2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Learning Agile Flight Maneuvers: Deep SE(3) Motion Planning and Control for Quadrotors

Yixiao Wang*[1], Bingheng Wang*[1], Shenning Zhang[1], Han Wei Sia[2], and Lin Zhao[1]

***Abstract*— Agile flights of autonomous quadrotors in cluttered environments require constrained motion planning and control subject to translational and rotational dynamics. Traditional model-based methods typically demand complicated design and heavy computation. In this paper, we develop a novel deep reinforcement learning-based method that tackles the challenging task of flying through a dynamic narrow gate. We design a model predictive controller with its adaptive tracking references parameterized by a deep neural network (DNN). These references include the traversal time and the quadrotor SE(3) traversal pose that encourage the robot to fly through the gate with maximum safety margins from various initial conditions. To cope with the difficulty of training in highly dynamic environments, we develop a reinforce-imitate learning framework to train the DNN efficiently that generalizes well to diverse settings. Furthermore, we propose a binary search algorithm that allows online adaption of the SE(3) references to dynamic gates in real-time. Finally, through extensive high-fidelity simulations, we show that our approach is adaptive to different gate trajectories, velocities, and orientations.**

I. Introduction

Agile flights of autonomous quadrotors in cluttered environments require constrained motion planning and control in the SE(3) space (special euclidean group) [1]–[5]. It is challenging due to their underactuated nature which couples the translational and rotational dynamics. It is further complicated by the dynamic environments that impose complex constraints and demand the adaptation of trajectories in real-time.

As a representative task, flying through narrow gates with quadrotors has attracted increasing research interest. Earlier works split the traversing process into several phases and proposed different methods to plan a sequence of trajectory segments with given constraints on the quadrotor attitude [6]–[8]. These constraints are hand-picked from extensive experimental trials, which only work for specific static scenarios. By considering the attitude constraints explicitly, Liu et al. [9] proposed a search-based trajectory planning method on SE(3). It models the quadrotor as an ellipsoid and searches for a collision-free trajectory from a set of motion primitives. However, the search-based approach generally suffers from high computational complexity. Direct nonlinear optimization with soft SE(3) constraints was adopted in [10], which relies on various reparameterizations and transforms for fast computation and still only considers static gates.

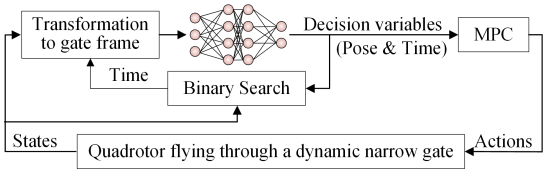


Fig. 1: Diagram of the proposed deep SE(3) motion planning and control method, which fuses a DNN with an MPC to address the gate traversing problem. The DNN adapts the SE(3) decision variables online for the MPC. Moreover, the binary search algorithm updates the DNN-predicted traversal time in real-time and is particularly effective for tracking a dynamic gate.

All the above methods adopt the traditional two-layered framework that addresses planning and control separately. This framework generally works well for regular flights but can lead to inconsistencies between layers for agile flights, thus causing performance degradation in highly dynamic environments [11]. In comparison, reinforcement learning (RL)-based methods have shown great potential in achieving agile flights as they can seamlessly integrate planning and control without excessive priors on desired trajectories. The authors in [12] and [13] showed that end-to-end deep neural networks (DNNs) enable a quadrotor to cross a static gate. While effective for a specific range of gate poses, these model-free RL methods generally lack the robustness that model-based methods have against uncertainties. Song et al. [14] proposed a hybrid method that trains a DNN to predict the traversal time as a decision variable for a model predictive controller (MPC). This method demonstrated efficacy in flying through a dynamic gate. However, it fixes the quadrotor's attitude to be always aligned with the gate orientation, which is conservative and limits the agility.

Driven by the dynamic narrow gate-traversing task, this paper proposes a novel deep SE(3) motion planning and control method for quadrotors. Specifically, we learn an MPC's SE(3) tracking references (meta-learning decision variables). These variables are parameterized by a portable DNN, including the traversal time and the quadrotor traversal pose, which can adapt online to diverse quadrotor states and gate parameters. The DNN is trained such that the generated references encourage the quadrotor to fly through a random gate with maximum safety margins, which fully exploits its agility. However, the receding horizon manner of MPC makes this learning problem computationally expensive; the dynamic environments can cause even unstable learning. To

∗ The first two authors contributed equally.

This work was supported by the Singapore Ministry of Education AcRF Tier 1 (A-0009030-01-00). (*Corresponding author: Lin Zhao.*)

1 These authors are with the Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, 117583 Singapore, Singapore {wangyixiao, wangbingheng, shenningzhang}@u.nus.edu, elezhli@nus.edu.sg

2 Han Wei Sia is with ST Engineering, 1 Ang Mo Kio Electronics Park Road, 567710 Singapore, Singapore siahanwei@hotmail.com

address these difficulties, we first learn the decision variables to traverse through a static gate and develop a reinforce-imitate learning framework that divides the training process into two tractable subproblems. Technical-wise, we train a first DNN via RL to generate the decision variables for a single-run open-loop MPC from stationary initial states. We then train a second DNN via supervised learning to imitate the first while making it adaptive to non-static states on the optimal MPC trajectory generated using the first DNN. Thus, the trained second DNN can generate the adaptive decision variables from dynamic states for an MPC in the receding horizon manner. Finally, we apply the MPC and the trained second DNN to a dynamic gate. As outlined in Fig.1, we view the gate as "static" at its traversal position and transform the network inputs into the gate frame of that position. We develop a binary search algorithm to predict the gate traversal position by updating the traversal time from the second DNN. Given the current state feedback, it can update the predictions in real-time. This property is particularly effective for improving the adaptation of the MPC to dynamic environments.

Our main contributions are summarized as follows:

1) We propose a novel deep SE(3) motion planning and control method for quadrotors and demonstrate its effectiveness via challenging tasks of traversing through fast-moving and rotating narrow gates in real-time under various settings;
2) We develop a computationally efficient reinforce-imitate learning framework for training the DNN that parameterizes the traversal SE(3) decision variables, which generalizes well to unseen environments with different initial quadrotor states, gate widths and orientations, etc;
3) We propose a binary search algorithm that enables fast real-time adaptation to dynamic gates of different moving and angular velocities;
4) We further validate the robustness of the proposed methods in high-fidelity simulations that consider more complex aerodynamics than training.

The rest of this paper is organized as follows. Section II formulates an MPC for the gate-traversing problem. Section III develops the reinforce-imitate learning framework. The binary search algorithm is designed in Section IV. Simulation results are demonstrated in Section V. We conclude this paper and discuss our future work in Section VI.

II. Model Predictive Control for Planning Agile Flight Maneuvers on SE(3)

A. Quadrotor Model

We model the quadrotor as a 6 degree-of-freedom (DoF) rigid body of mass m and moment of inertia $\mathbf{J} \in \mathbb{R}^{3\times3}$. The equations governing the system motions can be written as

$$\dot{\mathbf{p}}_w = \mathbf{v}_w, \qquad \dot{\mathbf{v}}_w = m^{-1}\mathbf{R}\left(\mathbf{q}\right) f_{\mathrm{r}}\mathbf{e}_z - g\mathbf{e}_z, \quad (1a)$$

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{\Omega}\left(\boldsymbol{\omega}_b\right)\mathbf{q}, \quad \dot{\boldsymbol{\omega}}_b = \mathbf{J}^{-1}\left(\boldsymbol{\tau}_{\mathrm{r}} - \boldsymbol{\omega}_b \times \mathbf{J}\boldsymbol{\omega}_b\right), \quad (1b)$$

where $\mathbf{p}_w = [x, y, z]^T$ and $\mathbf{v}_w = [v_x, v_y, v_z]^T$ are the position and velocity of the quadrotor's center-of-mass (CoM) in the world frame $\mathcal{W}$, $\mathbf{e}_z = [0, 0, 1]^T$, $\mathbf{R}\left(\mathbf{q}\right)$ is the rotation matrix from the body frame $\mathcal{B}$ to $\mathcal{W}$ parameterized by quaternions $\mathbf{q} = [q_0, q_x, q_y, q_z]$, g is the gravitational acceleration, $\boldsymbol{\omega}_b = [\omega_x, \omega_y, \omega_z]^T$ is the angular velocity in $\mathcal{B}$, and $\mathbf{\Omega}\left(\boldsymbol{\omega}_b\right)$ is a skew-symmetric matrix of $\boldsymbol{\omega}_b$. Finally, f_{r} and $\boldsymbol{\tau}_{\mathrm{r}} = [\tau_x, \tau_y, \tau_z]^T$ are the collective thrust and the torque produced by the rotor forces $f_i,\ \forall i \in [1, 4]$. We use $\mathbf{x} = \left[\mathbf{p}_w^T, \mathbf{v}_w^T, \mathbf{q}^T, \boldsymbol{w}_b^T\right]^T$ and $\mathbf{u} = [f_1, f_2, f_3, f_4]^T$ to represent the quadrotor's states and control inputs, respectively.

B. Gate Model

We model the dynamic gate as a rectangle which can move freely in space. Fig.2 depicts the gate frame $\mathcal{B}_g$ attached to the center and defines the gate on the $x_g o_g z_g$ plane. Suppose the gate is perpendicular to the ground and can rotate about its y_g axis freely by a pitch angle θ_g. Note that the gate body frame $\mathcal{B}_g$ is instantaneously attached to the gate at the beginning of movement (the gate in dashed line), which is a non-rotating frame. Let $\mathbf{p}_{g,w} = [x_g, y_g, z_g]^T$ denote the position of o_g in $\mathcal{W}$, $\mathbf{v}_{g,w} = [v_{gx}, v_{gy}, v_{gz}]^T$ the velocity of o_g in $\mathcal{W}$, and ω_g the angular rate. The motion of the dynamic gate is described by:

$$\dot{\mathbf{p}}_{g,w} = \mathbf{v}_{g,w}, \quad \dot{\theta}_g = \omega_g. \quad (2)$$

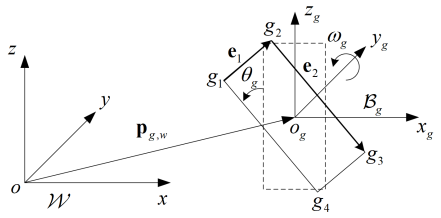


Fig. 2: Illustration of the rectangular gate model and the instantaneously attached body frame $\mathcal{B}_g$. The gate's vertices are denoted by $g_{i=1,2,3,4}$, and o_g is the gate center. For convenience, we define the vectors $\mathbf{e}_1$ and $\mathbf{e}_2$, which can be expressed in $\mathcal{W}$ in terms of g_1, g_2, and g_3 in $\mathcal{W}$. The axis $\mathbf{y}_g$ is not necessarily parallel to y of $\mathcal{W}$.

C. Model Predictive Control

We denote by $\mathbf{x}_T$ the quadrotor final states of hovering at a target point behind the gate. MPC can generate an optimal state trajectory $\mathbf{x}_k^*,\ \forall k \in [0, N]$ towards $\mathbf{x}_T$ and a sequence of optimal control commands $\mathbf{u}_k^*,\ \forall k \in [0, N-1]$ over a horizon N that covers the whole flight trajectory. In MPC, we view the gate as an intermediate waypoint and define the cost function as a sum over four quadratic components: a target-tracking cost $J_{x_k} = \|\mathbf{x}_k - \mathbf{x}_T\|^2_{\mathbf{Q}_x}$, two control regularization costs $J_{u_k} = \|\mathbf{u}_k\|^2_{\mathbf{Q}_u}$ and $J_{\Delta u_k} = \|\mathbf{u}_k - \mathbf{u}_{k-1}\|^2_{\mathbf{Q}_{\Delta u}}$, and a gate-traversing cost $J_{\mathrm{tra},k} = \|\boldsymbol{\delta}_{\mathrm{tra},k}\|^2_{\mathbf{Q}_{\mathrm{tra}}}$ with

$$\boldsymbol{\delta}_{\mathrm{tra},k} = \left[\left(\mathbf{p}_{w,k} - \mathbf{p}_{w,\mathrm{tra}}\right)^T, \mathrm{Tr}\left(\mathbf{I}_3 - \mathbf{R}\left(\mathbf{q}_k\right)^T \mathbf{R}\left(\mathbf{q}_{\mathrm{tra}}\right)\right)\right]^T$$

where $\mathbf{r}_{\mathrm{tra}} = [\mathbf{p}_{w,\mathrm{tra}}, \mathbf{q}_{\mathrm{tra}}]$ is the reference traversal pose, $\mathbf{I}_3 \in \mathbb{R}^{3\times3}$ is an identity matrix, and $\mathrm{Tr}\left(\cdot\right)$ takes the trace of

a given matrix. Therefore, we solve the following nonlinear optimization problem.

$$\min_{\mathbf{x}_{0:N},\mathbf{u}_{0:N-1}} \sum_{k=0}^{N-1} \left(J_{x_k} + J_{u_k} + J_{\Delta u_k} + J_{\mathrm{tra},k}\right) + J_{x_N} \quad (3a)$$

$$\text{s.t. } \mathbf{x}_{k+1} = \mathbf{x}_k + d_t * f_{\mathrm{dyn}}\left(\mathbf{x}_k, \mathbf{u}_k\right), \quad (3b)$$

$$\mathbf{x}_0 = \mathbf{x}_{\mathrm{init}},\ \mathbf{u}_{-1} = \mathbf{u}_{\mathrm{init}},\ \mathbf{x}_k \in \mathbb{X},\ \mathbf{u}_k \in \mathbb{U}, \quad (3c)$$

where d_t is the discrete time step, f_{dyn} is the quadrotor's dynamics model defined in (1), $\mathbf{x}_{\mathrm{init}}$ and $\mathbf{u}_{\mathrm{init}}$ denote the initial states and control commands, $\mathbb{X}$ and $\mathbb{U}$ represent the constraint sets for the quadrotor's states and control inputs, respectively.

These cost matrices $\mathbf{Q}_x \in \mathbb{R}^{13\times13}$, $\mathbf{Q}_u \in \mathbb{R}^{4\times4}$, $\mathbf{Q}_{\Delta u} \in \mathbb{R}^{4\times4}$, and $\mathbf{Q}_{\mathrm{tra}} \in \mathbb{R}^{4\times4}$ are positive definite. In particular, $\mathbf{Q}_x$, $\mathbf{Q}_u$, and $\mathbf{Q}_{\Delta u}$ are time-invariant while the traversal cost matrix $\mathbf{Q}_{\mathrm{tra}}$ is time-varying, defined as

$$\mathbf{Q}_{\mathrm{tra}}\left(t_{\mathrm{tra}}, k\right) = \mathbf{Q}_{\max} * \exp\left(-\gamma * \left(k * d_t - t_{\mathrm{tra}}\right)^2\right), \quad (4)$$

where $\mathbf{Q}_{\max} \in \mathbb{R}^{4\times4}$ is the maximum traversal cost matrix that should be larger than all the time-invariant cost matrices, $\gamma \in \mathbb{R}_+$ is the temporal spread of the traversal cost, and t_{tra} is the time at which the quadrotor traverses through the gate. As time approaches to t_{tra}, we have $\mathbf{Q}_{\mathrm{tra}}\left(t_{\mathrm{tra}}, k\right) \approx \mathbf{Q}_{\max}$ and the relatively large $\mathbf{Q}_{\max}$ encourages the quadrotor to track the gate. Instead, for time away from t_{tra}, the matrix $\mathbf{Q}_{\mathrm{tra}}$ decays exponentially, making the quadrotor arrive at the target point.

D. Deep Neural SE(3) Decision Variables for MPC

For Problem (3), t_{tra} and $\mathbf{r}_{\mathrm{tra}}$ are of great importance as they determine when, where, and how the quadrotor flies through the gate. Compared to the previous work [14], we learn the more challenging SE(3) traversal references (decision variables) $\mathbf{r}_{\mathrm{tra}}$ instead of merely the traversal position $\mathbf{p}_{w,\mathrm{tra}}$, which significantly improves the feasibility and optimality of SE(3) motion planning. We highlight that the Gaussian sampling based policy search in [14] is intractable to learn the SE(3) DNN policy in our case.

For gradient-based training of the attitude reference, we use the Rodrigues parameters (i.e., the $so(3)$ vector representation) $\boldsymbol{\rho}_{\mathrm{tra}} \in \mathbb{R}^3$ instead of the quaternion $\mathbf{p}_{\mathrm{tra}}$, due to the constraint-free optimization of the former [15]. For ease of reference, we define a vector $\mathbf{z} = [\mathbf{p}_{w,\mathrm{tra}}, \boldsymbol{\rho}_{\mathrm{tra}}, t_{\mathrm{tra}}] \in \mathbb{R}^7$ as the high-level decision variables, parameterize Problem (3) as $\mathrm{MPC}\left(\mathbf{z}\right)$ and denote the corresponding optimal state trajectory by $\boldsymbol{\xi}^*\left(\mathbf{z}\right)$ (i.e., $\boldsymbol{\xi}^*\left(\mathbf{z}\right) = \{\mathbf{x}_k^*\left(\mathbf{z}\right)\}_{k=0}^N$).

In this paper, we are interested in learning deep neural decision variables (i.e., modeling $\mathbf{z}$ using a DNN) that can adapt the MPC performance online in highly dynamic flight scenarios. The training is highly-nontrivial, and we will develop a novel reinforce-imitate learning framework in the following section. It trains the DNN efficiently using a static gate (Section III). Combining with a binary search algorithm, we can update the traversal time predicted by the DNN in real-time to improve the adaptation to dynamic environments (Section IV).

III. Reinforce-imitate Learning Framework

Fig.3 illustrates the proposed reinforce-imitate learning framework that divides the training process into two tractable subproblems. We train a first DNN via RL to generate the decision variables for a single-run open loop MPC from different stationary states. Then, we train a second DNN via supervised learning to imitate the first while making it adaptive to non-static quadrotor states on the optimal MPC trajectory generated using the first DNN.

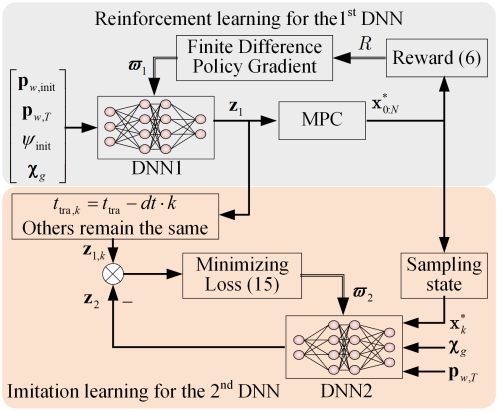


Fig. 3: Diagram of the proposed reinforce-imitate learning framework

A. Reinforcement Learning for Training the 1st DNN

We use subscript 1 to denote the decision variables modeled by the first DNN, which has the following form

$$\mathbf{z}_1 = f_{\boldsymbol{\varpi}_1}\left(\mathbf{p}_{w,\mathrm{init}}, \mathbf{p}_{w,T}, \psi_{\mathrm{init}}, \boldsymbol{\chi}_g\right). \quad (5)$$

Here $\mathbf{p}_{w,\mathrm{init}}$ and ψ_{init} denote the initial quadrotor position in $\mathcal{W}$ and yaw angle, $\mathbf{p}_{w,T}$ is the target position in $\mathcal{W}$, and $\boldsymbol{\varpi}_1$ denote the parameters of the first DNN. In training, we place the gate at the origin of $\mathcal{W}$ and use its width $\|\mathbf{e}_1\|_2$ and pitch angle θ_g to denote the gate states $\boldsymbol{\chi}_g$, i.e., $\boldsymbol{\chi}_g = [\|\mathbf{e}_1\|_2, \theta_g]$.

In RL, the reward function is designed as a weighted sum of three terms to evaluate the quality of $\boldsymbol{\xi}^*\left(\mathbf{z}_1\right)$:

$$R\left(\boldsymbol{\xi}^*\left(\mathbf{z}_1\right)\right) = R_{\max} - \alpha L_T\left(\boldsymbol{\xi}^*\left(\mathbf{z}_1\right)\right) - \beta L_{\mathrm{coll}}\left(\boldsymbol{\xi}^*\left(\mathbf{z}_1\right)\right) \quad (6)$$

where $\alpha \in \mathbb{R}_+$ and $\beta \in \mathbb{R}_+$ are the weighting coefficients. We define these three terms and explain their meaning in the following bullets.

1) Target penalty term $L_T \in \mathbb{R}_+$: We introduce this term to penalize the quadrotor's deviation away from the target point. It is defined using the last n successive points on the optimal trajectory, and thus takes the following form:

$$L_T = \sum_{k=N-n}^{N} \left\|\mathbf{p}^*_{w,k}\left(\mathbf{z}_1\right) - \mathbf{p}_{w,T}\right\|_2^2. \quad (7)$$

2) Collision penalty term $L_{\mathrm{coll}} \in \mathbb{R}_+$: To penalize the collision, we build this term upon a collision loss function that reflects the traversal performance and can apply to different gate dimensions. However, designing such a function is nontrivial as it requires considering the quadrotor's shape and attitude. Here, we view the quadrotor as a square (denoted

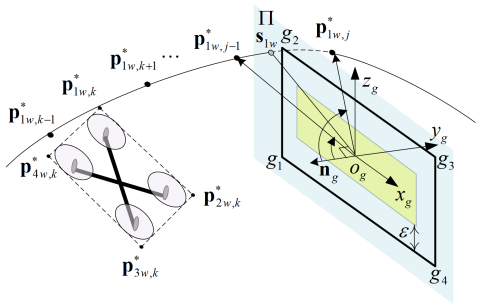


Fig. 4: Given an optimal trajectory of one vertex $\mathbf{p}^*_{iw,0:N}, i \in [1,4]$, we compute the inner product of $\mathbf{p}^*_{iw,k}$ and the normal vector $\mathbf{n}_g$. Suppose $\mathbf{p}^*_{iw,j}$ is the first waypoint behind the gate that makes the product negative. Then, the line connecting $\mathbf{p}^*_{iw,j-1}$ and $\mathbf{p}^*_{iw,j}$ intersects with Π at $\mathbf{s}_{iw}$.

by the dashed contour in Fig.4) and design the loss functions for the four vertexes. Specifically, given an optimal quadrotor pose $\mathbf{x}^*_{\text{pose},k} = \left[\mathbf{p}^*_{w,k}, \mathbf{q}^*_k\right]$, the position of a vertex in $\mathcal{W}$ at $\mathbf{x}^*_{\text{pose},k}$ can be obtained as

$$\mathbf{p}_{iw,k} = \mathbf{p}^*_{w,k} + \mathbf{R}\left(\mathbf{q}^*_k\right)\mathbf{p}_{ib,k},\ k \in [0, N] \tag{8}$$

where $\mathbf{p}_{ib,k}$ denotes the position of ith vertex in $\mathcal{B}$. Then, we compute the intersections $\mathbf{s}_{iw},\ \forall i \in [1,4]$ of the vertexes' optimal trajectories and the gate plane Π, in order to define the collision loss function as:

$$\text{coll}\left(\mathbf{s}_{iw}\right) = \begin{cases} \max\left(0, \varepsilon - d_i\right) & \mathbf{s}_{iw} \text{ inside the gate} \\ 2\varepsilon * d_i + \varepsilon^2 & \text{otherwise} \end{cases}, \tag{9}$$

where $\varepsilon \in \mathbb{R}_+$ is a safety margin that can be slightly larger than the quadrotor's height, and d_i is the shortest distance from $\mathbf{s}_{iw}$ to the gate sides. This loss function encourages the quadrotor to enter the safe region of the gate (denoted by the light yellow rectangle in Fig.4) with a more feasible and optimal pose. This property benefits the motion planning and control after flying through the gate, and thus substantially improves the quadrotor's agility in different environments. Using (9), we can design the collision penalty term as

$$L_{\text{coll}} = \sum_{i=1}^{4} \text{coll}\left(\mathbf{s}_{iw}\left(\mathbf{z}_1\right)\right). \tag{10}$$

3) Goal reward term $R_{\max} \in \mathbb{R}_+$: If the quadrotor flies through the gate's safe region and arrives at the target point accurately, this positive goal reward will be achieved.

Hence, the RL problem is to find optimal network parameters $\boldsymbol{\varpi}^*_1$ that maximize the reward $R\left(\boldsymbol{\xi}^*\left(\mathbf{z}_1\right)\right)$. It can be interpreted as the following optimization problem:

$$\max_{\boldsymbol{\varpi}_1} R\left(\boldsymbol{\xi}^*\left(\mathbf{z}_1\left(\boldsymbol{\varpi}_1\right)\right)\right) \tag{11a}$$

$$\text{s.t. } \boldsymbol{\xi}^*\left(\mathbf{z}_1\left(\boldsymbol{\varpi}_1\right)\right) \text{ generated by MPC}\left(\mathbf{z}_1\left(\boldsymbol{\varpi}_1\right)\right). \tag{11b}$$

We use gradient ascent to train $\boldsymbol{\varpi}_1$. The gradient of R with respect to $\boldsymbol{\varpi}_1$ can be computed using the chain rule $\frac{dR}{d\boldsymbol{\varpi}_1} = \frac{\partial R}{\partial \mathbf{z}_1}\frac{\partial \mathbf{z}_1}{\partial \boldsymbol{\varpi}_1}$. Here, calculating $\frac{\partial \mathbf{z}_1}{\partial \boldsymbol{\varpi}_1}$ is standard for the neural network via many existing machine learning tools. However, computing $\frac{\partial R}{\partial \mathbf{z}}$ is challenging as it requires differentiating through the nonlinear MPC optimization problem (3), which is computationally expensive for a long horizon. Therefore, we adopt a more efficient method: the finite difference policy gradient. It estimates the gradient $\frac{\partial R}{\partial \mathbf{z}_1}$ by applying a small perturbation $\delta\mathbf{z}_1$ to the decision variable vector $\mathbf{z}_1$, i.e., $\frac{\partial R}{\partial \mathbf{z}} = R\left(\boldsymbol{\xi}^*\left(\mathbf{z}_1 + \delta\mathbf{z}_1\right)\right) - R\left(\boldsymbol{\xi}^*\left(\mathbf{z}_1\right)\right)$. This method is typically powerful in the episode-based learning strategy, provided the reward is not too noisy [16].

B. Imitation Learning for Training the 2nd DNN

We design a second DNN to make the decision variables adaptive to non-static quadrotor states. To this end, the inputs of the second DNN incorporate the current optimal states $\mathbf{x}^*_k$ that move on $\boldsymbol{\xi}^*\left(f_{\boldsymbol{\varpi}^*_1}\right)$, and the resulting decision variables are defined as

$$\mathbf{z}_2 = f_{\boldsymbol{\varpi}_2}\left(\mathbf{x}^*_k, \mathbf{p}_{w,T}, \boldsymbol{\chi}_g\right), \tag{12}$$

where $\boldsymbol{\varpi}_2$ denote the parameters of the second DNN. We train the second DNN to imitate the first via supervised learning. Note that t_{tra} generated by the first DNN is relative to the initial states. Given the receding horizon manner of MPC, we shift t_{tra} by the time elapsed till $\mathbf{x}^*_k$, i.e.,

$$t_{\text{tra},k} = t_{\text{tra}} - k * d_t. \tag{13}$$

The reference traversal pose $\mathbf{r}_{\text{tra}}$ produced by the first DNN keeps the same. Thus, we can denote by

$$\mathbf{z}_{1,k} = \left[\mathbf{p}_{w,\text{tra}}, \boldsymbol{\rho}_{\text{tra}}, t_{\text{tra},k}\right],\ k \in [0, N], \tag{14}$$

the desired decision variables for the second DNN to imitate. The imitation learning problem is to find optimal network parameters $\boldsymbol{\varpi}^*_2$ that minimize the following loss:

$$L = \left\| f_{\boldsymbol{\varpi}_2}\left(\mathbf{x}^*_k, \mathbf{p}_{w,T}, \boldsymbol{\chi}_g\right) - \mathbf{z}_{1,k} \right\|_2^2. \tag{15}$$

We summarize the whole training procedures of the proposed reinforce-imitate learning framework in Algorithm 1.

IV. Binary Search for Traversal Time

We further design a binary search method that efficiently applies the second DNN to a dynamic gate. The idea is to view the gate as "static" at its traversal position (denoted by 'gate at t_{tra}' in Fig.5) and transform the network inputs[1]

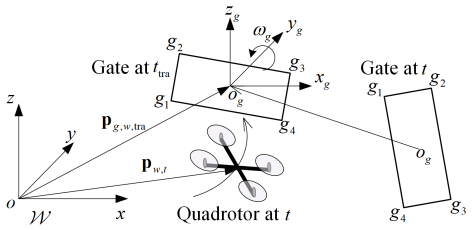


Fig. 5: A dynamic gate viewed as "static" at its traversal position.

to the gate frame of that position (see Fig.1). Such a transformation of the inputs enables the second DNN to generate the corresponding decision variables that guide the quadrotor towards the "static" gate.

[1]When using the binary search method, we solve the MPC in the receding horizon manner and replace $\mathbf{x}^*_k$ in (12) with the current quadrotor states $\mathbf{x}_t$.

Algorithm 1 Reinforce-Imitate Learning Framework

Input $f_{\boldsymbol{\varpi}_1}$
repeat Reinforcement learning for the 1st DNN
  Randomly sample $\mathbf{p}_{w,\text{init}}$, $\mathbf{p}_{w,T}$, ψ_{init}, and $\boldsymbol{\chi}_g$
  Compute $\mathbf{z}_1 = f_{\boldsymbol{\varpi}_1}(\mathbf{p}_{w,\text{init}}, \mathbf{p}_{w,T}, \psi_{\text{init}}, \boldsymbol{\chi}_g)$
  Solve MPC $(\mathbf{z}_1)$ to obtain $\boldsymbol{\xi}^*(\mathbf{z}_1)$
  Solve MPC $(\mathbf{z}_1 + \delta\mathbf{z}_1)$ to obtain $\boldsymbol{\xi}^*(\mathbf{z}_1 + \delta\mathbf{z}_1)$
  Compute $\frac{\partial R}{\partial \mathbf{z}_1} = R(\boldsymbol{\xi}^*(\mathbf{z}_1 + \delta\mathbf{z}_1)) - R(\boldsymbol{\xi}^*(\mathbf{z}_1))$
  Update $\boldsymbol{\varpi}_1$ using gradient-based optimization
until $f_{\boldsymbol{\varpi}_1}$ is well trained
Output $f_{\boldsymbol{\varpi}_1^*}$
Input $f_{\boldsymbol{\varpi}_2}$
repeat Imitation learning for the 2nd DNN
  Randomly sample $\mathbf{p}_{w,\text{init}}$, $\mathbf{p}_{w,T}$, ψ_{init}, and $\boldsymbol{\chi}_g$
  Compute $\mathbf{z}_1 = f_{\boldsymbol{\varpi}_1^*}(\mathbf{p}_{w,\text{init}}, \mathbf{p}_{w,T}, \psi_{\text{init}}, \boldsymbol{\chi}_g)$
  Solve MPC $(\mathbf{z}_1)$ to obtain $\boldsymbol{\xi}^*(\mathbf{z}_1)$
  for $k \leftarrow 0$ to N **do**
    Sample $\mathbf{x}_k^*$ from $\boldsymbol{\xi}^*(\mathbf{z}_1)$
    Compute $t_{\text{tra},k} = t_{\text{tra}} - k * d_t$ to obtain $\mathbf{z}_{1,k}$
    Compute $L = \|f_{\boldsymbol{\varpi}_2}(\mathbf{x}_k^*, \mathbf{p}_{w,T}, \boldsymbol{\chi}_g) - \mathbf{z}_{1,k}\|_2^2$
    Update $\boldsymbol{\varpi}_2$ using gradient-based optimization
  end for
until $f_{\boldsymbol{\varpi}_2}$ is well trained
Output $f_{\boldsymbol{\varpi}_2^*}$

Using the gate kinematics model (2), we can predict the gate traversal position by updating t_{tra} from the second DNN via the binary search method. We start with an initial guess t_1, which is proportional to the distance between the current quadrotor's CoM and the gate center, and calculate the future gate position at t_1. Then, the network inputs are transformed into the gate frame at t_1 and thus the second DNN can generate the corresponding traversal time[2] t_2. Finally, we update t_1 using the mean value of these two times. We repeat these steps until the difference between t_1 and t_2 is within a small threshold ϵ, meaning that both the times converge to the actual traversal time t_{tra}. The whole procedures of the binary search method are summarized in Algorithm 2. As a consequence of the well-trained second DNN, this algorithm can be run in real-time to substantially improve the performance of tracking a dynamic gate.

Algorithm 2 Binary Search Algorithm for Updating t_{tra}

Input $f_{\boldsymbol{\varpi}_2^*}$, $\mathbf{x}$, $\mathbf{v}_{g,w}$, and ω_g
Guess an initial value of t_1 and set $t_2 = 0$
while $|t_1 - t_2| > \epsilon$ **do**
  Predict $\mathbf{p}_{g,w}$ and θ_g of the gate using Eq.(2) and t_1
  Transform $\mathbf{x}_t$ and $\mathbf{p}_{w,T}$ into the gate frame at t_1
  Obtain t_2 from the second DNN $f_{\boldsymbol{\varpi}_2^*}$
  Update $t_1 \leftarrow \frac{1}{2}(t_1 + t_2)$
end while

[2]To avoid confusion with the actual traversal time t_{tra}, we denote by t_2 the traversal time from the second DNN in the binary search method.

V. Simulation Results

We validate the effectiveness of our approach via extensive simulations. In particular, we design a challenging scenario: traversing through a fast-moving and rotating gate to reach a random target point behind it. The gate moves from the origin of $\mathcal{W}$ at a time-varying velocity of $\mathbf{v}_{g,w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ $\text{m}\cdot\text{s}^{-1}$ where $\boldsymbol{\mu}$ is the gate mean velocity and $\boldsymbol{\sigma}$ is the gate velocity standard deviation, while rotating at a constant angular rate of $\frac{\pi}{2}$ $\text{rad}\cdot\text{s}^{-1}$ from the initial angle $\theta_{g,\text{init}}$ that is inversely proportional to the gate width. We place the target around $\mathbf{p}_{w,Tc} = [0, 6, 0]^T$ m subject to an uniform random deviation $\Delta\mathbf{p}_{w,T} \sim \mathcal{U}_{[-\mathbf{2},\mathbf{2}]}$ m where $\mathbf{2} = [2, 2, 2]$, such that $\mathbf{p}_{w,T} = \mathbf{p}_{w,Tc} + \Delta\mathbf{p}_{w,T}$. The quadrotor's position and yaw angle are randomly initialized by $\mathbf{p}_{w,\text{init}} = \mathbf{p}_{w,\text{init}c} + \Delta\mathbf{p}_{w,\text{init}}$ and $\psi_{\text{init}} \sim \mathcal{U}_{[-0.1,0.1]}$ rad, respectively, where $\mathbf{p}_{w,\text{init}c} = [0, -9, 0]^T$ m and $\Delta\mathbf{p}_{w,\text{init}} \sim \mathcal{U}_{[-\mathbf{5},\mathbf{5}]}$ m. The rest of the quadrotor initial states are fixed at zeros. For better visualization in 3D simulation, we scale up the gate's length to 2 m, and its width is sampled from a Gaussian distribution, i.e., $\|\mathbf{e}_1\|_2 \sim \mathcal{N}(0.9, 0.2)$ m, in the beginning of each flight. Accordingly, the quadrotor's dimension is scaled up to 1.5 m $\times$ 1.5 m, but not affecting its dynamics properties.

In the MPC, we use a prediction time horizon of $T = 5.0$ s and a discrete time step of $d_t = 0.1$ s. In the simulation environment, the quadrotor model (1) and the gate model (2) are integrated using the forward Euler method with the discrete time step of 0.01 s. To build those two DNNs, we adopt two multilayer perceptrons (MLPs) with rectified linear unit (ReLU) as the activation function. The architectures of the first and second DNNs are $9 \to 64 \to 64 \to 7$ and $18 \to 128 \to 128 \to 7$, respectively. We use CasADi [17] with `ipopt` to solve the MPC optimization problem (3) in Python. These two MLPs are built in PyTorch [18] and are trained using `Adam` [19].

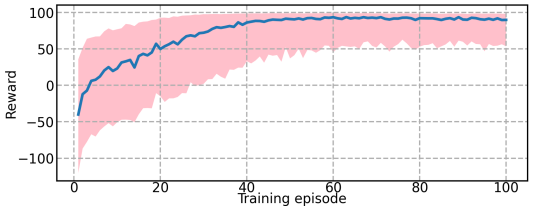
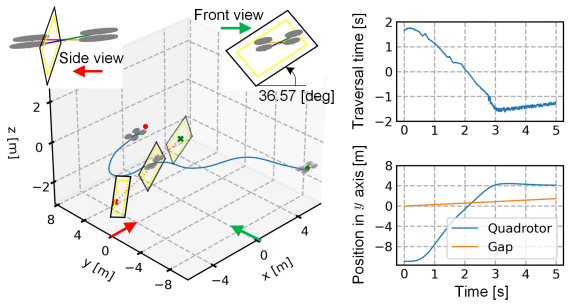


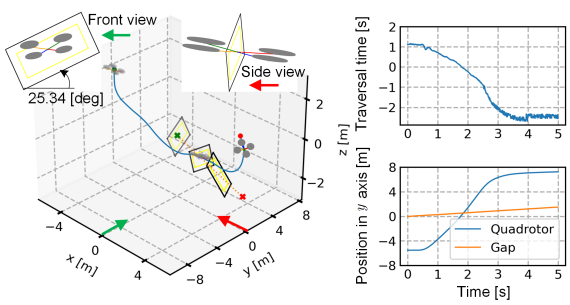
Fig. 6: Reward of training the first DNN. The goal reward $R_{\max}$ is 100. In each episode, we iteratively solve the MPC 100 times. We compute the median value (solid curve) and the interquartile range (pink region). Here we set the safe margin ε to be 0.2 m. A larger ε can reduce the collision risk but escalates the training difficulty, leading to a smaller steady reward.

Fig.6 shows the reward $R(\boldsymbol{\xi}^*(\mathbf{z}_1))$ during the training of the first DNN via the RL of Algorithm 1. The median reward increases rapidly in the beginning and converges to around 90 after 50 episodes, indicating efficient and stable learning. Based on the first DNN, we then train the second DNN using the imitation learning of Algorithm 1.

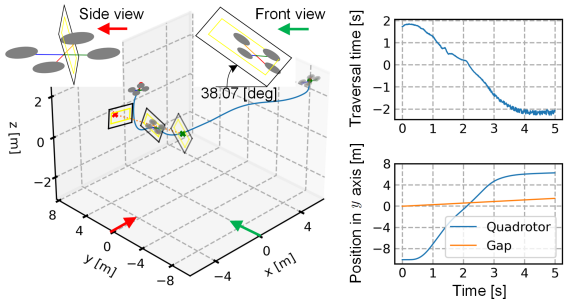
We evaluate the trained second DNN and Algorithm 2 on a dynamic gate whose velocity profile is defined above. In evaluation, the MPC is implemented in the receding horizon



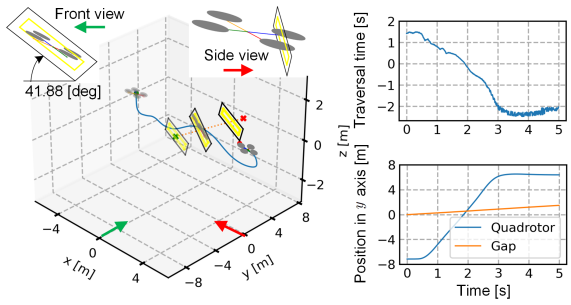
(a) Trial 1 with $\boldsymbol{\mu} = [-1, 0.3, -0.4]^T$ m $\cdot$ s^{-1}



(b) Trial 2 with $\boldsymbol{\mu} = [1, 0.3, -0.4]^T$ m $\cdot$ s^{-1}



(c) Trial 3 with $\boldsymbol{\mu} = [-1, 0.3, 0.4]^T$ m $\cdot$ s^{-1}



(d) Trial 4 with $\boldsymbol{\mu} = [1, 0.3, 0.4]^T$ m $\cdot$ s^{-1}

Fig. 7: Evaluation of the trained second DNN and the binary search algorithm with $\boldsymbol{\sigma} = \text{diag}\,(0.1, 0.1, 0.1)$ m $\cdot$ s^{-1}. The green and red dots represent the start and target points of the quadrotor, respectively, while the green and red crosses denote the initial and final positions of the gate. The yellow rectangles in the zoom-in sub-figures denote the safe regions.

manner where we set $\mathbf{x}_{\text{init}} = \mathbf{x}_t$ and $\mathbf{u}_{\text{init}} = \mathbf{u}_{t-1}$, and apply $\mathbf{u}_0^*$ to the quadrotor. We visualize four randomly sampled examples with different gate mean velocities in Fig.7. Our method enables the quadrotor to fly through the dynamic gate and approach the target as closely as possible, adapting to different gate orientations, trajectories, and dimensions. Specifically, the quadrotor can enter the pre-defined safe region of the gate with an optimal pose (see the zoom-in sub-figures). We further observe from the right columns in Fig.7 that the positions of the quadrotor and the gate in the y direction intersect when the traversal time decreases to zero (see Eq.(13) for the decreasing t_{tra}), indicating accurate updates of t_{tra} by the developed binary search algorithm.

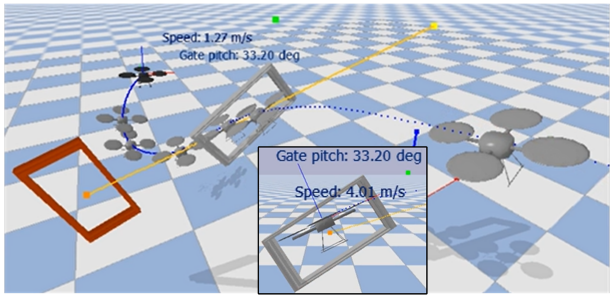


Fig. 8: A trial of flying through a fast-moving and rotating gate in the `gym-pybullet-drones` [20]. The zoom-in figure shows the quadrotor traversing through the tilted gate of 33.2 deg with an optimal pose.

We further test our method in `gym-pybullet-drones`, a high-fidelity open-source quadrotor simulator considering more complex aerodynamic effects than our training settings, such as drag and ground effect [20]. The MPC and the trained second DNN are deployed directly to the simulator without extra tuning. We randomly run four trials using the same gate mean velocities as those in Fig.7. Fig.8 visualizes a trial corresponding to $\boldsymbol{\mu} = [-1, 0.3, -0.4]^T$ m $\cdot$ s^{-1}, and the complete demo can be found in this video: `https://youtu.be/TCId8S22gic`. These successes demonstrate our approach's robustness to aerodynamic disturbances and great potential for practical applications.

VI. Conclusions

Driven by the challenging gate-traversing agile flight, this paper presented a novel deep SE(3) motion planning and control method for quadrotors. It learns an MPC's adaptive SE(3) decision variables parameterized by a portable DNN, encouraging the quadrotor to fly through the gate with maximum safety margins under diverse settings. Our reinforce-imitate learning framework and binary search algorithm allow efficient training with a static gate and then adaptation to highly dynamic environments. Extensive high-fidelity simulations suggest that our method is adaptive to different gate trajectories, velocities, orientations, and target positions. Our future work includes theoretical studies on the robustness and stability of the deep SE(3) MPC control framework. We also plan to develop more efficient training algorithms on SE(3) with analytical gradients.

References

[1] 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)*. IEEE, 2010, pp. 5420–5425.

[2] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2011, pp. 2520–2525.

[3] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE transactions on robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[4] J. C. Pereira, V. J. Leite, and G. V. Raffo, "Nonlinear model predictive control on SE(3) for quadrotor aggressive maneuvers," *Journal of Intelligent & Robotic Systems*, vol. 101, no. 3, pp. 1–15, 2021.

[5] Z. Han, Z. Wang, N. Pan, Y. Lin, C. Xu, and F. Gao, "Fast-racing: An open-source strong baseline for SE(3) planning in autonomous drone racing," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8631–8638, 2021.

[6] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[7] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2016.

[8] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 5774–5781.

[9] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in SE (3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[10] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Transactions on Robotics*, 2022.

[11] D. H. Shim, H. J. Kim, and S. Sastry, "Decentralized nonlinear model predictive control of multiple flying robots," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 4. IEEE, 2003, pp. 3621–3626.

[12] J. Lin, L. Wang, F. Gao, S. Shen, and F. Zhang, "Flying through a narrow gap using neural network: an end-to-end planning and control approach," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 3526–3533.

[13] C. Xiao, P. Lu, and Q. He, "Flying through a narrow gap using end-to-end deep reinforcement learning augmented with curriculum learning and sim2real," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[14] Y. Song and D. Scaramuzza, "Policy search for model predictive control with application to agile drone flight," *IEEE Transactions on Robotics*, 2022.

[15] E. O. Díaz, O. Díaz, and Ditzinger, *3D Motion of Rigid Bodies*. Springer, 2019.

[16] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[17] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to flya gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7512–7519.