BoundaryPredictor: Imitating Cost-Minimzed Trajectory Sampling for Autonomous Drone Racing

Shreepa Parthaje
Department of Computer Science
University of Virginia
Charlottesville, USA
shreepa@virginia.edu

Nicola Bezzo
Department of Systems Engineering
University of Virginia
Charlottesville, USA
nbezzo@virginia.edu

***Abstract*—Drone racing is a popular test-platform for deep neural planning and control methodologies because neural frameworks can maximize the vehicle's highly non-linear dynamics to make split-second maneuvers. Methods which have found the highest success operate nearly-entirely end-to-end in latent space, making it difficult to validate the safety in decision making. Prior to neural methods, min-snap trajectories were generated using boundary conditions obtained by a model-based optimization algorithm. This is paired with a low-level controller which follows trajectories. These methods have three main challenges: difficulties in modelling drone dynamics, difficulties in modelling the controller limitations, and the heavy computation cost for finding optimal boundary conditions. We propose the BoundaryPredictor framework which leverages deep neural methods for finding the boundary conditions of a min-snap trajectory, while still relying on the traditional trajectory generation and following methods to validate trajectory safety. BoundaryPredictor is trained to imitate low-cost trajectories in a dataset created through a time-intensive cost-minimizing sampling procedure in a high fidelity simulator. Furthermore, BoundaryPredictor is trained with the introduction of weighted soft thresholding in the loss function to encourage safe behavior when generating trajectories. This results in BoundaryPredictor having a skew in errors that favor slower trajectories. Furthermore, we observe in simulation and in real-life on a BitCraze Crazyflie 2.0, BoundaryPredictor's ability to generate fast, track-able trajectories which can navigate circular, straight, and snaked gate courses. Future works for BoundaryPredictor involve expanding the cost-minimizing sampling procedure to consider a wider array of potential trajectories.**

***Index Terms*—imitation learning, weighted soft thresholding loss, min-snap trajectories, drone racing**

I. Introduction

Autonomous quadcopters (drones) are comprised of three hierarchical components: perception, planning, and control. Perception systems turn sensor streams, such as cameras and depth sensors, into an understanding of the environment. The planner processes this understanding of the environment to a set of points, called a trajectory, for the drone to follow. Finally, the controller turns this trajectory into a high frequency series of commands to the motors in order to follow the trajectory.

Trajectory generation for autonomous aerial vehicles requires a fundamental understanding of highly non-linear system dynamics of the drone platform. Autonomous drone racing is an emerging competition where gates, posts with a square hole on top, are arranged into a course for drones to fly through. Winning a drone race requires maximizing the speed through a trajectory while still ensuring that the controller is able to follow the trajectory. As a consequence, research into autonomous drone racing has increased significantly because it provides the perfect environment to maximize the unique system dynamics at play. Computing agile trajectories through a set of gates presents a unique set of problems in collision avoidance, maximizing agility, and minimizing tracking error (i.e. ensuring the trajectory is followed as commanded).

II. Related Works

In mobile robotics path planning is a significant area of challenge due to the high dimensionality of the search space. [1] for instance relies on a RRT approach to get an initial trajectory followed by a quadratic programming approach to optimize a trajectory. In drone racing works have focused highly on imitating human FPV (first-person view) drone racers, who only operate on information from the drones front camera on any given time in a fixed gate-course as seen in [2]. Most works focus on generating agile plans in a fixed environment to get competitive if not better racing results than humans in head-to-head races as seen in [3] through reinforcement learning. What current drone racing papers aim for is agile trajectories in fixed environments allowing for repeated interactions in the same environment [2]. While UAV research has solutions to tackle a variety of environment, research into a unified solution which addresses agility in new environments provides new opportunity.

III. Background

A. Min-Snap Trajectories

A drones highly non-linear dynamics give way to trackable aggressive trajectories which only need to minimize snap, $\ddddot{x}$. A min-snap trajectory is comprised of two boundary conditions: b_0 and b_f and the time taken between the boundaries, T. Boundary conditions include position, velocity, acceleration, and jerk states as seen in (1).

$$b_i = \begin{pmatrix} x_i & \dot{x}_i & \ddot{x}_i & \dddot{x}_i \end{pmatrix}^T \tag{1}$$

The parameterized representation of the min-snap trajectory along $t \in [0, T]$ can be constructed as a 7^{th} order polynomial.

We solve for the coefficients of the polynomial by creating a system of equations as seen below in (2).

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ T^7 & T^6 & T^5 & T^4 & T^3 & T^2 & T & 1 \\ 7 \cdot T^6 & 6 \cdot T^5 & 5 \cdot T^4 & 4 \cdot T^3 & 3 \cdot T^2 & 2 \cdot T & 1 & 0 \\ 42 \cdot T^5 & 30 \cdot T^4 & 20 \cdot T^3 & 12 \cdot T^2 & 6 \cdot T & 2 & 0 & 0 \\ 210 \cdot T^4 & 120 \cdot T^3 & 60 \cdot T^2 & 24 \cdot T & 6 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} p_7 & p_6 & \dots & p_0 \end{bmatrix}^T = A^{-1} \begin{bmatrix} \vec{b_0} & \vec{b_f} \end{bmatrix}^T \qquad (2)$$

It is important to note this representation is fully differentiable for $t \in (0, T)$. Min snap segments can be joined together so long as the final boundary condition of one segment is the initial boundary condition of the next segment, allowing for differentiability at the joint point. With a boundary condition set, $B = \{b_0, b_1, \dots, b_n\}$ and $T = \{T_0, T_1, \dots, T_{n-1}\}$, we can create a $\sum_{i=1}^{n-1} t_i$ second fully differentiable trajectory. Three dimensional trajectories can be constructed with B_x, B_y, and B_z, as along as a shared set T is used when creating $x(t)$, $y(t)$, and $z(t)$.

B. Mellinger Controller

The Mellinger controller is a feedback-based control approach for drones, designed to achieve precise trajectory tracking by leveraging nonlinear control laws for attitude and position. It computes thrust and angular velocity commands to stabilize the drones along a desired path by minimizing trajectory errors, first introduced in [4].

IV. Proposed Formulation

Computing the required boundary conditions and time segments, B_x, B_y, B_z, and T, is a challenging. Traditional methods involve using a large non-linear model of the vehicle and using a solver to find the optimal values. This involves expensive computation along with arduous system identification to find the parameters. Another strategy involves repeatedly sampling potential values for the boundary conditions and validating in simulation, but this is also an expensive process to do at runtime.

We propose the BoundaryPredictor framework which teaches a network to predict optimal conditions for the next gate, G_i with knowledge of just G_{i-1}, G_i, and G_{i+1}. This is done through imitation learning of the lowest-cost trajectories in simulation along with a weighted soft-thresholding loss function. On deployment, the start position followed by the entire gate course parsed through a sliding window of three gates to create the inputs for the BoundaryPredictor network. The network will then predict the boundary conditions at each gate sequentially. These boundary conditions are fed to create a trajectory which is subsequently fed to a Mellinger Controller. In real environments, a Vicon System provides state estimations for the controller

V. Methods

A. Simulation

The training data is collected from a PyBullet Gym environment first introduced by [5]. The default gate setups are modified to have two heights: 0.3 and 0.525 meters. The tracking gap in simulation and real-life specific to the input commands given a variety of trajectories are used to create an upper-bound acceleration limit to impose on the simulator as seen in (3).

$$a(t) < -0.3v(t)^3 + 2.0 \qquad (3)$$

We deploy experiments with inertial property randomization and disturbances seen in Table I in simulation. Furthermore, tracking inconsistencies exist when $\psi_0 = 0$ versus $\psi_0 = \frac{\pi}{4}$. The dataset sampling procedure (explained in subsection B), requires the same behavior between two orientations of the same three gate setup. To address this, we linearly interpolating the drone's yaw between gates G_{i-1} and G_i as seen in (4). We found a successful ratio for interpolation to be $R = 0.25$.

Property	Distribution	Low	High
Mass (M)	Uniform	-0.01	0.01
Moments (I_{xx}, I_{yy}, I_{zz})	Uniform	-0.000001	0.000001
Disturbance Force X, Y, Z	Uniform	-0.000001	0.000001
Property	**Distribution**	**Standard Deviation**	
Motor Thrust Additive Noise	White Noise	0.000001	

TABLE I: Disturbances sampled from during simulation

$$\psi(t) = \begin{cases} \left(1 - \frac{t}{RT}\right)\psi_{i-1} + \left(\frac{t}{RT}\right)\psi_i & \text{if } t < RT \\ \psi_i & \text{if } t \geq RT \end{cases}, \text{for } t \in [0, T] \qquad (4)$$

B. Cost-Minimizing Sampling Procedure

The sampling procedures makes some assumptions regarding the *optimal* trajectory:

1: A gate can be described in polar (d_i, θ_i) centered at G_{i-1}.
2: The *optimal* v_i and G_i are perpendicular as seen in (5).
3: The *optimal* acceleration and jerk at every gate is $\vec{0}$.
4: The *optimal* B_i for G_i is a factor of B_{i-1}, d_{i+1}, θ_{i+1} .
5: When sampling B_i, we decelerate to G_{i+1} so $\vec{v}_{i+1} = \vec{0}$.

$$v_i(s_i, \psi_i) = \begin{pmatrix} s_i sin(\psi_i) & s_i cos(\psi_i) & 0 \end{pmatrix}^T \qquad (5)$$

We can store the required data to find B_i in the vector E_i as seen in (6). In this equation, (d_0, θ_0) is the position of G_i and (d_f, θ_f) is the position of G_{i+1}

$$E_i = \begin{pmatrix} v_{i-1} & z_{i-1} & d_0 & \theta_0 & z_i & d_f & \theta_f & z_{i+1} \end{pmatrix}^T \qquad (6)$$

We construct a dataset of 129600 unique samples, E_i. Table II itemizes constraints on E_i used for the NumPy linspace method, along with the number of discretized points.

For each E_i, 70 sampled trajectories are evaluated using (7) in simulation. N_{crash} counts crashes, $N_{skipped}$ counts gates

Constraint	Discrete Points per Field
$0 \leq v_{i-1} \leq 1$	10
$z_{i-1}, z_i, z_{i+1} \in \{0.3, 0.525\}$	2
$0.8 \leq d_0, d_f \leq 1.5$	6
$\frac{-\pi}{4} \leq \theta_0 \leq \frac{\pi}{4}$	15
$\frac{-\pi}{4} \leq \theta_1 \leq \frac{\pi}{4}$	3

TABLE II: Inputs for NumPy linspace to construct $\{E\}$.

skipped in course, $E_{tracking}$ is the mean position error in the current state to the previously commanded state, and $T = t_0$. The full procedure, Algorithm 1 was deployed on an AWS EC2 C6i.24xlarge instance with 96 vCPUs based on Intel Ice Lake processors. The algorithm was parallelized splitting the large number of samples across the 96 cores, collecting the total samples in a handful of hours.

$$C = 1000000N_{crash} + 100N_{skipped} + 4E_{tracking} + T \quad (7)$$

Algorithm 1 Optimal Value Sampling for E_i

```
Given $E_i$, $\epsilon = 1e{-}10$, initialize empty $v_{i:optimal}$, $t_{0:optimal}$,
and $C_{min} = \infty$
for $v_i \in$ linspace$(0.0, 2.0, 10)$ do
    $t_{linear} = (v_{i-1} + v_i < \epsilon)\ ?\ 1.5 : 2d_0/(v_{i-1} + v_i)$
    for $t_0 \in$ linspace$(0.5t_{linear}, 1.1t_{linear}, 7)$ do
        $t_f = (v_i < \epsilon)\ ?\ 2d_f : 2d_f/v_1$
        $\vec{v}_0 = v(v_0, \psi_1)$, $\vec{v}_1 = v(v_1, \psi_1)$, $\vec{v}_2 = \vec{0}$ using (5).
        Construct $B_{i-1}$, $B_i$, $B_f$ using (1)
        if Equation (3) holds then
            Construct trajectory with (2) and simulate for $C$
            if $C < C_{min}$ then
                Update $C_{min}$, $v_{i:optimal}$, $t_{0:optimal}$
            end if
        end if
    end for
end for
return $v_{i:optimal}$, $t_{0:optimal}$
```

C. BoundaryPredictor Network

BoundaryPredictor predicts $(v_{i:optimal}, t_{0:optimal})$ from an E_i normalized between the values in Table II. Similarly, outputs are normalized between $[[0, 0], [2ms^{-1}, 2s]]$.

Layer	Type	Output Size	Activation Function
Input	Normalized E vector	8	-
1	Linear	512	Leaky ReLU ($\alpha = 0.1$)
2	Linear + Dropout	256	Leaky ReLU ($\alpha = 0.04$)
3	Linear + Dropout	32	Leaky ReLU ($\alpha = 0.02$)
4	Linear	32	Leaky ReLU ($\alpha = 0.02$)
Output	Linear	2	Sigmoid

TABLE III: Architecture of the BoundaryPredictor neural network

The per item loss is defined in (8) which is then weighted to encourage safe behavior. Over-predicting velocity or under-predicting time results in a higher chance of a crash. The aggregate loss function in (9) weights errors to favor negative percent errors (i.e. safe error). Here, W_S and W_U are the weights for the safe and unsafe errors respectively, N is the number of samples, and $\epsilon = 0.1$ is a small constant to avoid division by zero. A sigmoid sum along with an α which adjusts the sharpness around $e_i = 0$ allows for differentiable scaling by sign. $(W_S, W_U, \alpha, \epsilon) = (0.8, 2.0, 10.0, 0.1)$ resulted in the best balance of fast trajectories and safe trajectories.

$$e_i = \begin{cases} \frac{y_i - \hat{y}_i}{y_i + \epsilon} & \text{if } y_i \text{ is a velocity estimate} \\ \frac{-(y_i - \hat{y}_i)}{y_i + \epsilon} & \text{if } y_i \text{ is a time estimate} \end{cases} \quad (8)$$

$$\text{loss} = \frac{1}{N} \sum_{i=1}^{N} (\sigma(\alpha \cdot e_i) \cdot W_S \cdot e_i + (1 - \sigma(\alpha \cdot e_i)) \cdot (-W_U \cdot e_i)) \quad (9)$$

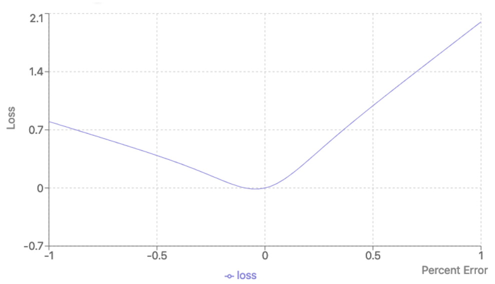

Fig. 1: Soft-sigmoid threshold scaling for per item loss e_i

We define additional hyperparameters outlined in Table IV for training. Patience defines how many epochs to tolerate no decrease in test loss before breaking out of training loop. The random bound, B, defines an upper bound of noise added to the normalized E_i as seen in (10). When training, we use the Adam optimizer and a step learning rate scheduler.

Hyperparameter	**Value**
EPOCHS	1000
BATCH_SIZE	1280
PATIENCE (Early Stopping)	100
B (Noise Factor)	0.1
lr_0	0.01
Scheduler Step Size	50
Scheduler Gamma	0.8
Test / Train Split	92 / 8

TABLE IV: Model Hyperparameters used to balance training speed and stability

$$E_{training} = E_{normalized} + B \cdot \text{Unif}(0, 1) \cdot \mathcal{N}_8(0, I_8) \quad (10)$$

VI. Results

A. Simulation

BoundaryPredictor performs well on the test dataset as seen in with near zero instances of unsafe behavior as seen in Fig. 2. The time error distribution is fairly right skewed indicating a preference towards slower trajectories. The velocity distribution is more Guassian centered around a negative percent error,

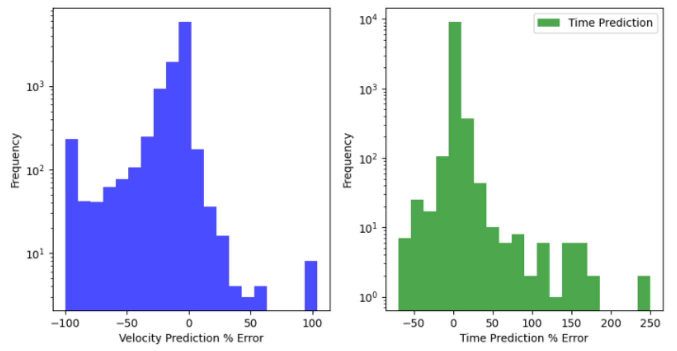

Fig. 2: Percent error log-distribution in test dataset (excluding labels of 0 due to divide by zero)

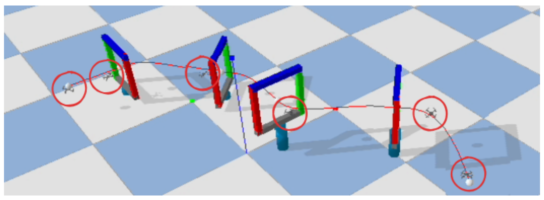

Fig. 3: Pathing in simulation through gate course with frequent change in direction and height

similarly indicating a preference towards slower trajectories. The higher rate of -100% error in the velocity prediction is explained by the use of the Sigmoid activation function on the outputs which caps the error in the velocity dimension. Even when the network is most incorrect, the output action is still safe. In simulation, we try a variety of gate courses and observe the BoundaryPredictor network preferring to take slower trajectories out of gates which change in heading rapidly, while accelerating through gates which are aligned. An example of a successful pass through a more challenging gate course can be seen in Fig. 3.

B. Deployment In Lab

When deployed on real vehicles we noticed a similar performance to that in the sim (with the yaw normalization from (4) removed). We use the Crazyswarm ROS 1 wrapper for the Crazyflie 2.0 API [6], along with localization data from a Vicon motion capture system. On the same course as Fig. 3, you can observe successful behavior in real life in Fig. 4.

C. Sim2Real

When specifically evaluating performance degradation from simulation to real deployment, we can observe Fig. 5 which shows that there exists similar shapes of error between commanded state and measured state in both simulation and in real-life. The difference in magnitude of errors appears to be fairly similar and stochastic.

VII. Discussion and Conclusion

The proposed method for imitation learning on cost minimized trajectory sampling is shown to work in the

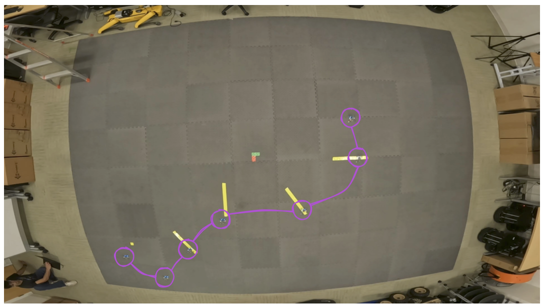

Fig. 4: Pathing in deployment. Note image top-down distortion

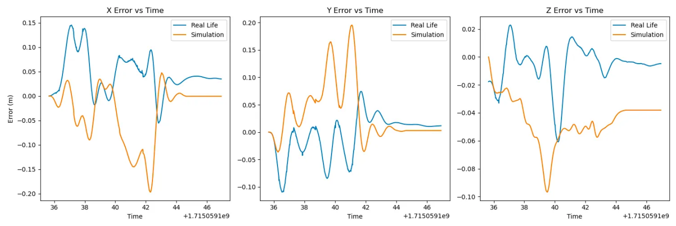

Fig. 5: Position error when following commanded trajectory in simulation and real life

BoundaryPredictor framework. Furthermore, BoundaryPredictor shows that the high success values in a drone racing scenario can be generalized from a three gate setup without considering long-horizon ideas. The network shows a strong ability to generalize the space of gate setups shown with high performance on test dataset. Future works will need to consider that a large number of constraints were placed in the cost sampling procedure which limit what actions can be taken. Revisiting this space will be critical to expand BoundaryPredictor to become competitive with other methods and generalize to any three dimensional trajectory.

References

[1] Bohui Shi, Youmin Zhang, Lingxia Mu, Jing Huang, Jing Xin, Yingmin Yi, Shangbin Jiao, Guo Xie, and Han Liu. Uav trajectory generation based on integration of rrt and minimum snap algorithms. In *2020 Chinese Automation Congress (CAC)*, pages 4227–4232. IEEE, 2020.

[2] Drew Hanover, Antonio Loquercio, Leonard Bauersfeld, Angel Romero, Robert Penicka, Yunlong Song, Giovanni Cioffi, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing: A survey. *IEEE Transactions on Robotics*, 2024.

[3] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[4] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[5] Zhaocong Yuan, Adam W Hall, Siqi Zhou, Lukas Brunke, Melissa Greeff, Jacopo Panerati, and Angela P Schoellig. Safe-control-gym: A unified benchmark suite for safe learning-based control and reinforcement learning in robotics. *IEEE Robotics and Automation Letters*, 7(4):11142–11149, 2022.

[6] James A. Preiss*, Wolfgang Hönig*, Gaurav S. Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3299–3304. IEEE, 2017. Software available at https://github.com/USC-ACTLab/crazyswarm.