Article | Received 13 March 2025; Accepted 23 June 2025; Published 30 June 2025
https://doi.org/10.55092/aias20250005

Neural network-based model predictive control for unmanned aerial vehicles

Yiwei Zhou

Department of Electrical and Computer Engineering, National University of Singapore, Singapore

Correspondence author; E-mail: e1323440@u.nus.edu.

Highlights:

- Quadrotor dynamic model based on data and neural network differential equation methods.
- Faster neural network-based Model Predictive Control with the Jacobian method.

Abstract: This study presents a neural network-based predictive control (abbreviated as NNMPC in subsequent content) approach for quadrotor tracking. By learning dynamical behaviors from experimental flight data, a neural ordinary differential model (NODM) is built first, which is particularly suitable for situations where aerodynamic conditions are complex and difficult to model formulaically. Subsequently, the NODM is used for designing the predictive control by considering the constraint conditions. Here, the proposed method uses a linearized model of the NODM established to effectively handle the neural network system for predicting the states, reducing the computation time. Finally, simulation results show that the proposed method can achieve a good tracking performance.

Keywords: model predictive control; neural ordinary differential equations; trajectory tracking; UAV

1. Introduction

The significance of unmanned aerial vehicles (UAVs), commonly referred to as drones, is growing across numerous domains. Their applications include object tracking, exploration of hazardous or inaccessible areas, atmospheric detection, post-disaster operations, *etc.* [1, 2]. Drones are becoming more stable and reliable, making them ideal for use in a variety of settings.

The key problem in UAV control is to do navigation [3]. This requires the system to have an accurate tracking control function. Researchers have developed various methods to improve the position tracking accuracy of quadrotor UAVs. One of the most popular approaches is to use the model predictive control (MPC) method [4, 5, 6, 7]. Because of its capacity to handle MIMO systems with constraints, including input/output limits to ensure physical feasibility and state constraints to preserve desired operating conditions. This capability makes the MPC method ideal for real-world applications [8].

Furthermore, the MPC method benefits from its capacity to consider both the target path and the anticipated system behavior over a future time window, known as the prediction horizon [9]. This enables the controller to proactively adjust actions by anticipating future needs and constraints, resulting in improved tracking performance and stability, particularly amid disturbances or delays.

However, the MPC method depends significantly on the accuracy of the system model [10], and quadrotor dynamics is influenced by complex aerodynamic effects such as hub force, rolling moment effect, and blade flapping, which cannot be fully captured by linear and non-linear models [11]. The main challenging work in the control design is the modeling of quadrotors, especially in aerodynamics modeling. The work in reference [12] employed a state-dependent coefficient representation to incorporate non-linear characteristics within a quasi-linear system matrix, facilitating the derivation of an equivalent linearized model suitable for the MPC method. In reference [13], the authors develop a dynamic model that accounts for both linear and complex nonlinear effects to capture the full translational dynamics, thereby mitigating the impact of UAV irregular shapes and low Reynolds number behavior. Another possible approach to dynamic optimization is to apply learning-based methods, which is an important approach for modeling nonlinear systems [14]. In reference [15] the authors established a neural network (NN)-based MPC to demonstrate the effectiveness of deep learning in inferring helicopter dynamics, while in reference [16] the authors employed Gaussian processes to create a hybrid model that integrates dynamics with a neural network, enabling real-time feedback control to account for aerodynamic effects at high speeds. In reference [17] the authors present a modified MPC method with a neural network compensation to handle the disturbances. However, the methods presented in references [13,14,17] fail to capture complex aerodynamic behaviors, while the methods presented in references [15,16] rely on large-scale neural networks, which may lead to increased computational demands.

This paper presents a linearized neural model-based MPC method trained by neural network differential equations (NNODEs), which can better fit physical behavior in nature [18]. Unlike existing MPC methods, the proposed method is to use the neural network-based linear model for designing MPC for the quadrotor control. The advantages of the proposed method include: (1) It can still include complex aerodynamic behaviors; (2) It can reduce the computational load. The contributions of the paper are summarized as follows:

- Model the quadrotor dynamic characteristics based on data and neural network differential equation methods and linearize the neural model by the Jacobian method.
- An improved MPC method is developed based on the neural linear model and the neural network-based Model Predictive Control (NNMPC) controller is implemented for tracking different kinds of trajectories.

The subsequent sections of this paper are structured in this manner: Section 2 provides a mathematical description of the system model and discusses relevant complex aerodynamic influences. Section 3 introduces some adopted algorithms and methods for designing the controller. Section 4 describes the experimental process in detail, evaluating the performance of the NNMPC controller under different conditions.

2. Model of quadrotor

2.1. Newton-Euler model of quadrotor

The basic configuration of a quadrotor is shown in Figure 1, the length of each arm length is l, and the attitude of the quadrotor is represented by three Euler angles η: roll ϕ, pitch θ, and yaw ψ. Two frames of reference are defined: the inertia frame (x,y,z), and the body frame (x_B,y_B,z_B), whose origin is attached to the center of mass of the quadrotor. The position of the quadrotor in the inertial frame is defined as $\xi = [x,y,z] \in R^3$, while linear velocities are defined as $V_B = [v_{x,B}, v_{y,B}, v_{z,B}] \in R^3$. Transformation from the body frame to the inertial frame is achieved using the rotation matrix $\mathbf{R} \in R^{3\times3}$ belonging to the special orthogonal group $SO(3) := \{\mathbf{R} \in R^{3\times3} | \mathbf{R}^T\mathbf{R} = \mathbf{I}, det(\mathbf{R}) = \mathbf{1}\}$. The angular velocities in the body frame are defined as $\omega_B = [p,q,r]^T$.

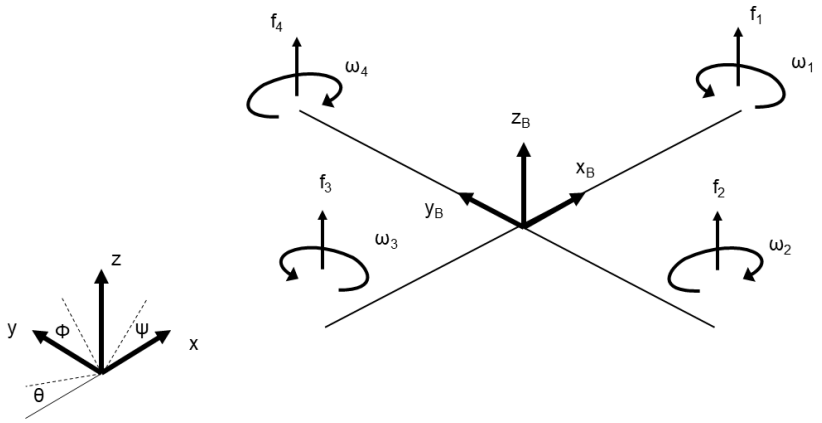

Figure 1. The inertial and body frames of a quadrotor.

The rotors generate thrust T along the z-axis Z_B in the body frame, while torque τ_B is the set of the torques τ_ϕ, τ_θ, and τ_ψ corresponding to rotations in the body frame. The angular velocity and acceleration of the rotor also generate torque $\tau_{M_i} = b\omega_i^2 + I_M\dot{\omega}_i$ around the rotor axis:

$$T = \sum_{i=1}^{4} f_i = k\sum_{i=1}^{4} \omega_i^2, \quad T_B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \tag{1}$$

$$\tau_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} lk(-\omega_2^2 + \omega_4^2) \\ lk(-\omega_1^2 + \omega_3^2) \\ \sum_{i=1}^{4} \tau_{M_i} \end{bmatrix} \tag{2}$$

where k represents the lift constant and b denotes the drag constant, and I_M is the rotor's moment of inertia, which is omitted in this case because its effect is too small. The square of the rotational speed ω_i^2 is defined as the control input u_i. Assuming the quadrotor behaves as a rigid body allows its dynamics to be characterized using the Newton-Euler (N-E) equations, presented as follows:

$$m\ddot{\xi} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi S_\theta C_\phi - C_\psi S_\phi \\ C_\theta C_\phi \end{bmatrix} T_B = mg + RT_B + f_{aero} \tag{3}$$

$$\tau_B = I\dot{\omega}_B + \omega_B \times (I\omega_B) \tag{4}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} (I_{yy} - I_{zz})qr/I_{xx} \\ (I_{zz} - I_{xx})pr/I_{yy} \\ (I_{xx} - I_{yy})pq/I_{zz} \end{bmatrix} + \begin{bmatrix} \tau_\phi/I_{xx} \\ \tau_\theta/I_{yy} \\ \tau_\psi/I_{zz} \end{bmatrix} + \tau_{aero} \tag{5}$$

where m defines the mass of the quadrotor, g is the acceleration of the gravity, and $I = diag(I_{xx}, I_{yy}, I_{zz})$ is defined as the rotational inertia matrix.

2.1.1. Aerodynamic effects

The commonly used mathematical model is a simplified ideal situation omitting the effect of f_{aero} and τ_{aero}. In order to give a more realistic representation of the dynamics of a quadrotor, the effect of air resistance needs to be taken into account. An object's speed significantly influences the aerodynamic forces acting upon it, which in turn directly impacts its acceleration through drag, lift, and airflow separation [19], which is difficult to model. In [16], the authors model complex aerodynamics using Gaussian processes while empirical equations based on aerodynamic coefficients can also be used, which will be discussed in the simulation part.

2.2. Neural model

A neural state-space model is a type of nonlinear state-space model where neural networks can be used to model the state-transition and measurement functions like (3) and (5): An ordinary differential equation (ODE) can model the evolution of a state y over time t in a dynamical system in the real world, while a neural ODE arises when the system's state function $f(\cdot)$ is replaced by a neural network $NN(\cdot)$, effectively learning and computing the solution to the ODE.

We can get a neural ODE simply by converting it into integral form:

$$x(t) = x(t_0) + \int_{t_0}^{t} NN(x(t'), \theta(t'), t')dt' \tag{6}$$

During the learning process, the deep learning ODE solver works based on an explicit Runge-Kutta formula. For an initial value problem (IVP) of first order:

$$\begin{cases} \dot{x} = f(x, u, t) \\ x(t_0) = x_0 \end{cases} \tag{7}$$

Based on the initial conditions, we can follow the recursive relationship to find the numerical solution $x(t), t = 1, 2, ..., N, ...$ in a sequence and optimize a scalar-valued loss function $\mathscr{L}(\cdot)$, whose input is the result of an ODE solver:

$$\mathscr{L}(x(t_1)) = \mathscr{L}\left(x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta)dt\right) = \mathscr{L}(\text{ODESolve}(x(t_0), f, t_0, t_1, \theta)) \tag{8}$$

Methods in Figure 2 obtain the state at the next moment from previous state-changing rates and control inputs through an integration step. The method used in this manuscript replaces the state function $f(\cdot)$ with a neural network $(NN(\cdot)$, which means that the dynamical model of the system is learned or represented by a neural network. During the training process, an MLP network $NN(\cdot)$ that has a structure as shown in Figure 3 characterizes the target quadrotor state function built with N-E equations. Then the deep learning ODE solver imposes the initial conditions x_0 at the initial time t_0, and integrates the ODE from t_0 to t_l, and finally provides the solution evaluated at specified instants for calculating the loss. In cases where the state $\dot{X} = [\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}, \dot{v_x}, \dot{v_y}, \dot{v_z}, \dot{p}, \dot{q}, \dot{r}]$ is difficult to get an analytical solution, data-driven modeling (gray- or black-box modeling) can be a useful alternative, and this process will be discussed in the simulation study section.

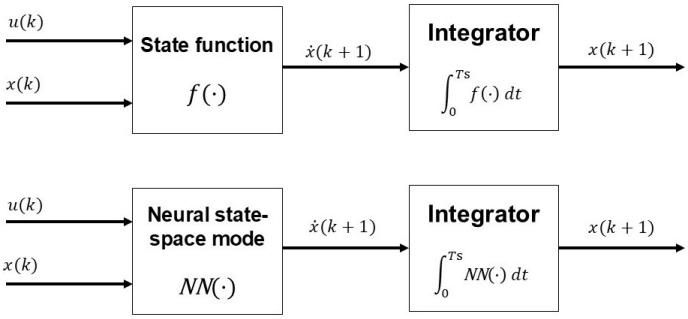

Figure 2. Neural state-space inputs and outputs.

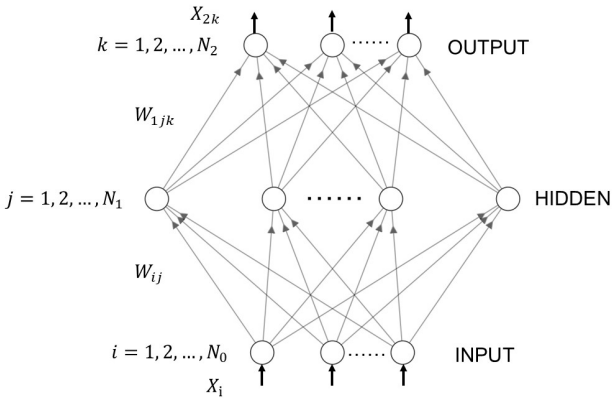

Figure 3. Basic structure of an MLP network.

3. Methodology

This section describes the methodologies for the NNMPC method, covering the design of the model predictive control system, the training process for the neural ODE, and the Jacobian linearization of the trained object.

The controller follows a cascade feedback control involving two controllers: the position controller is designed to generate a thrust T_d, while the attitude controller is used to generate the desired rotor control inputω_i. The whole diagram of the proposed approach is given in Figure 4, and the overall system will be analyzed.

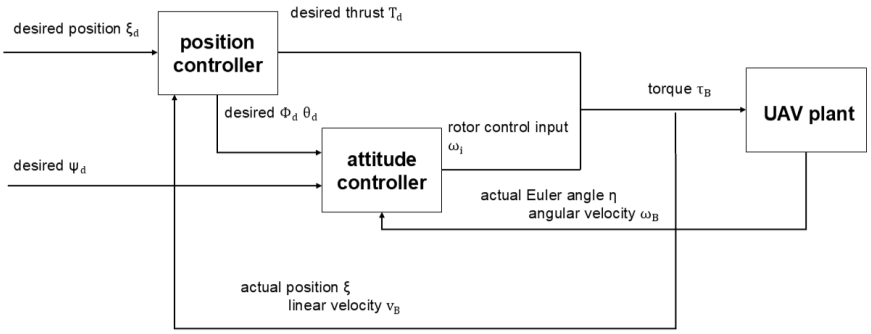

Figure 4. Closed-loop scheme of quadrotor control.

3.1. Nonlinear MPC algorithm design

MPC leverages online optimization to guide system behavior. At every discrete time instant k, the MPC algorithm addresses an optimization problem utilizing the current system state as a starting point. The goal is to determine the optimal sequence of control inputs within a specified control horizon N_u that will steer the system towards a desired setpoint over a predefined prediction horizon N. After computing this optimal sequence, only the first control input is applied to the system, and the process is repeated at the next time step, incorporating the updated system state. By sampling at specific time points, a continuous state function can be discretized. Consider a system represented by the following nonlinear state-space model in discrete time:

$$\begin{aligned} \dot{x}(k) &= f(x(k), u(k)) \\ y(k) &= g(x(k)) \end{aligned} \tag{9}$$

Here, the functions $f(\cdot,\cdot)$ and $g(\cdot)$ are assumed to be continuously differentiable. The state vector $x(k)$ encompassing positions, angles, and velocities $[x, y, z, \phi, \theta, \psi, v_x, v_y, v_z, p, q, r] \in \mathbb{R}^n$ are the state variables, $u(k) \in \mathbb{R}^m$ are the control inputs, and $y(k) \in \mathbb{R}^p$ are the output variables. The equilibrium control input in the case of hover is $u_i = \frac{mg}{4}$. The nonlinear model predictive control (NMPC) hlmethod aims to stabilize the dynamic system described by $\dot{x} = f(x,u)$ such that it follows a reference trajectory $\bar{r}(t)$, by minimizing the cost function $\mathscr{L}(x,u)$. The NMPC problem can be expressed in the form of:

$$\begin{aligned} \min_{u} \quad & \int J(x,u) \quad (9) \\ \text{subject to} \quad & \dot{x} = f_{dyn}(x,u) \quad x(t_0) = x_{init} \\ & r(x,u) = 0 \\ & h(x,u) \leq 0 \end{aligned} \tag{10}$$

where x_0 denotes the initial condition and h, r are equality or inequality constraints like input limitations.

3.2. Linear MPC algorithm design

Denote the operating point as $[u_0(k), x_0(k), y_0(k)]$ such that the original condition of the system is:

$$\begin{aligned} f(x_0(k), u_0(k)) &= 0 \\ y_0(k) &= g(x_0(k)) \end{aligned} \tag{11}$$

with

$$A = \frac{\partial f}{\partial x}(x,u)\Big|_{x=x_0(k),u=u_0(k)}, B = \frac{\partial f}{\partial u}(x,u)\Big|_{x=x_0(k),u=u_0(k)}, C = \frac{\partial g}{\partial x}(x,u)\Big|_{x=x_0(k)} \tag{12}$$

And the corresponding linear time variant model is

$$\begin{aligned} x(k+1) &= A(k)x(k) + B(k)u(k), \\ y(k) &= C(k)x(k) \end{aligned} \tag{13}$$

For the above process model, the purpose of designing the controller is to minimize the cost function $\mathscr{J}$ and get the optimal input trajectory over the control horizon:

$$J(k) = \sum_{j=1}^{N} ||r(k+j|k) - y_p(k+j|k)||_Q^2 + \sum_{j=0}^{N_u-1} ||\Delta u(k+j|k)||_R^2, \tag{14}$$

This involves the weighting matrices $Q \in \mathbb{R}^{p\times p}$, semi-positive definite and $R \in \mathbb{R}^{m\times m}$, positive definite. The prediction of the future states of the control system is partially calculated by Equation (13): During the optimization process, $\bar{r}(k) = [r(k+1|k)\cdots r(k+N|k)]$ denotes the reference trajectory of the system at the sampling instant k, $\bar{y}_0(k) = [y_0(k+1|k)\cdots y_0(k+N|k)]$ denotes the fundamental component, $\bar{y}_p(k) = [y_p(k+1|k)\cdots y_p(k+N|k)]$ denotes the predicted output, and $\Delta\bar{u}(k) = [\Delta u(k|k)\cdots\Delta u(k+N_u-1|k)]$ denotes the increment of the control inputs sequence, where $\Delta u(k+j|k) = u(k+j|k) - u(k-1+j|k)$. The system employs a predictive horizon N where $1 \le N$ and a control horizon N_u subject to $0 < N_u \le N$, and the optimization process must also adhere to certain constraints:

$$u_{min} \le u(k+j|k) \le u_{max}, \quad j = 0,1,...,N_u-1, \tag{15}$$

$$\Delta u_{min} \le \Delta u(k+j|k) \le \Delta u_{max}, \quad j = 0,1,...,N_u-1, \tag{16}$$

$$y_{min} \le y(k+j|k) \le y_{max}, \quad j = 1,2,...,N. \tag{17}$$

Then the cost optimization problem can be expressed in the following form:

$$\begin{aligned} \min_{\Delta\bar{u}(k)} \quad & ||\bar{r}(k) - \bar{y}_0(k) - M(k)\Delta\bar{u}(k)||_Q^2 + ||\Delta\bar{u}(k)||_R^2, \\ \text{s.t.} \quad & -\Delta\bar{u}_{\max} \le \Delta\bar{u}(k) \le \Delta\bar{u}_{\max} \\ & \bar{u}_{\min} \le \bar{u}_0(k) + H\Delta\bar{u}(k) \le \bar{u}_{\max} \\ & \bar{y}_{\min} \le \bar{y}_0(k) + M(k)\Delta\bar{u}(k) \le \bar{y}_{\max} \\ & \dot{\delta}_x(k) = A(k)\delta_x(k) + B\delta_u(k) \end{aligned} \tag{18}$$

where $\delta_x(t)$ and $\delta_u(t)$ are system deviation variables and

$$H = \begin{bmatrix} I & 0 & \cdots & 0 \\ I & I & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ I & I & I & I \end{bmatrix} \in \Re^{N_u m \times N_u m} \tag{19}$$

$$M(k) = \begin{bmatrix} CB & \cdots & 0 \\ C(A+I)B & \cdots & 0 \\ \vdots & \ddots & \vdots \\ C(A^{N_u-1}+\cdots+I)B & \cdots & CB \\ C(A^{N_u}+\cdots+I)B & \cdots & C(A+I)B \\ \vdots & \ddots & \vdots \\ C(A^{N-1}+\cdots+I)B & \cdots & C(A^{N-N_u}+\cdots+I)B \end{bmatrix} \in \Re^{N_p \times N_u m} \tag{20}$$

The standard quadratic programming problem form is:

$$\begin{aligned} \min \quad & \frac{1}{2}\Delta U^T W \Delta U + c^T \Delta U \\ \text{s.t.} \quad & \Delta U_{\min} \le \Delta U \le \Delta U_{\max} \\ & G\Delta U \le b \end{aligned} \tag{21}$$

where the coefficient matrices and vectors are

$$\begin{aligned} \Delta U &= \Delta \bar{u}(k) \in \Re^{N_u m} \\ W &= 2(M(k)^T Q M(k) + R) \in \Re^{N_u m \times N_u m}, \\ c &= -2M(k)^T Q(\bar{r}(k) - \bar{y}_0(k)) \in \Re^{N_u m}, \\ G &= \begin{bmatrix} -H & H & -M(k) & M(k) \end{bmatrix}^T \in \Re^{(2N_p+2N_u m)\times N_u m} \\ b &= \begin{bmatrix} -\bar{u}_{\min} + \bar{u}_0(k) \\ \bar{u}_{\max} - \bar{u}_0(k) \\ -\bar{y}_{\min} + \bar{y}_0(k) \\ \bar{y}_{\max} - \bar{y}_0(k) \end{bmatrix} \in \Re^{2N_p+2N_u m} \end{aligned} \tag{22}$$

The positive definiteness of W ensures that the objective function (22) is strictly convex, and the optimization problem (21) possesses a unique solution that satisfies the KKT optimality conditions. Solving the quadratic programming problem (21) yields the control action vector $\Delta\bar{u}(k)$, which is subsequently used to determine the optimal control inputs.

3.3. Dynamic linearization of neural network

Since the nonlinear neural network-based MPC needs a high computational load, a neural linear model-based MPC is proposed. The idea of the dynamic linearization process is to extract a linear model from the trained neural network object $NN(\cdot)$ at each time instant k. The goal is to calculate the time-variant parameters $a_i(k)$ and $b_i(k)$ of the IVP system (7), and these parameters are computed by

first-order Taylor series approximation from the measurement of the system outputs:

$$a_n(k) = -\frac{\partial y(k)}{\partial y(k-i)}|_{\chi=\chi(k)}, b_n(k) = -\frac{\partial y(k)}{\partial u(k-i)}|_{\chi=\chi(k)} \tag{23}$$

where $\chi = [y(k), ..., y(k-N), u(k), ..., u(k-N_u)]^T$ is the regression vector, and the derivation can be derived from the back propagation process of the neural network [20].

Consider the nonlinear neural differential equation $\dot{x}(t) = f(x(t), u(t))$ where $f(\cdot)$ is the neural network which maps the state from the previous instant x_{k-1} and the current control input u_k to the current state derivative $\dot{x}_k$, and there exists a specific control input $\bar{u} \in \mathbb{R}^m$ such that $f(\bar{x}, \bar{u}) = 0$. Then define small perturbations near the equilibrium $\bar{x}$:

$$\delta_x(t) = x(t) - \bar{x}, \delta_u(t) = u(t) - \bar{u} \tag{24}$$

The Taylor expansion form of the NODE can be written as:

$$\dot{\delta}_x(t) \approx f(\bar{x}, \bar{u}) + \left.\frac{\partial f}{\partial x}\right|_{x=\bar{x}, u=\bar{u}} \delta_x(t) \quad + \quad \left.\frac{\partial f}{\partial u}\right|_{x=\bar{x}, u=\bar{u}} \delta_u(t) \tag{25}$$

This linearized, time-invariant differential equation effectively governs the deviation variables $\delta_x(t)$ and $\delta_u(t)$, provided they remain small. This approximation neglects higher-order terms. Consequently, the derivatives $de\dot{l}ta_x$ are expressed solely as linear combinations of the state deviations δ_x and input deviations δ_u. For more complicated situations, there are state equations in the form of:

$$\dot{x}(t) = f(x(t), u(t), d(t), t) \tag{26}$$

where $d(t)$ is the external disturbance that causes small deviations. In the real world, pre-planned input u is needed to correct for these deviations. Consequently, it is crucial for the model to capture how minor variations in the input, $\bar{u}(t) + \delta_u(t)$, and the disturbance, $\bar{d}(t) + \delta_d(t)$, influence the system's resulting trajectory. Denoting $x(t)$ as a deviation from $\bar{x}(t)$ and defining $\delta x(t) = x(t) - \bar{x}(t)$, we have $x(t) = \bar{x}(t) + \delta(t)$, now the real-world problem is:

$$\dot{\bar{x}}(t) + \dot{\delta}_x(t) = f(\bar{x}(t) + \delta_x(t), \bar{u}(t) + \delta_u(t), \bar{d}(t) + \delta_d(t), t) \tag{27}$$

and the disturbance $\dot{\delta}_x$ can be approximated:

$$\dot{\delta}_x(t) = \left.\frac{\partial f}{\partial x}\right|_{\substack{x(t)=\bar{x}(t) \\ u(t)=\bar{u}(t) \\ d(t)=\bar{d}(t)}} \delta_x(t) + \left.\frac{\partial f}{\partial u}\right|_{\substack{x(t)=\bar{x}(t) \\ u(t)=\bar{u}(t) \\ d(t)=\bar{d}(t)}} \delta_u(t) + \left.\frac{\partial f}{\partial d}\right|_{\substack{x(t)=\bar{x}(t) \\ u(t)=\bar{u}(t) \\ d(t)=\bar{d}(t)}} \delta_d(t) \tag{28}$$

The Jacobian matrix A, B_1, and B_2 can be extracted from the mapping relations of the neural network. Finally, the deviation variables can be approximately estimated:

$$\dot{\delta}_x(t) = A(t)\delta_x(t) + \begin{bmatrix} B_1(t) & B_2(t) \end{bmatrix} \begin{bmatrix} \delta_u(t) \\ \delta_d(t) \end{bmatrix} \tag{29}$$

This is the linearization of the system with respect to the trajectory $[\bar{x}(t), \bar{u}(t), \bar{d}(t)]$, which is a generalization of the linearization about a set of equilibrium points: the linearization about one equilibrium point can produce linear time-invariant (LTI) systems, while with the aid of matrices $A(t)$, $B_1(t)$ and

$B_2(t)$ consisting of time-varying elements, the linearization about trajectories can be expanded to a linear time-varying (LVI) system.

Figure 5 shows the whole NNMPC scheme based on the neural network method: The NN model is trained using the generated data X and U. NODEs are applied to capture the nonlinear dynamics of the quadrotor that might not be fully represented by the linear model, and then Jacobian matrices are extracted through the dynamic linearization process, providing a linearized model for designing the NNMPC controller. Finally, a Gaussian noise signal d, representing potential external disturbances, is added during the process to assess the effectiveness of the NNMPC controller.

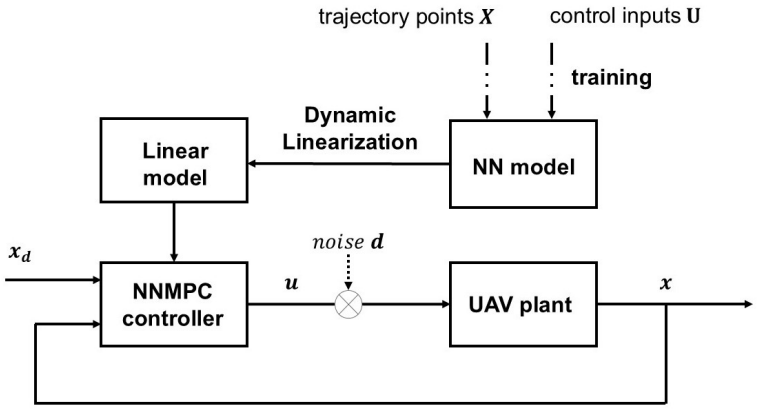

Figure 5. Block diagram of neural network approach to design the NNMPC controller.

4. Simulation study

In this section, the control performance of the proposed algorithm is illustrated by simulations, and all of these are trained and implemented in MATLAB. To qualitatively evaluate the developed NNMPC, the tracking flight is simulated using both the NMPC controller based on the N-E model and the NNMPC controller based on the trained neural state model with different types of trajectories.

4.1. Aerodynamic model

As is discussed in Section 2.2, the original linear accelerations $[\ddot{x}_o, \ddot{y}_o, \ddot{z}_o]^T$ are calculated from Equation (3) and angular accelerations $[\dot{p}_o, \dot{q}_o, \dot{r}_o]^T$ are calculated from equation (5), while a more complex aerodynamic model given by empirical equations can be used to simulate these impacts:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \ddot{x}_o \\ \ddot{y}_o \\ \ddot{z}_o \end{bmatrix} + A_k \begin{bmatrix} \dot{x} + \dot{x}|\dot{x}| \\ \dot{y} + \dot{y}|\dot{y}| \\ \dot{z} + \dot{z}|\dot{z}| + \sqrt{\dot{x}^2 + \dot{y}^2} + (\dot{x}^2 + \dot{y}^2) \end{bmatrix} \tag{30}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \dot{p}_o \\ \dot{q}_o \\ \dot{r}_o \end{bmatrix} + A_k \begin{bmatrix} \dot{y} + \dot{y}|\dot{y}| \\ \dot{x} + \dot{x}|\dot{x}| \\ \dot{x} + \dot{y} \end{bmatrix} \tag{31}$$

where A_k is the aerodynamic coefficient.

Table 1. Related quadrotor parameters.

Parameter	Value	Remark
g	9.81	m/s^2
m	2.0	kg
l	0.25	m
k	1	
b	0.2	
A_k	0.2	kg/s
I_{xx}	1.2	kg m^2
I_{yy}	1.2	kg m^2
I_{zz}	2.3	kg m^2

Related quadrotor parameters in equation (1) and (2) used in the simulation are listed in Table 1.

4.2. Neural state model training

For training the neural state model, raw data generated from the simulation experiments are divided into chronological groups as training data, each containing l consecutive trajectory points $x_i = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r]$, corresponding control inputs, u_i and timestamps k_i. These timestamps of trajectory points with the time span of $Ts * l$. During the training process of neural differential equations, modified akima interpolation is used to supplement discrete points x_i with approximate gradient information $\dot{\hat{x}}$ to provide smooth curves and avoid the Runge phenomenon [21].

Take sample 1 as an example, as is illustrated in Figure 6, sample 1 contains l trajectory points $[x_1, x_2, ..., x_l]$ and corresponding control inputs $[u_1, u_2, ..., u_l]$. These data are packed with a time span from 1 to $Ts * l$, where Ts is the sampling interval during simulation.

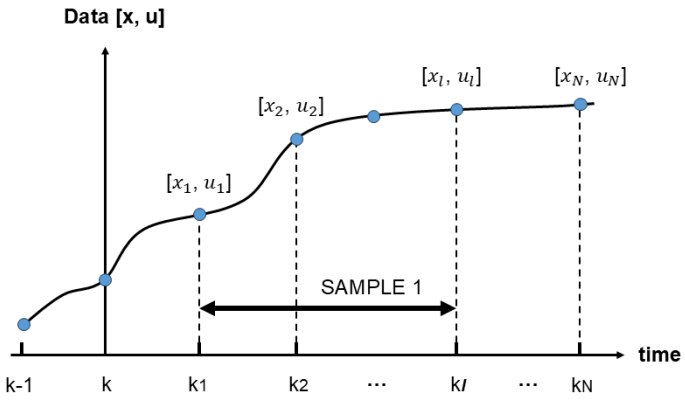

Figure 6. The structure of training data.

The loss function with the regularization term is given by:

$$\widehat{V}_N(\theta) = \frac{1}{N}\sum_{t=1}^{N} |\varepsilon(t,\theta)| + \frac{1}{N}\lambda||\theta||^2 \tag{32}$$

where t is the timestamp, N is the batch size, ε is the mean absolute loss, θ is a concatenated vector of weights and biases of the neural network, and λ is the regularization constant. During the modeling process, a MLP network characterizes the target neural ODE function, and the hidden layer size is set to be $[24, 32, 64, 128, 128, 128, 64, 32, 24]$ in the MATLAB Deep Learning Toolbox.

Figures 7 to 10 illustrates the neural network model prediction outputs during the training process. By treating data as continuous, neural ODEs more flexibly and accurately capture subtle trends over extended time horizons, and the neural network is capable of modeling the corresponding dynamic equations for different orders of magnitude of data from $1e^{-1}$ to $1e^{1}$ in training samples.

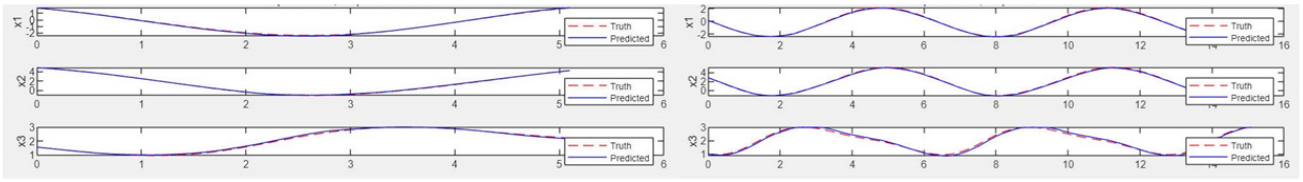

Figure 7. Validation results for position $[x, y, z]$.

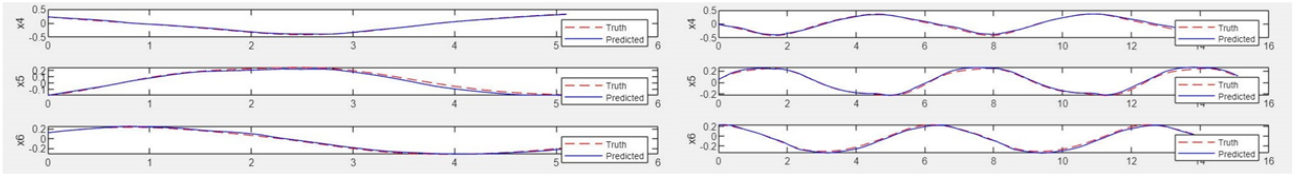

Figure 8. Validation results for angle $[\phi, \theta, \psi]$.

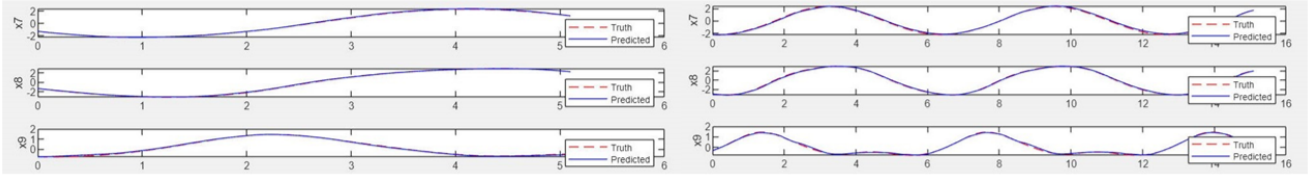

Figure 9. Validation results for linear velocity $[v_x, v_y, v_z]$.

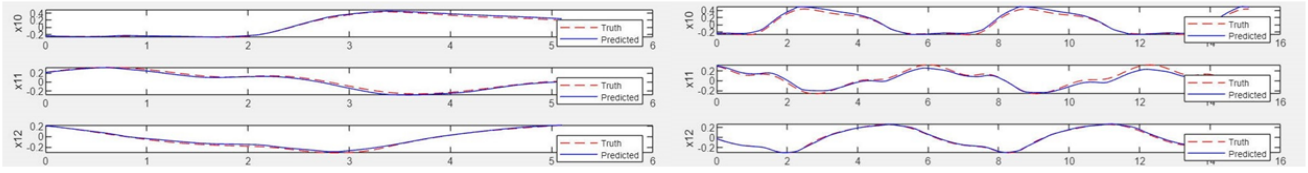

Figure 10. Validation results for angular velocity $[p, q, r]$.

Figure 11 shows that the MAE loss is drastically reduced during training.

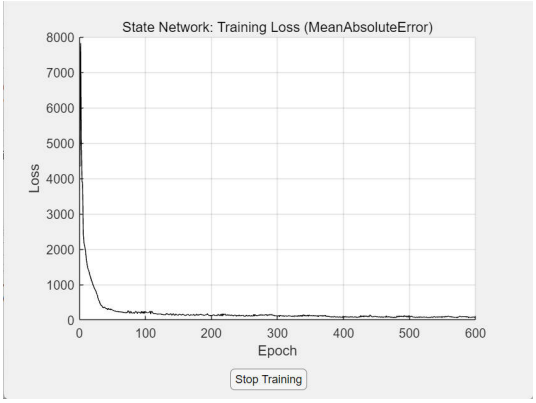

Figure 11. Trend of loss.

Figure 12 compares the prediction outputs of the neural state model with the measured position in the test data under the same control input u_{test}. The result indicates that the neural network model is able to predict the non-linear dynamic characteristics of the quadrotor with good accuracy for given inputs.

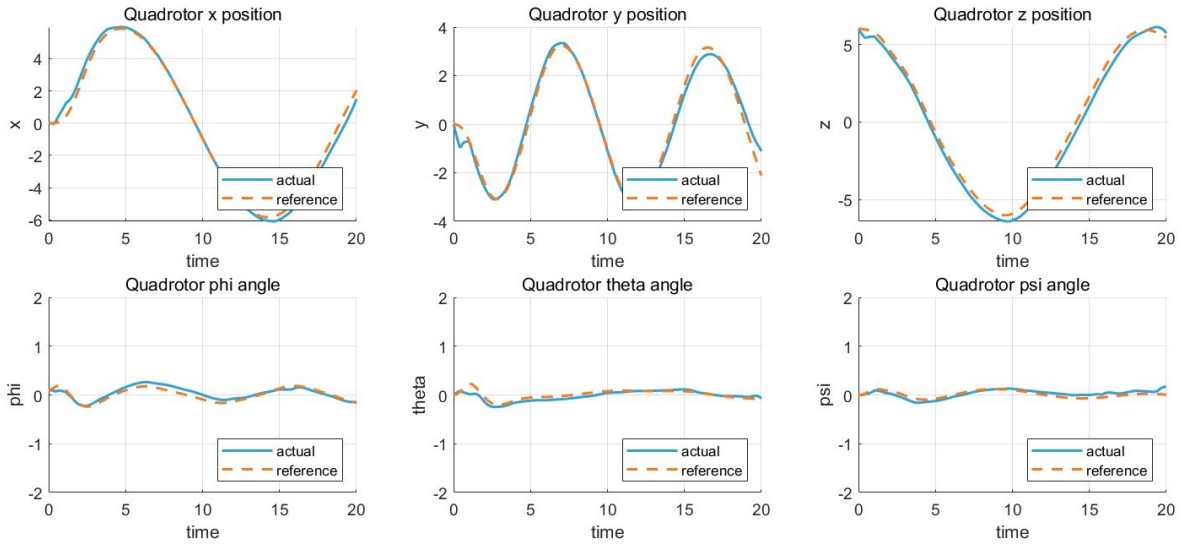

Figure 12. Comparison between neural model output and measured data.

4.3. MPC implement

The design and evaluation of the MPC controller focus on investigating two key aspects: (i) assessing the impact of the dynamics learned by the NNMPC within a closed-loop trajectory tracking scenario, and (ii) evaluating the performance of the NNMPC relative to conventional nonlinear MPC approaches. Related simulation parameters used in the MATLAB Control System Toolbox are listed in Table 2.

Table 2. Control parameters used in simulation.

Parameter	Value	Remark
T_s	0.1	sample time (s)
Q	diag[1.5, 5, 5, 1, 1, 1, 0, 0, 0, 0, 0]	weights over x_k
R	diag[0.1, 0.1, 0.1, 0.1]	weights over u_k

Each control input u_i is limited within $0 \leq u_i \leq 3$, which means that each rotor force is within $0 \leq f_i \leq 9N$, and the total thrust T is within $0 \leq T \leq 36N$. In the simulation, the UAV plant executes three different shapes of trajectories: orbital trajectory in the horizontal plane R_{d1} defined by $[x(t) = 3\sin(0.2\pi t); y(t) = 3\sin(0.2\pi t); z(t) = 3$,orbital trajectory in 3D space R_{d2} defined by $[x(t) = 3\sin(0.2\pi t); y(t) = 2 + 3\sin(0.2\pi t); z(t) = 2 + \cos(0.2\pi t)]$, and lemniscate in 3D space R_{d3} defined by $[x(t) = -6\sin(0.2\pi t); y(t) = -6\sin(0.2\pi t)\cos(0.2\pi t); z(t) = 6\cos(0.2\pi t)]$. These trajectories are illustrated in Figure 13.

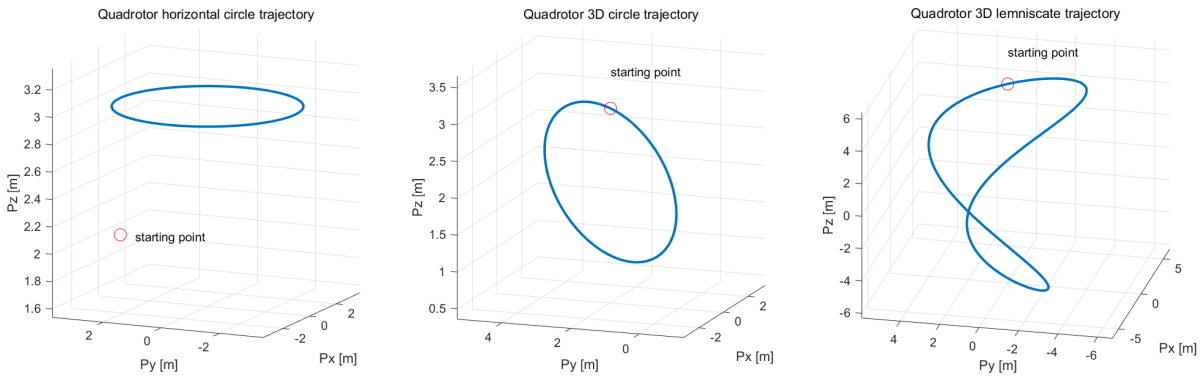

Figure 13. Applied trajectories in simulation, from left to right are R_{d1}, R_{d2} and R_{d3}.

4.3.1. Case 1: tracking performance

In this simulation, we evaluate the tracking performance of NNMPC with trajectory R_{d1}, the position $[x, y, z]$ and the attitude $[\phi, \theta, \psi]$ of the quadrotor are illustrated sequentially in the simulation result along the time axis.

It is observed in Figure 14 that the controller demonstrates good trajectory tracking capabilities in the horizontal plane $[x, y]$, as evidenced by the close alignment between the actual trajectory (blue solid line) and the desired trajectory (orange dashed line). Initially, the quadrotor needs to ascend to reach the target trajectory plane since it takes off from a lower height with an error of $1m$. Despite this initial climb, the controller successfully stabilizes the position, highlighting its effective tuning and ability to maintain a consistent height.

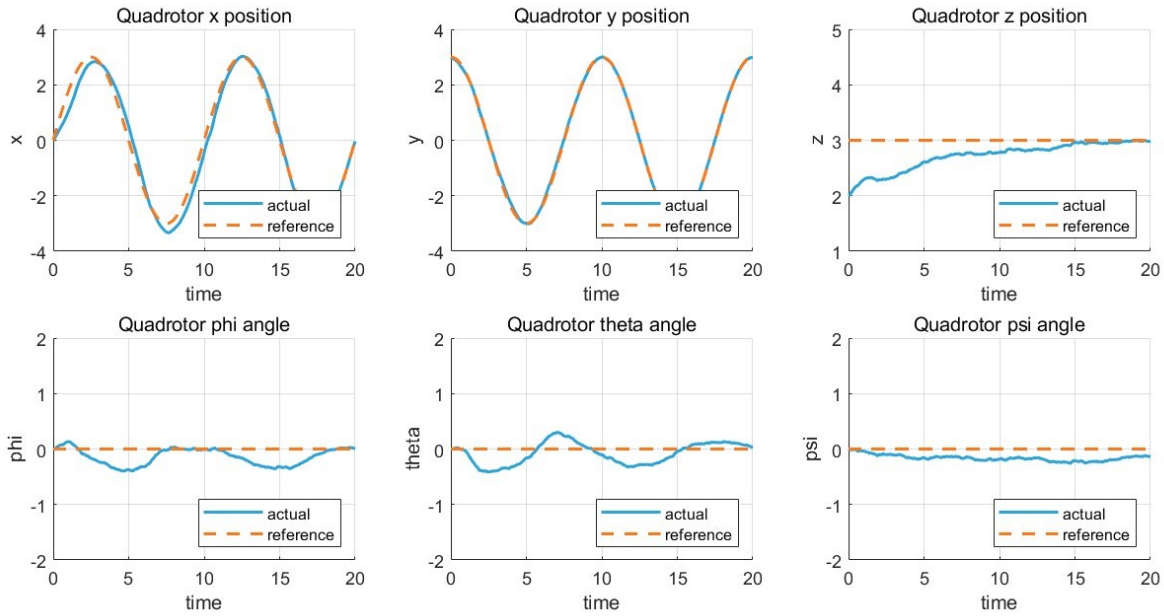

Figure 14. The tracking performance of trajectory R_{d1}

4.3.2. Case 2: influence of the sample length

For the orbital trajectory R_{d2} in 3D space, Figures 15 and 16 illustrate how the length l of each sample in the training data impacts the effectiveness of the NN model for the controller: when trained by shorter samples, as is shown in Figure 15, the model struggles to track the desired trajectory accurately, as seen in the tracking error exceeding 1.5 m between the actual output R_2 and reference path R_{d2}, particularly at the peaks and troughs of the trajectory.

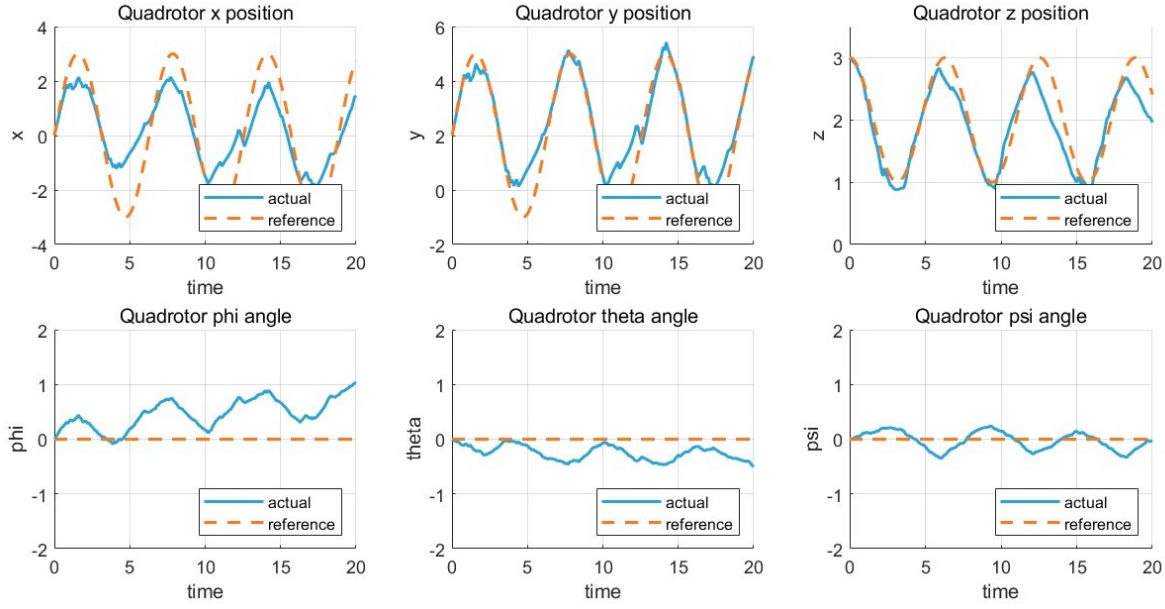

Figure 15. The tracking performance of trajectory R_{d2} with NN model trained with sample length $l = 51$.

By contrast, increasing the length of each training sample can significantly improve the model's

effectiveness. The predicted output $X = [x, y, z]$ aligns more closely with the actual positions, demonstrating better accuracy in all three dimensions. Furthermore, the quadrotor's attitude $\eta = [\phi, \theta, \psi]$ is also controlled more precisely.

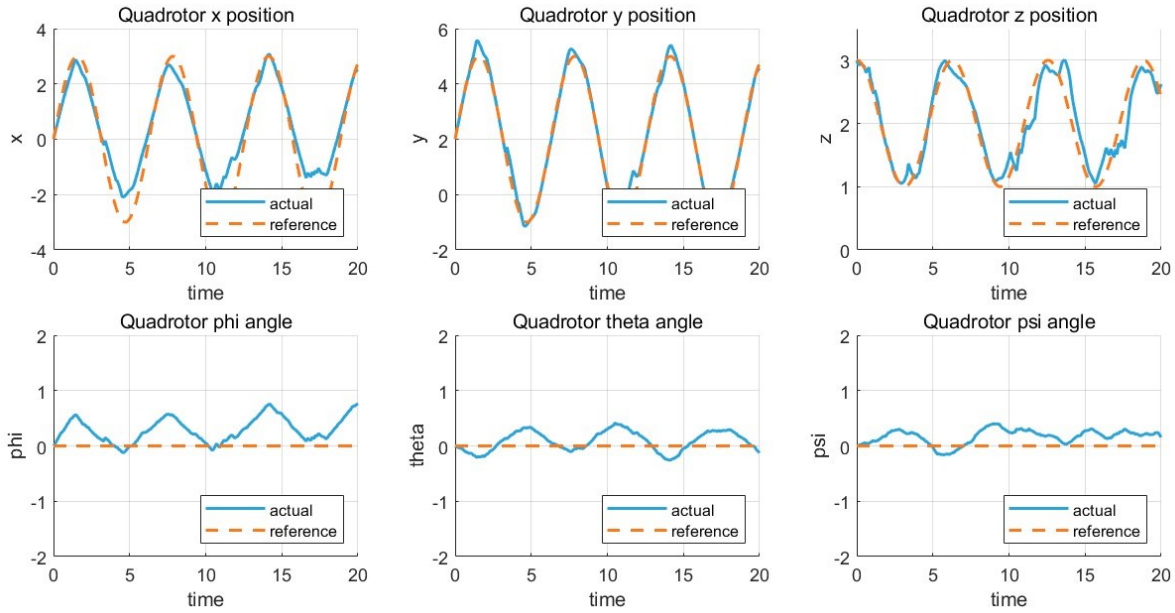

Figure 16. The tracking performance of trajectory R_{d2} with NN model trained with sample length l = 151.

4.3.3. Case 3: comparison between NMPC and NNMPC

For the lemniscate trajectory R_{d3} in 3D space, Figure 17 illustrates that for more complex nonlinear features like $\sin(t) * \cos(t)$, the NNMPC controller can also perform well, while Figure 18 shows the tracking performance of the NMPC controller on the same trajectory.

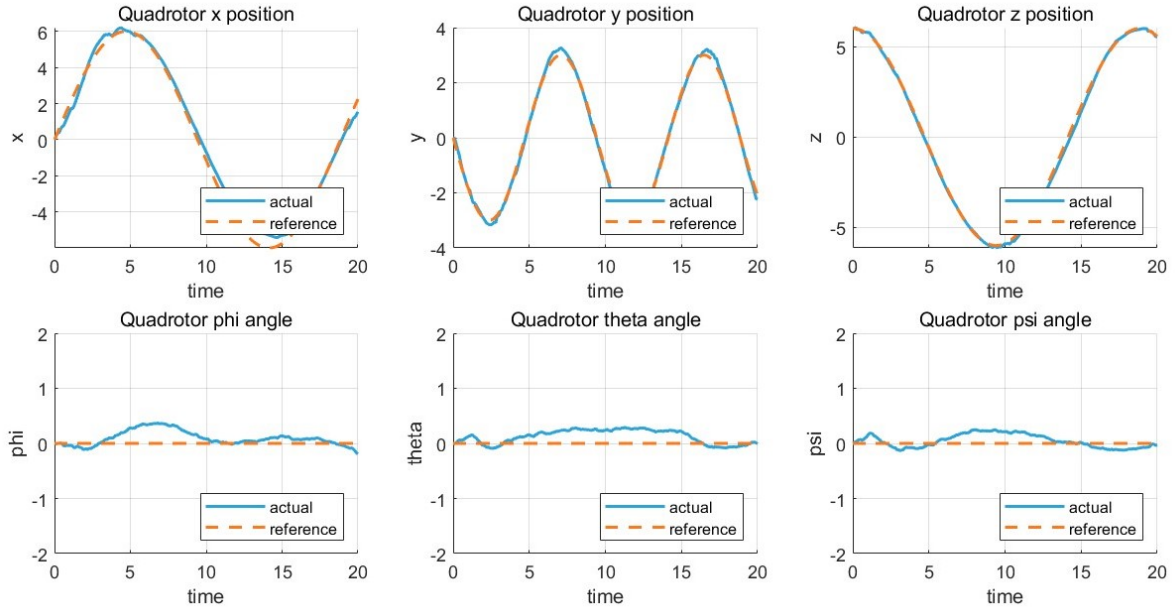

Figure 17. The tracking performance of NNMPC controller on trajectory R_{d3}.

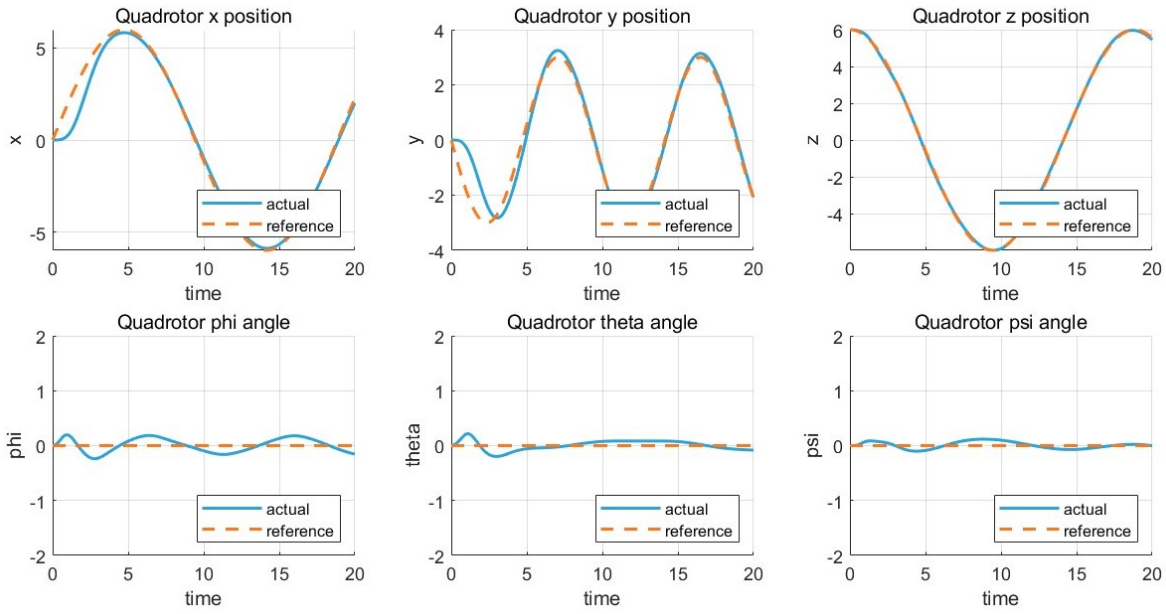

Figure 18. The performance of traditional NMPC controller on trajectory R_{d3}.

Figure 19 presents the tracking errors, specifically the norm of the position error $E_x = |X_d - X|$ and the attitude error $E_\eta = |\eta_d - \eta|$. A key observation, particularly within the initial five seconds, is the significantly faster response of the NNMPC compared to the NMPC, which exhibits a large initial position error (approximately 2 m). While the NNMPC trajectory shows minor oscillations after the initial phase, this behavior might stem from factors including the linearization process which simplifies dynamics by omitting higher-order information and the influence of inherent noise within the training data used to build the neural network model.

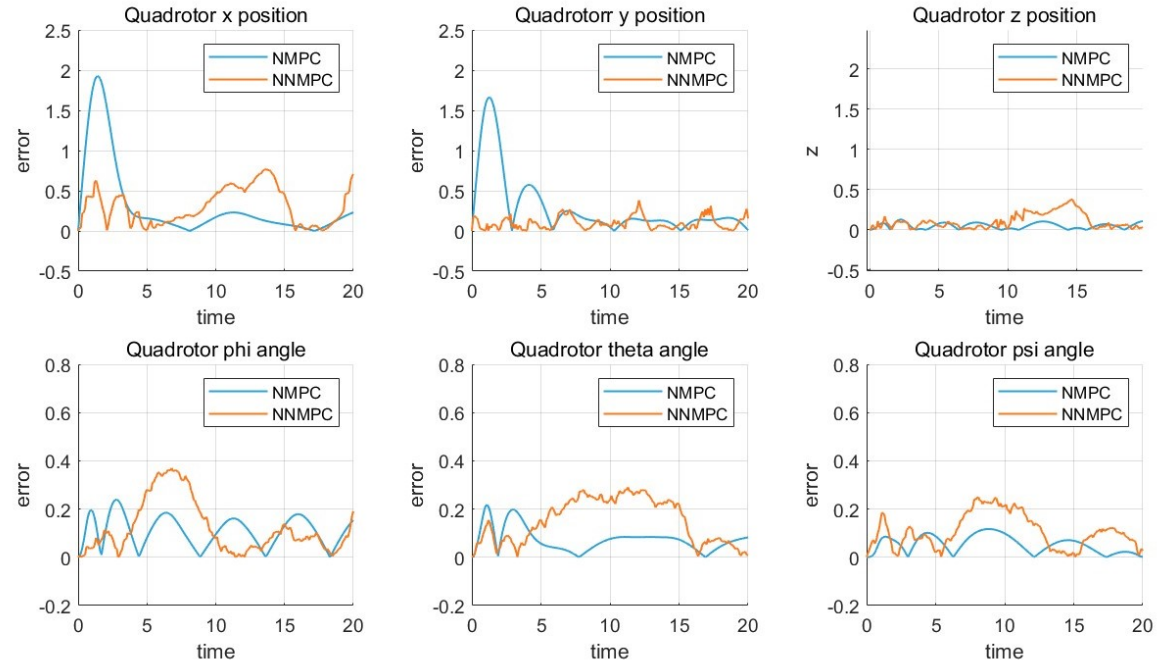

Figure 19. The absolute tracking error contrast between NMPC and NNMPC.

When applying the NNMPC controller, the best practice to improve computational efficiency is to implement Jacobian matrices A and B with time-varying elements defined in Equation (29). The tracking performance contrast between controllers with and without Jacobian linearization is shown in Table 3. It can be found that regardless of the type of desired trajectory R_d, the linearized NNMPC controller can reduce the calculation time by 40–50 percent while ensuring the tracking effect.

Table 3. Comparison of control computing time (unit: seconds).

Trajectory	With Jacobian Linearization	Without Jacobian Linearization
Horizontal Circle R_{d1}	287.758	656.801
Spatial Circle R_{d2}	342.340	616.832
Spatial Lemniscate R_{d3}	360.976	564.008

Moreover, linearized NNMPC demonstrates superior robustness compared to NMPC when subjected to external noise, successfully completing tracking tasks where NMPC failed. Therefore, the minor oscillations can be seen as a trade-off for achieving faster response times and enhanced robustness in more realistic, noisy conditions.

Based on the evaluations across the three test cases, the proposed NNMPC controller demonstrates several improved features: (1) the capability to effectively manage complex aerodynamic behaviors learned from data; (2) It can significantly reduce computational demands compared to standard NMPC; (3) Enhanced robustness against disturbances.

5. Conclusion

This study presents a neural network-based MPC controller based on the state model trained by neural ordinary differential equations and linearized with Jacobian matrices. In the proposed approach, the NNMPC controller is able to track trajectories with different nonlinear features and can reduce computation time by nearly half compared to the traditional nonlinear MPC method. Furthermore, this approach also exhibits enhanced robustness against disturbances. Its reliance on flight data allows it to implicitly learn and accommodate complex aerodynamic effects, bypassing the significant challenge of constructing accurate analytical aerodynamic models under varying flight conditions, making it a practical alternative to conventional techniques.

Some aspects are still expected to be explored in future work. For example, developing online training capabilities for the predictive model to allow adaptation to time-varying conditions such as when the self-weight of a drone changing during flight. Moreover, efforts need to be made to improve the algorithm's generalization ability to handle more complex real-world autonomous control scenarios.

Acknowledgments

The author would like to thank Dr. Huang Sunan and Prof. Xiang Cheng for their helpful guidance on the control algorithm and 5003 project.

Author's contribution

Conceptualization, Y.Z.; methodology, Y.Z.; validation, Y.Z.; formal analysis, Y.Z.; resources, Y.Z.; data curation, Y.Z.; writing, Y.Z. All authors have read and agreed to the published version of the manuscript.

Conflicts of interests

The authors declare no conflict of interest.

References

[1] Mohsan SAH, Khan MA, Noor F, Ullah I, Alsharif MH. Towards the unmanned aerial vehicles (UAVs): a comprehensive review. *Drones* 2022, 6(6):147.

[2] Huang S, Teo RSH, Leong WWL. Multi-camera networks for coverage control of drones. *Drones* 2022, 6(3):67.

[3] Huang S, Teo RSH, Tan KK. Collision avoidance of multi unmanned aerial vehicles: a review. *Annu. Rev. Control* 2019, 48:147–164.

[4] Kamel M, Stastny T, Alexis K, Siegwart R. Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system. In *Robot Operating System (ROS): The Complete Reference (Volume 2)*, 1st ed. Cham: Springer International Publishing, 2017. pp. 3–39.

[5] Richalet J, Rault A, Testud JL, Papon J. Model predictive heuristic control: applications to industrial processes. *Automatica* 1978, 14(5):413–428.

[6] Hang P, Huang S, Chen X, Tan KK. Path planning of collision avoidance for unmanned ground vehicles: a nonlinear model predictive control approach. *Proc. Inst. Mech. Eng., Part I: J. Syst. Control Eng.* 2021, 235(2):222–236.

[7] Zhang X, Ma J, Cheng Z, Huang S, Ge SS, *et al.* Trajectory generation by chance-constrained nonlinear MPC with probabilistic prediction. *IEEE Trans. Cybern.* 2021, 51(7):3616–3629.

[8] Cesari G, Schildbach G, Carvalho A, Borrelli F. Scenario model predictive control for lane change assistance and autonomous driving on highways. *IEEE Intell. Transp. Syst. Mag.* 2017, 9(3):23–35.

[9] Camacho EF, Bordons C. Nonlinear model predictive control: an introductory review. In *Assessment and Future Directions of Nonlinear Model Predictive Control*, 1st ed. Berlin: Springer Berlin Heidelberg, 2007. pp. 1–16.

[10] Zhou W, Chen S, Chang CW, Wen CY, Chen CK, *et al.* System identification and control for a tail-sitter unmanned aerial vehicle in the cruise flight. *IEEE Access* 2020, 8:218348–218359.

[11] Fay G. Derivation of the Aerodynamic Forces for the Mesicopter Simulation. Available: https://media.gradebuddy.com/documents/340594/59482841-8400-4d1d-890d-8932a64a24d1.pdf (accessed on 28 May 2025).

[12] Ru P, Subbarao K. Nonlinear model predictive control for unmanned aerial vehicles. *Aerospace* 2017, 4(2):31.

[13] Jiang B, Li B, Zhou W, Lo LY, Chen CK, *et al.* Neural network based model predictive control for a quadrotor UAV. *Aerospace* 2022, 9(8):460.

[14] Liu S, Wang J. A simplified dual neural network for quadratic programming with its KWTA application. *IEEE Trans. Neural Networks* 2006, 17(6):1500–1510.

[15] Bansal S, Akametalu AK, Jiang FJ, Laine F, Tomlin CJ. Learning quadrotor dynamics using neural network for flight control. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, Las Vegas, USA, December 12–14, 2016, pp. 4653–4660.

[16] Torrente G, Kaufmann E, Föhn P, Scaramuzza D. Data-driven MPC for quadrotors. *IEEE Rob. Autom. Lett.* 2021, 6(2):3769–3776.

[17] Zhang L, Huang S, Xiang C, Teo R, Srigrarom S, *et al.* AI-based adaptive nonlinear MPC for quadrotors. In *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, Chania, Greece, June 4–7, 2024, pp. 216–223.

[18] Chen RTQ, Rubanova Y, Bettencourt J, Duvenaud D. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Montréal, Canada, December 2–8, 2018, pp. 6572–6583.

[19] Panagiotou P, Kaparos P, Salpingidou C, Yakinthos K. Aerodynamic design of a MALE UAV. *Aerosp. Sci. Technol.* 2016, 50:127–138.

[20] Huang S, Tan KK, Lee TH. *Applied predictive control*, 1st ed. London: Springer London, 2013. pp. 1–264.

[21] Akima H. A new method of interpolation and smooth curve fitting based on local procedures. *J. ACM* 1970, 17(4):589–602.