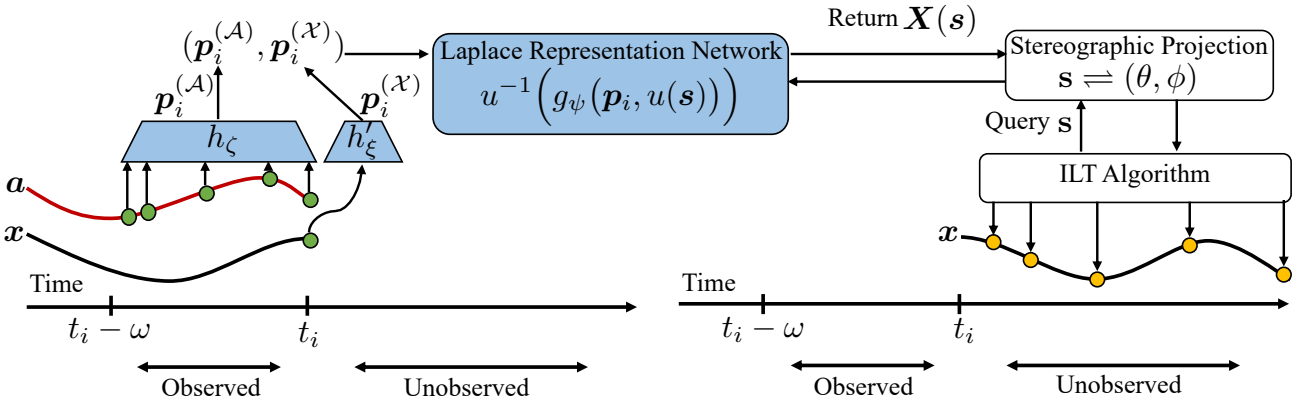


Figure 1: Block diagram of Neural Laplace Control. The query points $\boldsymbol{s}$ are given by the inverse Laplace transform (ILT) algorithm based on the time points of the future state trajectory to predict. The gradients can be back-propagated through the ILT algorithm and stereographic projection to train networks $h_\zeta, h'_\xi, g_\psi$.

2021), assumes deterministic environments with no observation noise—however, also consider observation noise in Appendix J, and find NLC still performant. Furthermore, we note that an environment that has either a constant action delay or a constant observation state delay are both equivalent and refer to a constant action delay throughout (Katsikopoulos & Engelbrecht, 2003). This is evident as an agent that interacts in an environment that *only* has an action delay of $\tau$ will only observe the effect of its selected action $a(t)$ $\tau$ seconds later in the state observation—therefore the agent has to decide an action based on a state observation that is delayed by $\tau$. Therefore, given an initial state $\boldsymbol{x}(0)$ as well as an action trajectory $\boldsymbol{a}_{(-\tau,t-\tau)}$, the state at time $t \in (0, \infty)$ can be written as

$$\boldsymbol{x}(t) = \boldsymbol{x}(0) + \int_0^t f(\boldsymbol{x}(t'), \boldsymbol{a}(t' - \tau))dt' \tag{2}$$

Consider some policy $\pi : \cup_{t\in\mathbb{R}}\mathcal{X}^{(-\infty,t]} \to \mathcal{A}$ that controls the system by mapping state trajectories $\boldsymbol{x}_{(-\infty,t]}$ to actions $a(t) = \pi(\boldsymbol{x}_{(-\infty,t]})$. Then similarly, given an initial state trajectory $\boldsymbol{x}_{(-\infty,0]}$, the state at time $t \in (0, \infty)$ for when the system is controlled by policy $\pi$ can be written as $\boldsymbol{x}^{(\pi)}(t) = \boldsymbol{x}(0) + \int_0^t f(\boldsymbol{x}(t'), \pi(\boldsymbol{x}_{(-\infty,t'-\tau]}))dt'$.

**Offline Dataset** We consider the case where the environment dynamics $f$ including the action delay $\tau$ are unknown. Instead, we observe irregularly-sampled in time state-action trajectories: $\mathcal{D} = \{(\boldsymbol{x}^{(k)}(t_i^{(k)}), \boldsymbol{a}^{(k)}(t_i^{(k)}))\}_{i=0}^{n^{(j)}}$ where $\boldsymbol{x}^{(k)}, \boldsymbol{a}^{(k)}$ denotes the $k$-th state-action trajectory sampled at irregular times $\{t_0^{(k)}, t_1^{(k)}, \ldots, t_{n^{(k)}}^{(k)}\}$; we drop the trajectory index $k$ unless explicitly needed. Letting $\Delta_i = t_i - t_{i-1}$ denote the time interval between two consecutive samples, having irregular samples entails that $\Delta_i \neq \Delta_j$ for some $i, j \in \{1, \ldots, n\}$. We also denote with $\boldsymbol{x}_i = \boldsymbol{x}(t_i)$ as the $i$-th state observation and with $\boldsymbol{a}_i = \boldsymbol{a}(t_i)$ as the $i$-th action observation.

**Control O bjective** Our overall objective is to control the environment to a given goal state $\boldsymbol{x}^* \in \mathcal{X}$. This is achieved by defining an instantaneous reward function $r : \mathcal{X} \to \mathbb{R}$ of the current state. A common reward function in continuous-time control is the exponential of the negative distance from the current state to the goal state, that is $r(\boldsymbol{x}(t)) = e^{-||\boldsymbol{x}(t)-\boldsymbol{x}^*||_2}$—so that the instantaneous reward is maximized when $\boldsymbol{x}(t) = \boldsymbol{x}^*$. Consequently, to achieve our objective, we seek to find the *optimal policy* $\pi^*$ within a feasible set of policies $\Pi$ that maximizes the reward integral for a given final time $T \in \mathbb{R}_+$:

$$\pi^* = \text{argmax}_{\,\pi\in\Pi}\left(R^{(\pi)} \doteq \int_0^T r(\boldsymbol{x}^{(\pi)}(t'))dt'\right) \tag{3}$$

given the offline dataset $\mathcal{D}$ but without access to the environment dynamics $f$ or the action delay $\tau$.

In practice, policies cannot observe continuous state trajectories in their entirety and hence have to work with point-wise samples instead. As such, we restrict our search space $\Pi$ to practical policies that only update their actions whenever a state observation is made. Given sampling times $\{t_0, t_1, \ldots\}$, these are policies of the form

$$\pi(\boldsymbol{x}_{(-\infty,t)}) = \begin{cases} \mathbf{0} & \text{if } t \in (-\infty, t_0) \\ \boldsymbol{a}_i & \text{if } t \in [t_i, t_{i+1}) \end{cases} \tag{4}$$

where actions $\boldsymbol{a}_i$ are generated recursively given $\boldsymbol{x}_{(-\infty,t_i]}$ by an auxiliary policy $\pi' : \mathcal{X} \times \cup_{j\in\mathbb{Z}_+}\mathcal{A}^j \to \mathcal{A}$ such that $\boldsymbol{a}_i = \pi'(\boldsymbol{x}_i, \bar{\boldsymbol{a}}_{i-1} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}\})$. Note that it is sufficient for the auxiliary policy to keep track of only the most recent state observation $\boldsymbol{x}_i$ due to the Markovianity of environment dynamics $f$, but it needs to keep track of all the past actions due to the (unknown) action delay $\tau$—keeping consistent with our convention of modeling action delays, however, note that this is equivalent for state delays. However, in the case of a having a fixed previous time window

$\omega \in \mathbb{R}_+$, the implicit learnable delay $\tau$ is bounded by $\omega$, i.e., $\tau < \omega$.

# 4 NEURAL LAPLACE CONTROL

We follow the standard model-based framework setup (Lutter et al., 2021). First, we learn a dynamics model as outlined in Section 4.1. Then in Section 4.2, we use this learnt model to plan a policy via model predictive control.

## 4.1 Learning the Dynamics Model

In the following we propose a way to incorporate *actions* into the Neural Laplace (Holt et al., 2022) model for modeling diverse DE systems, also detailed in Appendix B. We build on the Neural Laplace framework, which was originally designed to only model the state differential equation evolution. Specifically, Neural Laplace Control involves three main components: **(1)** an encoder that learns to infer and represent the initial representation of the current state-action trajectory up to time $t$, **(2)** a Laplace representation network that learns to represent the solutions of the state trajectory in the Laplace domain conditioned on the input state-action trajectories, and **(3)** an inverse Laplace transform (ILT) algorithm that converts the Laplace representation back to the time domain. We also note that Neural Laplace Control is preferable for nonlinear dynamics as nonlinear delay DE's can be solved in the Laplace domain, through the Laplace Adomian decomposition method (Yousef & Ismail, 2018). We provide a block diagram in Figure 1 and now discuss each component in detail.

**(1) Learning to Represent Initial Conditions at Time $t$** The future state trajectory solution depends on the *initial condition* of the state-action trajectories. To model the delay differential equation environment dynamics fully, we seek to encode this initial condition, whereby the dynamics implicitly depend on the *past* state-action histories. Therefore, Neural Laplace Control uses an encoder network to learn a *representation* of the current initial condition at time $t$ by encoding the recent state-action history of the trajectory up to a fixed previous time window $\omega \in \mathbb{R}+$. Note that ideally, we want $\omega > \tau$ since previous actions at least up to a time window of $\tau$ affect how states evolve in the future.

For a sample observed at time $t_i$, instead of encoding both state and action histories, we note that we only need to encode one history and follow the convention of Walsh et al. (2009) to only encode the current state $\boldsymbol{x}_i = \boldsymbol{x}(t_i)$, and the action history $\mathcal{H}_i = \{(\boldsymbol{a}_j, t_j - t_i) : t_j \in [t_i - \omega, t_i]\}$ up to the previous time window $\omega$. We highlight that the actions encoded can be at irregular times. As the action history varies in time, we encode it with a recurrent neural network, that of a reverse time gated recurrent unit (Chen et al., 2018a; Holt et al., 2022)—denoted as $h_\zeta$ with parameters $\zeta$—and encode the current state with a linear neural network layer—denoted as $h'_\xi$ with parameters $\xi$—and concatenate both into a latent dimension representing the initial condition of the state-action trajectory:

$$\boldsymbol{p}_i = (\boldsymbol{p}_i^{(\mathcal{A})} \doteq h_\zeta(\mathcal{H}_i),\ \boldsymbol{p}_i^{(\mathcal{X})} \doteq h'_\xi(\boldsymbol{x}_i)) \tag{5}$$

The vector $\boldsymbol{p}_i \in \mathcal{P} = \mathbb{R}^{d_\mathcal{P}}$ is the learned initial condition representation, where $d_\mathcal{P} \geq d_\mathcal{X}$ is a hyper-parameter. The encoders $h_\zeta, h'_\xi$ have trainable weights $\zeta, \xi$ respectively. Neural Laplace Control is agnostic to the exact choice of encoder architecture.

**(2) Learning DE Solutions in the Laplace Domain** Given an initial condition representation $\boldsymbol{p} \in \mathcal{P}$, we need to learn a function $l : \mathcal{P} \times \mathbb{C}^{d_\mathcal{S}} \to \mathbb{C}^{d_\mathcal{X}}$ that models the Laplace representation of the delay DE solution, i.e., when we take the inverse Laplace transform (ILT) of $\boldsymbol{X}(\boldsymbol{s}) = l(\boldsymbol{p}, \boldsymbol{s})$, it approximates $\boldsymbol{x}(t)$ well for future $t$. Here, $d_\mathcal{S} \in \mathbb{N}_+$ denotes the number of reconstruction terms per time point and is specific to the ILT algorithm (Appendix B). However, the Laplace representation $\boldsymbol{X}(\boldsymbol{s})$ often involves singularities (Schiff, 1999), which are difficult for neural networks to approximate or represent (Baker & Patil, 1998). Therefore, we use the proposed stereographic projection onto a Riemann sphere to mitigate this (Holt et al., 2022). With the stereographic projection, we introduce a feed-forward neural network $g$ to learn the Laplace representation of the dynamics model solution:

$$\boldsymbol{X}(\boldsymbol{s}) = u^{-1}\Big(g_\psi\big(\boldsymbol{p}, u(\boldsymbol{s})\big)\Big), \tag{6}$$

where $u$ is the stereographic projection and $u^{-1}$ is the inverse stereographic projection (Appendix B), the vector $\boldsymbol{p}$ is the output of the encoders (Equation 5), and $\psi$ is the trainable weights. Here the neural network's inputs and outputs are the coordinates on the Riemann Sphere $(\theta, \phi) \in \mathcal{D} = (-\pi, \pi) \times (-\frac{\pi}{2}, \frac{\pi}{2})$, which are bounded and free from singularities (Holt et al., 2022).

**(3) Inverse Laplace transform** After obtaining the Laplace representation $\boldsymbol{X}(\boldsymbol{s})$, we compute the predicted or reconstructed state values $\hat{\boldsymbol{x}}(t) = \mathcal{L}^{-1}\{\boldsymbol{X}\}(t)$ for future $t$, where $\mathcal{L}^{-1}$ is the inverse Laplace transform, using a numerical inverse Laplace transform algorithm. We highlight that we can evaluate $\hat{\boldsymbol{x}}(t)$ *at any future time* $t \in \mathbb{R}_+$ as the Laplace representation is independent of time once learnt. In practice, we use the well-known ILT Fourier series inverse algorithm (ILT-FSI), which can obtain the most general time solutions whilst remaining numerically stable (Dubner & Abate, 1968; De Hoog et al., 1982; Kuhlman, 2013; Holt et al., 2022).

**Loss Function** Neural Laplace Control trains its dynamics model end-to-end using the mean squared error loss of

the next step ahead prediction error,

$$\mathcal{J}(\zeta, \xi, \psi) = \sum\nolimits_{t \in \{t_{i+1}, \ldots, t_n\}} \|\hat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)\|_2^2 \quad (7)$$

$$\text{where} \quad \hat{\boldsymbol{x}}(t) = \mathcal{L}^{-1}\{\boldsymbol{X}(\cdot) = u^{-1}(g_\psi(\boldsymbol{p}_i, u(\cdot))\}(t) \quad (8)$$

We minimize the above loss function $\mathcal{J}$ to learn the encoders $h_\zeta, h_\xi$ and the Laplace representation network $g_\psi$. This training is summarized in Appendices B and G.

### 4.2 Planning with the Learnt Dynamics Model

Once we have learnt the dynamics model, any arbitrary model-based reinforcement learning framework can be used for planning. The straightforward choice would be to perform deep Q-learning using the dynamics model as a simulator. Ideally, we want to pursue a more principled approach that can accommodate the Laplace-domain representation of the dynamics more readily. Specifically, this Laplace-domain representation provides us with the important benefit of being able to simulate state trajectories for arbitrarily long control signals without requiring any additional compute. This is *not* the case for neural ODEs (Holt et al., 2022), as shown in Section 5.2.

**Laplace-model with MPC** We opt to use the model predictive controller of Model Predictive Path Integral (MPPI) (Williams et al., 2017). This uses a zeroth order particle-based trajectory optimizer method with our learned Laplace dynamics model. Specifically, this computes a discrete action sequence up to a fixed time horizon of $H \in \mathbb{R}_+$, and then executes the first element in the planned action sequence. We note that our continuous-time Laplace-based dynamics model can be used to reconstruct a trajectory at any future time horizon up to $H$ seconds, however it requires that the corresponding control input up to that future time horizon is input into the dynamics model, i.e., $\boldsymbol{a}_{[t,t+H)}$. To simplify the planning problem, we assume a state, and action history tuple is given to the Laplace-based dynamics model, along with the time interval to predict the dynamics model next future state at, i.e., an input of $(\boldsymbol{x}_t, \boldsymbol{a}_{[t-\omega:t]}, \delta)$, to predict $\boldsymbol{x}_{t+\delta}$. Where we denote $\delta \in \mathbb{R}_+$ as the observation time interval, that is the time between two consecutive state observations [1]. It is natural for online control problems to be controlled at discrete-time steps of $\delta$, where $\delta$ can be varied. Therefore, the MPPI plans actions at discrete-time steps $\delta$, up to a fixed horizon $H$ by planning ahead $N \in \mathbb{Z}_+$ steps into the future, thus the planning horizon is determined by $H = \delta \cdot N$. This leverages a number of parallel roll-outs $M \in \mathbb{Z}_+$, a hyper parameter, which can be tuned. As MPPI is a Monte Carlo based sampler, increasing the number of roll-outs improves the input trajectory optimization, however scales the runtime complexity as $\mathcal{O}(NM)$. Although planning benefits from having a longer time horizon to use when optimizing the next action trajectory, it becomes computationally infeasible to do so for a large $N$. Naturally with the Neural Laplace Control dynamics model representation we can change the observation interval $\delta$ time step, to increase it to enable planning at a longer time horizon. Clearly, however, planning at a longer horizon can compound model inaccuracies (Williams et al., 2017)—which can become significant, rendering the model uncontrollable within that planning regime, and is further explored in Section 5.2. Additionally, we also detail the MPC MPPI pseudocode and planner implementation in Appendix C.

[1] We note that the observation state time interval $\delta$ is the same as the time interval between the executed actions.

## 5 EXPERIMENTS AND EVALUATION

**Benchmark Environments** We use the continuous-time control environments from the ODE-RL suite (Yildiz et al., 2021), as they provide true irregular samples in time of state observations and are fully continuous in time, unlike discrete environments (Brockman et al., 2016). We adapt these to incorporate an arbitrary fixed delayed action time, turning the ODE environments into delay DE environments. This ODE-RL suite consists of three environments of the Pendulum, Cartpole and Acrobot. The starting state for all tasks is hanging down and the goal is to swing up and stabilize the pole(s) upright (Yildiz et al., 2021). Here, each environment uses the reward function of the exponential of the negative distance from the current state to the goal state $\boldsymbol{x}^*$, whilst also penalizing the magnitude of action, and we assume we are given this reward function when planning. We detail all environments in Appendix D.

**Benchmark Dynamic Models** We select benchmark dynamics models for our specific setting of having both *states observed at irregular time intervals* $\Delta_i \neq \Delta_j$ and an *unknown fixed delay* $\tau > 0$ *in the environment*. We benchmark against a *discrete-delay method* of a RNN over the action buffer and current state (Chen et al., 2021), and adapt it to model continuous-time with a new input of the time increment to predict the next state for ($\Delta t-$**RNN**) [2]. We also compare with the true environment dynamics (**Oracle**), an augmented Neural-ODE (**NODE**) (Chen et al., 2018b), Latent-ODE (**Latent-ODE**) (Rubanova et al., 2019b) and our Neural Laplace Control (**NLC**) model. We plan all dynamics models with the MPC MPPI method (Williams et al., 2017), and further compare against a random policy (**Random**). We provide further details of model selection, hyperparameter selection and implementation details in Appendix E.

[2] We note to adapt discrete-time models to continuous-time we add an additional input parameter, that of the time difference between the current time and the next state observation to predict, i.e., $\delta$, e.g., $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{a}_i, \delta)$. (Yildiz et al., 2021)

Table 2: Normalized scores $\mathcal{R}$ of the offline model-based agents, where the irregularly-sampled (P1) offline dataset consists of an action delay (P2) of $\{1, 2, 3\}$ multiples of the environments observation interval time step $\bar{\Delta} = 0.05$ seconds. Averaged over 20 random seeds, with $\pm$ standard deviations. Scores are un-discounted cumulative rewards normalized to be between 0 and 100, where 0 corresponds to the Random agent and 100 corresponds to the expert with the *known* world model (Oracle+MPC). Negative normalized scores, i.e., worse than random are set to zero. Full results are included in the Appendix H.

| | Action Delay $\tau = \bar{\Delta}$ | | | Action Delay $\tau = 2\bar{\Delta}$ | | | Action Delay $\tau = 3\bar{\Delta}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| Dynamics Model | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot |
| Random | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |
| Oracle | 100.0±0.15 | 100.0±3.14 | 100.0±2.19 | 100.0±0.04 | 100.0±2.57 | 100.0±1.79 | 100.0±0.08 | 100.0±2.57 | 100.0±1.26 |
| $\Delta t-$RNN | 95.28±0.4 | 1.14±6.31 | 18.95±7.6 | 97.01±0.31 | 9.94±2.48 | 28.39±9.73 | 97.8±0.25 | 11.81±11.93 | 3.89±6.72 |
| Latent-ODE | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 1.24±20.67 | 8.91±13.62 | 41.56±47.07 | 3.26±12.24 | 9.19±9.08 |
| NODE | 85.09±7.95 | 0.63±5.16 | 23.07±6.94 | 90.75±1.34 | 0.0±0.0 | 10.92±10.09 | 94.55±1.08 | 1.97±4.01 | 11.78±8.33 |
| **NLC (Ours)** | **99.83±0.19** | **98.31±3.51** | **99.12±1.7** | **99.88±0.1** | **93.28±4.96** | **100.44±2.13** | **99.92±0.12** | **98.98±1.32** | **99.46±1.88** |

**Offline Dataset Generation** For each environment we generate an offline state-action trajectory dataset by using an agent that uses an oracle dynamics model combined with MPC and has additional noise added to the agents selected action, $\bar{\pi}(t) = \pi(t) + \epsilon, \epsilon \sim \mathcal{N}(0, \boldsymbol{a}_{\max})$. This "noisy expert" agent interacts with the environment and observes observations at irregular unknown times, where we sample the time interval to the next observation from an exponential distribution, i.e., $\Delta \sim \text{Exp}(\bar{\Delta})$, with a mean of $\bar{\Delta} = 0.05$ seconds [3] (Yildiz et al., 2021). We assume a fixed action delay $\tau$, and evaluate discrete multiples of this delay of the mean sampling time $\bar{\Delta}$, i.e., $\tau = \bar{\Delta}$ for one step delay, $\tau = 2\bar{\Delta}$ for two step delay etc. We enforce the observed action history buffer that includes past actions back to $\omega = 4\bar{\Delta}$ seconds. We provide further details on the dataset generation and model training in Appendix G.

**Evaluation** For each environment, with a different delay setting (described above) we collect an offline dataset of irregularly-sampled trajectories, consisting of 1e6 samples from the "noisy expert" agent interacting within that environment. For each benchmark dynamics model, we follow the same two step evaluation process of, firstly, training the dynamics model on that environment's collected offline dataset using a MSE error loss for the next step ahead state prediction $\hat{\boldsymbol{x}}(t_{i+1})$. Then, secondly, taking the same pre-trained model and freezing the weights, and only using it for planning with the MPPI (MPC) planner at run-time in an environment episode, that lasts for 10 seconds. In total, we evaluate our model-based control algorithms online in the same environment, running each one for a fixed observation interval of $\delta = \bar{\Delta} = 0.05$ seconds (as is the nominal value for these environments (Yildiz et al., 2021; Brockman et al., 2016)), and take the cumulative reward value after running one episode of the planner (policy) and repeat this for 20 random seed runs for each result. We quote the normalized score $\mathcal{R}$ (Yu et al., 2020) of the policy in the environment, averaged over the 20 random seed run episodes, with standard deviations throughout. The scores are un-discounted cumulative rewards normalized to lie roughly between 0 and 100, where a score of 0 corresponds to a random policy, and 100 corresponds to an expert (oracle with a MPC planner). We further detail our evaluation metrics and experimental setup in Appendix F.

[3] We note other irregular sampling types are possible, however Yildiz et al. (2021) has shown they are approximately equivalent.

### 5.1 Main results

We compared all the benchmark methods against each environment, which consists of a continuous-time environment with a specific delay—with normalized scores $\mathcal{R}$ are tabulated in Table 2. Neural Laplace Control achieves high normalized scores (high episode rewards) on all the environments. Specifically, NLC is able to model naturally a variety of different delay environment dynamics (P2), whereas existing *delay methods* adapted to continuous-time ($\Delta t-$RNN) struggle to learn appropriate dynamics models for a range of different challenging environments. Importantly, NLC performs well by learning a *good* dynamics model from the irregularly-sampled offline datasets, whereas the standard *continuous-time methods* (NODE, Latent-ODE) struggle to learn such a model from environments that have an inherent delay. We also observe similar patterns from additional experiments in Appendix J.

### 5.2 Insight and Understanding of How Neural Laplace Control Works

In this section we seek to gain further insight into *how* Neural Laplace Control outperforms the benchmarks. In the following we seek to understand if NLC is able to learn from irregularly-sampled state-action offline datasets (P1), whilst learning the delayed dynamics of the environment (P2). Furthermore, we also explore the benefits of the NLC approach for planning at longer time horizons with a fixed amount of compute and being sample efficient.

**Can NLC Learn a Good Dynamics Model?** To explore if NLC is able to learn a suitable dynamics model, we plot the trained models next step ahead prediction error with that of the ground truth for a varying observation interval $\delta$ for the Cartpole environment with a delay of $\bar{\Delta}$, as shown

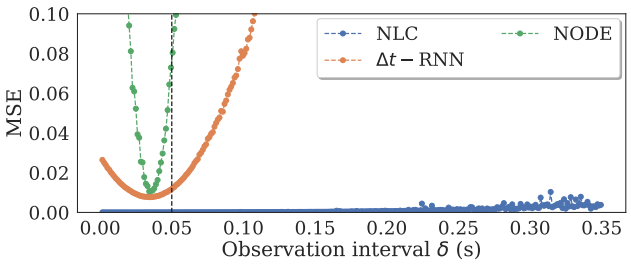


Figure 2: Next step ahead validation error (MSE) at a variable time step of an observation interval $\delta$ of the learnt baseline dynamics models, for the irregularly-sampled Cartpole environment with a fixed action delay of $\tau = \bar{\Delta}$. The black dotted line indicates the environments run-time observation interval $\delta = \bar{\Delta} = 0.05$ s. Here, we observe Neural Laplace Control learns a good dynamics model over a wide range of observation intervals $\delta$, correctly learning from the irregularly-sampled offline dataset (P1).

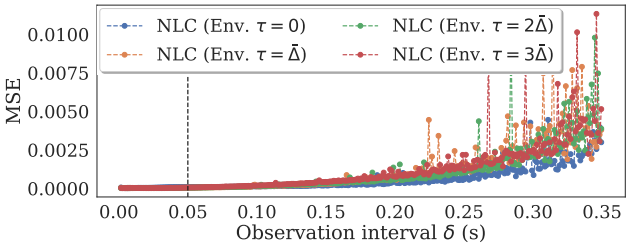


Figure 3: Next step ahead validation error (MSE) at a variable time step of an observation interval $\delta$ of the learnt Neural Laplace Control dynamics models, for each delayed environment versions $\tau = \{0, \bar{\Delta}, 2\bar{\Delta}, 3\bar{\Delta}\}$ of the specific Cartpole environment. The black dotted line indicates the environments run-time observation interval $\delta = \bar{\Delta} = 0.05$ s. Here, Neural Laplace Control is able to correctly learn and capture the delayed dynamics (P2), as the forward MSE errors are low and similar—whereas neural-ODE methods have a greater increasing forward MSE, Appendix I.

in Figure 2. Empirically we observe that NLC using its Laplace-based dynamics model is able to better approximate a wider range of observation intervals $\delta$ and achieve a good *global* approximation compared to the recurrent neural network and ODE based models. We note that due to the offline dataset being sampled with trajectories that have irregular sampling times (P1), where the sampling times are defined by an exponential distribution with a mean of $\bar{\Delta} = 0.05$ seconds; the other competing methods seem to over-fit purely to the median sample time of the exponential distribution, i.e., $0.05 \cdot \ln(2) = 0.034$ s. Other works have shown a more accurate next step prediction model correlates to a higher environment episode reward (Williams et al., 2017).

**Can NLC Learn Delay Environment Dynamics?** To investigate this, we similarly plot the trained NLC dynamics models next step ahead prediction error with that of the ground truth for a varying observation interval $\delta$, for each of the delayed environment versions of the specific Cartpole environment, as show in Figure 3. Empirically

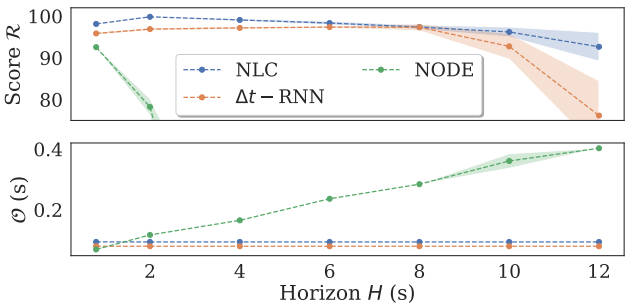


Figure 4: Normalized score $\mathcal{R}$ of the baseline methods on the Cartpole environment with an action delay of $\tau = \bar{\Delta} = 0.05$ seconds, plotted against an increasing time horizon $H$, by increasing the observation interval $\delta$. NLC, maintains a high performing policy at a longer time horizon—whilst using the same amount of *constant* planning time per action $\mathcal{O}$ as a $\Delta t-$RNN.

we observe that the NLC dynamics models correctly learnt the delay dynamics (P2) of each individual environment, as they each have a similar low forward MSE error for the varying levels of inherent delay. In contrast, neural-ODE models are unable to model the delay dynamics correctly, and we observe that they have a higher rate of increasing forward MSE error, that can also increase for an increasing environment delay and is shown further in Appendix I.

**Can NLC Plan with a Longer Time Horizon Using a Fixed Amount of Compute?** We investigate this by planning with a MPC planner, increasing the observation interval $\delta$ and keeping $N$ fixed, therefore the time horizon $H$ increases—as shown in Figure 4. Here we measure the total planning time taken to plan the next action as $\mathcal{O}$ seconds [4] and observe that planning with the NLC dynamics model takes the same amount of planning time, and hence a *fixed amount compute* for planning at a greater time horizon $H$—which is the same as a $\Delta t-$RNN. This is achieved by the Laplace-based dynamics model that can predict a future state at *any* future time interval using the same number of forward model evaluations, and hence the same amount of compute. In contrast, this is *not* readily achievable with neural-ODE continuous-time methods that use a larger number of numerical forward steps with a numerical ODE step-wise solver for a increasing time horizon—leading to an increasing planning time for an increasing time horizon, i.e., $\mathcal{O} \propto H$. Furthermore, we highlight, that there exists a trade-off of the time horizon $H$ to plan at—as we wish to use a large "enough" horizon that captures sufficient future dynamics, whilst minimizing compounded model inaccuracies at a larger planning time horizon. Therefore these two opposing factors, give rise to the maxima of the normalized score $\mathcal{R}$ at a time horizon $H = 2$ seconds, as seen in Figure 4.

We further investigate an alternative setup in Figure 5, and

[4] We perform all results using a Intel Core i9-12900K CPU @ 3.20GHz, 64GB RAM with a Nvidia RTX3090 GPU 24GB.

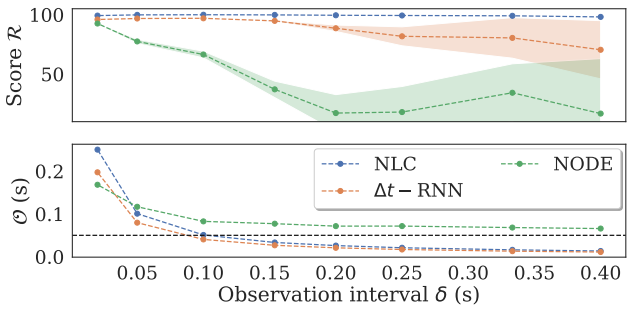


Figure 5: Normalized score $\mathcal{R}$ of the baseline methods on the Cartpole environment with an action delay of $\tau = \bar{\Delta} = 0.05$s, plotted against an increasing observation interval $\delta$. Here, the time horizon is fixed at $H = 2$s, thus increasing the observation interval $\delta$ decreases the number of MPC forward planning steps needed (i.e., $N = \frac{H}{\delta}$). The black dotted line indicates the environments run-time observation interval $\delta = \bar{\Delta} = 0.05$ s. NLC demonstrates that it can still outperform the baselines, achieving a near optimal policy—whilst reducing the planning time taken $\mathcal{O}$ needed to generate the next action.

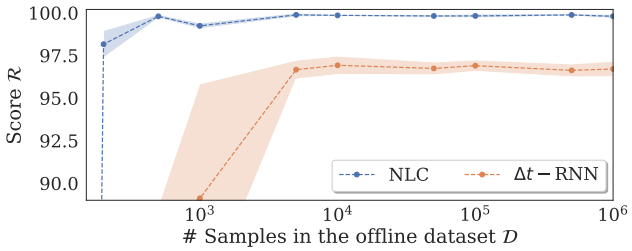


Figure 6: Normalized score $\mathcal{R}$ of the baseline methods on the Cartpole environment with an action delay of $\tau = \bar{\Delta} = 0.05$s, plotted against the number of samples in the irregularly-sampled offline dataset used to train the dynamics model of each method. The two closest highest performing baselines are plotted here, and refer to Appendix I for others. NLC can maintain a high performing policy—even from the challenging case of only learning a dynamics model from 200 samples from an irregularly-sampled in time offline dataset $\mathcal{D}$.

keep the time horizon fixed at $H = 2$ seconds and increase the observation interval $\delta$—allowing us to reduce $N$ the number of MPC forward planning steps (i.e., $N = \frac{H}{\delta}$). Importantly, this *reduces the planning time* $\mathcal{O}$ needed to generate the next action, enabling a method to use a higher frequency of executing actions to control the dynamics—whilst still planning at the *same fixed time horizon* $H$. NLC is able to still outperform the baselines, achieving a high performing policy—even when using a lesser amount of planning compute per action. The numeric values, along with those for other environments are provided in Appendix I.

**Is NLC Sample Efficient?** We observe in Figure 6 that NLC can still learn a suitable dynamics model, and perform well on the Cartpole environment with a delay of $\tau = \bar{\Delta} = 0.05$ seconds, when trained with an offline irregularly-sampled in time dataset that contains only 200 random samples—which corresponds 10 seconds of interaction time of a noisy expert (expert with random action noise) agent from the delayed environment. We further detail full results, including other environment results in Appendix I.

**Can NLC Incorporate Adaptive State-based Constraints** Using an MPC planner, NLC can naturally handle unseen state-based constraints, and we show this in Appendix I.

## 6 DISCUSSION AND FUTURE WORK

**Discussion** In this work, we have proposed and validated a novel model-based offline RL method, which combines a Neural Laplace dynamics model with a MPC planner. This novel method performs RL in continuous-time, training only on offline irregularly-sampled data that has an inherent delay. We have shown experimentally that these Laplace-domain models outperform their neural-ODE and recurrent neural-network based counterparts on irregularly-sampled datasets, and make model predictive control feasible for longer time horizons, with a fixed compute budget.

**Future Works** Our current focus is on continuous-time environments with unknown *fixed* delays. We consider learning in environments with unknown and *variable* delays an important area for future work. Furthermore, in the current work, we solely explored using only one instance of the dynamics model. However, we note that we can extend NLC trivially to create an *ensemble* of Neural Laplace Control models, thereby providing uncertainty estimation (epistemic uncertainty) of the future state prediction.

**Societal Impact** We envisage Neural Laplace Control as a tool to perform offline RL in realistic continuous-control settings, although emphasize that the dynamics action control trajectory proposed would need to be further verified by a human expert or via experimentation.

### Acknowledgements

SH would like to acknowledge and thank AstraZeneca for funding. This work was additionally supported by the Office of Naval Research (ONR) and the NSF (Grant number: 1722516). Moreover, we would like to warmly thank all the anonymous reviewers, alongside research group members of the van der Scaar lab, for their valuable input, comments and suggestions as the paper was developed—where all these inputs ultimately improved the paper.

### References

Mridul Agarwal and Vaneet Aggarwal. Blind decision making: Reinforcement learning with delayed observations. *Pattern Recognition Letters*, 150:176–182, 2021.

Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. In *International Conference on Learning Representations*, 2020.

Karl Johan Åström and Richard M Murray. *Feedback systems*. Princeton university press, 2010.

Karl Johan Åström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.

Mark R Baker and Rajendra B Patil. Universal approximation theorem for interval neural networks. *Reliable Computing*, 4(3):235–239, 1998.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

Fawzi P Bayan, Anthony D Cornetto, Ashley Dunn, and Eric Sauer. Brake timing measurements for a tractor-semitrailer under emergency braking. *SAE International Journal of Commercial Vehicles*, 2(2):245–255, 2010.

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *International conference on learning representations*, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Benjamin Bruder and Huyên Pham. Impulse control problem on finite horizon with intervention lag and execution delay. 2007.

Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. Delay-aware model-based reinforcement learning for continuous control. *Neurocomputing*, 450:119–128, 2021.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018a.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018b.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Frank R De Hoog, JH Knight, and AN Stokes. An improved method for numerical inversion of laplace transforms. *SIAM Journal on Scientific and Statistical Computing*, 3(3):357–366, 1982.

Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies. In *International Conference on Learning Representations*, 2020.

J Du, J Futoma, and F Doshi-Velez. Model-based reinforcement learning for semi-Markov decision processes with neural ODEs. In *Proceedings of the 34th Conference on Neural Information Processing Systems*, 2020.

Harvey Dubner and Joseph Abate. Numerical inversion of laplace transforms by relating them to the finite fourier cosine transform. *Journal of the ACM (JACM)*, 15(1): 115–123, 1968.

Vlad Firoiu, Tina Ju, and Josh Tenenbaum. At human speed: Deep reinforcement learning with action delay. *arXiv preprint arXiv:1810.07286*, 2018.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

Samuel I Holt, Zhaozhi Qian, and Mihaela van der Schaar. Neural laplace: Learning diverse classes of differential

equations in the laplace domain. In *International Conference on Machine Learning*, pp. 8811–8832. PMLR, 2022.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Konstantinos V Katsikopoulos and Sascha E Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE transactions on automatic control*, 48(4):568–574, 2003.

Li Kexue and Peng Jigen. Laplace transform and fractional differential equations. *Applied Mathematics Letters*, 24 (12):2019–2023, 2011.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020.

Patrick Kidger, Ricky TQ Chen, and Terry Lyons. " hey, that's not an ode": Faster ode adjoints with 12 lines of code. *arXiv preprint arXiv:2009.09457*, 2020a.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. In *Conference on Neural Information Processing Systems*. Neural Information Processing Systems Foundation, 2020b.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Marin Kobilarov. Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871, 2012.

Kristopher L Kuhlman. Review of inverse laplace transform algorithms for laplace-space numerical approaches. *Numerical Algorithms*, 63(2):339–355, 2013.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

H Kwakernaak and R Sivan. *Linear Optimal Control Systems*. Wiley InterScience, New York, 1972.

Wook Hyun Kwon, Jin Won Kang, Young Sam Lee, and Young Soo Moon. A simple receding horizon control for state delayed systems and its stability criterion. *Journal of Process Control*, 13(6):539–551, 2003.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*, pp. 143–156. PMLR, 2017.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Pierre Liotet, Erick Venneri, and Marcello Restelli. Learning a belief representation for delayed reinforcement learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2021.

Pierre Liotet, Davide Maran, Lorenzo Bisi, and Marcello Restelli. Delayed reinforcement learning by imitation. In *International Conference on Machine Learning*, pp. 13528–13556. PMLR, 2022.

Michael Lutter, Leonard Hasenclever, Arunkumar Byravan, Gabriel Dulac-Arnold, Piotr Trochim, Nicolas Heess, Josh Merel, and Yuval Tassa. Learning dynamics models for model predictive agents. *arXiv preprint arXiv:2109.14311*, 2021.

Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Igor Podlubny. The laplace transform method for linear differential equations of the fractional order. *arXiv preprint funct-an/9710005*, 1997.

Alexander D Poularikas. *Transforms and applications handbook*. CRC press, 2018.

Tobias Raff, Carsten Angrick, Rolf Findeisen, Jung-Su Kim, and Frank Allgower. Model predictive control for nonlinear time-delay systems. *IFAC Proceedings Volumes*, 40(12):60–65, 2007.

Jacques Richalet, André Rault, JL Testud, and J Papon. Model predictive heuristic control: Applications to industrial processes. *Automatica*, 14(5):413–428, 1978.

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019a.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019b.

Walter Rudin. *Real and Complex Analysis, 3rd Ed.* McGraw-Hill, Inc., USA, 1987. ISBN 0070542341.

Tim Salzmann, Elia Kaufmann, Marco Pavone, Davide Scaramuzza, and Markus Ryll. Neural-mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. *arXiv preprint arXiv:2203.07747*, 2022.

Joel L Schiff. *The Laplace transform: theory and applications*. Springer Science & Business Media, 1999.

Nabeel Seedat, Fergus Imrie, Alexis Bellot, Zhaozhi Qian, and Mihaela van der Schaar. Continuous-time modeling of counterfactual outcomes using neural controlled differential equations. *arXiv preprint arXiv:2206.08311*, 2022.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2019.

Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, USA, 1997. ISBN 0966017633.

Thomas J Walsh, Ali Nouri, Lihong Li, and Michael L Littman. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems*, 18(1):83–105, 2009.

Jianhao Wang, Wenzhe Li, Haozhe Jiang, Guangxiang Zhu, Siyuan Li, and Chongjie Zhang. Offline reinforcement learning with reverse model-based imagination. *Advances in Neural Information Processing Systems*, 34: 29420–29432, 2021.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1433–1440. IEEE, 2016.

Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721. IEEE, 2017.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Sun Yi, A Galip Ulsoy, and Patrick W Nelson. Solution of systems of linear delay differential equations via laplace transformation. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 2535–2540. IEEE, 2006.

Sun Yi, Patrick W Nelson, and A Galip Ulsoy. Controllability and observability of systems of linear delay differential equations via the matrix lambert w function. *IEEE Transactions on Automatic Control*, 53(3):854–860, 2008.

Cagatay Yildiz, Markus Heinonen, and Harri Lähdesmäki. Continuous-time model-based reinforcement learning. In *International Conference on Machine Learning*, pp. 12009–12018. PMLR, 2021.

Hamood M Yousef and AIB MD Ismail. Application of the laplace adomian decomposition method for solution system of delay differential equations with initial value problem. In *AIP Conference Proceedings*, volume 1974, pp. 020038. AIP Publishing LLC, 2018.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33: 14129–14142, 2020.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. *arXiv preprint arXiv:1909.12077*, 2019.

Contents of supplementary materials:



**Code** We have released a PyTorch implementation (Paszke et al., 2019) at https://github.com/samholt/NeuralLaplaceControl. Additionally, we have a research group codebase, available at https://github.com/vanderschaarlab/NeuralLaplaceControl.

**AISTATS 2022 Checklist** For all models and algorithms presented, check if you include:

1. A clear description of the mathematical setting, assumptions, algorithm, and/or model. (Yes, see Section 3.)
2. An analysis of the properties and complexity (time, space, sample size) of any algorithm. (Yes, see Section 5.2 and Appendix I.)
3. (Optional) Anonymized source code, with specification of all dependencies, including external libraries. (https://github.com/samholt/NeuralLaplaceControl.)

For any theoretical claim, check if you include:

1. A statement of the result. (Not applicable.)
2. A clear explanation of any assumptions. (Not applicable.)
3. A complete proof of the claim. (Not applicable.)

For all figures and tables that present empirical results, check if you include:

1. A complete description of the data collection process, including sample size. (Yes, see Section 5 and Appendix G.)
2. A link to a downloadable version of the dataset or simulation environment. (Yes, see Appendix D.)
3. An explanation of any data that were excluded, description of any pre-processing step. (Yes, see Appendix G.)
4. An explanation of how samples were allocated for training / validation / testing. (Yes, see Appendix G.)
5. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results. (Yes, see Appendix E.)
6. The exact number of evaluation runs. (Yes, see Section 5.)
7. A description of how experiments were run. (Yes, see Section 5 and Appendix F.)
8. A clear definition of the specific measure or statistics used to report results. (Yes, see Section 5 and Appendix F.)
9. Clearly defined error bars. (Yes, see Section 5 and Appendix F.)
10. A description of results with central tendency (e.g., mean) & variation (e.g., stddev). (Yes, see Section 5, and results throughout.)
11. A description of the computing infrastructure used. (Yes, see Section 5.2. and Appendix F.)

# A EXTENDED RELATED WORK

Table 3: Comparison with related model-based approaches to RL. **(P1) Learn from irregular samples**—can it learn from an offline dataset sampled at irregular times, $\Delta_i \neq \Delta_j$? **(P2) Learn delayed dynamics**—can it learn environments that contain a delay $\tau > 0$? Neural Laplace Control is the only method that can both learn from irregular samples (P1) as well as learn environments that contain a delay (P2).

| Approach | True Dynamics | Data Available | Reference | Model | (P1) $\Delta_i \neq \Delta_j$ | (P2) $\tau > 0$ |
|---|---|---|---|---|---|---|
| Conventional model-based RL | $\boldsymbol{x}_{t+1} \sim f(\boldsymbol{x}_t, \boldsymbol{a}_t)$ | $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{a}_i)\}_{i=0}^n$ | Williams et al. (2017) | MDP / Neural Network | ✗ | ✗ |
| Discrete-time delay methods | $\boldsymbol{x}_{t+1} \sim f(\boldsymbol{x}_t, \boldsymbol{a}_{t-\tau})$ | $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{a}_i)\}_{i=0}^n$ | Chen et al. (2021) | DA-MDP / RNN | ✗ | ✓ |
| Continuous-time methods | $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{a}(t))$ | $\mathcal{D} = \{(\boldsymbol{x}(t_i), \boldsymbol{a}(t_i))\}_{i=0}^n$ s.t. $\exists i,j : t_{i+1} - t_i \neq t_{j+1} - t_j$ | Yildiz et al. (2021) | Neural ODE | ✓ | ✗ |
| | | | Du et al. (2020) | Latent ODE | ✓ | ✗ |
| **Neural Laplace Control** | $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{a}(t-\tau))$ | $\mathcal{D} = \{(\boldsymbol{x}(t_i), \boldsymbol{a}(t_i))\}_{i=0}^n$ s.t. $\exists i,j : t_{i+1} - t_i \neq t_{j+1} - t_j$ | **(Ours)** | Neural Laplace Control | ✓ | ✓ |

In the following we summarize the key related work in Table 3 and provide an extended discussion of additional related works including a review of the benefits of using model-based RL and using model predictive control, which happens to be our preferred strategy for planning policies.

**Why model-based reinforcement learning?** Model-based RL holds the promise to enable creating policies for real world tasks in the offline setting where the true environment dynamics are *unknown*, and instead we require to learn a dynamics model from a dataset of demonstrations from agents acting within the environment. Although existing model-free RL approaches have shown effective performance (Fujimoto et al., 2018), they have the inherent disadvantages of being less sample efficient (Lutter et al., 2021) where they require interacting either online or with the *known* environment dynamics, and often require millions or billions of interactions with the environment to learn a good policy. Furthermore, learning a dynamics model of the environment, allows planning over that dynamics model to optimize actions (MPC) rather than learning a specific policy—by planning, a model can easily adapt to different goals or tasks at run-time. Whereas a policy trained for a specific task or goal would often have to be re-trained for a new task or goal, making it difficult for a policy to adapt to multi-task settings (Lutter et al., 2021). Naturally in continuous-control settings the dynamics model (e.g., the physics of the environment) is independent of the reward function (e.g., the goal state to reach), therefore changing tasks are straightforward by changing the reward function arbitrarily. An important property of model-based reinforcement learning is that in general it is more sample-efficient than model-free methods in conventional control tasks (Wang et al., 2019; Moerland et al., 2020). While model-free methods learn to master challenging tasks (Mnih et al., 2015; Lillicrap et al., 2015) and improves learning efficiency in high-dimensional continuous control tasks (Fujimoto et al., 2018; Haarnoja et al., 2018), it was later shown in Ha & Schmidhuber (2018); Janner et al. (2019) that model-based methods have much higher sample efficiency once properly tuned. Furthermore, Hafner et al. (2019); Sharma et al. (2019) propose to learn dynamics in a latent space, and similar insights have been applied to further improve the model-based reinforcement learning performance (Hansen et al., 2022).

In offline reinforcement learning, an agent learns from a fixed replay buffer and is not permitted to interact with the environment (Wu et al., 2019). While both model-free (Kumar et al., 2019; 2020; Fujimoto & Gu, 2021) and model-based (Kidambi et al., 2020; Wang et al., 2021) approaches have been proposed for offline RL, in general, model-based methods have been shown to be more sample efficient than model-free methods (Moerland et al., 2020). The main challenge in model-based RL is known as "extrapolation error" (Fujimoto et al., 2019), whereby the learnt dynamics model inaccuracies compound for a larger number of future predicted time steps. Hence, it is crucial in model-based RL to learn an appropriate dynamics model that is capable of accurately capturing the unique characteristics of an environment. However, even though many environments operate in continuous-time by nature and contain action or observation delays, almost all the existing approaches to model-based RL consider dynamics models only suited to the conventional discrete-time $\Delta_i = \Delta_j$ setting with no delays $\tau = 0$. We review here some of the few approaches that go beyond the conventional setting, namely (i) *discrete-time delay methods* and (ii) *continuous-time methods*.

**Discrete-time delay methods** One approach for handling environment observation delays is to increase the time step till the next action is performed, that is to synchronize an agent's actions with its delayed observations. However, such an approach is infeasible in most environments (for example dynamics involving momentum), and even when it is feasible, such a "wait agent" is often sub optimal, as it is possible to perform better by acting before receiving the most recent observation (Walsh et al., 2009). We note that modeling environments with either delayed observations $\boldsymbol{x}(t+\tau)$ or delayed actions $\boldsymbol{a}(t+\tau)$ are equivalent in form (Katsikopoulos & Engelbrecht, 2003). Prior work models regular sampled $\Delta_i = \Delta_j$ (discrete time) environments with constant time delays $\tau > 0$, and provides the agent with the current state $\boldsymbol{x}(t)$, and a

history of past actions performed in the environment $\bar{\boldsymbol{a}}_{i-1} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}\}$, whereby the history action window is larger than or equal to the observation or action delay in the environment (Walsh et al., 2009; Firoiu et al., 2018; Bouteiller et al., 2020). Recently, Chen et al. (2021) proposed delay-aware Markov decision processes (MDPs) that are capable of modeling delayed dynamics in discrete-time based on regularly sampled data, with an RNN encoding the history of past actions and the current state.

**Continuous-time methods** A standard approach for applying model-based RL to irregular sampled time series is to divide the timeline into equally sized intervals and impute or aggregate state and action tuples using averages (Rubanova et al., 2019b). Thus, turning a continuous-time environment into a discrete-time environment approximation; however, such pre-processing destroys information, particularly about the timing of measurements and the specific underlying environment dynamics. Real world data is often sampled irregularly $\Delta_i \neq \Delta_j$, as such (Yildiz et al., 2021) propose to use Neural-ODEs (Chen et al., 2018b) as their continuous-time dynamics model can model irregularly-sampled environments with no delays $\tau = 0$. Similarly, the work of Du et al. (2020) uses an a Latent-ODE model when planning policies. However, these existing approaches are limiting, as an ODE-based model by definition cannot handle a delay differential equation, necessitating the need for a model that can learn and model more diverse classes of differential equations. Recent models, of modeling diverse classes of differential equations is made possible with the work of Neural Laplace (Holt et al., 2022) by representing them in the Laplace domain. These Laplace-based models have been shown to be able to model such systems, be more accurate and scale better with increasing time horizons in time complexity. Our approach, namely Neural Laplace Control, essentially extends Neural Laplace to the setting of controlled systems—that is systems that evolve based on an action signal $\boldsymbol{a}(t)$—so that it can be used in planning policies in a RL setting.

**Model predictive control (MPC)** Principally relies on a good dynamics model, historically using simple first principle *known* dynamic models (Richalet et al., 1978; Salzmann et al., 2022). Recently, MPC Model Predictive Path Integral (MPPI) (Williams et al., 2017) is a zeroth order particle-based trajectory optimizer method that is capable of handling complex cost criteria and general nonlinear dynamics. Specifically, Williams et al. (2017) showed it could be used with a neural network learned dynamics model and used to drive a toy vehicle on a dirt track. MPC, and hence MPPI is often computationally infeasible for long time horizons, therefore often only being run for a fixed receding time horizon optimization into the future with a dynamics model. Using an MPC planner benefits from being able to handle arbitrary state constraints, and changing goals. Here MPPI is the state-of-the-art for MPC with a learned dynamics model, improving upon the previous cross-entropy method (CEM) (Kobilarov, 2012) MPC method. These naturally can incorporate new state-based constraints at run time.

**Hybrid MPC** Various ways of combining a powerful MPC planner with an accurate dynamics model is another fruitful thread (Argenson & Dulac-Arnold, 2020). All existing hybrid works, work only on discrete domains where demonstration data is collected on regular time intervals. These include MBOP (Argenson & Dulac-Arnold, 2020), TD-MPC (Hansen et al., 2022) and DADS (Sharma et al., 2019). We highlight that hybrid methods that plan in the latent space, i.e., TD-MPC and DADS are unable to incorporate state-based constraints. However, these methods still struggle with scaling MPC computational complexity forwards for longer time horizons.

**Control literature** We perform full system identification, i.e., learning the nonlinear dynamics model that has an *unknown* inherent delay. Whereas the existing control literature provides control algorithms for known forms (often linear) dynamics models for a *known* delay (Kwon et al., 2003; Raff et al., 2007)—therefore are not comparable. Moreover, there exists a wealth of orthogonal related work on stability analysis in Control (Åström & Murray, 2010).

**Learning from noisy demonstrations** It is preferable to learn a dynamics model on state-action trajectories that come from a "noisy" expert. As a noisy expert can provide better trajectories than an expert as it shows how to recover from "bad" states (Laskey et al., 2017)—specifically we assume the true expert is *unknown*. Moreover, as we only have access to trajectories from a noisy expert, performing imitation learning (Liotet et al., 2022) would propagate the noisy behavior, achieving a poor performance.

## B NEURAL LAPLACE BACKGROUND

In the following we provide a brief Laplace background, specifically from that of the Neural Laplace (Holt et al., 2022) model for modeling diverse differential equation (DE) systems—in the context of Neural Laplace Control. We defer the reader to the work of Holt et al. (2022) for a full comprehensive explanation of the original Neural Laplace model.

**States & actions** For a system with *state* space $\mathcal{X} = \mathbb{R}^{d_\mathcal{X}}$ and *action* space $\mathcal{A} = \mathbb{R}^{d_\mathcal{A}}$, the state at time $t \in \mathbb{R}$ is denoted as $\boldsymbol{x}(t) = [x_1(t), \ldots, x_{d_\mathcal{X}}(t)] \in \mathcal{X}$ and the action at time $t \in \mathbb{R}$ is denoted as $\boldsymbol{a}(t) = [a_1(t), \ldots, a_{d_\mathcal{A}}(t)] \in \mathcal{A}$. We elaborate that *state trajectory* $\boldsymbol{x} : \mathbb{R} \to \mathcal{X}$ and *action trajectory* $\boldsymbol{a} : \mathbb{R} \to \mathcal{A}$ are both functions of time, where an individual state $\boldsymbol{x}(t) \in \mathcal{X}$ or an individual action $\boldsymbol{a}(t) \in \mathcal{A}$ are points on these trajectories. Given a time interval $\mathcal{I} \subseteq \mathbb{R}$, $\boldsymbol{x}_\mathcal{I} \in \mathcal{X}^\mathcal{I}$ and $\boldsymbol{a}_\mathcal{I} \in \mathcal{A}^\mathcal{I}$ we denote the partial state and action trajectories on that interval such that $\boldsymbol{x}_\mathcal{I}(t) = \boldsymbol{x}(t)$ and $\boldsymbol{a}_\mathcal{I}(t) = \boldsymbol{a}(t)$ for $t \in \mathcal{I}$.

**Laplace Transform** The Laplace transform of a trajectory $\boldsymbol{x}$ is defined as (Schiff, 1999)

$$\boldsymbol{X}(\boldsymbol{s}) = \mathcal{L}\{\boldsymbol{x}\}(\boldsymbol{s}) = \int_0^\infty e^{-\boldsymbol{s}t}\boldsymbol{x}(t)dt, \tag{9}$$

where $\boldsymbol{s} \in \mathbb{C}^{d_\mathcal{X}}$ is a vector of *complex* numbers and $\boldsymbol{X}(\boldsymbol{s}) \in \mathbb{C}^{d_\mathcal{X}}$ is called the *Laplace representation*. The $\boldsymbol{X}(\boldsymbol{s})$ may have singularities, i.e., points where $\boldsymbol{X}(\boldsymbol{s}) \to \boldsymbol{\infty}$ for one component (Schiff, 1999). Importantly, the Laplace transform is well-defined for trajectories that are *piecewise continuous*, i.e., having a finite number of isolated and finite discontinuities (Poularikas, 2018). This property allows a learned Laplace representation to model a dynamics model that can have delay differential equation solutions (Holt et al., 2022).

**Inverse Laplace Transform** The inverse Laplace transform (ILT) is defined as

$$\hat{\boldsymbol{x}}(t) = \mathcal{L}^{-1}\{\boldsymbol{X}(\boldsymbol{s})\}(t) = \frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty} \boldsymbol{X}(\boldsymbol{s})e^{\boldsymbol{s}t}d\boldsymbol{s}, \tag{10}$$

where the integral refers to the Bromwich contour integral in $\mathbb{C}^{d_\mathcal{X}}$ with the contour $\sigma > 0$ chosen such that all the singularities of $\boldsymbol{X}(\boldsymbol{s})$ are to the left of it (Schiff, 1999). Many algorithms have been developed to numerically evaluate Equation 10. On a high level, they involve two steps: (Dubner & Abate, 1968; De Hoog et al., 1982; Kuhlman, 2013).

$$\mathcal{Q}(t) = \text{ILT-Query}(t) \tag{11}$$

$$\hat{\boldsymbol{x}}(t) = \text{ILT-Compute}\big(\{\boldsymbol{X}(\boldsymbol{s})|\boldsymbol{s} \in \mathcal{Q}(t)\}\big) \tag{12}$$

To evaluate $\boldsymbol{x}(t)$ on time points $t \in \mathcal{T} \subset \mathbb{R}_+$, the algorithms first construct a set of *query points* $\boldsymbol{s} \in \mathcal{Q}(\mathcal{T}) \subset \mathbb{C}$. They then compute $\hat{\boldsymbol{x}}(t)$ using the $\boldsymbol{X}(\boldsymbol{s})$ evaluated on these points. The number of query points scales *linearly* with the number of time points, i.e., $|\mathcal{Q}(\mathcal{T})| = d_\mathcal{S}|\mathcal{T}|$, where the constant $d_\mathcal{S} > 1$, denotes the number of reconstruction terms per time point and is specific to the algorithm. Importantly, the computation complexity of ILT only depends on the *number* of time points, but not their values (e.g., ILT for $t = 0$ and $t = 100$ requires the same amount of computation). The vast majority of ILT algorithms are differentiable with respect to $\boldsymbol{X}(\boldsymbol{s})$, which allows the gradients to be back propagated through the ILT transform (Holt et al., 2022).

Intuitively, the inverse Laplace transform (ILT) (Equation 10) reconstructs the dynamics model time solution with the basis functions of complex exponentials $e^{\boldsymbol{s}t}$, which exhibit a mixture of *sinusoidal* and *exponential* components (Schiff, 1999; Smith, 1997; Kuhlman, 2013).

**Solving control of differential equations in the Laplace domain** A key application of the Laplace transform is to solve broad classes of DEs (Podlubny, 1997; Yousef & Ismail, 2018; Yi et al., 2006; Kexue & Jigen, 2011). Due to the Laplace derivative theorem (Schiff, 1999), the Laplace transform can convert a DE into an *algebraic equation* even when the DE contains historical states $\boldsymbol{x}(t - \tau)$ (as in a delayed DE). It also applies to coupled DEs and can allow decoupled solutions to coupled DEs for dynamical systems (Åström & Murray, 2010). The resulting algebraic equation can either be solved analytically or numerically to obtain the solution of the DE, $\boldsymbol{X}(\boldsymbol{s})$, in the Laplace domain. Finally, one can obtain the time solution $\boldsymbol{x}(t)$ by applying the ILT on $\boldsymbol{X}(\boldsymbol{s})$. For instance, we could use the concise *Laplace transform method* to solve the (delay) differential equations to get solutions for the state trajectories conditioned on a control input trajectory (Yi et al., 2008).

**Stereographic projection** However, the Laplace representation $\boldsymbol{X}(\boldsymbol{s})$ often involves singularities (Schiff, 1999), which are difficult for neural networks to approximate or represent (Baker & Patil, 1998). We instead propose to use a stereographic projection $u(s) = (\theta, \phi)$ to translate any complex number $s \in \mathbb{C}$ into a coordinate on the Riemann Sphere

$(\theta, \phi) \in \mathcal{D} = (-\pi, \pi) \times (-\frac{\pi}{2}, \frac{\pi}{2})$ (Rudin, 1987), i.e.,

$$u(s) = \left( \arctan\left( \frac{\text{Im}(s)}{\text{Re}(s)} \right), \arcsin\left( \frac{|s|^2 - 1}{|s|^2 + 1} \right) \right) \tag{13}$$

Where the associated inverse transform, $u^{-1} : \mathcal{D} \to \mathbb{C}$, is given as

$$s = u^{-1}(\theta, \phi) = \tan\left( \frac{\phi}{2} + \frac{\pi}{4} \right) e^{i\theta} \tag{14}$$

A nice example of this map is the function of $1/s$, which corresponds to a rotation of the Riemann-sphere $180°$ about the real axis. Therefore, a representation of $1/s$ under this transformation becomes the map $\theta, \phi \mapsto -\theta, -\phi$ (Rudin, 1987).

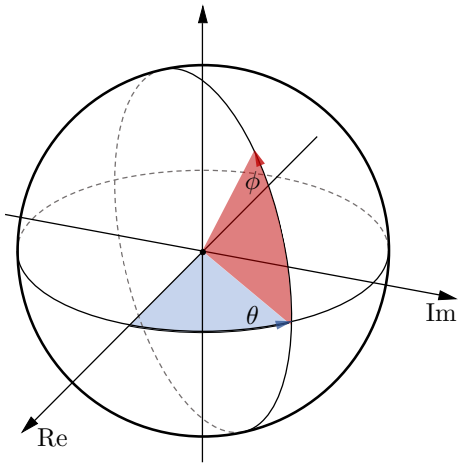


Figure 7: Geometry of the Riemann sphere map for a complex number $\mathbb{C}$ into a spherical co-ordinate representation of $\theta, \phi$.

**Inverse Laplace transform** After obtaining the Laplace representation $\boldsymbol{X}(\boldsymbol{s})$ from Equation 6 (see main paper), we compute the predicted or reconstructed state values $\hat{\boldsymbol{x}}(t)$ using the ILT. We highlight that we can evaluate $\hat{\boldsymbol{x}}(t)$ *at any future time* $t \in \mathbb{R}_+$ as the Laplace representation is independent of time once learnt. In practice, we use the well-known ILT Fourier series inverse algorithm (ILT-FSI), which can obtain the most general time solutions whilst remaining numerically stable (Dubner & Abate, 1968; De Hoog et al., 1982; Kuhlman, 2013). We use the specific ILT Fourier series algorithm from Holt et al. (2022) and use their code implementation of the ILT algorithm.

## C MPC MPPI PSEUDOCODE AND PLANNER IMPLEMENTATION DETAILS

We opt to use the model predictive controller of Model Predictive Path Integral (MPPI) (Williams et al., 2017). This uses a zeroth order particle-based trajectory optimizer method with our learned Laplace dynamics model. Specifically, this computes a discrete action sequence up to a fixed time horizon of $H \in \mathbb{R}_+$ seconds, and then executes the first element in the planned action sequence. Where we denote $\delta \in \mathbb{R}_+$ as the observation time interval, that is the time between two consecutive state observations. It is natural for online control problems to be controlled at discrete-time steps of $\delta$, where $\delta$ can be varied. Therefore, the MPPI plans actions at discrete-time steps $\delta$, up to a fixed time horizon $H$ by planning ahead $N \in \mathbb{Z}_+$ steps into the future, thus the planning time horizon is determined by $H = \delta \cdot N$. This leverages a number of parallel roll-outs $M \in \mathbb{Z}_+$, a hyper parameter, which can be tuned. As MPPI is a Monte Carlo based sampler, increasing the number of roll-outs improves the input trajectory optimization, however, scales the run-time complexity as $\mathcal{O}(NM)$.

We use the standard MMPI algorithm (Williams et al., 2017) with our dynamics model $F$, with the slight modification where we provide the dynamics model the current state, action and a buffer of previous actions back to $\omega$ seconds, i.e., $\boldsymbol{x}_{t+\delta} = F(\boldsymbol{x}_t, \boldsymbol{a}_{[t-\omega:t]}, \delta)$. For simplification of notation, as MPPI plans at discrete time steps of $\delta$ seconds, we relax the $\delta$ notation to discrete time steps of $\delta$, where $\delta = \bar{\Delta} = 0.05$ seconds at run-time. Specifically, for $\omega = 4\bar{\Delta} = 4\delta$ we denote the discrete multiple of $\delta$ as $\bar{\omega} = 4$, the next state estimate is given by $\boldsymbol{x}_{t+1} = F(\boldsymbol{x}_t, \boldsymbol{a}_{[t-\bar{\omega}:t]})$. Furthermore, MPPI requires us to keep in memory the global action trajectory $\boldsymbol{T} \in \mathbb{R}^{(N+\bar{\omega}) \times d_{\mathcal{A}}}$ buffer of the previously planned action

trajectory. With slight abuse of notation, we define the global action trajectory that is used to plan ahead $N$ discrete time steps to also contain the past $\bar{\omega}$ action histories, i.e., $\boldsymbol{T}_{[0-\bar{\omega}:N]}$. We detail the MPPI pseudocode with the high-level policy in Algorithm 1 and the MPPI action trajectory optimizer in Algorithm 2.

**Algorithm 1** High-Level MPPI Policy

**Input:** Pre-trained dynamics model $F$.
$\boldsymbol{T}^0 \leftarrow [\mathbf{0}_{0-\bar{\omega}}, \dots, \mathbf{0}_{N-1}]$ ▷ Initialize planned action trajectory.
**for** $t = 1 \dots \infty$ **do**
  $\boldsymbol{x}_t \leftarrow$ Observe $\boldsymbol{x}_t$
  $\boldsymbol{T}^t \leftarrow$ MPPI-Trajectory-Optimization$(F, \boldsymbol{x}_t, \boldsymbol{T}^{t-1})$ ▷ Update planned trajectory $\boldsymbol{T}^t$ starting with $\boldsymbol{T}_{[0-\bar{\omega}:0]}$.
  $\boldsymbol{a}_t \leftarrow \boldsymbol{T}_0^t$ ▷ Use first action $\boldsymbol{T}_0$ as $\pi(\boldsymbol{x}_t)$.
**end for**

**Algorithm 2** MPPI-Trajectory-Optimization

**Input:** Pre-trained dynamics model $F$, starting state $\boldsymbol{x}$, previous global action trajectory $\boldsymbol{T}$, steps to plan ahead for $N$, number of parallel roll-outs $M$, noise covariance $\boldsymbol{\Sigma}$, hyper parameters $\lambda$, action max $\boldsymbol{a}_{\max}$, action min $\boldsymbol{a}_{\min}$.
$\boldsymbol{R}_M \leftarrow \mathbf{0}_{\mathbf{M}}$ ▷ This holds our $M$ trajectory returns.
$\boldsymbol{A}_{M,N+\bar{\omega}} \leftarrow \mathbf{0}_{\mathbf{M},\mathbf{N}+\bar{\omega}}$ ▷ This holds our $M$ action trajectories of length $N$.
$\boldsymbol{A}'_{M,[0-\bar{\omega}:N-1]} \leftarrow \mathbf{0}_{\mathbf{M},\mathbf{N}+\bar{\omega}}$ ▷ This holds $M$ action trajectories of length $N$ that are perturbed by noise.
$\boldsymbol{\varepsilon}_{M,N+\bar{\omega}} \leftarrow \mathbf{0}_{\mathbf{M},\mathbf{N}+\bar{\omega}}$ ▷ This holds the generated scaled action noise.

$\boldsymbol{A}_{M,[0-\bar{\omega}:N-1]} \leftarrow \mathbf{T}_{[\mathbf{1}-\bar{\omega}:\mathbf{N}]}/\mathbf{a}_{\max}$ ▷ Broadcast previous scaled down action trajectory $\boldsymbol{T}$ to $M$ roll-outs.
**for** $m = 0 \dots M-1$ **do** ▷ Sample $M$ trajectories over the horizon $N$.
  $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$
  **for** $n = 0 \dots N-1$ **do**
    $\boldsymbol{\varepsilon}_{m,n} \leftarrow \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ ▷ Sample action noise.
    $\boldsymbol{A}'_{m,n} \leftarrow \boldsymbol{A}_{m,n} + \boldsymbol{\epsilon}_{m,n}$ ▷ Perturb action by noise.
    $\boldsymbol{A}'_{m,n} \leftarrow \min(\max(\boldsymbol{A}'_{m,n}, -1), +1)$ ▷ Clip normalized perturbed noise (to bound actions to their limits).
    $\boldsymbol{\varepsilon}_{m,n} \leftarrow \boldsymbol{A}'_{m,n} - \boldsymbol{A}_{m,n}$ ▷ Update noise after bounding, so we do not penalize clipped noise.
  **end for**
  **for** $n = 0 \dots N-1$ **do**
    $\boldsymbol{x}_{n+1} \leftarrow F(\boldsymbol{x}_n, \boldsymbol{A}'_{[n-\bar{\omega}:n]} \cdot \boldsymbol{a}_{\max})$ ▷ Sample next state from pre-trained dynamics model $F$.
    $\boldsymbol{R}_m \leftarrow \boldsymbol{R}_m + r(\boldsymbol{x}_n, \boldsymbol{A}_{m,n}) - \lambda \boldsymbol{A}_{m,n}^T \Sigma^{-1} \boldsymbol{\varepsilon}_{m,n}$ ▷ Accumulate the current state reward.
  **end for**
**end for**
$\kappa \leftarrow \min_m[\boldsymbol{R}_m]$
$\boldsymbol{T}'_n = \boldsymbol{T}_n + \frac{\sum_{m=0}^{M-1} \exp(\frac{1}{\lambda}(\boldsymbol{R}_m - \kappa))\boldsymbol{\varepsilon}_{m,n}}{\sum_{m=0}^{M-1}(\frac{1}{\lambda}(\boldsymbol{R}_m - \kappa))} \cdot \boldsymbol{a}_{\max}, \forall n \in [0, N-1]$ ▷ Generate the return-weighted trajectory update.
**Return:** $\boldsymbol{T}'$

# D ENVIRONMENT SELECTION AND DETAILS

In the following we discuss our reasoning for why we selected the delay $\tau > 0$ adapted continuous-time control environments from the ODE-RL suite (Yildiz et al., 2021) [5] and the reasoning behind our choice of sampling irregularly in time $\Delta_i \neq \Delta_j$ of state-action trajectories $(\boldsymbol{x}(t+\Delta_i), \boldsymbol{a}(t+\Delta_i))$ from these environments. We first outline why existing environment offline datasets are not suitable.

**Why we cannot use an offline dataset of agent trajectories in an un-delayed discrete-time environment** It is straightforward, and there exists standard datasets (Fu et al., 2020) of state-action trajectories of (expert) agents interacting with environments that have no delays $\tau = 0$ and are sampled at regular times $\Delta_i = \Delta_j$. In the following we outline why these datasets cannot be used:

[5] The ODE-RL suite of the environments used can be downloaded freely available from `https://github.com/cagatayyildiz/oderl`.

- **Presence of a constant delay $\tau > 0$ in the environment.** Intuitively one might suggest taking a standard dataset and shift the actions by a fixed delay. However, we note this dataset is then unrealistic, as it violates causal information—as a hypothetical action would know how its current action affected future states before it had even observed them. Due to this fact, this prevents us from using existing standard offline datasets (Fu et al., 2020). Thus, this motivates the need to sample agents that interact within an environment that has an inherent *delay* of a constant delayed action (or constant delayed state observation).
- **Presence of irregularly sampled in time $\Delta_i \neq \Delta_j$ state-action trajectories.** Let us hypothetically imagine that there exists a regularly-sampled $\Delta_i = \Delta_j$ in time state-action trajectory offline dataset from an environment that has an inherent constant action delay $\tau > 0$. Can we then sample them irregularly? One could suppose that we use some form of interpolation (e.g., splines or similar) to interpolate to irregular time steps between states and actions, however doing so would lead to errors in the sampled state-action trajectories in comparison to the true irregularly-sampled state-action trajectories at those non-uniform time points. These errors could compound over a dynamics model being trained on these; therefore, we highlight that this approach is unsuitable. Another approach would be to start with a regularly sampled state-action trajectory $t \in \mathcal{T} \in \{0, \bar{\Delta}, 2\bar{\Delta}, \dots, N\bar{\Delta}\}$ then sub-sample state-action times from that regular grid of collected times $t \subset \mathcal{T}$. Again we indicate this approach unsuitable, as often environments are captured at run-time with a particular observation interval $\bar{\Delta}$ seconds, and only observing multiples of this would mean gaps between observations, where the mean of the observation intervals is larger than that of the environments nominal run-time observation interval $\bar{\Delta}_{\text{Sub-sample}} > \bar{\Delta}_{\text{Original Trajectory}}$. We note that this becomes a different problem, and as there is less information in the state-action trajectories with large observation interval gaps. Instead to mitigate both of these issues we prefer to collect an offline dataset ourselves of an agent interacting with the delay environments with true irregular observation intervals, where we sample the time interval to the next observation from an exponential distribution, i.e., $\Delta \sim \text{Exp}(\bar{\Delta})$, with a mean of $\bar{\Delta} = 0.05$ seconds.

Given the above reasoning, we take the approach to collect offline datasets of a noisy agent interacting with the delay environments, where the observations occur at irregular time intervals given by $\Delta \sim \text{Exp}(\bar{\Delta})$, with a mean of $\bar{\Delta} = 0.05$ seconds. We note this approach is similar to Fu et al. (2020) and provides a more realistic offline dataset to train our dynamics models on.

We use the continuous-time control environments from the ODE-RL suite (Yildiz et al., 2021), as they provide true irregular samples in time of state observations and are fully continuous in time, unlike discrete environments (Brockman et al., 2016). We adapt these to incorporate an arbitrary fixed delayed action time, turning the ODE environments into delay DE environments. We do this by keeping a buffer of the previous actions that have been produced by the policy, which captures the previous actions generated in the past $\omega = 4\bar{\Delta}$ seconds. In practice this buffer is 4 action elements long and is fed into the dynamics model in its entirety. The true environment then executes the action at the previous time of the constant action delay, which is one of $\tau = \{0, \bar{\Delta}, 2\bar{\Delta}, 3\bar{\Delta}\}$ and is *unknown* to the dynamics model. Thus, the dynamics model sees the entire action buffer and must instead learn implicitly the delay of the environment by modeling the dynamics of the environment accurately.

The starting state for all tasks is hanging down and the goal is to swing up and stabilize the pole(s) upright (Yildiz et al., 2021) in each environment. In all environments the actions are continuous and bounded to a defined range $[\boldsymbol{a}_{\min}, \boldsymbol{a}_{\max}$. Here we assume a given state $\boldsymbol{x}(t)$ is composed of the position state $\boldsymbol{q}$ and their respective velocities $\dot{\boldsymbol{q}}$, i.e., $\boldsymbol{x}(t) = \{\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)\}$. Here, each environment uses the reward function of the exponential of the negative distance from the current state to the goal state $\boldsymbol{q}^*$, whilst also penalizing the magnitude of action, and we assume that we are given this reward function when planning—as we often know the desired goal state $\boldsymbol{q}^*$ and our current state $\boldsymbol{q}$. Therefore, the reward function for the environments has the following form:

$$r(\{\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)\}, \boldsymbol{a}(t)) = \exp(-||\boldsymbol{q}(t) - \boldsymbol{q}^*||_2^2 - b||\dot{\boldsymbol{q}}(t)||_2^2) - c||\boldsymbol{a}(t)||_2^2 \tag{15}$$

Where $b$ and $c$ are specific environment constants (Yildiz et al., 2021). Specifically, when we use our MPC planner we observe that it plans better without the exponential operator, therefore remove it, and use the following reward function throughout, $r(\{\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)\}, \boldsymbol{a}(t)) = -||\boldsymbol{q}(t) - \boldsymbol{q}^*||_2^2 - b||\dot{\boldsymbol{q}}(t)||_2^2 - c||\boldsymbol{a}(t)||_2^2$. Yildiz et al. (2021) set the environments parameters of $b, c$ to penalize large values and enforce exploration from trivial states, and we use their same values which are also tabulated in Table 4.

The goal states for all environments is when the poles, each of length $L$ are fully upright, such that their $x, y$ co-ordinates of the tip of the pole reach the goal state. Where $\boldsymbol{q}^*$ is: $[0, L]$ for the Pendulum environment, $[0, 0, L]$ for the Cartpole environment—where the additional 0 is zero for the cart's $x$ location and in Acrobot is $[0, 2L]$ as there are two poles

Table 4: Environment specification parameters of the ODE-RL suite (Yildiz et al., 2021).

| Base Environment | $b$ | $c$ | $\boldsymbol{a}_{\max}$ | $\boldsymbol{x}_{\text{Init}}$ | $\boldsymbol{q}^*$ |
|---|---|---|---|---|---|
| Pendulum | $1e-2$ | $1e-2$ | $[2]$ | $[0.1, 0.1]$ | $[0, L]$ |
| Cartpole | $1e-2$ | $1e-2$ | $[3]$ | $[0.05, 0.05, 0.05, 0.05]$ | $[0, 0, L]$ |
| Acrobot | $1e-4$ | $1e-2$ | $[4, 4]$ | $[0.1, 0.1, 0.1, 0.1]$ | $[0, 2L]$ |

connected to each other. Furthermore, upon restarting the environment the initial state $x$ is sampled from the uniform distribution of $x_0 \sim \mathcal{U}[-\boldsymbol{x}_{\text{Init}}, \boldsymbol{x}_{\text{Init}}]$ (Yildiz et al., 2021), then the $\theta$ states are added with set angle such that the pole(s) are pointing downwards (i.e., Cartpole $\theta' = \theta_{\text{Init}} + \pi$).

We note that existing offline RL methods have only been developed for discrete-time $\Delta_i = \Delta_j$ settings and environments that do not possess a delay dynamics $\tau = 0$ (Argenson & Dulac-Arnold, 2020), therefore they are not applicable—instead we opt to train an environment dynamics model and use a planner to select the next action to take.

In the following we describe each of our environments introducing the vanilla environment with a screenshot figure.

### D.1 Cartpole (swing up) Environment

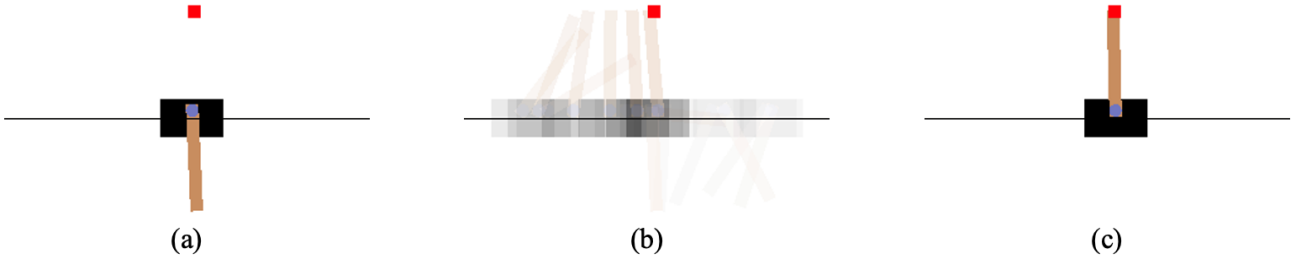


Figure 8: Screen shots of the Cartpole environment. The task is to swing up a pole attached to a cart that can move horizontally along a rail. In the following we see: (a) the starting downward state with an additional small amount of perturbation, (b) the optimal trajectory solution found by a policy that scores $\mathcal{R} = 100\%$ including our NLC model and an expert + MPC and (c) the final goal state that has been reached, that is, to swing up the pole and stabilize it upwards—which is a challenging control task. We note that the control actuator is bounded and limited, and the force is such that the Cartpole cannot be directly swung up—rather it must gain momentum through a swing and then stabilize this swing to not overshoot when stabilizing the pole upwards in the goal position, as indicated when the tip of the pole reaches the centre of the red target square. Furthermore, we note this environment is an underactuated system.

We can see in Figure 8, an illustration of the starting state Figure 8 (a) with a small perturbed random initial start. Here a pole is attached to an un-actuated joint to a cart that moves along a frictionless track (Barto et al., 1983). The pendulum starts in the downward position Figure 8 (a) and the goal is to swing the pendulum upwards and then balance the pole upright by applying forces to the left or right horizontal direction of the cart. This environment has the state of $[x, \dot{x}, \theta, \dot{\theta}]$ and a corresponding observation of $[x, \dot{x}, \cos(\theta), \sin(\theta), \dot{\theta}]$, where $\theta \in (-\pi, \pi)$ is measured from the upward vertical of the pole. We note that this environment is an underactuated system, as it has two degrees of freedom $[x, \theta]$, however only the carts position is actuated, leaving $\theta$ indirectly controlled.

### D.2 Pendulum Environment

We can see in Figure 9, an illustration of the starting state Figure 9 (a) with a small perturbed random initial start. Here a pole (pendulum) is attached to a fixed point at one end with the other end being free (Barto et al., 1983; Yildiz et al., 2021). The pendulum starts in the downward position Figure 9 (a) and the goal is to swing the pendulum upwards and then balance the pole upright by applying torques about the fixed point, as indicated in the Figure 9 with a visualisation showing the torque direction and magnitude based on the size of the arrow. This environment has the state of $[\theta, \dot{\theta}]$ and a corresponding observation of $[\sin(\theta), \cos(\theta), \dot{\theta}]$.

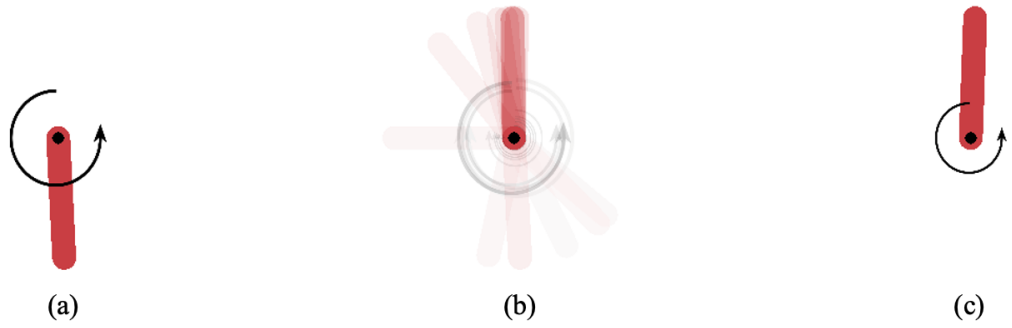


Figure 9: Screen shots of the Pendulum environment. The task is to swing up the pole (pendulum). In the following we see: (a) starting downward state with an additional small amount of perturbation, (b) the optimal trajectory solution found by a policy that scores $\mathcal{R} = 100\%$ including our NLC model and an expert + MPC and (c) the final goal state that has been reached, that is, to swing up the pole and stabilize it upwards. We note that the control actuator is bounded and limited, and the force is such that the Pendulum cannot be directly swung up—rather it must gain momentum through a swing and then stabilize this swing to not overshoot when stabilizing the pole upwards in the goal position.

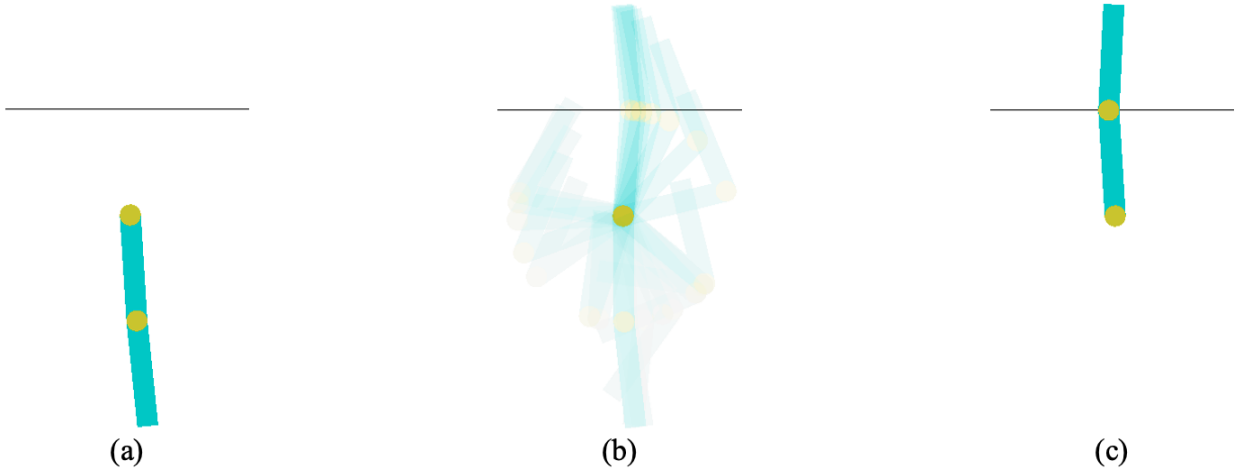


Figure 10: Screen shots of the Acrobot environment. The task is to swing up the 2-link pendulum. In the following we see: (a) starting downward state with an additional small amount of perturbation, (b) the optimal trajectory solution found by a policy that scores $\mathcal{R} = 100\%$ including our NLC model and an expert + MPC and (c) the final goal state that has been reached, that is, to swing up the 2-link pendulum and stabilize it upwards. We note that the control actuator is bounded and limited, and the force is such that the 2-link pendulum cannot be directly swung up—rather it must gain momentum through a 2-link swing and then stabilize this swing to not overshoot when stabilizing the 2-link pendulum upwards in the goal position.

### D.3 Acrobot Environment

We can see in Figure 10, an illustration of the starting state Figure 10 (a) with a small perturbed random initial start. It is a 2-link pendulum with the individual joints actuated (Brockman et al., 2016). The 2-link pole starts in the downward position Figure 10 (a) and the goal is to swing the 2-link pendulum upwards and then balance the pole(s) upright by applying torques about their fixed points. This environment has the state of $[\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2]$ and a corresponding observation of $[\sin(\theta_1), \cos(\theta_1), \dot{\theta}_1, \sin(\theta_2), \cos(\theta_2), \dot{\theta}_2]$. Here the Acrobot environment is fully actuated, as no method has been able to solve the underactuated balancing problem (Yildiz et al., 2021; Zhong et al., 2019).

## E BENCHMARK METHOD IMPLEMENTATION DETAILS

We tuned all the baseline dynamics models to have the same approximate number of parameters, as can be seen in Table 5—to ensure fair comparison for any gains in modeling complexity. We train and evaluate all the dynamics models and all data used with double point floating precision, as is recommended when using an inverse Laplace transform (ILT) to aid the ILT stability (Holt et al., 2022; Kuhlman, 2013). Further all dynamics models are implemented in PyTorch (Paszke et al.,

Table 5: Benchmark dynamic models implemented and their number of parameters for each model.

| Dynamics model | # Parameters |
|---|---|
| $\Delta t-$RNN | 79,075 |
| NODE | 76,956 |
| Latent-ODE | 76,453 |
| Neural Laplace Control | 81,772 |

2019), and trained with an Adam optimizer (Kingma & Ba, 2017) with a learning rate of 1e-4. Each baseline dynamics model are:

**Discrete-delay method** We implemented the discrete-delay method similar to Chen et al. (2021), a RNN over the action buffer and current state, and adapt it to model continuous-time with a new input of the time increment to predict the next state for ($\Delta t-$**RNN**). We note to adapt discrete-time models to continuous-time we add an additional input parameter, that of the time difference between the current time and the next state observation to predict, i.e., $\delta$, e.g., $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{a}_i, \delta)$. (Yildiz et al., 2021). Specifically, we feed the action buffer $\mathcal{H}_i$ into a gated recurrent neural (GRU) network with a hidden size of 160 features in the hidden state and feed the final hidden state and concatenate it with the current observed state and time increment to predict the next state which is all input into a linear layer to produce the output state prediction.

**Continuous-time methods** We implemented an augmented Neural-ODE (**NODE**) (Chen et al., 2018b), using their code and corresponding implementation provided, and set their ODE function $\boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{a}(t))$ to be a 3-layer multilayer perceptron (MLP), of 270 units, with $\tanh$ activation functions—with an additional augmented dimension of zeros. As neural-ODE does not have an encoder, we feed the most recent action taken at time $t$, i.e., $\boldsymbol{a}(t)$ into the MLP $\boldsymbol{f}$ function instead. Further, to allow for fair comparison we use the semi-norm trick for faster back propagation Kidger et al. (2020a), and use the 'euler' solver throughout. We also use the reconstruction MSE for training.

We also compare with **Latent-ODE** (Rubanova et al., 2019a), which uses an ODE-RNN encoder and an ODE model decoder. We feed this the action history buffer $\mathcal{H}_i$ concatenated with an equivalent state history buffer $\mathcal{H}'_i = \{(\boldsymbol{x}_j, t_j - t_i) : t_j \in [t_i - \omega, t_i]\}$ for the same sample times. We use their code provided, setting the units to be 128 for the GRU and ODE function $\boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{a}(t))$ net, with $\tanh$ activation functions and use the 'dopri5' solver. We also use their reconstruction variational loss function for training.

**Neural Laplace Control** This paper uses a GRU encoder $h_\zeta$ (Cho et al., 2014), to encode the action buffer $\mathcal{H}_i$ with 2 layers and a hidden size of 64 features in the hidden state, with a final linear layer on the final hidden to output $\boldsymbol{p}_i^{(\mathcal{A})}$. We do not encode the state and instead feed it directly as $\boldsymbol{p}_i^{(\mathcal{X})} = \boldsymbol{x}_i$. Therefore, we encode both into a latent dimension $\boldsymbol{p}_i = (\boldsymbol{p}_i^{(\mathcal{A})}, \boldsymbol{p}_i^{(\mathcal{X})})$. For the Laplace representation model, $g_\psi$ we use a 3-layer MLP with 128 units, with $\tanh$ activations. As recommended by Holt et al. (2022) we further use a $\tanh$ on the output to constrain the output domain to be $(\theta, \phi) \in \mathcal{D} = (-\pi, \pi) \times (-\frac{\pi}{2}, \frac{\pi}{2})$ for each output state prediction. For a given state prediction, we encode the action buffer and state into $\boldsymbol{p}_i$ and concatenate this with $u(\boldsymbol{s})$ as the input to $g_\psi$, i.e., $g_\psi\big(\boldsymbol{p}, u(\boldsymbol{s})\big)$, where $\boldsymbol{s}$ is given by the ILT algorithm for a future time point to predict the state for. Specifically, we use the Fourier-series inverse algorithm (ILT-FSI), with $d_{\mathcal{S}} = 17$ reconstruction terms. Where we use the specific ILT-FSI from Holt et al. (2022) and use their code implementation of the ILT algorithm.

**MPPI Implementation** We use the MPPI algorithm, with pseudocode and is further described in Appendix C. Specifically, as is recommended by Lutter et al. (2021) we optimized the MPPI hyperparameters through a grid search with the true (Oracle) dynamics model for a single environment setting, that of the Cartpole environment with a delay of $\tau = \bar{\Delta} = 0.05$ seconds, and fix these for planning with all the learned dynamics models throughout. Specifically, our final optimized hyperparameter combination is $N = 40, M = 1,000, \lambda = 1.0, \sigma = 1.0$. Where $\Sigma$ the MPPI action noise is defined as:

$$\Sigma = \begin{cases} [\sigma^2] & \text{if } d_{\mathcal{A}} = 1 \\ \begin{bmatrix} \sigma^2 & 0.5\sigma^2 \\ 0.5\sigma^2 & \sigma^2 \end{bmatrix} & \text{if } d_{\mathcal{A}} = 2 \end{cases} \tag{16}$$

Where the Cartpole and Pendulum environments have $d_{\mathcal{A}} = 1$ and Acrobot environment has $d_{\mathcal{A}} = 2$. These hyperparameters were found by searching over a grid of possible values, which is detailed in Table 6.

Table 6: MPPI hyperparameter grid search sweep values.

| Hyperparameter | Grid values searched over |
|---|---|
| $N$ | $\{1, 2, 4, 8, 16, 20, 40, 50, 60, 70, 80, 90, 100, 128, 256, 512, 1024, 2048\}$ |
| $M$ | $\{1, 2, 4, 8, 16, 20, 40, 50, 60, 70, 80, 90, 100, 128, 256, 500, 1000, 2000, 4000, 8000\}$ |
| $\lambda$ | $\{0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 0.8, 1.0, 1.5, 2.0, 10.0, 100.0, 1000.0\}$ |
| $\sigma$ | $\{0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 0.8, 1.0, 1.5, 2.0, 10.0, 100.0, 1000.0\}$ |

# F EVALUATION METRICS

For each environment, with a different delay setting we collect an offline dataset of irregularly-sampled in time trajectories, consisting of 1e6 samples from the "noisy expert" agent interacting within that environment (Section 5 & Appendix G). For each benchmark dynamics model, we follow the same two step evaluation process of, firstly, training the dynamics model on that environment's collected offline dataset using a MSE error loss for the next step ahead state prediction $\hat{\boldsymbol{x}}(t_{i+1})$. Then, secondly, taking the same pre-trained model and freezing the weights, and only using it for planning with the MPPI (MPC) planner at run-time in an environment episode, that lasts for 10 seconds. In total, we evaluate our model-based control algorithms online in the same environment, running each one for a fixed observation interval of $\delta = \bar{\Delta} = 0.05$ seconds (as is the nominal value for these environments (Yildiz et al., 2021; Brockman et al., 2016), unless specified otherwise), and take the cumulative reward value after running one episode of the planner (policy) and repeat this for 20 random seed runs for each result. We quote the normalized score $\mathcal{R}$ (Yu et al., 2020) of the policy in the environment, averaged over the 20 random seed run episodes, with standard deviations throughout. The scores are un-discounted cumulative rewards normalized to lie roughly between 0 and 100, where a score of 0 corresponds to a random policy, and 100 corresponds to an expert (oracle with a MPC planner). Specifically, when we evaluate the insights experiments in Section 5.2, where we change the planning observation interval $\delta$, this changes the number of steps taken in an episode, therefore we quote the un-discounted average rewards normalized to lie roughly between 0 and 100, where a score of 0 corresponds to a random policy, and 100 corresponds to an expert (oracle with a MPC planner). We note that a normalized cumulative reward for an episode and an average reward for an episode for the same number of steps in an episode are equivalent. Furthermore, we also track the metric of total planning time taken to plan the next action as $\mathcal{O}$ seconds and perform all experiments using a single Intel Core i9-12900K CPU @ 3.20GHz, 64GB RAM with a Nvidia RTX3090 GPU 24GB.

# G DATASET GENERATION AND MODEL TRAINING

For each environment we generate an offline state-action trajectory dataset by using an agent that uses an oracle dynamics model combined with MPC and has additional noise added to the agents selected action, $\bar{\pi}(t) = \pi(t) + \epsilon, \epsilon \sim \mathcal{N}(0, \boldsymbol{a}_{\max})$. This "noisy expert" agent interacts with the environment and observes observations at irregular unknown times, where we sample the time interval to the next observation from an exponential distribution, i.e., $\Delta \sim \text{Exp}(\bar{\Delta})$, with a mean of $\bar{\Delta} = 0.05$ seconds. We note other irregular sampling types are possible, however Yildiz et al. (2021) has shown they are approximately equivalent. We assume a fixed action delay $\tau$, and evaluate discrete multiples of this delay of the mean sampling time $\bar{\Delta}$, i.e., $\tau = \bar{\Delta}$ for one step delay, $\tau = 2\bar{\Delta}$ for two step delay etc. We enforce the observed action history buffer that includes past actions back to $\omega = 4\bar{\Delta}$ seconds. For each specific delay version of each base environment class (Cartpole, Pendulum and Acrobot) we collect a total of 1e6 irregular state-action samples in time (unless otherwise specified). Using the whole collected dataset we pre-process this by a standardization step, to make each dimension of the samples have zero mean and unit variance (by taking away the mean for each dimension and then dividing by the standard deviation for each dimension)—we also use this step during run-time for each dynamics model. Furthermore, we train all the baseline models on all the samples collected in the offline dataset (all samples are training data), training the models parameters with the Adam optimizer (Kingma & Ba, 2017) with a learning rate of 1e-4 throughout. Specifically, we train all the baseline dynamics models on a given offline dataset by training each model for 2 hours and 15 minutes. We also ran a further experiment where we trained all the models for a fixed number of epochs instead, detailed further in Appendix J.1. For the insights experiments in Section 5.2 that quote a validation dataset error, we achieve this by generating a new offline dataset using the same setup described above with a different random seed to use as a validation dataset.

# H RAW RESULTS

The full results from Table 2, are in Table 7 along with their un-normalized versions in Table 8.

**High variance in Latent-ODE and NODE** We detail their poor performance to: (1) both these models do not support batches of trajectories evaluated at different non-uniform time points—therefore, they are trained with a batch size of 1, (2) we only train a single dynamics model, unlike other works (Yildiz et al., 2021) that train an ensemble of models, and (3) Latent-ODE is trained using the recommended variational loss of the next step ahead prediction.

Table 7: Normalized scores $\mathcal{R}$ of the offline model-based agents, where the irregularly-sampled (P1) offline dataset consists of an action delay (P2) of $\{0, 1, 2, 3\}$ multiples of the environments observation interval time step $\bar{\Delta} = 0.05$ seconds. Averaged over 20 random seeds, with $\pm$ standard deviations. Scores are un-discounted cumulative rewards normalized to be between 0 and 100, where 0 corresponds to the Random agent and 100 corresponds to the expert with the *known* world model (oracle+MPC). Negative normalized scores, i.e., worse than random are set to zero.

| | Action Delay $\tau = 0$ | | | Action Delay $\tau = \bar{\Delta}$ | | | Action Delay $\tau = 2\bar{\Delta}$ | | | Action Delay $\tau = 3\bar{\Delta}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dynamics Model | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot |
| Random | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |
| Oracle | 100.0±0.04 | 100.0±3.34 | 100.0±1.84 | 100.0±0.15 | 100.0±3.14 | 100.0±2.19 | 100.0±0.04 | 100.0±2.57 | 100.0±1.79 | 100.0±0.08 | 100.0±2.57 | 100.0±1.26 |
| $\Delta t-$RNN | 96.76±0.34 | 32.73±7.09 | 12.61±4.65 | 95.28±0.4 | 1.14±6.31 | 18.95±7.6 | 97.01±0.31 | 9.94±2.48 | 28.39±9.73 | 97.8±0.25 | 11.81±11.93 | 3.89±6.72 |
| Latent-ODE | 0.0±0.0 | 8.08±8.45 | 4.45±8.81 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 1.24±20.67 | 8.91±13.62 | 41.56±47.07 | 3.26±12.24 | 9.19±9.08 |
| NODE | 70.08±2.94 | 12.89±4.19 | 0.0±0.0 | 85.09±7.95 | 0.63±5.16 | 23.07±6.94 | 90.75±1.34 | 0.0±0.0 | 10.92±10.09 | 94.55±1.08 | 1.97±4.01 | 11.78±8.33 |
| **NLC (Ours)** | **99.87±0.1** | **101.52±3.3** | **99.16±1.91** | **99.83±0.19** | **98.31±3.51** | **99.12±1.7** | **99.88±0.1** | **93.28±4.96** | **100.44±2.13** | **99.92±0.12** | **98.98±1.32** | **99.46±1.88** |

Table 8: Un-normalized scores $\mathcal{R}$ of the offline model-based agents, where the irregularly-sampled (P1) offline dataset consists of an action delay (P2) of $\{0, 1, 2, 3\}$ multiples of the environments observation interval time step $\bar{\Delta} = 0.05$ seconds. Averaged over 20 random seeds, with $\pm$ standard deviations. Scores are un-discounted cumulative rewards.

| | Action Delay $\tau = 0$ | | | Action Delay $\tau = \bar{\Delta}$ | | | Action Delay $\tau = 2\bar{\Delta}$ | | | Action Delay $\tau = 3\bar{\Delta}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dynamics Model | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot |
| Random | -14246.3±0.0 | -616.77±0.0 | -2948.64±0.0 | -9713.19±0.0 | -575.98±0.0 | -2910.5±0.0 | -15097.54±0.0 | -584.8±0.0 | -2938.69±0.0 | -20798.89±0.0 | -596.38±0.0 | -2885.99±0.0 |
| Oracle | -139.69±5.27 | -121.05±16.54 | -571.11±43.7 | -146.26±13.95 | -123.44±14.2 | -558.76±51.41 | -145.56±5.26 | -125.08±11.79 | -582.01±42.1 | -153.19±16.72 | -133.51±11.88 | -588.61±28.84 |
| $\Delta t-$RNN | -597.21±47.95 | -454.51±35.13 | -2648.73±110.56 | -598.27±37.96 | -570.82±28.57 | -2464.87±178.77 | -592.66±46.17 | -539.12±11.39 | -2269.64±229.3 | -607.96±51.11 | -541.71±55.22 | -2796.69±154.29 |
| Latent-ODE | -152488.09±89951.95 | -576.69±41.91 | -2842.9±209.42 | -814206.21±292756.78 | -696.61±71.72 | -2911.94±263.25 | -1539109.96±288324.7 | -579.11±95.01 | -2728.8±320.88 | -12217.77±9717.18 | -581.29±56.67 | -2674.88±208.65 |
| NODE | -4359.77±415.24 | -552.86±20.78 | -3048.13±79.08 | -1572.47±760.41 | -573.12±23.33 | -2368.03±163.25 | -1528.85±200.28 | -613.56±16.01 | -2681.39±237.74 | -1277.55±222.61 | -587.27±18.57 | -2615.45±191.46 |
| **NLC (Ours)** | **-158.31±14.66** | **-113.52±16.35** | **-591.16±45.46** | **-162.47±18.31** | **-131.07±15.9** | **-579.39±39.92** | **-163.0±14.67** | **-155.96±22.82** | **-571.64±50.1** | **-170.17±25.59** | **-138.23±6.13** | **-601.04±43.17** |

# I INSIGHT EXPERIMENTS

In this section we seek to gain further insight into *how* Neural Laplace Control outperforms the benchmarks. In the following we seek to understand if NLC is able to learn from irregularly-sampled state-action offline datasets (P1), whilst learning the delayed dynamics of the environment (P2). Furthermore, we also explore the benefits of the NLC approach for planning at longer time horizons with a fixed amount of compute and being sample efficient.

## I.1 Can the baseline dynamics models learn a good model?

To explore if NLC is able to learn a suitable dynamics model, we plot the trained dynamics models next step ahead prediction error with that of the ground truth for a varying observation interval $\delta$ for the Cartpole environment with a delay of $\bar{\Delta}$, as shown in Figure 11. Empirically we observe that NLC using its Laplace-based dynamics model is able to better approximate a wider range of observation intervals $\delta$ and achieve a good *global* approximation compared to the recurrent neural network and ODE based models. We note that due to the offline dataset being sampled with trajectories that have irregular sampling times (P1), where the sampling times are defined by an exponential distribution with a mean of $\bar{\Delta} = 0.05$ seconds; the other competing methods seem to over-fit purely to the median sample time of the exponential distribution, i.e., $0.05 \cdot \ln(2) = 0.034$ s. Other works have shown a more accurate next step prediction model correlates to a higher environment episode reward (Williams et al., 2017).

## I.2 Can the baseline dynamics models learn delay environment dynamics?

To investigate this, we similarly plot the trained NLC dynamics models next step ahead prediction error with that of the ground truth for a varying observation interval $\delta$, for each of the delayed environment versions of the specific Cartpole environment, as show in Figure 12 (d). Empirically we observe that the NLC dynamics models correctly learnt the delay dynamics (P2) of each individual environment, as they each have a similar low forward MSE error for the varying levels of

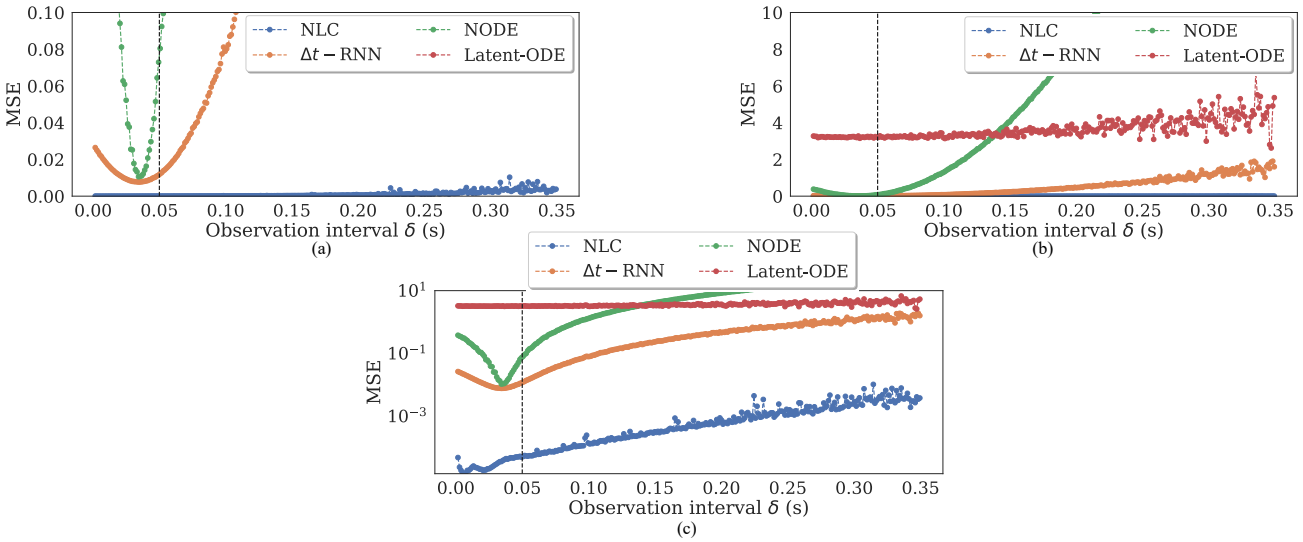


Figure 11: Next step ahead validation error (MSE) at a variable time step of an observation interval $\delta$ of the learnt baseline dynamics models, for the irregularly-sampled Cartpole environment with a fixed action delay of $\tau = \bar{\Delta}$. Where in each sub-figure we have the same results plotted at: (a) a zoomed-in y-limit, (b) a zoomed-out y-limit and (c) a log scale plot. The black dotted line indicates the environments run-time observation interval $\delta = \bar{\Delta} = 0.05$ s. Here, we observe Neural Laplace Control learns a good dynamics model over a wide range of observation intervals $\delta$, correctly learning from the irregularly-sampled offline dataset (P1).

inherent delay. In contrast, neural-ODE models are unable to model the delay dynamics correctly, and we observe that they have a higher rate of increasing forward MSE error Figure 12 (a) & (b), that can also increase for an increasing environment delay and is shown further in Figure 12 for the Latent-ODE model—intuitively this may occur as increasing the delay in a delay differential equation driven environment dynamics becomes less like that of an ordinary differential equation driven environment dynamics, which neural-ODE methods implicitly assume.

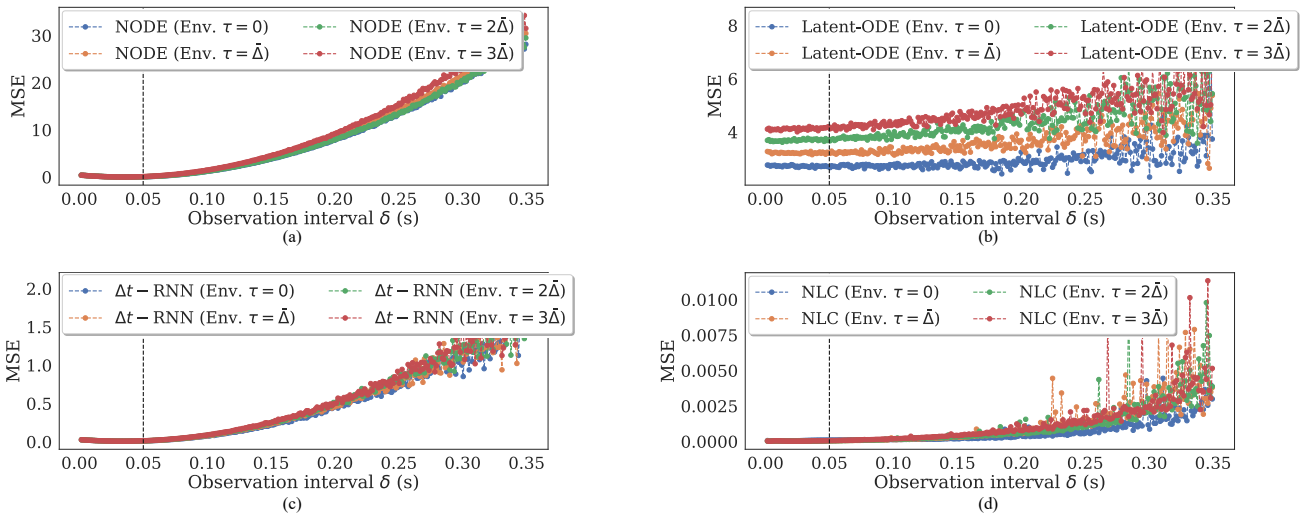


Figure 12: Next step ahead validation error (MSE) at a variable time step of an observation interval $\delta$ of the learnt baseline dynamics models, for each delayed environment versions $\tau = \{0, \bar{\Delta}, 2\bar{\Delta}, 3\bar{\Delta}\}$ of the specific Cartpole environment. Where in each sub-figure we have: (a) NODE, (b) Latent-ODE, (c) $\Delta t-$RNN and (d) NLC. The black dotted line indicates the environments run-time observation interval $\delta = \bar{\Delta} = 0.05$ s. Here, Neural Laplace Control is able to correctly learn and capture the delayed dynamics (P2), as the forward MSE errors are low and similar—whereas neural-ODE methods (a) & (b) have a greater increasing forward MSE.

### I.3 Can NLC plan with a longer time horizon using a fixed amount of compute?

We investigate this by planning with a MPC planner, increasing the observation interval $\delta$ and keeping $N$ fixed, therefore the time horizon $H$ increases—as shown in Figure 4. Here we measure the total planning time taken to plan the next action as $\mathcal{O}$ seconds [6] and observe that planning with the NLC dynamics model takes the same amount of planning time, and hence a *fixed amount compute* for planning at a greater time horizon $H$—which is the same as a $\Delta t-$RNN. This is achieved by the Laplace-based dynamics model that can predict a future state at *any* future time interval using the same number of forward model evaluations, and hence the same amount of compute. In contrast, this is *not* readily achievable with neural-ODE continuous-time methods that use a larger number of numerical forward steps with a numerical ODE step-wise solver for an increasing time horizon—leading to an increasing planning time for an increasing time horizon, i.e., $\mathcal{O} \propto H$. Furthermore, we highlight, that there exists a trade-off of the time horizon $H$ to plan at—as we wish to use a large "enough" horizon that captures sufficient future dynamics, whilst minimizing compounded model inaccuracies at a larger planning time horizon. Therefore, these two opposing factors, give rise to the maxima of the normalized score $\mathcal{R}$ at a time horizon $H = 2$ seconds, as seen in Figure 13. The numeric values for each environment are tabulated in Tables 9, 10 & 11.

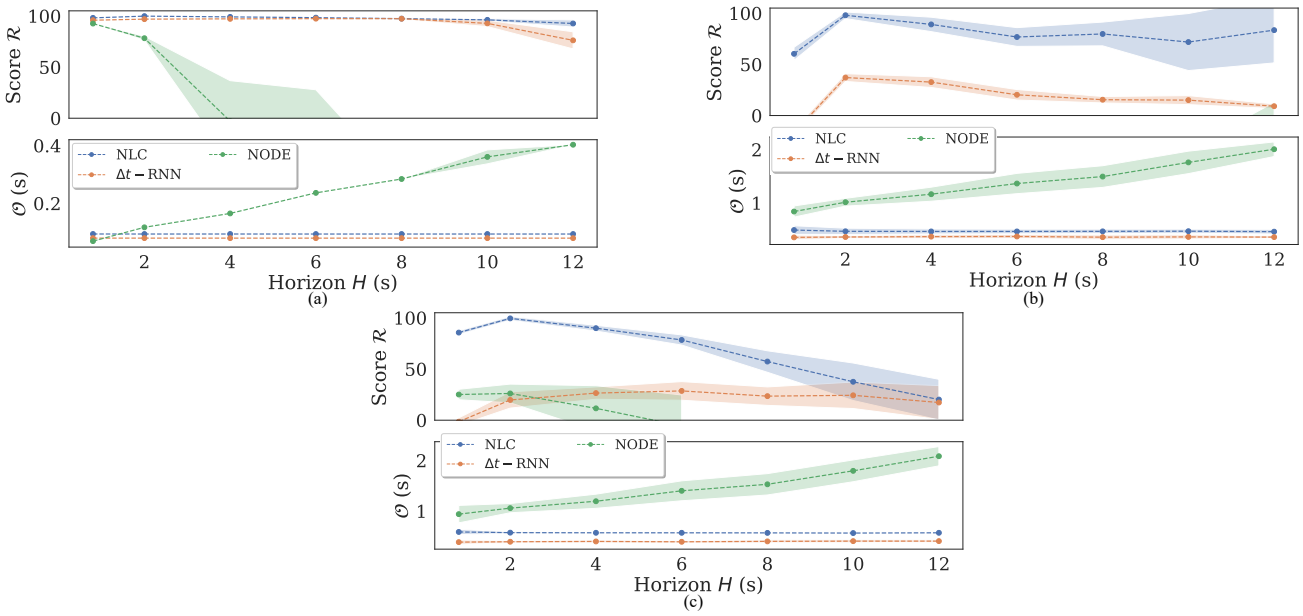


Figure 13: Normalized score $\mathcal{R}$ of the baseline methods on the three environments with an action delay of $\tau = \bar{\Delta} = 0.05$ seconds, plotted against an increasing time horizon $H$, by increasing the observation interval $\delta$. Specifically in: (a) the Cartpole environment, (b) the Pendulum environment and (c) the Acrobot environment. NLC, maintains a high performing policy at a longer time horizon—whilst using the same amount of *constant* planning time per action $\mathcal{O}$ as a $\Delta t-$RNN.

Table 9: Numerical results of the Cartpole environment, of the planning time taken $\mathcal{O}$ to generate the next action, and normalized scores $\mathcal{R}$ of the baseline methods with an environment action delay of $\tau = \bar{\Delta} = 0.05$s, varying against an increasing time horizon $H$—by increasing the observation interval $\delta$. NLC, maintains a high performing policy at a longer time horizon—whilst using the same amount of *constant* planning time per action $\mathcal{O}$ as a $\Delta t-$RNN.

| Dynamics Model | | $H$=0.8 s<br>$N$=40, $\delta$=0.02 s | $H$=2.0 s<br>$N$=40, $\delta$=0.05 s | $H$=4.0 s<br>$N$=40, $\delta$=0.1 s | $H$=6.0 s<br>$N$=40, $\delta$=0.15 s | $H$=8.0 s<br>$N$=40, $\delta$=0.2 s | $H$=10.0 s<br>$N$=40, $\delta$=0.25 s | $H$=12.0 s<br>$N$=40, $\delta$=0.3 s |
|---|---|---|---|---|---|---|---|---|
| $\Delta t-$RNN | $\mathcal{O}$ | 0.08±0.0 | 0.08±0.0 | 0.08±0.0 | 0.08±0.0 | 0.08±0.0 | 0.08±0.0 | 0.08±0.0 |
| | $\mathcal{R}$ | (95.85±0.34) | (96.88±0.34) | (97.18±0.35) | (97.37±0.26) | (97.32±0.84) | (92.75±3.09) | (76.16±8.36) |
| NODE | $\mathcal{O}$ | 0.069±0.0 | 0.117±0.0 | 0.165±0.0 | 0.236±0.0 | 0.284±0.0 | 0.361±0.02 | 0.403±0.0 |
| | $\mathcal{R}$ | (92.56±0.16) | (78.24±1.96) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) |
| NLC **(Ours)** | $\mathcal{O}$ | 0.094±0.0 | 0.094±0.0 | 0.094±0.0 | 0.094±0.0 | 0.094±0.0 | 0.094±0.0 | 0.094±0.0 |
| | $\mathcal{R}$ | (98.13±0.16) | (99.83±0.14) | (99.08±0.29) | (98.34±0.43) | (97.38±0.53) | (96.2±1.19) | (92.65±3.45) |

[6] We perform all results using a Intel Core i9-12900K CPU @ 3.20GHz, 64GB RAM with a Nvidia RTX3090 GPU 24GB.

Table 10: Numerical results of the Pendulum environment, of the planning time taken $\mathcal{O}$ to generate the next action, and normalized scores $\mathcal{R}$ of the baseline methods with an environment action delay of $\tau = \bar{\Delta} = 0.05$s, varying against an increasing time horizon $H$—by increasing the observation interval $\delta$. NLC, maintains a high performing policy at a longer time horizon—whilst using the same amount of *constant* planning time per action $\mathcal{O}$ as a $\Delta t-$RNN.

| Dynamics Model | | $H$=0.8 s $N$=40, $\delta$=0.02 s | $H$=2.0 s $N$=40, $\delta$=0.05 s | $H$=4.0 s $N$=40, $\delta$=0.1 s | $H$=6.0 s $N$=40, $\delta$=0.15 s | $H$=8.0 s $N$=40, $\delta$=0.2 s | $H$=10.0 s $N$=40, $\delta$=0.25 s | $H$=12.0 s $N$=40, $\delta$=0.3 s |
|---|---|---|---|---|---|---|---|---|
| $\Delta t-$RNN | $\mathcal{O}$ | 0.355±0.04 | 0.362±0.03 | 0.369±0.03 | 0.372±0.04 | 0.359±0.04 | 0.364±0.04 | 0.361±0.03 |
| | $\mathcal{R}$ | (0.0±0.0) | (37.11±3.79) | (32.67±5.23) | (20.28±5.13) | (15.46±3.15) | (15.07±4.4) | (9.12±2.22) |
| NODE | $\mathcal{O}$ | 0.84±0.1 | 1.014±0.07 | 1.164±0.13 | 1.364±0.19 | 1.494±0.2 | 1.76±0.21 | 2.007±0.14 |
| | $\mathcal{R}$ | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) |
| **NLC (Ours)** | $\mathcal{O}$ | 0.493±0.08 | 0.469±0.06 | 0.466±0.05 | 0.466±0.04 | 0.468±0.04 | 0.471±0.04 | 0.462±0.04 |
| | $\mathcal{R}$ | (60.31±6.39) | (97.98±3.21) | (89.04±7.04) | (76.73±9.15) | (79.7±11.59) | (71.8±27.86) | (83.44±32.02) |

Table 11: Numerical results of the Acrobot environment, of the planning time taken $\mathcal{O}$ to generate the next action, and normalized scores $\mathcal{R}$ of the baseline methods with an environment action delay of $\tau = \bar{\Delta} = 0.05$s, varying against an increasing time horizon $H$—by increasing the observation interval $\delta$. NLC, maintains a high performing policy at a longer time horizon—whilst using the same amount of *constant* planning time per action $\mathcal{O}$ as a $\Delta t-$RNN.

| Dynamics Model | | $H$=0.8 s $N$=40, $\delta$=0.02 s | $H$=2.0 s $N$=40, $\delta$=0.05 s | $H$=4.0 s $N$=40, $\delta$=0.1 s | $H$=6.0 s $N$=40, $\delta$=0.15 s | $H$=8.0 s $N$=40, $\delta$=0.2 s | $H$=10.0 s $N$=40, $\delta$=0.25 s | $H$=12.0 s $N$=40, $\delta$=0.3 s |
|---|---|---|---|---|---|---|---|---|
| $\Delta t-$RNN | $\mathcal{O}$ | 0.387±0.04 | 0.394±0.02 | 0.4±0.03 | 0.393±0.02 | 0.401±0.03 | 0.406±0.03 | 0.407±0.02 |
| | $\mathcal{R}$ | (0.0±0.0) | (19.78±7.95) | (26.49±6.0) | (28.62±9.1) | (23.51±9.02) | (24.34±12.93) | (17.28±16.53) |
| NODE | $\mathcal{O}$ | 0.94±0.17 | 1.058±0.09 | 1.194±0.14 | 1.401±0.2 | 1.529±0.21 | 1.795±0.22 | 2.084±0.19 |
| | $\mathcal{R}$ | (25.1±5.12) | (26.13±9.13) | (11.59±22.12) | (0.0±0.0) | NA | NA | NA |
| **NLC (Ours)** | $\mathcal{O}$ | 0.588±0.05 | 0.574±0.02 | 0.571±0.01 | 0.571±0.01 | 0.569±0.01 | 0.566±0.01 | 0.571±0.01 |
| | $\mathcal{R}$ | (85.62±1.85) | (99.49±1.87) | (89.93±2.8) | (78.4±4.96) | (57.32±10.5) | (37.56±18.32) | (20.16±19.83) |

### I.4 Can NLC use less compute to plan with the same time horizon?

We further investigate an alternative setup in Figure 14, and keep the time horizon fixed at $H = 2$ seconds and increase the observation interval $\delta$—allowing us to reduce $N$ the number of MPC forward planning steps (i.e., $N = \frac{H}{\delta}$). Importantly, this *reduces the planning time* $\mathcal{O}$ needed to generate the next action, enabling a method to use a higher frequency of executing actions to control the dynamics—whilst still planning at the *same fixed time horizon* $H$. NLC is able to still outperform the baselines, achieving a high performing policy—even when using a lesser amount of planning compute per action. The numeric values for each environment are tabulated in Tables 12, 13 & 14.

Table 12: Numerical results of the Cartpole environment, of the planning time taken $\mathcal{O}$ to generate the next action, and normalized scores $\mathcal{R}$ of the baseline methods with an environment action delay of $\tau = \bar{\Delta} = 0.05$s, varying against an increasing observation interval $\delta$. Here, the time horizon is fixed at $H = 2$s, thus increasing the observation interval $\delta$ decreases the number of MPC forward planning steps needed (i.e., $N = \frac{H}{\delta}$). NLC demonstrates that it can still outperform the baselines, achieving a near optimal policy—whilst reducing the planning time taken $\mathcal{O}$ needed to generate the next action.

| Dynamics Model | | $H$=2.0 s $N$=100, $\delta$=0.02 s | $H$=2.0 s $N$=40, $\delta$=0.05 s | $H$=2.0 s $N$=20, $\delta$=0.1 s | $H$=2.0 s $N$=13, $\delta$=0.15 s | $H$=2.0 s $N$=10, $\delta$=0.2 s | $H$=2.0 s $N$=8, $\delta$=0.25 s | $H$=2.0 s $N$=6, $\delta$=0.33 s | $H$=2.0 s $N$=5, $\delta$=0.4 s |
|---|---|---|---|---|---|---|---|---|---|
| $\Delta t-$RNN | $\mathcal{O}$ | 0.197±0.0 | 0.08±0.0 | 0.04±0.0 | 0.027±0.0 | 0.021±0.0 | 0.017±0.0 | 0.013±0.0 | 0.011±0.0 |
| | $\mathcal{R}$ | (95.95±0.4) | (96.79±0.25) | (96.95±0.27) | (94.79±0.49) | (88.59±2.68) | (81.81±7.97) | (80.47±17.05) | (70.48±23.23) |
| NODE | $\mathcal{O}$ | 0.168±0.0 | 0.117±0.0 | 0.083±0.0 | 0.077±0.0 | 0.072±0.0 | 0.072±0.0 | 0.068±0.0 | 0.066±0.0 |
| | $\mathcal{R}$ | (92.57±0.13) | (77.49±1.77) | (66.49±2.8) | (37.18±6.9) | (17.23±15.49) | (18.08±21.49) | (34.32±24.34) | (16.81±38.05) |
| **NLC (Ours)** | $\mathcal{O}$ | 0.25±0.0 | 0.101±0.0 | 0.051±0.0 | 0.033±0.0 | 0.026±0.0 | 0.021±0.0 | 0.016±0.0 | 0.013±0.0 |
| | $\mathcal{R}$ | (99.23±0.3) | (99.85±0.13) | (99.95±0.12) | (99.82±0.15) | (99.51±0.18) | (99.32±0.27) | (99.06±0.44) | (98.13±0.16) |

### I.5 Can NLC learn from few samples?

We observe in Figure 15 that NLC can still learn a suitable dynamics model and perform well across the environments with a delay of $\tau = \bar{\Delta} = 0.05$ seconds, when trained with an offline irregularly-sampled in time dataset that contains a limited number of samples. Specifically, it is able to learn with only 200 random samples on the Cartpole and Pendulum environments—which corresponds 10 seconds of interaction time of a noisy expert (expert with random action noise) agent from the delayed environment. Also, on the Acrobot environment, a more challenging environment it is able to learn a sufficient dynamics model from only 1,000 random samples.

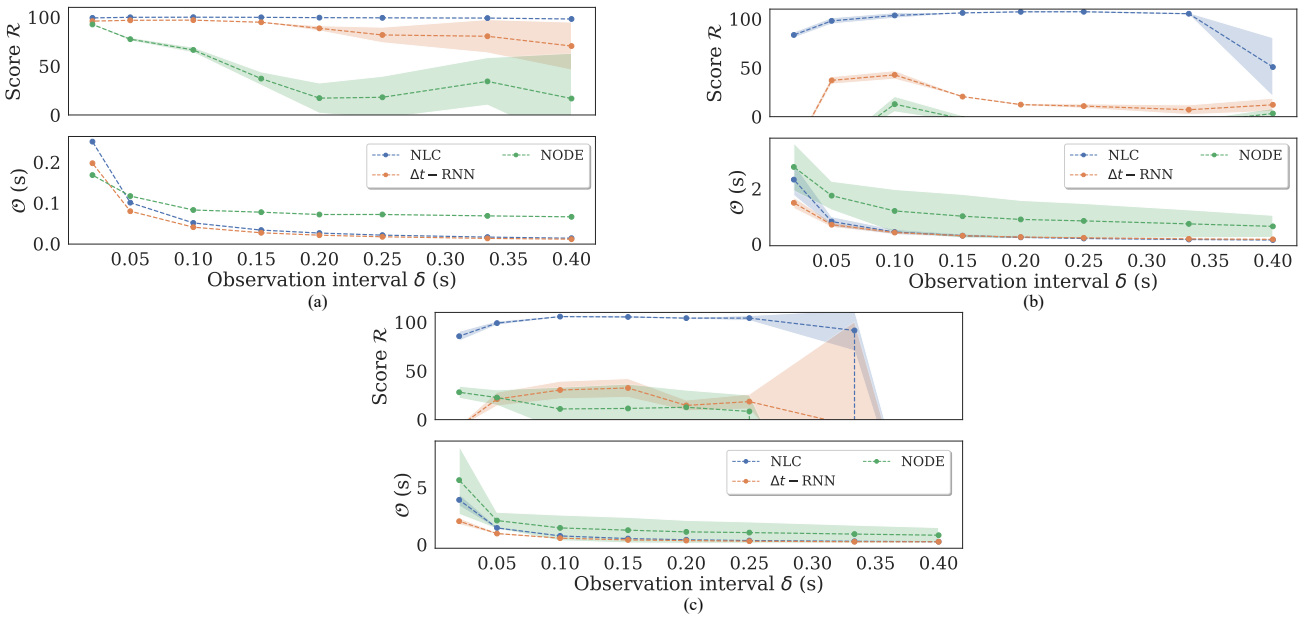


Figure 14: Normalized score $\mathcal{R}$ of the baseline methods on the three environments in each sub-figure with an action delay of $\tau = \bar{\Delta} = 0.05$s, plotted against an increasing observation interval $\delta$. Specifically in: (a) the Cartpole environment, (b) the Pendulum environment and (c) the Acrobot environment. Here, the time horizon is fixed at $H = 2$s, thus increasing the observation interval $\delta$ decreases the number of MPC forward planning steps needed (i.e., $N = \frac{H}{\delta}$). NLC demonstrates that it can still outperform the baselines, achieving a near optimal policy—whilst reducing the planning time taken $\mathcal{O}$ needed to generate the next action.

Table 13: Numerical results of the Pendulum environment, of the planning time taken $\mathcal{O}$ to generate the next action, and normalized scores $\mathcal{R}$ of the baseline methods with an environment action delay of $\tau = \bar{\Delta} = 0.05$s, varying against an increasing observation interval $\delta$. Here, the time horizon is fixed at $H = 2$s, thus increasing the observation interval $\delta$ decreases the number of MPC forward planning steps needed (i.e., $N = \frac{H}{\delta}$). NLC demonstrates that it can still outperform the baselines, achieving a near optimal policy—whilst reducing the planning time taken $\mathcal{O}$ needed to generate the next action.

| Dynamics Model | | $H$=2.0 s $N$=100, $\delta$=0.02 s | $H$=2.0 s $N$=40, $\delta$=0.05 s | $H$=2.0 s $N$=20, $\delta$=0.1 s | $H$=2.0 s $N$=13, $\delta$=0.15 s | $H$=2.0 s $N$=10, $\delta$=0.2 s | $H$=2.0 s $N$=8, $\delta$=0.25 s | $H$=2.0 s $N$=6, $\delta$=0.33 s | $H$=2.0 s $N$=5, $\delta$=0.4 s |
|---|---|---|---|---|---|---|---|---|---|
| $\Delta t-$RNN | $\mathcal{O}$ | 1.49±0.2 | 0.699±0.1 | 0.414±0.06 | 0.299±0.04 | 0.251±0.04 | 0.223±0.03 | 0.186±0.03 | 0.169±0.02 |
| | $\mathcal{R}$ | (0.0±0.0) | (37.11±3.79) | (42.7±4.37) | (20.51±0.6) | (12.32±1.09) | (10.81±2.27) | (7.07±4.99) | (12.12±6.86) |
| NODE | $\mathcal{O}$ | 2.788±0.86 | 1.749±0.52 | 1.196±0.78 | 1.005±0.79 | 0.889±0.69 | 0.836±0.63 | 0.728±0.51 | 0.638±0.4 |
| | $\mathcal{R}$ | (0.0±0.0) | (0.0±0.0) | (12.78±7.86) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) | (0.0±0.0) | (3.22±5.19) |
| **NLC (Ours)** | $\mathcal{O}$ | 2.329±0.57 | 0.813±0.18 | 0.434±0.1 | 0.301±0.06 | 0.241±0.05 | 0.2±0.04 | 0.163±0.03 | 0.145±0.03 |
| | $\mathcal{R}$ | (83.55±3.02) | (97.98±3.21) | (103.55±2.84) | (106.13±0.41) | (107.22±1.03) | (107.28±0.93) | (105.27±0.98) | (50.73±30.43) |

Table 14: Numerical results of the Acrobot environment, of the planning time taken $\mathcal{O}$ to generate the next action, and normalized scores $\mathcal{R}$ of the baseline methods with an environment action delay of $\tau = \bar{\Delta} = 0.05$s, varying against an increasing observation interval $\delta$. Here, the time horizon is fixed at $H = 2$s, thus increasing the observation interval $\delta$ decreases the number of MPC forward planning steps needed (i.e., $N = \frac{H}{\delta}$). NLC demonstrates that it can still outperform the baselines, achieving a near optimal policy—whilst reducing the planning time taken $\mathcal{O}$ needed to generate the next action.

| Dynamics Model | | $H$=2.0 s $N$=100, $\delta$=0.02 s | $H$=2.0 s $N$=40, $\delta$=0.05 s | $H$=2.0 s $N$=20, $\delta$=0.1 s | $H$=2.0 s $N$=13, $\delta$=0.15 s | $H$=2.0 s $N$=10, $\delta$=0.2 s | $H$=2.0 s $N$=8, $\delta$=0.25 s | $H$=2.0 s $N$=6, $\delta$=0.33 s | $H$=2.0 s $N$=5, $\delta$=0.4 s |
|---|---|---|---|---|---|---|---|---|---|
| $\Delta t-$RNN | $\mathcal{O}$ | 2.037±0.29 | 0.96±0.08 | 0.558±0.06 | 0.405±0.04 | 0.348±0.03 | 0.292±0.03 | 0.243±0.02 | 0.243±0.02 |
| | $\mathcal{R}$ | (0.0±0.0) | (21.04±7.06) | (30.4±9.03) | (32.44±9.69) | (14.59±5.5) | (18.49±7.6) | (0.0±0.0) | (0.0±0.0) |
| NODE | $\mathcal{O}$ | 5.631±2.99 | 2.095±0.73 | 1.455±1.12 | 1.256±1.12 | 1.108±1.01 | 1.045±0.93 | 0.915±0.78 | 0.815±0.67 |
| | $\mathcal{R}$ | (28.09±6.18) | (22.63±7.97) | (10.96±22.24) | (11.48±24.95) | (12.67±17.63) | (8.41±16.82) | (0.0±0.0) | (0.0±0.0) |
| **NLC (Ours)** | $\mathcal{O}$ | 3.902±0.52 | 1.452±0.09 | 0.755±0.06 | 0.521±0.05 | 0.419±0.04 | 0.349±0.04 | 0.283±0.03 | 0.251±0.03 |
| | $\mathcal{R}$ | (85.6±4.98) | (99.05±2.06) | (105.71±0.92) | (105.39±1.24) | (104.15±0.9) | (104.18±2.61) | (91.49±20.89) | (0.0±0.0) |

### I.6 Can NLC incorporate state-based constraints?

We show that using a MPC planner, we can easily incorporate a new state-based constraint at test time (run-time) in Figure 16. Here, using the Cartpole environment we add an additional constraint on the horizontal $x$ position of the cart.

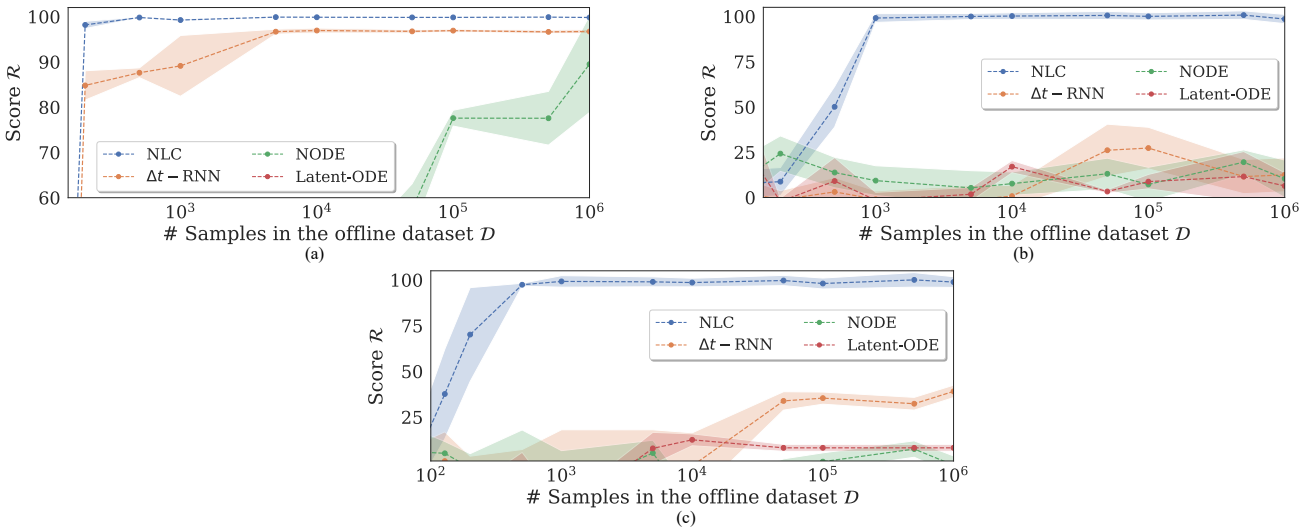


Figure 15: Normalized score $\mathcal{R}$ of the baseline methods on the three environments in each sub-figure with an action delay of $\tau = \bar{\Delta} = 0.05$s, plotted against the number of samples in the irregularly-sampled offline dataset used to train the dynamics model of each method. Specifically, in: (a) the Cartpole environment, (b) the Pendulum environment and (c) the Acrobot environment. NLC can maintain a high performing policy on the Cartpole and Pendulum environments—even from the challenging case of only learning a dynamics model from 200 samples from an irregularly-sampled in time offline dataset $\mathcal{D}$.

Specifically, we illustrate that a "left constrained" ($x < 0$) version at run-time can still be solved by our NLC method and similarly for a "right constrained" ($x > 0$) version can be solved—this is made possible as the planner only generates feasible trajectories where these additional state-based constraints are satisfied, which is further illustrated in Figure 16. Therefore, using a MPC planner we can easily incorporate new additional unseen state constraints at run-time, whereas using a learnt policy (q-learner) would be unable to do this and would have to re-train a new policy for each new state constraint. We note that this benefit of using a MPC planner to incorporate additional state-based constraints has been shown by others (Lutter et al., 2021).

# J ADDITIONAL EXPERIMENTS

In this section we seek to gain further insight into how the baseline dynamics models compare when trained using the same number of fixed training epochs on each dataset, and an ablation by training the dynamics models on regularly-sampled offline datasets.

## J.1 Training dynamics models for a fixed number of epochs

Here we changed the training setup to train each of the dynamics models for the same number of 10 epochs, with the same batch size, that of 1 (As NODE and Latent-ODE are only able to support a batch size of 1 using our offline dataset) on the irregularly sampled offline dataset consisting of only 10,000 random state-action samples. The benchmark methods against each environment, which consists of a continuous-time environment with a specific delay—with normalized scores $\mathcal{R}$ are tabulated in Table 15. We observe that the results are consistent and similar to the original results reported in Table 2 in the paper—where each dynamics model received the same number of training iterations, and this is less training than our original results tabulated in Table 2 in the paper. Given less training iterations, NLC is still able to outperform the baseline dynamics models.

## J.2 Ablation by training dynamics models on regularly-sampled offline datasets

We further investigate how the baseline models perform by training on regularly-sampled offline datasets, i.e., datasets that are collected with a noisy agent that observes the next observation $\boldsymbol{x}(t + \Delta_i)$ with the same discrete time observation interval $\Delta_i = \Delta_j$. These results are tabulated in Table 16. We observe a similar pattern, where NLC is still able to learn a *good* dynamics model and outperform the baselines.

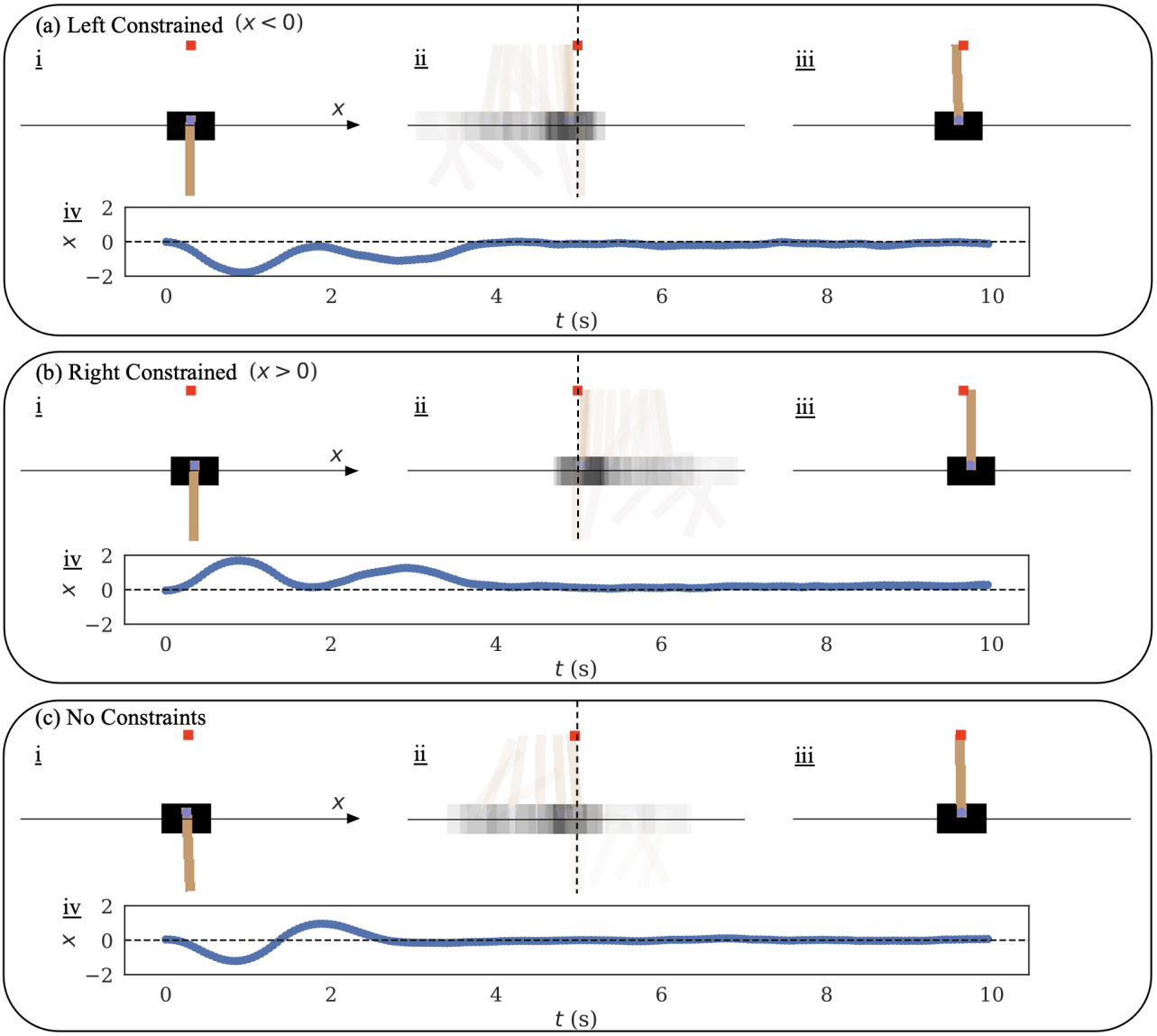


Figure 16: Screen shots of the Cartpole environment, with an action delay of $\tau = \bar{\Delta} = 0.05$s using the NLC dynamics model. We see additional new state-based constraints given at run-time, and the MPC planner is able to incorporate these such that the action trajectories generated and executed satisfy the state constraint, here on the horizontal $x$ position of the cart. Specifically, in sub-figures we observe: (a) a "left constrained" ($x < 0$) version, (b) a "right constrained" ($x > 0$) version and in (c) a standard version with no state constraints. Specifically in each sub-figure, we have further sub-figures where: (i) shows the starting screenshot, (ii) a visualization of the trajectory followed (with past screenshots superimposed, where the more faded image is oldest), (iii) the final state reached and in (iv) the state $x$ trajectory plotted over the entire episode's length of 10 seconds. Therefore, using a MPC planner we can easily incorporate new additional unseen state constraints at run-time, whereas using a learnt policy (q-learner) would be unable to do this and would have to re-train a new policy for each new state constraint.

### J.3 Environments with observational noise

We performed an additional experiment of adding observation noise, specifically, perturbing the state with Gaussian noise $\mathcal{N}(0, 0.01^2)$. We observe in Table 17 that NLC is still performant under this observation noise. Here, the noisy expert datasets are collected with this observation noise, and the same observation noise is used at run-time evaluation for each method. We also see this same effect for increasing noise, in Figure 17.

Table 15: Normalized scores $\mathcal{R}$ of the offline model-based agents, where the irregularly-sampled (P1) offline dataset consists of an action delay (P2) of $\{0, 1, 2, 3\}$ multiples of the environment's observation interval time step $\bar{\Delta} = 0.05$ seconds. Averaged over 20 random seeds, with $\pm$ standard deviations. Scores are un-discounted cumulative rewards normalized to be between 0 and 100, where 0 corresponds to the Random agent and 100 corresponds to the expert with the *known* world model (oracle+MPC). Negative normalized scores, i.e., worse than random are set to zero. Specifically, we trained each dynamics model for the same number of 10 epochs, with a batch size of 1 from an offline dataset consisting of only 10,000 random state-action samples—we note that each dynamics model received the same number of training iterations, and this is less training than our original results tabulated in Table 2 in the paper. Given less training iterations, NLC is still able to outperform the baseline dynamics models.

| | Action Delay $\tau = 0$ | | | Action Delay $\tau = \bar{\Delta}$ | | | Action Delay $\tau = 2\bar{\Delta}$ | | | Action Delay $\tau = 3\bar{\Delta}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dynamics Model | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot |
| Random | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |
| Oracle | 100.0±0.04 | 100.0±3.59 | 100.0±2.61 | 100.0±0.02 | 100.0±3.15 | 100.0±1.8 | 100.0±0.03 | 100.0±2.42 | 100.0±1.64 | 100.0±0.04 | 100.0±2.6 | 100.0±1.42 |
| $\Delta t-$RNN | 97.13±0.27 | 19.86±5.2 | 30.9±8.19 | 98.59±0.13 | 13.02±11.0 | 25.16±9.86 | 98.02±0.17 | 19.64±6.21 | 33.67±7.71 | 96.89±0.29 | 13.18±4.4 | 26.27±6.03 |
| Latent-ODE | 0.0±0.0 | 0.0±0.0 | 5.99±9.64 | 0.0±0.0 | 5.26±11.63 | 6.45±13.78 | 0.0±0.0 | 2.86±14.97 | 7.97±9.31 | 35.81±55.95 | 5.29±12.05 | 12.22±6.08 |
| NODE | 95.63±0.19 | 0.0±0.0 | 16.04±10.68 | 93.43±4.1 | 0.0±0.0 | 1.7±6.73 | 0.0±0.0 | 6.19±3.16 | 5.5±10.52 | 95.49±0.12 | 4.4±5.13 | 23.98±8.18 |
| **NLC (Ours)** | **99.54±0.12** | **92.65±4.43** | **57.1±22.72** | **99.39±0.02** | **95.78±2.35** | **66.55±6.59** | **99.71±0.07** | **99.01±4.13** | **57.73±20.73** | **97.16±0.24** | **91.56±4.73** | **80.48±14.03** |

Table 16: Normalized scores $\mathcal{R}$ of the offline model-based agents, where we use a regularly-sampled offline dataset—that consists of an action delay (P2) of $\{0, 1, 2, 3\}$ multiples of the environments observation interval time step $\bar{\Delta} = 0.05$ seconds. Averaged over 20 random seeds, with $\pm$ standard deviations. Scores are un-discounted cumulative rewards normalized to be between 0 and 100, where 0 corresponds to the Random agent and 100 corresponds to the expert with the *known* world model (oracle+MPC). Negative normalized scores, i.e., worse than random are set to zero.

| | Action Delay $\tau = 0$ | | | Action Delay $\tau = \bar{\Delta}$ | | | Action Delay $\tau = 2\bar{\Delta}$ | | | Action Delay $\tau = 3\bar{\Delta}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dynamics Model | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot | Cartpole | Pendulum | Acrobot |
| Random | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |
| Oracle | 100.0±0.06 | 100.0±3.59 | 100.0±1.98 | 100.0±0.03 | 100.0±3.05 | 100.0±1.18 | 100.0±0.1 | 100.0±2.57 | 100.0±2.27 | 100.0±0.03 | 100.0±2.7 | 100.0±1.7 |
| $\Delta t-$RNN | 97.17±0.42 | 0.0±0.0 | 3.07±6.09 | 97.18±0.32 | 0.0±0.0 | 7.03±7.53 | 96.54±0.35 | 0.0±0.0 | 5.39±6.0 | 98.01±0.31 | 0.0±0.0 | 10.47±7.09 |
| Latent-ODE | 0.0±0.0 | 0.0±0.0 | 12.26±9.47 | 0.0±0.0 | 0.0±0.0 | 3.34±25.49 | 0.0±0.0 | 0.0±0.0 | 12.81±14.84 | 0.0±0.0 | 0.0±0.0 | 19.47±10.4 |
| NODE | 0.0±0.0 | 49.8±7.27 | 27.18±10.36 | 0.0±0.0 | 39.12±5.54 | 32.06±10.84 | 54.19±51.77 | 0.0±0.0 | 24.31±10.38 | 0.0±0.0 | 0.0±0.0 | 25.49±11.89 |
| **NLC (Ours)** | **100.01±0.04** | **99.95±3.74** | **99.39±2.31** | **99.97±0.1** | **101.14±3.7** | **101.38±2.14** | **100.03±0.05** | **99.45±2.98** | **99.74±1.79** | **99.93±0.08** | **99.81±3.17** | **100.11±1.92** |

Table 17: Normalized scores $\mathcal{R}$ of the offline model-based agents, where the irregularly-sampled (P1) offline dataset consists of an action delay (P2) of $\{0, 1, 2, 3\}$ multiples of the environments observation interval time step $\bar{\Delta} = 0.05$ seconds. Averaged over 20 random seeds, with $\pm$ standard deviations. Here we add observation noise—whereby the state is perturbed with Gaussian noise $\mathcal{N}(0, 0.01^2)$. Scores are un-discounted cumulative rewards normalized to be between 0 and 100, where 0 corresponds to the Random agent and 100 corresponds to the expert with the *known* world model (Oracle+MPC). Negative normalized scores, i.e., worse than random are set to zero.

| Dynamics Model | Action Delay $\tau = 0$ Cartpole | Action Delay $\tau = \bar{\Delta}$ Cartpole | Action Delay $\tau = 2\bar{\Delta}$ Cartpole | Action Delay $\tau = 3\bar{\Delta}$ Cartpole |
|---|---|---|---|---|
| Random | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |
| Oracle | 100.0±0.04 | 100.0±0.02 | 100.0±0.02 | 100.0±0.02 |
| $\Delta t-$RNN | 98.33±0.26 | 97.96±0.24 | 97.8±0.25 | 97.99±0.27 |
| Latent-ODE | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |
| NODE | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 9.86±56.04 |
| **NLC (Ours)** | **99.99±0.03** | **99.97±0.04** | **99.91±0.08** | **99.9±0.03** |

### J.4 Environment with friction

We also performed an additional experiment of adding friction to the environment. We observe in Figure 18 that NLC is still performant under this friction for the Cartpole environment. Here, the expert datasets are collected with this friction, and the same friction is used at run-time evaluation for each method. Specifically, we used a fiction coefficient of 5e-4 for the cart and a friction coefficient of 2e-6 for the pole.

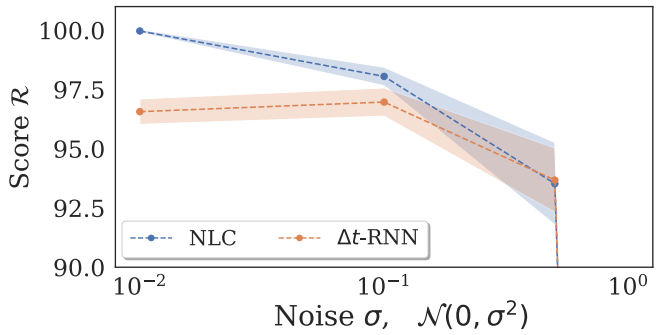


Figure 17: Normalized score $\mathcal{R}$ of the baseline methods on the Cartpole environment with an action delay of $\tau = \bar{\Delta} = 0.05$s. Here adding observation noise, specifically, perturbing the state with Gaussian noise $\mathcal{N}(0, \sigma^2)$, plotted against noise $\sigma$. NLC can maintain a high performing policy on the Cartpole with noise—however degrades like the other closest performing policy with increasing noise, as there becomes less signal to noise in the enviroment observations.

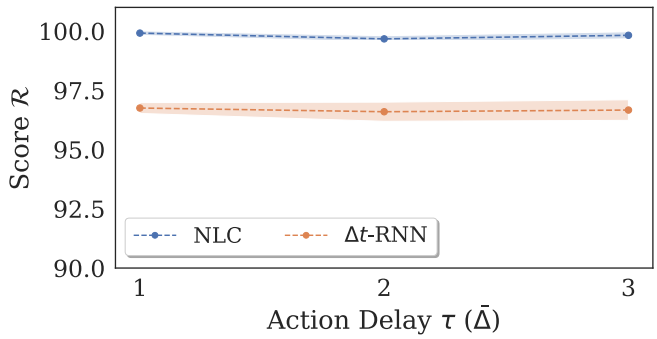


Figure 18: Normalized score $\mathcal{R}$ of the baseline methods on the Cartpole environment against an action delay of $\{0, 1, 2, 3\}$ multiples of the environments observation interval time step $\bar{\Delta} = 0.05$ seconds. Here the Cartpole environment has friction added to it. NLC can maintain a high performing policy on the Cartpole with friction.