DATT: Deep Adaptive Trajectory Tracking for Quadrotor Control

Anonymous Author(s)
Affiliation
Address
email

Abstract: Precise arbitrary trajectory tracking for quadrotors is challenging due to unknown nonlinear dynamics, trajectory infeasibility, and actuation limits. To tackle these challenges, we present DATT, a learning-based approach that can precisely track arbitrary, potentially infeasible trajectories in the presence of large disturbances in the real world. DATT builds on a novel feedforward-feedback-adaptive control structure trained in simulation using reinforcement learning. When deployed on real hardware, DATT is augmented with a disturbance estimator using $\mathcal{L}_1$ adaptive control in closed-loop, without any fine-tuning. DATT significantly outperforms competitive adaptive nonlinear and model predictive controllers for both feasible smooth and infeasible trajectories in unsteady wind fields, including challenging scenarios where baselines completely fail. Moreover, DATT can efficiently run online with an inference time less than $3.2\,\mathrm{ms}$, less than 1/4 of the adaptive nonlinear model predictive control baseline[1].

Keywords: Quadrotor, Reinforcement Learning, Adaptive Control

1 Introduction

Executing precise and agile flight maneuvers is important for the ongoing commoditization of uninhabited aerial vehicles (UAVs), in applications such as drone delivery, rescue and search, and urban air mobility. In particular, accurately following *arbitrary trajectories* with quadrotors is among the most notable challenges to precise flight control for the following reasons. First, the quadrotor dynamics are highly nonlinear and underactuated. Moreover, such nonlinearity is often hard to model due to unknown system parameters (e.g., motor characteristics) and uncertain environments (e.g., complex aerodynamics from unknown wind gusts). Second, aggressive trajectories demand operating at the limits of system performance, requiring awareness and proper handling of actuation constraints, especially for quadrotors with small thrust-to-weight ratios. Finally, the arbitrary desired trajectory might not be *dynamically feasible* (i.e., it is impossible to stay on such a trajectory), which necessities long-horizon reasoning and optimization in real-time. For instance, to stay close to the five-star trajectory in Fig. 1, which is infeasible due to the sharp changes of direction, the quadrotor must predict, plan, and react online before the sharp turns.

Traditionally, there are two commonly deployed control strategies for accurate trajectory following with quadrotors: nonlinear control based on differential flatness and model predictive control (MPC). However, nonlinear control methods, despite their proven stability and efficiency, are constrained to differentially flat trajectories (i.e., smooth trajectories with bounded velocity, acceleration, jerk, and snap) satisfying actuation constraints [1, 2, 3]. On the other hand, MPC approaches can potentially incorporate constraints and non-smooth arbitrary trajectories [4, 5], but their performances heavily rely on the accuracy of the model and the optimality of the solver for the underlying nonconvex optimization problems, which could also be expensive to run online.

Reinforcement learning (RL) has shown its potential flexibility and efficiency in trajectory tracking problems [6, 7, 8]. However, most existing works focus on tracking smooth trajectories in stationary

[1] Videos and demonstrations in `https://sites.google.com/view/deep-adaptive-traj-tracking` and code for experiments and analysis will be released upon acceptance.

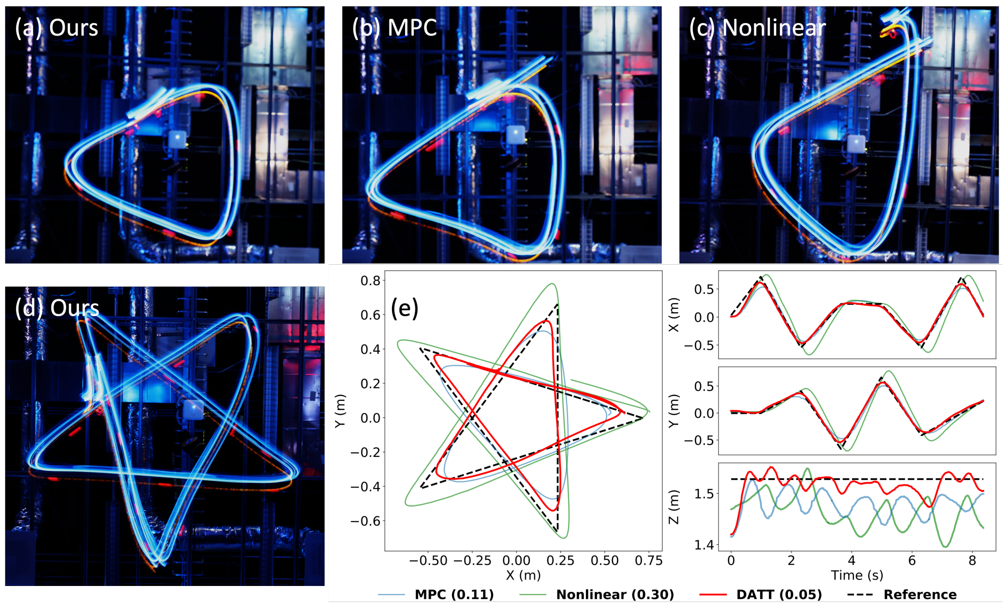

Figure 1: Trajectory visualizations for example infeasible trajectories. (a-c) Long-exposure photos of different methods for an equilateral triangle reference trajectory. (d) Long-exposure photo of our method for a five-pointed star reference trajectory. (e) Quantitative comparisons between our approach and baselines for the five-pointed star. Numbers indicate the tracking error in meters.

environments. In this work, we aim to design an RL-based flight controller that can (1) follow feasible trajectories as accurately as traditional nonlinear controllers and MPC approaches; (2) accurately follow arbitrary infeasible and dynamic trajectories to the limits of the hardware platform; and (3) adapt to unknown system parameters and uncertain environments online. Our contributions are:

- We propose DATT, a novel feedforward-feedback-adaptive policy architecture and training pipeline for RL-based controllers to track arbitrary trajectories. In training, this policy is conditioned on ground-truth translational disturbance in a simulator, and such a disturbance is estimated in real using $\mathcal{L}_1$ adaptive control in closed-loop;
- On a real, commercially available, lightweight, and open-sourced quadrotor platform (Crazyflie 2.1 with upgraded motors), we show that our approach can track feasible smooth trajectories with 27%-38% smaller errors than adaptive nonlinear or adaptive MPC baselines. Moreover, our approach can effectively track infeasible trajectories where the nonlinear baseline completely fails, with a 39% smaller error than MPC and 1/4th the computational time;
- On the real quadrotor platform, we show that our approach can adapt zero-shot to unseen turbulent wind fields with an extra cardboard drag plate for both smooth desired trajectories and infeasible trajectories. Specifically, for smooth trajectories, our method achieves up to 22% smaller errors than the state-of-the-art adaptive nonlinear control method. In the most challenging scenario (infeasible trajectories with wind and drag plate), our method significantly outperforms the adaptive MPC approach with 15% less error and 1/4th of the computation time.

2 Problem Statement and Related Work

2.1 Problem Statement

In this paper, we let $\dot{\boldsymbol{x}}$ denote the derivative of a continuous variable $\boldsymbol{x}$ regarding time. We consider the following quadrotor dynamics:

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \qquad m\dot{\boldsymbol{v}} = m\boldsymbol{g} + \boldsymbol{R}\boldsymbol{e}_3 f_\Sigma + \boldsymbol{d} \tag{1a}$$

$$\dot{\boldsymbol{R}} = \boldsymbol{R}S(\boldsymbol{\omega}), \qquad \boldsymbol{J}\dot{\boldsymbol{\omega}} = \boldsymbol{J}\boldsymbol{\omega} \times \boldsymbol{\omega} + \boldsymbol{\tau}, \tag{1b}$$

where $\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{g} \in \mathbb{R}^3$ are position, velocity, and gravity vectors in the world frame, $\boldsymbol{R} \in \mathrm{SO}(3)$ is the attitude rotation matrix, $\boldsymbol{\omega} \in \boldsymbol{R}^3$ is the angular velocity in the body frame, $m, \boldsymbol{J}$ are mass and inertia matrix, $\boldsymbol{e}_3 = [0; 0; 1]$, and $S(\cdot) : \mathbb{R}^3 \to \mathrm{so}(3)$ maps a vector to its skew-symmetric matrix form. Moreover, $\boldsymbol{d}$ is the time-variant translational disturbance, which includes parameter mismatch (e.g., mass error) and environmental perturbation (e.g., wind perturbation) [9, 10, 11, 12]. The control input is the total thrust f_Σ and the torque $\boldsymbol{\tau}$ in the body frame. For quadrotors, there is a linear invertible actuation matrix between $[f_\Sigma; \boldsymbol{\tau}]$ and four motor speeds.

We let $\boldsymbol{x}_t$ denote the temporal discretization of $\boldsymbol{x}$ at time step $t \in \mathbb{Z}_+$. In this work, we focus on the 3-D trajectory tracking problem with the desired trajectory $\boldsymbol{p}^d_1, \boldsymbol{p}^d_2, \cdots, \boldsymbol{p}^d_T$, with average tracking error as the performance metric: $\frac{1}{T}\sum_{t=1}^{T} \|\boldsymbol{p}_t - \boldsymbol{p}^d_t\|$. We do not have any assumptions on the desired trajectory $\boldsymbol{p}^d$. In particular, $\boldsymbol{p}^d$ is not necessarily differentiable or smooth.

2.2 Differential Flatness

The differential flatness property of quadrotors allows efficient generation of control inputs to follow smooth trajectories [1, 5]. Differential flatness has been extended to account for unknown linear disturbances [3], learned nonlinear disturbances [13], and also to deal with the singularities associated with pitching and rolling past 90 degrees [14]. While differential-flatness-based methods can show impressive performance for smooth and aggressive trajectories, they struggle with nondifferentiable trajectories or trajectories that require reasoning about actuation constraints.

2.3 Model Predictive Control (MPC)

Optimal control is a powerful methodology for achieving precise trajectory tracking in robotics by minimizing a cost function that quantifies the deviation from the desired path. MPC is a widely used optimal control approach that online optimizes control inputs over a finite time horizon, considering system dynamics and constraints [15].

Model Predictive Path Integral Control (MPPI) [4, 16] is a sampling-based MPC incorporating path integral control formulation and stochastic sampling. Unlike deterministic optimization, MPPI employs a stochastic optimization approach where control sequences are sampled from a distribution. These samples are then evaluated based on a cost function, and the distribution is iteratively updated to improve control performance. Recently MPPI has been applied to quadrotor control [17, 18].

Gradient-based nonlinear MPC techniques have been widely used for rotary-winged-based flying robots or drones. Hanover et al. [12] and Sun et al. [5] have shown good performance of nonlinear MPC in agile trajectory tracking of drones and adaptation to external perturbations. Moreover, these techniques are being used for vision-based agile maneuvers of drones [19, 7].

However, for either sampling-based or gradient-based MPC, the control performance heavily relies on the optimality of the optimizer for the underlying nonconvex problems. Generally speaking, MPC-based approaches require much more computing than differential-flatness-based methods [5]. Moreover, MPC's robustness and adaptability for infeasible trajectories remain unclear since existing works consider smooth trajectory tracking. In this paper, we implemented MPPI [4] and $\mathcal{L}_1$ augmented MPPI [17] for our baselines.

2.4 Adaptive Control and Disturbance Estimation

Adaptive controllers aim to improve control performance through online estimation of unknown system parameters in closed-loop. For quadrotors, adaptive controllers typically estimate a three-dimensional force disturbance $\boldsymbol{d}$ [20, 10, 21, 22, 17]. Most recently, $\mathcal{L}_1$ adaptive control for quadrotors [11] has been shown to improve trajectory tracking performance in the presence of complex and time-varying disturbances such as sloshing payloads and mismatched propellers. Recently, deep-learning-based adaptive flight controllers have also emerged [10, 23].

Learning dynamical models is a common technique to improve quadrotor trajectory tracking performance [9, 10, 24, 25] and can provide more accurate disturbance estimates than purely reactive adaptive control, due to the model of the disturbance over the state and control space. In this work, we use the disturbance estimation from $\mathcal{L}_1$ adaptive control, but we notice that our method can leverage any disturbance estimation or model learning techniques.

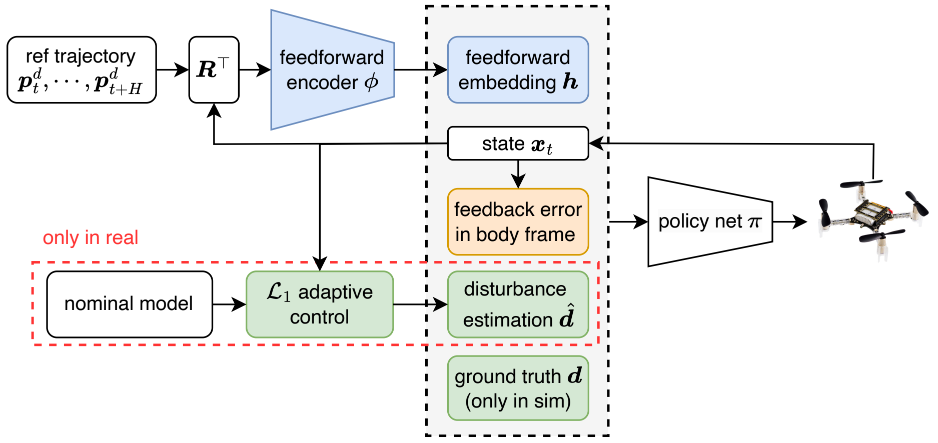
ref trajectory $p_t^d, \cdots, p_{t+H}^d$
$R^\top$
feedforward encoder ϕ
feedforward embedding h
state x_t
feedback error in body frame
policy net π
only in real
nominal model
$\mathcal{L}_1$ adaptive control
disturbance estimation $\hat{d}$
ground truth d (only in sim)

Figure 2: Algorithm Overview. Blue, yellow, and green blocks represent feedforward, feedback, and adaptation modules respectively. In training the policy has access to the true disturbance $\boldsymbol{d}$ whereas in real we use $\mathcal{L}_1$ adaptive control to get the disturbance estimation $\hat{\boldsymbol{d}}$ in closed-loop.

In particular, Rapid Motor Adaptation (RMA) is a supervised learning-based approach that aims to predict environmental parameters using a history of state-action pairs, which are then inputted to the controller [26]. This approach has been shown to work for real legged-robots, but can be susceptible to domain shift during sim2real transfer on drones.

2.5 Reinforcement Learning for Quadrotor Control

Reinforcement learning for quadrotor stabilization is studied in [6, 27, 23]. Molchanov et al. [27] uses domain randomization to show policy transfer between multiple quadrotors. Kaufmann et al. [28] compares three different policy formulations for quadrotor trajectory tracking and finds that outputting body thrust and body rates outperforms outputting desired linear velocities and individual rotor thrusts. [28] only focuses on feasible trajectories while in this work, we aim to track infeasible trajectories as accurately as possible. Simulation-based learning with imitation learning to an expert MPC controller is used to generate acrobatic maneuvers in [7]. In this work, we focus on trajectories and environments for which obtaining an accurate expert even in simulation is difficult or expensive and thus use reinforcement learning to learn the controller.

3 Methods

3.1 Algorithm Overview

A high-level overview of DATT is given in Fig. 2. Using model-free RL, DATT learn a neural network quadrotor controller $\boldsymbol{\pi}$ capable of tracking arbitrary reference trajectories, including infeasible trajectories, while being able to adapt to various environmental disturbances, even those unseen during training. We condition our policy on a learned *feedforward embedding* $\boldsymbol{h}$, which encodes the desired reference trajectory, in the body frame, over a fixed time horizon, as well as the force disturbance $\boldsymbol{d}$ in Eq. (1). We also input the position feedback error in the body frame to our controller.
The state $\boldsymbol{x}_t$ consists of the position $\boldsymbol{p}$, the velocity $\boldsymbol{v}$, and the orientation $\boldsymbol{R}$, represented as a quaternion $\boldsymbol{q}$. We convert $\boldsymbol{p}, \boldsymbol{v}$ to the body frame and input them to $\boldsymbol{\pi}$. Our policy controller outputs $\boldsymbol{u}$ which includes the desired total thrust $f_{\Sigma,\text{des}}$, and the desired body rates $\boldsymbol{\omega}_{\text{des}}$. In summary, our controller functions as follows:

$$\boldsymbol{h}_t = \boldsymbol{\phi}(\boldsymbol{R}_t^\top(\boldsymbol{p}_t - \boldsymbol{p}_t^d)), \ldots, \boldsymbol{R}_t^\top(\boldsymbol{p}_t - \boldsymbol{p}_{t+H}^d)) \tag{2a}$$
$$\boldsymbol{u}_t = \boldsymbol{\pi}(\boldsymbol{R}_t^\top \boldsymbol{p}_t, \boldsymbol{R}_t^\top \boldsymbol{v}_t, \boldsymbol{q}_t, \boldsymbol{h}_t, \boldsymbol{R}_t^\top(\boldsymbol{p}_t - \boldsymbol{p}_t^d), \boldsymbol{d}_t) \tag{2b}$$

We define the expected reward for our policy conditioned on the reference trajectory as follows:

$$J(\boldsymbol{\pi}|\boldsymbol{p}^d_{t:t+H}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{u})\sim\boldsymbol{\pi}}\left[\sum_{t=0}^{\infty} r(\boldsymbol{x}_t, \boldsymbol{u}_t|\boldsymbol{p}^d_{t:t+H})\right] \tag{3a}$$

$$r(\boldsymbol{x}_t, \boldsymbol{u}_t|\boldsymbol{p}^d_{t:t+H}) = \|\boldsymbol{p}_t - \boldsymbol{p}^d_t\| + 0.5\|\psi_t\| + 0.1\|\boldsymbol{v}_t\| \tag{3b}$$

ψ_t denotes the yaw of the drone. The reward function optimizes for accurate position and yaw tracking, with a small velocity regularization penalty. $\boldsymbol{\pi}$ and $\boldsymbol{\phi}$ are jointly optimized with respect to J using the Proximal Policy Optimization (PPO) algorithm [29].

3.2 Arbitrary Trajectory Tracking

Classical controllers, such as differential-flatness controllers, rely on higher-order position derivatives of the reference trajectory for accurate tracking (velocity, acceleration, jerk, and snap), which are needed for incorporating future information about the reference, i.e., feedforward control. However, arbitrary trajectories can have undefined higher order derivatives, and exact tracking may not be feasible. With RL, a controller can be learned to optimally track an arbitrary reference trajectory, given just the desired future positions $\boldsymbol{p}^d_t$. Thus, we input just the desired positions into a feedforward encoder $\boldsymbol{\phi}$, which learns the feedforward embedding that contains the information of the desired future reference positions. For simplicity, we assume the desired yaw for all trajectories is zero. The reference positions are provided evenly spaced from the current time t to the feedfoward horizon $t + H$, and are transformed into the body frame.

3.3 Adaptation to Disturbance

During training in simulation, we add a constant force perturbation $\boldsymbol{d}$ to the environment, which is randomized at the start of each episode. The policy is conditioned on the ground truth value of $\boldsymbol{d}$ during training. During inference in the real world, we use $\mathcal{L}_1$ adaptive control [11] to estimate $\boldsymbol{d}$, which is directly passed into our policy network. The adaptation law is given by:

$$\dot{\hat{\boldsymbol{v}}} = \boldsymbol{g} + \boldsymbol{R}\boldsymbol{e}_3 f_\Sigma/m + \hat{\boldsymbol{d}}/m + \boldsymbol{A}_s(\hat{\boldsymbol{v}} - \boldsymbol{v}) \tag{4a}$$

$$\hat{\boldsymbol{d}}_{\text{new}} = -(e^{\boldsymbol{A}_s dt} - \boldsymbol{I})^{-1}\boldsymbol{A}_s e^{\boldsymbol{A}_s dt}(\hat{\boldsymbol{v}} - \boldsymbol{v}) \tag{4b}$$

$$\hat{\boldsymbol{d}} \leftarrow \text{low pass filter}(\hat{\boldsymbol{d}}, \hat{\boldsymbol{d}}_{\text{new}}) \tag{4c}$$

where $\boldsymbol{A}_s$ is a Hurwitz matrix, dt is the discretization step length and $\hat{\boldsymbol{v}}$ is the velocity prediction. Generally speaking, (4a) is a velocity predictor using the estimated disturbance $\hat{\boldsymbol{d}}$, and (4b) and (4c) update and filter $\hat{\boldsymbol{d}}$. Compared to other sim-to-real techniques such as domain randomization [27] and student-teacher adaptation [23], the adaptive-control-based disturbance adaptation method in DATT tends to be more reactive and robust, thanks to the closed-loop nature and provable stability and convergence of $\mathcal{L}_1$ adaptive control.

4 Experiments

4.1 Simulation and Training

Training is done in a custom quadrotor simulator that implements (1) using on-manifold integration, with body thrust and angular velocity as the inputs to the system. In order to convert the desired body thrust $f_{\Sigma,\text{des}}$ and body rate $\boldsymbol{\omega}_{\text{des}}$ output from the controller to the actual thrust and body rate for the drone in simulation, we use a first-order time delay model:

$$\boldsymbol{\omega}_t = \boldsymbol{\omega}_{t-1} + k(\boldsymbol{\omega}_{\text{des}} - \boldsymbol{\omega}_{t-1}) \tag{5a}$$

$$f_{\Sigma,t} = f_{\Sigma,t-1} + k(f_{\Sigma,\text{des}} - f_{\Sigma,t-1}) \tag{5b}$$

We set k to a fixed value of 0.4, which we found worked well on the real drone. In practice, the algorithm generalizes well to a large range of k, even when training on fixed k. Our simulator effectively runs at 50 Hz, with $dt = 0.02$ for each simulation step.

We train across a series of xy-planar smooth and infeasible reference trajectories. The smooth trajectories are randomized degree-five polynomials and series of degree-five polynomials chained together. The infeasible trajectories are we refer to as *zigzag trajectories*, which are trajectories that linearly connect a series of random waypoints, and have either zero or undefined acceleration. The average speed of the infeasible trajectories is approximately $2\,\mathrm{m/s}$. See Appendix C for more details on the reference trajectories.

During each episode, we apply a constant force perturbation $\boldsymbol{d}$ with randomized direction and strength in the range of $[-3.5\,\mathrm{m/s^2}, 3.5\,\mathrm{m/s^2}]$, representing translational disturbances. Randomization occurs only at the start of each episode. We run each episode for a total of 500 steps, corresponding to 10 seconds. By default, we set H to $0.6\,\mathrm{s}$ with 10 feedforward reference terms. In Appendix A, we show ablation results for various different horizons.

We also note that stable training and best performance require fixing an initial trajectory for the first 2.5M steps of training (see Appendix A for more details). Only after that initial time period do we begin randomizing the trajectory. We train the policy using PPO for a total of 20M steps. Training takes slightly over 3 hours on an NVIDIA 3080 GPU.

4.2 Hardware Setup and the Low-level Attitude Rate Controller

We conduct hardware experiments with the Bitcraze Crazyflie 2.1 equipped with the longer $20\,\mathrm{mm}$ motors from the thrust upgrade bundle for more agility. The quadrotor as tested weighs $40\,\mathrm{g}$ and has a thrust-to-weight ratio of slightly under 2.

Position and velocity state estimation feedback is provided by the OptiTrack motion capture system at $50\,\mathrm{Hz}$ to an offboard computer that runs the controller. The Crazyflie quadrotor provides orientation estimates via a $2.4\,\mathrm{GHz}$ radio and control commands are sent to the quadrotor over the same radio at $50\,\mathrm{Hz}$. Communication with the drone is handled using the Crazyswarm API [30]. Body rate commands $\boldsymbol{\omega}_{\text{des}}$ received by the drone are converted to torque commands $\boldsymbol{\tau}$ using a custom low-level PI attitude rate controller on the firmware: $\boldsymbol{\tau} = -K_P^{\boldsymbol{\omega}}(\boldsymbol{\omega} - \boldsymbol{\omega}_{\text{des}}) - K_I^{\boldsymbol{\omega}} \int(\boldsymbol{\omega} - \boldsymbol{\omega}_{\text{des}})$. Finally, this torque command and the desired total thrust $f_{\Sigma,\text{des}}$ from the RL policy are converted to motor thrusts using the invertible actuation matrix.

4.3 Baselines

We compare our reinforcement learning approach against two nonlinear baselines: differential flatness-based feedback control and sampling-based Model Predictive Control (MPC) [4].

Nonlinear Tracking Controller and $\mathcal{L}_1$ Adaptive Control The differential flatness-based controller baseline consists of a PID position controller, which computes a desired acceleration vector, and a tilt-prioritized nonlinear attitude controller, which computes the body thrust f_Σ and desired body angular velocity $\boldsymbol{\omega}_{\text{des}}$.

$$\boldsymbol{a}_{\text{fb}} = -K_P(\boldsymbol{p} - \boldsymbol{p}^d) - K_D(\boldsymbol{v} - \boldsymbol{v}^d) - K_I \int (\boldsymbol{p} - \boldsymbol{p}^d) + \boldsymbol{a}^d - \boldsymbol{g} - \hat{\boldsymbol{d}}/m, \tag{6a}$$

$$\boldsymbol{z}_{\text{fb}} = \frac{\boldsymbol{a}_{\text{fb}}}{||\boldsymbol{a}_{\text{fb}}||}, \quad \boldsymbol{z} = \boldsymbol{R}\boldsymbol{e}_3, \quad f_\Sigma = \boldsymbol{a}_{\text{fb}}^\top \boldsymbol{z} \tag{6b}$$

$$\boldsymbol{\omega}_{\text{des}} = -K_R \boldsymbol{z}_{\text{fb}} \times \boldsymbol{z} + \psi_{\text{fb}} \boldsymbol{z}, \quad \psi_{\text{fb}} = -K_{\text{yaw}}(\psi \ominus \psi_{\text{ref}}) \tag{6c}$$

where $\hat{\boldsymbol{d}}$ is the disturbance estimation. For the nonlinear baseline, we set $\hat{\boldsymbol{d}} = 0$, and for $\mathcal{L}_1$ adaptive control [11] we use (4) to compute $\hat{\boldsymbol{d}}$ in real time [11]. For our experiments, we set $K_P = \text{diag}([6\ 6\ 6])$, $K_I = \text{diag}([1.5\ 1.5\ 1.5])$, $K_D = \text{diag}([4\ 4\ 4])$, $K_R = \text{diag}([120\ 120\ 0])$, and $K_{\text{yaw}} = 13.75$. PID gains were empirically tuned on the hardware platform to track both smooth and infeasible trajectories while minimizing crashes.

Nonlinear MPC and Adaptive Nonlinear MPC We use Model Predictive Path Integral (MPPI) [4] control as our second nonlinear baseline. MPPI is a sampling-based nonlinear optimal control technique that computes the optimal control sequence w.r.t. a known dynamics model and specified cost function. In our implementation, we use (1) ($\boldsymbol{d} = 0$) as the dynamics model with the body thrust f_Σ and angular velocity $\boldsymbol{\omega}$ as the control input. The cost function is the sum of the position

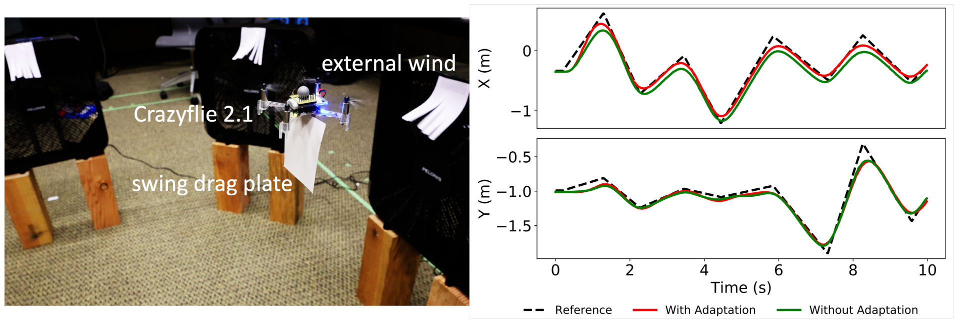

Figure 3: **Left**: Crazyflie 2.1 with a swinging cardboard drag plate in an unsteady wind field. **Right**: Comparison between our methods with and without adaptation with the drag plate on a zigzag trajectory. With wind added, adaptation is needed, otherwise the drone crashes.

error norms along $k = 40$ horizon steps. We use 8192 samples, $dt = 0.02$, and a temperature of 0.05 for the softmax. For adaptive MPC, similar to prior works [17, 12], we augment the standard MPPI with the disturbance estimation $\hat{\boldsymbol{d}}$ from $\mathcal{L}_1$ adaptive control, which we refer to as $\mathcal{L}_1$-MPC.

4.4 Arbitrary Trajectory Tracking

We first evaluate the trajectory tracking performance of DATT compared to the baselines in the absence of disturbances. We test on both infeasible zigzag trajectories and smooth polynomial trajectories. Each controller is run 2 times on the same bank of 10 random zigzag trajectories and 10 random polynomials.

Results are shown in Table 1. For completeness, we also compare with the tracking performance of adaptive controllers in the absence of any distrubances. We also compare our method to a version without adaptation, meaning that we enforce $\hat{\boldsymbol{d}} = \mathbf{0}$.

Arbitrary trajectory tracking without external disturbances

Method	Smooth trajectory	Infeasible trajectory	Inference time (ms)
Nonlinear tracking control	0.098 ± 0.012	*crash*	0.21
$\mathcal{L}_1$ adaptive control	0.091 ± 0.009	*crash*	0.93
MPC	0.104 ± 0.009	0.183 ± 0.027	12.62
$\mathcal{L}_1$-MPC	0.088 ± 0.010	0.181 ± 0.031	13.10
DATT (w/ $\hat{\boldsymbol{d}} = 0$)	$\mathbf{0.065} \pm 0.014$	$\mathbf{0.110} \pm 0.024$	2.41
DATT	$\mathbf{0.064} \pm 0.015$	$\mathbf{0.112} \pm 0.028$	3.17

Table 1: Tracking error (in m) of DATT vs. baselines, without any environmental disturbances (no wind or plate). *crash* indicates a crash for all ten trajectory seeds.

We see that DATT achieves the most accurate tracking, with a fraction of the compute cost of MPC. As expected, the addition of adaptive control does little in this setting, as there are no environmental disturbances introduced. With our current gains, the nonlinear and $\mathcal{L}_1$ adaptive control baselines are unable to track the infeasible trajectory. With reduced controller gains, it is possible these controllers would not crash when tracking the infeasible trajectories, but doing so would greatly decrease their performance for smooth trajectories. In contrast, our method works well across all trajectories, with no fine tuning required.

4.5 Adaptation Performance in Unknown Wind Fields with a Drag Plate

To evaluate the ability of DATT to compensate for unknown disturbances, we test the Crazyflie in a high wind scenario with three fans and an attached soft cardboard plate hanging below the vehicle body. Figure 3 shows this experimental setup. We note that this setup differs significantly from simulation — the placement of the fans and the soft cardboard plate creates highly dynamic and state dependent force disturbances, as well as torque disturbances, yet in simulation we only model a constant force disturbance. However, our policy is able to generalize well zero-shot to this domain, as shown in Table 2.

Arbitrary trajectory tracking with external disturbances

Method	Smooth traj. w/ plate	Smooth traj. w/ plate & wind	Infeasible traj. w/ plate	Infeasible traj. w/ plate & wind
$\mathcal{L}_1$ adaptive control	0.163 ± 0.013	0.184 ± 0.020	*crash*	*crash*
$\mathcal{L}_1$-MPC	$\mathbf{0.121} \pm 0.010$	0.181 ± 0.04	0.216 ± 0.028	0.243 ± 0.026
DATT (w/ $\hat{\boldsymbol{d}} = 0$)	0.161 ± 0.021	0.173 ± 0.026	0.194 ± 0.037	*crash*
DATT	$\mathbf{0.127} \pm 0.039$	$\mathbf{0.146} \pm 0.083$	$\mathbf{0.171} \pm 0.052$	$\mathbf{0.206} \pm 0.011$

Table 2: Tracking error (in m) of DATT vs. baselines, with an attached plate and/or wind. Results are effectively for zero-shot generalization, as we do not model a plate, non-constant force disturbances, or torque disturbances in simulation.

In Table 2, we see that the baseline nonlinear adaptive controller is unable to track infeasible trajectories, similar to the experiment without adaptation. While setting $\hat{\boldsymbol{d}} = \mathbf{0}$ still allows our policy controller to track some trajectories with disturbances, it crashes for infeasible trajectories with both the plate and wind, where the disturbances are the highest. However, our method with adaptation enabled is able to track all the trajectories tested, with the lowest tracking error. Figure 3 shows the difference in tracking performance between our method using adaptive control and our method without, on an example zigzag trajectory with a drag plate. We see that our approach of integrating $\mathcal{L}_1$ adaptive control with our policy controller is effective in correcting the error introduced by the presence of the turbulent wind field and plate. Our method performs better than $\mathcal{L}_1$-MPC without any knowledge of the target domain, and with a fraction of the compute cost. Figures 5 and 6 in the Appendix visualizes the tracking performance of DATT vs. $\mathcal{L}_1$-MPC on a infeasible and smooth trajectory, respectively.

5 Limitations and Future Work

Our choice of hardware presents some inherent limitations. The relatively low thrust-to-weight ratio of the Crazyflie (less than 2) means that we are unable to fly very agile or aggressive trajectories on the real drone or perform complex maneuvers such as a drone flip mid-trajectory. For this reason, we focused on xy-planar trajectories in this paper, and did not vary the z direction, although our method also improves the performance in the z direction (Fig. 1). However, our method provides the framework for performing accurate tracking for any trajectory, as we note we are able to perform a much larger range of agile maneuvers in simulation, including flips.

Our simulator is only an approximation of the true dynamics. For example, we model the lower-level angular velocity controller with a simplified first-order time delay model, which limits sim2real generalization for very agile tasks. Furthermore, we only model a constant force disturbance in sim, which does not model the highly time- and state-dependent force and torque disturbances the drone can encounter in reality. With better modeling, we could likely greatly improve our performance on the plate and wind task. However, we show that we can already achieve good zero-shot generalization to a highly dynamic environment and challenging tasks.

Reinforcement learning also has drawbacks compared to classical methods. We note that our training process has fairly high variance and can be sensitive to the hyperparameters of the PPO algorithm. As seen in Appendix A, we use a few tricks for stable learning, including fixing the reference trajectory for the first 2.5M training steps. Future work is needed to understand the role of these architectural and training features and help inform the best algorithm design and training setup.

Finally, for disturbance estimation, more sophisticated modeling techniques can be used that better predict the acceleration disturbance as a function of state and action. For adaptive control, end-to-end learning methods that optimize a learned adaptation model for trajectory tracking performance can perhaps do better than the $\mathcal{L}_1$ baseline we use here.

References

[1] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2520–2525. IEEE, 2011. URL http://ieeexplore.ieee.org/abstract/document/5980409/.

[2] T. Lee, M. Leok, and N. H. McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *49th IEEE conference on decision and control (CDC)*, pages 5420–5425. IEEE, 2010.

[3] M. Faessler, A. Franchi, and D. Scaramuzza. Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories. *IEEE Robotics and Automation Letters*, 3(2):620–626, Apr. 2018. ISSN 2377-3766, 2377-3774. doi:10.1109/LRA.2017.2776353. URL http://arxiv.org/abs/1712.02402. arXiv: 1712.02402.

[4] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[5] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza. A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight. *IEEE Transactions on Robotics*, 38(6):3357–3373, 2022. doi:10.1109/TRO.2022.3177279.

[6] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a Quadrotor with Reinforcement Learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, Oct. 2017. ISSN 2377-3766, 2377-3774. doi:10.1109/LRA.2017.2720851. URL http://arxiv.org/abs/1707.05110. arXiv:1707.05110 [cs].

[7] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza. Deep Drone Acrobatics. In *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, July 2020. ISBN 978-0-9923747-6-1. doi:10.15607/RSS.2020.XVI.040. URL http://www.roboticsproceedings.org/rss16/p040.pdf.

[8] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis. Optimal and autonomous control using reinforcement learning: A survey. *IEEE transactions on neural networks and learning systems*, 29(6):2042–2062, 2017.

[9] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural Lander: Stable Drone Landing Control using Learned Dynamics. *2019 International Conference on Robotics and Automation (ICRA)*, pages 9784–9790, May 2019. doi:10.1109/ICRA.2019.8794351. URL http://arxiv.org/abs/1811.08027. arXiv: 1811.08027.

[10] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural-fly enables rapid learning for agile flight in strong winds. *Science Robotics*, 7(66): eabm6597, 2022.

[11] Z. Wu, S. Cheng, P. Zhao, A. Gahlawat, K. A. Ackerman, A. Lakshmanan, C. Yang, J. Yu, and N. Hovakimyan. $\mathcal{L}_1$ quad: $\mathcal{L}_1$ adaptive augmentation of geometric control for agile quadrotors with performance guarantees. *arXiv preprint arXiv:2302.07208*, 2023.

[12] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza. Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors. *IEEE Robotics and Automation Letters*, 7 (2):690–697, 2022. doi:10.1109/LRA.2021.3131690.

[13] A. Spitzer and N. Michael. Inverting Learned Dynamics Models for Aggressive Multirotor Control. In *Robotics: Science and Systems XV*. Robotics: Science and Systems Foundation, June 2019. ISBN 978-0-9923747-5-4. doi:10.15607/RSS.2019.XV.065. URL http://www.roboticsproceedings.org/rss15/p65.pdf. arXiv: 1905.13441.

[14] B. Morrell, M. Rigter, G. Merewether, R. Reid, R. Thakker, T. Tzanetos, V. Rajur, and G. Chamitoff. Differential Flatness Transformations for Aggressive Quadrotor Flight. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5204–5210, Brisbane, QLD, May 2018. IEEE. ISBN 978-1-5386-3081-5. doi:10.1109/ICRA.2018.8460838. URL `https://ieeexplore.ieee.org/document/8460838/`.

[15] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer science & business media, 2013.

[16] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[17] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou. $\mathcal{L}_1$-adaptive mppi architecture for robust and agile control of multirotors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7661–7666, 2020. doi:10.1109/IROS45743.2020.9341154.

[18] K. Lee, J. Gibson, and E. A. Theodorou. Aggressive perception-aware navigation using deep optical flow dynamics and pixelmpc. *IEEE Robotics and Automation Letters*, 5(2):1207–1214, 2020. doi:10.1109/LRA.2020.2965911.

[19] Y. Zhang, W. Wang, P. Huang, and Z. Jiang. Monocular vision-based sense and avoid of uav using nonlinear model predictive control. *Robotica*, 37(9):1582–1594, 2019. doi:10.1017/S0263574719000158.

[20] B. Michini and J. How. L1 Adaptive Control for Indoor Autonomous Vehicles: Design Process and Flight Testing. In *Proceeding of AIAA Guidance, Navigation, and Control Conference*, pages 5754–5768, 2009. URL `https://arc.aiaa.org/doi/pdf/10.2514/6.2009-5754`.

[21] C. D. McKinnon and A. P. Schoellig. Unscented external force and torque estimation for quadrotors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5651–5657, Daejeon, South Korea, Oct. 2016. IEEE. ISBN 978-1-5090-3762-9. doi:10.1109/IROS.2016.7759831. URL `http://ieeexplore.ieee.org/document/7759831/`.

[22] E. Tal and S. Karaman. Accurate Tracking of Aggressive Quadrotor Trajectories using Incremental Nonlinear Dynamic Inversion and Differential Flatness. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4282–4288, Miami Beach, FL, Dec. 2018. IEEE. ISBN 978-1-5386-1395-5. doi:10.1109/CDC.2018.8619621. URL `https://arxiv.org/abs/1809.04048`. ISSN: 0743-1546.

[23] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller. A zero-shot adaptive quadcopter controller. *arXiv preprint arXiv:2209.09232*, 2022.

[24] G. Torrente, E. Kaufmann, P. Foehn, and D. Scaramuzza. Data-Driven MPC for Quadrotors. *IEEE Robotics and Automation Letters*, 2021. ISSN 2377-3766, 2377-3774. doi:10.1109/LRA.2021.3061307. URL `http://arxiv.org/abs/2102.05773`. arXiv: 2102.05773.

[25] A. Spitzer and N. Michael. Feedback Linearization for Quadrotors with a Learned Acceleration Error Model. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6042–6048, May 2021. doi:10.1109/ICRA48506.2021.9561708. URL `https://ieeexplore.ieee.org/document/9561708`. ISSN: 2577-087X.

[26] A. Kumar, Z. Fu, D. Pathak, and J. Malik. RMA: Rapid Motor Adaptation for Legged Robots, July 2021. URL `http://arxiv.org/abs/2107.04034`. arXiv:2107.04034 [cs].

[27] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme. Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors. *arXiv:1903.04628 [cs]*, Apr. 2019. URL http://arxiv.org/abs/1903.04628. arXiv: 1903.04628.

[28] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight, Feb. 2022. URL http://arxiv.org/abs/2202.10796. arXiv:2202.10796 [cs].

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[30] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3299–3304, 2017. doi:10.1109/ICRA.2017.7989376.

[31] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

A Ablations

Ablation	Tracking error (sim) (m)
No body frame	*failed*
No fixed intial reference	0.437 ± 0.08
No feedback term	0.077 ± 0.011
Feedforward horizon 1 ($H = 0.02s$)	*failed*
Feedforward horizon 5 ($H = 0.3s$)	0.240 ± 0.008
Feedforward horizon 10 ($H = 0.6s$) (used in main experiments)	0.055 ± 0.007
Feedforward horizon 15 ($H = 0.9s$)	0.073 ± 0.010
Feedforward horizon 20 ($H = 1.2s$)	0.101 ± 0.018
Base policy (no ablation)	0.046

Table 3: Tracking error (in m), in simulation, of various ablations after 15M training steps. *Failed* indicates the drone diverges from the reference trajectory. Tracking error is with respect to infeasible zigzag trajectories. The ablations are done without adaptation, and with no disturbances in the environment. 5 runs were attempted for each ablation.

We test various ablations of our primary method, with results shown in Table 3. In particular, we test

- **No body frame**: With our training setup, we found that transforming all state inputs (except for the orientation) into the body frame was necessary for accurate trajectory tracking. This ablation tests our method, but with the position $\boldsymbol{p}$, velocity $\boldsymbol{v}$, and reference positions in the world frame instead of the body frame.
- **No fixed initial reference** This ablation removes the initial 2.5M training steps where we do not randomize the reference trajectory. We see that PPO converges to a much worse tracking performance. We note that the choice of the initial fixed reference does not have much impact on the variance of training, only the existence of the fixed reference.
- **No feedback term** We remove the feedback term $\boldsymbol{R}^\top(\boldsymbol{p}_t - \boldsymbol{p}_t^d)$ from our controller inputs. This term might appear redundant with the reference trajectory, but we find explicitly conditioning on the feedback error consistently results in slightly more accurate tracking.
- **Feedforward horizon** We test varying sizes of our feedforward horizon. In Table 3, Feedforward horizon N refers to passing in N future reference positions. As described in Section 3.2, we linearly space the N reference positions across time from t to $t + H$.
- **Base policy** For comparison, we list the tracking error in sim of the main policy that we use in our experiments section.

B Training Details and Network Architecture

Training is done with the PPO implementation in the Stable Baselines3 library [31]. All PPO parameters are left as default.

The feedforward encoder architecture consists of 3 1-D convolution layers with ReLU activations that project the reference positions into a 32-dim representation for input to the main policy. Each 1-D convolution has 16 filters with a kernel size of 3. The main policy network is a 3-layer MLP with 64 neurons per layer and ReLU activations, and the value network shares this structure.

C Reference Trajectory Details

C.1 Smooth Trajectory

For smooth trajectories, we include a mix of degree 5 polynomials and *chained polynomials*. Polynomials start at $x = 0$ and $y = 0$, and return to the origin after $10\,\mathrm{s}$, corresponding to our episode length. They are randomly generated by randomly selecting initial and end conditions. Chained

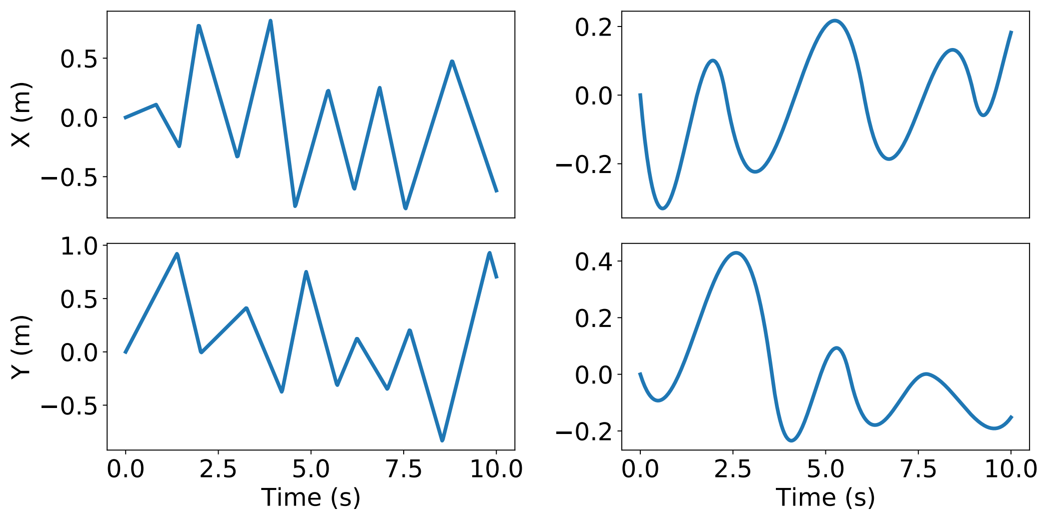

Figure 4: **Left:** Example of a random zigzag trajectory (infeasible). **Right:** Example of a random chained polynomial trajectory (smooth).

polynomials are a series of random polynomials. We generate these trajectories by randomly selecting ”nodes” at $x = 0$ and $y = 0$ at random times between $0\,\mathrm{s}$ and $10\,\mathrm{s}$, and fitting degree 5 polynomials between each node, ensuring that first, second, and third order derivatives are continuous at each node. Note that these trajectories are not guaranteed to be feasible, although in practice they are easy to track as they are highly smooth.

C.2 Infeasible Trajectory

We use a class of what we refer to as *zigzag trajectories*. We generate these trajectories by randomly selecting time intervals between 0.5 and 1.5 seconds, randomly generating waypoints after each time interval, and linearly connecting each waypoint. The waypoints can vary from $-1\,\mathrm{m}$ to $1\,\mathrm{m}$ in both the x and y directions. By training on these zigzags, we are able to generalize well to a wide variety of trajectories, including polygons and stars as seen in Figure 1, which are similar to random zigzags.

C.3 Additional Figures of Results

We show additional figures from our results from Table 2. Figure 7 shows the values of the predicted $\hat{\boldsymbol{d}}$ over time on an environment with wind versus one without wind. Our method is able to incorporate disturbance estimates of varying frequency despite only being trained with a constant disturbance per training episode. Figure 5 and Figure 6 show our tracking performance against $\mathcal{L}_1$-MPC for a smooth and infeasible trajectory, respectively.

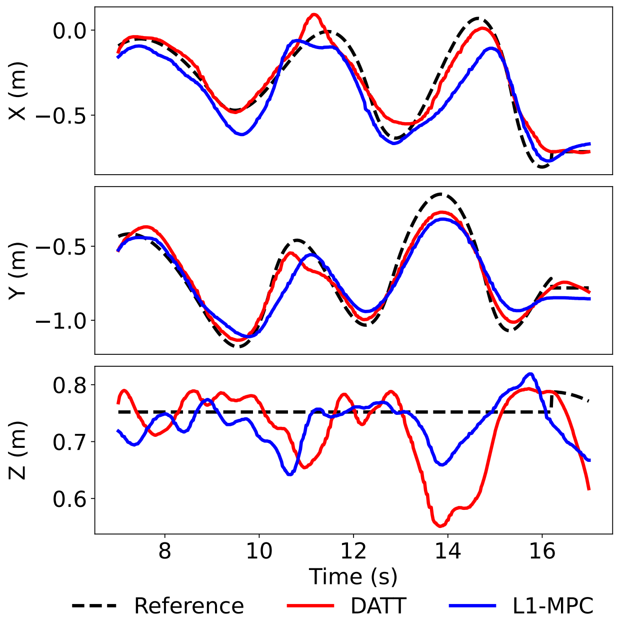

Figure 5: Performance of DATT against $\mathcal{L}_1$-MPC on a smooth trajectory with both wind and a plate attached.

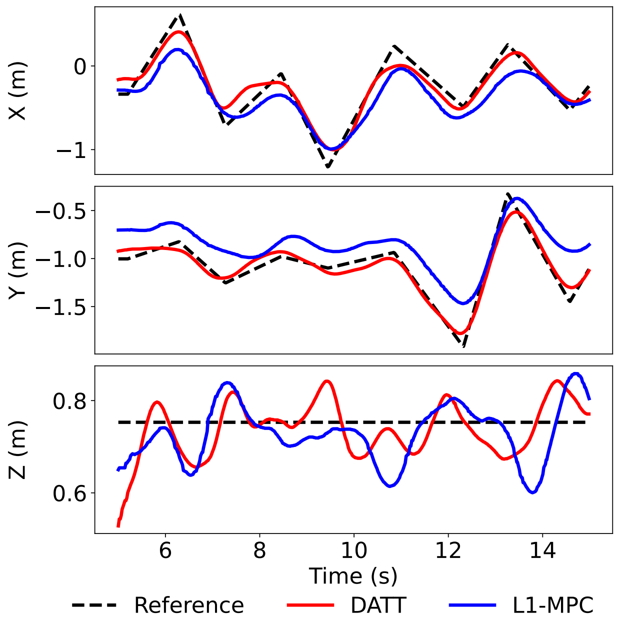

Figure 6: Performance of DATT against $\mathcal{L}_1$-MPC on an infeasible trajectory with both wind and a plate attached.

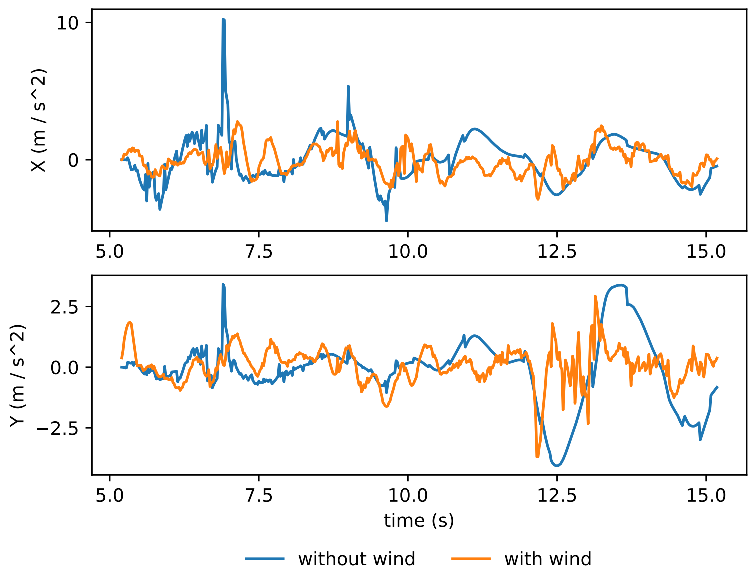

Figure 7: Predicted $\hat{d}$ terms on two infeasible trajectories, one with wind, one without wind but with an air drag plate.