Summary

Hanli Zhang

September 2023

Abstract

This work presents a novel framework for quadrotor trajectory tracking that compensates for unmodeled aerodynamic effects, particularly crucial in package delivery applications. Leveraging a layered control architecture, our approach employs a learned neural network model to accurately predict tracking errors due to asymmetric payloads, thereby enhancing the tracking controller's performance in dynamic and disturbance-rich environments.

1 Introduction

Quadrotors are increasingly employed in complex tasks where they encounter unmodeled aerodynamic forces, such as those caused by payloads, which can lead to substantial tracking errors. Traditional control methods often fall short in such scenarios, failing to compensate for the unpredictable nature of these forces. In response, we introduce a data-driven method that utilizes a neural network to learn cost functions reflective of these aerodynamic effects. The aim is to create a controller that can predict and minimize tracking errors, leading to more accurate and reliable quadrotor flight paths, especially in aerodynamically challenging environments.

2 Methodology

Our methodology combines the use of a neural network with a layered control architecture to learn a cost function that accurately captures the effects of aerodynamic disturbances on trajectory tracking. This cost function is derived from a dataset of trajectories, which is generated to include the impact of asymmetric aerodynamic drag. The neural network is trained using Stochastic Gradient Descent (SGD) and then integrated into a trajectory optimization process using Projected Gradient Descent (PGD). The optimized trajectories minimize the learned cost, resulting in improved tracking accuracy and robustness against aerodynamic disturbances. Our approach aims to create a predictive model that enhances the performance of the tracking controller under a variety of operating conditions.

3 Results and Discussion

3.1 What Worked

Our data collection and training pipeline works. We trained network model with SGD and incorporated into optimization via projected gradient descent.

Data Collection The generation of a dataset comprising 4000 trajectories to simulate asymmetric aerodynamic drag forces using RotorPy simulator was successful. Introducing asymmetric parasitic drag coefficients to simulate an oblong payload worked well to model real-world scenarios.

Training Pipeline Success The training pipeline was effective. It involved training a neural network model using Stochastic Gradient Descent (SGD) and incorporating it into optimization via projected gradient descent (PGD).

Training Details Network architecture with three hidden layers (500, 400, 200 units) was successful in capturing complex relationships. Training with a batch size of 32 facilitated efficient processing. Some network configurations did not yield successful learning outcomes. Using SGD with a learning rate of 0.001 was effective in updating network parameters. We observed the following training loss, as displayed in Tensorboard, as shown in Figure 1.

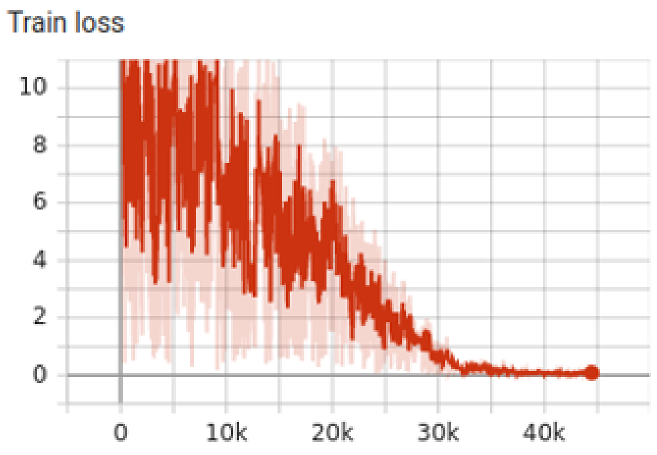


Figure 1: Train loss

Training the neural network over 2000 epochs allowed effective learning. The cost function that considered both predicted and actual tracking costs, including controller effort, was beneficial.

3.2 What Failed

During the inference phase, we observed two scenarios:

1. When the cost function includes contributions from both minimum snap and the neural network, the modified reference trajectory does not continue smoothly and exhibits a notably high tracking error.

2. However, when the cost function for pgd relies solely on the neural network without considering minimum snap costs, the modified reference trajectory remains smooth and uninterrupted. Comparatively, this approach yields a slight improvement in trajectory tracking errors when compared to minimum snap trajectories that do not account for aerodynamic forces, albeit with some variability in robustness.

Inference Visualization: When using the minimum jerk trajectory, the resulting trajectory is notably smooth. And it may be unfair to directly compare this trajectory with the initial trajectory generated from minimum snap:

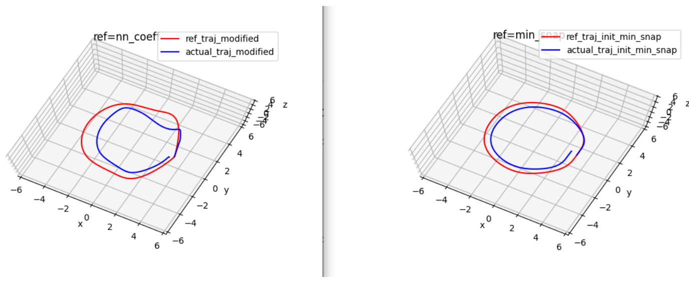


Figure 2: Comparison of Trajectories: Neural Network Modified from Minimum jerk

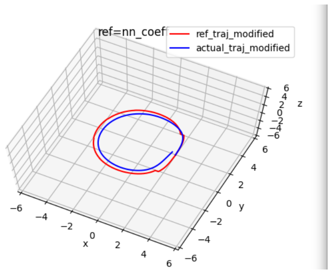
In the case of the minimum snap trajectory, the cost function is defined as follows:

$$\text{Cost} = \text{Cost from minimum snap} + \text{Cost from network} \tag{1}$$

The cost function combines terms from both the minimum snap approach and the neural network, as shown in Figure 3. Specifically, the cost from minimum snap is $\text{coeffs}^T \cdot \text{cost\_mat} \cdot \text{coeffs}$, while the cost from neural network is $\exp\left(\text{regularizer}\left(\text{reference points}\right)\right)$.

After conducting experiments where the cost associated with minimum snap is removed from the inference's cost function, and only the neural network cost is considered, we observed improvements in trajectory smoothness:

$$\text{Cost} = \text{Cost from minimum snap} \tag{2}$$



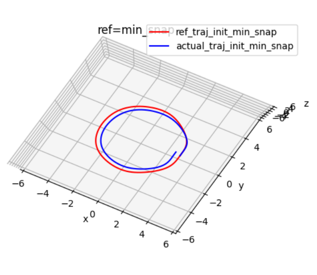
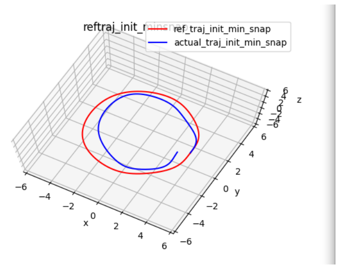


Figure 3: Comparison of Trajectories: Neural Network Modified from Minimum Snap (Cost from minimum snap and cost from network)



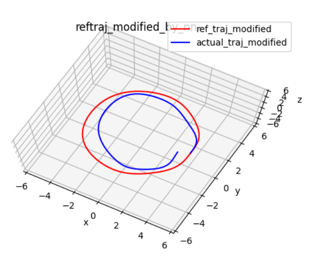
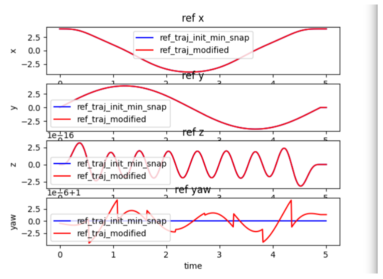


Figure 4: Comparison of Trajectories: Neural Network Modified from Minimum Snap(Cost from network)



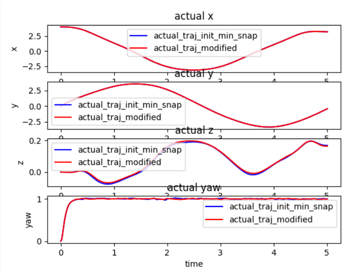


Figure 5: Comparison of different axis

These plots illustrate the differences between the trajectories generated with and without the influence of the minimum snap cost. Additionally, we analyzed the tracking error and control effort, and in some cases, observed improvements compared to the minimum snap trajectory:

Metric	**Value**
Tracking Error for Initial Minimum Snap	424.69
Input Trajectory Error for Initial Minimum Snap	32.20
Tracking Error for Neural Network Modified	428.58
Input Trajectory Error for Neural Network Modified	28.52

Table 1: Comparison of Tracking and Input Trajectory Errors

These results suggest that removing the minimum snap cost from the cost function can lead to smoother trajectories with potentially better performance, as indicated by reduced tracking error and input trajectory error.

4 Next Step

Here's a detailed description of what to do for the next steps:

Modify Value Function Linearize SE-3 Position Controller: Begin by exploring the possibility of representing the SE-3 (Spatial Euclidean-3) position controller as a linear system. This involves simplifying the controller equations and deriving a closed-form expression that approximates the control behavior in a linear fashion. This linearization process can significantly simplify the control problem and may make it more amenable to analysis and optimization.

Benchmark Using Neural Network: As an alternative or complementary approach, consider using a neural network to learn the cost-to-go for the linearized SE-3 controller. This neural network can serve as a benchmark for evaluating the performance of the linearized controller. It can capture complex relationships between inputs and desired outcomes, helping to fine-tune the control strategy.

Data Collection Expansion Controller Experiments: Conduct additional experiments with the controller to explore its behavior in more complex scenarios. This could involve scenarios with varying levels of aerodynamic drag compensation or introducing randomness, such as a yaw dot (yaw rate) random walk. These experiments can help assess the controller's robustness and performance under different conditions.

Data Loader for Large .npz Files: Develop a robust data loader capable of handling large .npz files. Large datasets are crucial for training and evaluating neural networks effectively. A well-designed data loader can efficiently process and manage these datasets, making it easier to scale up data collection efforts.

Minimum Snap Improvements Cost Matrix Optimization: Address issues related to the cost matrix used for minimum snap trajectory generation. Explore ways to optimize this cost matrix to better align with the desired trajectory characteristics. This may involve fine-tuning the cost weights or constraints to improve trajectory quality.

Constraint Relaxation: Consider relaxing certain constraints within the minimum snap trajectory generation algorithm. Relaxing constraints can sometimes lead to more flexible and smoother trajectory solutions while still meeting essential requirements.

Network Architecture Design and Fine-Tuning: Continue the process of designing and fine-tuning the neural network architecture. Experiment with different network architectures, including variations in the number of layers, units per layer, and activation functions. The goal is to enhance the network's ability to capture complex relationships between inputs and tracking costs.