Hybrid Gradient-Based Policy Optimization for Sample-Efficient Policy Learning in Autonomous Systems

Kuan-Yu Tseng[1], Jeff S. Shamma[2], and Geir E. Dullerud[1]

***Abstract*— This paper introduces HyGIPO, a novel gradient-based iterative policy optimization technique designed for efficient policy learning in autonomous systems, especially in the presence of modeling errors. Performance of control algorithms for autonomous systems is often limited by mismatches between a simplified nominal model and a complex real system. To address this degradation, HyGIPO leverages a hybrid gradient optimization approach, combining gradients of dynamics from a nominal model with real-world data to optimize control policies. We apply this method to the quadcopter waypoint tracking problem, with the controller parameterized by a neural network, demonstrating its effectiveness in both simulation and hardware experiments. In simulation, HyGIPO rapidly learns the policy within a hundred samples, showing orders of magnitude higher sample efficiency compared to reinforcement learning methods. The hardware experiments further validate the method, achieving successful tracking results in just tens of samples.**

I. Introduction

Research on controlling autonomous and robotic systems has been ongoing for decades, yet it remains a popular area due to the complexity of both the systems themselves and the tasks they are designed to perform. Control algorithms are often developed using nominal models in laboratory settings, while real-world systems often exhibit parametric or non-parametric variations from these models, leading to diminished performance. Model-free approaches, such as Deep Reinforcement Learning (DRL) [1]–[3], enable robotic systems to learn complex policies by directly interacting with their environments. However, these algorithms are typically data-inefficient and require extensive hyperparameter tuning. Additionally, the methods often need to address the so-called sim-to-real gap [4] when transferring learned policies to hardware platforms. In recent years, model-based approaches have been complemented by data-driven techniques to reduce the discrepancies between models and real-world environments. For instance, Model Predictive Control (MPC) are combined with machine learning techniques to learn the system dynamics from the real-world experiences [5]–[8]. Iterative Learning Control (ILC) [9]–[11] provides another route to address mismatches by utilizing a model to compute an optimal control policy, which is then refined through feedback from real-world executions. In the previous work, a hybrid gradient-based ILC technique is proposed to efficiently compute optimal control policies, by combining gradients from a nominal dynamics and trajectories and costs from a real system [12]. This algorithm is validated through real car experiments and extended to efficiently solve various episodic problems [13]. Typically, those model-based methods are limited to optimizing low-level control policies, which are task-specific and often lack the ability to generalize to higher-level control tasks.

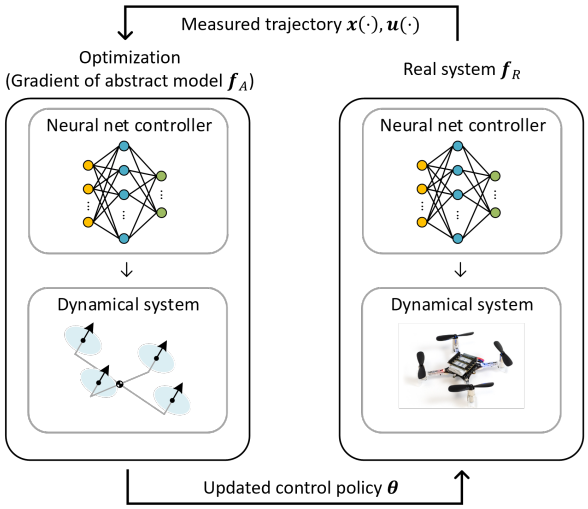

Fig. 1. The diagram of HyGIPO framework.

In this paper, we extend the framework from [12], which learns low-level action trajectories, and introduce Hybrid Gradient-based Iterative Policy Optimization (HyGIPO), a technique that learns general control policies through hybrid gradient optimization. HyGIPO computes the gradient information from an abstract nominal dynamics model and gathers state and control trajectories from a real system to update the controller parameters. Real system rollouts with the updated parameters are then performed, and the resulting system trajectories are fed back to the optimizer iteratively. We apply HyGIPO to the quadcopter waypoint tracking problem with the neural network controller design as illustrated in Fig. 1. The simulation demonstrates good tracking performance for waypoints randomly generated in a 3D space over fewer than a hundred rollouts. HyGIPO also shows significantly better data efficiency compared to the DRL algorithm Proximal Policy Optimization (PPO) [3] under the presence of the dynamical mismatch. Additionaly, the hardware experiments also demonstrate the efficacy of HyGIPO with only a few dozen of samples.

[1]Kuan-Yu Tseng and Geir E. Dullerud are with the Department of Mechanical Science and Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA {kuanyut2, dullerud}@illinois.edu

[2]Jeff S. Shamma is with the Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA jshamma@illinois.edu

II. Related Work

Model-based methods are commonly used to compute control policies for autonomous systems by modeling the real system with a mathematical model, synthesizing policies, and implementing them on the real system. To reduce the discrepancies between the model and the system, these methods are often paired with data-driven techniques to learn unmodeled terms from real-world data. In the context of MPC, [5], [7] use Gaussian process regression to account for environment variations in car and quadcopter systems. In [6], [8], a neural network is integrated into the MPC framework to model nonlinear system behaviors. ILC [9] provides an alternative strategy based on updating control policies from real-world executions without adjusting a model. In [10], the authors implement an optimization-based ILC on a quadcopter's trajectory tracking problem by updating the control policy using the linearized model from the previous trial. In [12], a hybrid gradient-based ILC combines trajectories collected from real robot operations with gradients computed via a simiplfied model to efficiently refine the control policy. Generally, model-based methods are used to solve low-level control policies, such as actuator outputs or setpoints, and there are limited works on applying these methods to learn more general types of control policies.

In recent years, DRL [1]–[3] has proven effective at learning approximated optimal control policies by allowing agents to interact directly with the environment and update policies based on experience [14]–[16]. These algorithms are often trained in a simulated environment before being applied to real robots. To reduce the sim-to-real gap, various strategies are employed, such as dynamics randomization [14], [17], Bayesian optimization for adapting kinematic parameters [18], and adjusting simulated parameters during training with real-world rollouts [19], [20]. In general, training DRL policies require extensive hyperparameter tuning and large amounts of data samples to achieve optimal performance.

III. Hybrid Gradient-based Iterative Policy Optimization

In this section, we derive HyGIPO in a general form. The derivation procedure is similar to the previous work [12]. However, the framework in this report is extended to optimize a general control policy. We consider a nonlinear discrete-time system

$$\boldsymbol{x}(t+1) = \boldsymbol{f}_R(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{1}$$

where the function $\boldsymbol{f}_R(\cdot,\cdot)$ describes the actual dynamics of the real system, $\boldsymbol{x}(t) \in \mathbb{R}^n$ and $\boldsymbol{u}(t) \in \mathbb{R}^m$ denote the system state and control signal to the system, respectively. The general state feedback controller is represented as

$$\boldsymbol{u}(t) = \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{x}_{\text{ref}}(t); \boldsymbol{\theta}) \tag{2}$$

which is parameterized by some parameters $\boldsymbol{\theta} \in \mathbb{R}^k$. $\boldsymbol{x}_{\text{ref}}$ represents the reference point that the controller aims to track. The controller's objective is to minimize a certain cost through an optimization problem,

$$J = \min_{\boldsymbol{\theta}} \; L\left(\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot)\right) \tag{3a}$$

$$\text{s.t.} \quad \boldsymbol{x}(t+1) = \boldsymbol{f}_R(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{3b}$$

$$\boldsymbol{u}(t) = \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{x}_{\text{ref}}(t); \boldsymbol{\theta}) \tag{3c}$$

$$\boldsymbol{x}(0) \in \mathcal{X}_S \tag{3d}$$

$$\boldsymbol{x}(t+1) \in \mathcal{X}, \boldsymbol{u}(t) \in \mathcal{U} \quad \forall t = \{0, \dots, N-1\} \tag{3e}$$

subject to the dynamics $\boldsymbol{f}_R$, the control function $\boldsymbol{h}$, initial conditions and constraints. L is some cost function. N is the length of the trajectory. $\mathcal{X}_S$ represents the set of the initial states, and $\mathcal{X}$ and $\mathcal{U}$ represent the state and input constraints, respectively. We assume the real system $\boldsymbol{f}_R$ is unknown in the optimization, but the state can be measured. Instead, we have access to an abstract nominal model $\boldsymbol{f}_A(\cdot,\cdot)$, which we regard as an approximation to $\boldsymbol{f}_R$. Specifically, we assume $\boldsymbol{f}_A$ and $\boldsymbol{f}_R$ have positive dot product across the state and control space. For simplicity, we consider $\boldsymbol{f}_A$ and $\boldsymbol{f}_R$ to have the same dimension. If $\boldsymbol{f}_R$ includes unmodeled terms and has a higher dimension than $\boldsymbol{f}_A$, the framework still holds under the assumption that $\boldsymbol{f}_A$ and $\boldsymbol{f}_R$ maintain positive dot product in the corresponding subspace. Similar to the procedure shown in [12], the goal is to predict the evolution of the real state and control trajectories by combining the trajectories collected from $\boldsymbol{f}_R$ and the gradient computed from $\boldsymbol{f}_A$. Starting with a predetermined feasible trajectory, $\{\boldsymbol{x}^j(t), \boldsymbol{u}^j(t)\}$, $t \in \{0, \dots, N\}, N < \infty, j \in \mathbb{N}$ denoting the number of iteration. The deviations of the state and control signal across iterations are denoted by $\hat{\boldsymbol{x}}(\cdot) = \boldsymbol{x}^{j+1}(\cdot) - \boldsymbol{x}^j(\cdot)$ and $\hat{\boldsymbol{u}}(\cdot) = \boldsymbol{u}^{j+1}(\cdot) - \boldsymbol{u}^j(\cdot)$. Assuming the motion of the system in a new iteration stays close to the previous trajectory, the deviation can be linearized by a first-order Taylor Expansion along the previous trajectory,

$$\hat{\boldsymbol{x}}(t+1) = \frac{\partial \boldsymbol{f}_A(t)}{\partial \boldsymbol{x}(t)}\hat{\boldsymbol{x}}(t) + \frac{\partial \boldsymbol{f}_A(t)}{\partial \boldsymbol{u}(t)}\hat{\boldsymbol{u}}(t) \tag{4a}$$

$$\hat{\boldsymbol{u}}(t) = \frac{\partial \boldsymbol{h}(t)}{\partial \boldsymbol{x}(t)}\hat{\boldsymbol{x}}(t) + \frac{\partial \boldsymbol{h}(t)}{\partial \boldsymbol{\theta}}\hat{\boldsymbol{\theta}} \tag{4b}$$

where $\hat{\boldsymbol{\theta}}$ is the deviation of the parameters. Note that $\boldsymbol{f}_A$ is used here to compute the Jacobian matrices. Substituting (4b) into (4a), we obtain

$$\begin{aligned}\hat{\boldsymbol{x}}(t+1) = &\underbrace{\left(\frac{\partial \boldsymbol{f}_A(t)}{\partial \boldsymbol{x}(t)} + \frac{\partial \boldsymbol{f}_A(t)}{\partial \boldsymbol{u}(t)}\frac{\partial \boldsymbol{h}(t)}{\partial \boldsymbol{x}(t)}\right)}_{\boldsymbol{A}(t)}\hat{\boldsymbol{x}}(t) \\ &+ \underbrace{\left(\frac{\partial \boldsymbol{f}_A(t)}{\partial \boldsymbol{u}(t)}\frac{\partial \boldsymbol{h}(t)}{\partial \boldsymbol{\theta}}\right)}_{\boldsymbol{B}(t)}\hat{\boldsymbol{\theta}}\end{aligned} \tag{5}$$

Hence, the evolution of the real state trajectory can be approximated with the deviation of parameters and the initial state,

$$\boldsymbol{X}^{j+1} \approx \boldsymbol{X}^j + \boldsymbol{M}_{\boldsymbol{\theta}}\hat{\boldsymbol{\theta}} + \boldsymbol{M}_{\boldsymbol{x}(0)}\hat{\boldsymbol{x}}(0), \tag{6}$$

where $\boldsymbol{X}^{j+1} = \left[\boldsymbol{x}^{j+1}(1), \boldsymbol{x}^{j+1}(2), \dots, \boldsymbol{x}^{j+1}(N)\right]$ is the vectorized state trajectory predicted in the next iteration,

$\boldsymbol{X}^j = \left[\boldsymbol{x}^j(1), \boldsymbol{x}^j(2), \ldots, \boldsymbol{x}^j(N)\right]$ is the measured state trajectory on $\boldsymbol{f}_R$. $\boldsymbol{M}_{\boldsymbol{\theta}} \in \mathbb{R}^{Nn \times k}$ and $\boldsymbol{M}_{\boldsymbol{x}(0)} \in \mathbb{R}^{Nn \times n}$ are lifted matrices containing the matrices $\boldsymbol{A}(\cdot)$ and $\boldsymbol{B}(\cdot)$. Specifically, $\boldsymbol{M}_{\boldsymbol{\theta}}$ can be represented as

$$\boldsymbol{M}_{\boldsymbol{\theta}} = \begin{bmatrix} \boldsymbol{M}_{\boldsymbol{\theta}}(0) \\ \boldsymbol{M}_{\boldsymbol{\theta}}(1) \\ \vdots \\ \boldsymbol{M}_{\boldsymbol{\theta}}(N-1) \end{bmatrix} \tag{7}$$

where $\boldsymbol{M}_{\boldsymbol{\theta}}(0) = \boldsymbol{B}(0)$ and $\boldsymbol{M}_{\boldsymbol{\theta}}(t) = \boldsymbol{A}(t)\boldsymbol{M}_{\boldsymbol{\theta}}(t-1) + \boldsymbol{B}(t)$ for $1 \leq t \leq N-1$. $\boldsymbol{M}_{\boldsymbol{x}(0)}$ can be viewed as

$$\boldsymbol{M}_{\boldsymbol{x}(0)} = \begin{bmatrix} \boldsymbol{M}_{\boldsymbol{x}(0)}(0) \\ \boldsymbol{M}_{\boldsymbol{x}(0)}(1) \\ \vdots \\ \boldsymbol{M}_{\boldsymbol{x}(0)}(N-1) \end{bmatrix} \tag{8}$$

where $\boldsymbol{M}_{\boldsymbol{x}(0)}(t) = \prod_{i=0}^{t} \boldsymbol{A}(i)$ for $0 \leq t \leq N-1$.

Similarly, we can derive the evolution of the control trajectory as

$$\boldsymbol{U}^{j+1} \approx \boldsymbol{U}^j + \boldsymbol{N}_{\boldsymbol{\theta}}\hat{\boldsymbol{\theta}} + \boldsymbol{N}_{\boldsymbol{x}(0)}\hat{\boldsymbol{x}}(0), \tag{9}$$

where $\boldsymbol{U}^{j+1} = \left[\boldsymbol{u}^{j+1}(0), \boldsymbol{u}^{j+1}(1), \ldots, \boldsymbol{u}^{j+1}(N-1)\right]$ and $\boldsymbol{U}^j = \left[\boldsymbol{u}^j(0), \boldsymbol{u}^j(1), \ldots, \boldsymbol{u}^j(N-1)\right]$ are the vectorized control trajectories in the next and current iteration. $\boldsymbol{N}_{\boldsymbol{\theta}}$ and $\boldsymbol{N}_{\boldsymbol{x}(0)}$ are lifted matrices. Then the optimization problem (3a) can be reformulated as

$$\min_{\hat{\boldsymbol{\theta}}} \; L\left(\boldsymbol{X}^{j+1}, \boldsymbol{U}^{j+1}\right) \tag{10}$$

where $\boldsymbol{X}^{j+1}$ and $\boldsymbol{U}^{j+1}$ are defined in (6) and (9) respectively. The problem is subject to (3d) and (3e), and a trust region constraint to bound the deviation of the variable $|\hat{\boldsymbol{\theta}}| \leq \hat{\boldsymbol{\theta}}_{\max}$. Therefore, starting from an initial feasible trajectory on $\boldsymbol{f}_R$, HyGIPO solves the optimization problem (10) with the gradient information of $\boldsymbol{f}_A$. Assuming that $\boldsymbol{f}_A$ and $\boldsymbol{f}_R$ are aligned across the state and control space, HyGIPO's optimization direction remains consistent with that of the real system dynamics, ensuring efficient and stable policy optimization. The updated control policy is then implemented in $\boldsymbol{f}_R$, and the new state and control trajectories are measured. Since there may exist a mismatch between $\boldsymbol{f}_R$ and $\boldsymbol{f}_A$, the trajectory propagated on $\boldsymbol{f}_R$ can deviate from the predicted one. If the real trajectory deviates significantly or violates the constraint, we can tighten the trust region $\hat{\boldsymbol{\theta}}_{\max}$ and re-optimize. Additionally, considering a batch update over several trajectory samples can also improve stability. The workflow is depicted in the Fig. 1.

IV. Quadcopter Waypoint Tracking Application

In this section, we describe the quadcopter dynamics and formulate the waypoint tracking problem.

A. Quadcopter Dynamics

The quadcopter is assumed to have 6 degrees of freedom and contains the following dynamics with 18-dimensional state space:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{R}} \\ \dot{\boldsymbol{v}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{R}\boldsymbol{\omega}^{\times} \\ -g\boldsymbol{e}_3 + \frac{F_T}{m}\boldsymbol{R}\boldsymbol{e}_3 \\ \boldsymbol{I}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega}) \end{bmatrix} \tag{11}$$

where $\boldsymbol{p} \in \mathbb{R}^3$ and $\boldsymbol{v} \in \mathbb{R}^3$ represent the position and velocity of the quadcopter. $\boldsymbol{R} \in SO(3)$ is the rotation matrix representing the quadcopter's orientation. $\boldsymbol{\omega} \in \mathbb{R}^3$ denotes the angular velocity. The dynamical parameters m and $\boldsymbol{I} \in \mathbb{R}^{3\times3}$ denote the vehicle mass and the moment of inertia. g is the gravity acceleration. $\boldsymbol{e}_3$ represent the unit vector in z direction. $\boldsymbol{\omega}^{\times}$ is the skew-symmetric matrix derived from the vector $\boldsymbol{\omega}$. The total thrust and torques are denoted by F_T and $\boldsymbol{\tau} \in \mathbb{R}^3$.

We consider the Mellinger controller [21] as the onboard lower-level controller of the quadcopter that computes F_T and $\boldsymbol{\tau}$ from the control command $\boldsymbol{u}$. The Mellinger controller is a two-stage controller consisting of the position PD and the attitude PD controllers. The first stage is fed with some reference state to compute the desired motor thrust and quadcopter's orientation. In the second stage, the desired torques are then calculated from the difference of the desired and the actual orientation, as well as the angular velocity. In this work, we choose the control command consisting of the desired linear velocity and yaw rate, i.e., $\boldsymbol{u} = [v_{\text{cmd},x}, v_{\text{cmd},y}, v_{\text{cmd},z}, \omega_{\text{cmd},z}]$. Sending desired velocity commands, rather than directly sending rotor thrust and torques, addresses communication latency and stability issues, particularly in hardware experiments [22].

B. Waypoint Tracking Problem

The waypoint tracking problem is formulated in this subsection. We assume the quadcopter starts from a fixed hovering state. Given a target waypoint position $\boldsymbol{p}_{\text{ref}}$, the quadcopter aims to track this target point within a fixed time horizon. The optimization problem is formulated as

$$\begin{aligned} J = \min_{\boldsymbol{\theta}} \sum_{t=0}^{N-1} (\boldsymbol{x}(t) - \boldsymbol{x}_{\text{ref}})^{\top} \boldsymbol{Q} (\boldsymbol{x}(t) - \boldsymbol{x}_{\text{ref}}) + \\ (\boldsymbol{x}(N) - \boldsymbol{x}_{\text{ref}})^{\top} \boldsymbol{P} (\boldsymbol{x}(N) - \boldsymbol{x}_{\text{ref}}) \end{aligned} \tag{12}$$

subject to the system dynamics, the controller function, the initial condition, and the constraints (3b-3e). Here, $\boldsymbol{Q}$ and $\boldsymbol{P}$ are diagonal weight matrices penalizing the intermediate and terminal states relative to the target. In this work, we select $\boldsymbol{Q} = \text{diag}\{1 \cdot \boldsymbol{1}_{3\times1}, \boldsymbol{0}_{15\times1}\}$ and $\boldsymbol{P} = \text{diag}\{1000 \cdot \boldsymbol{1}_{3\times1}, \boldsymbol{0}_{15\times1}\}$ as we aim for the quadcopter to move toward and stay near the waypoint at the final time. The target state contains the waypoint position $\boldsymbol{x}_{\text{ref}} = [\boldsymbol{p}_{\text{ref}}, \boldsymbol{0}_{15\times1}]$. We do not penalize the energy cost on the control signals $\boldsymbol{u}$. The constraints include limits on the linear and angular velocities of the quadcopter. Following the procedure in Section III, (12) can be transformed into a quadratic problem in terms of solving variable $\hat{\boldsymbol{\theta}}$.

C. Neural Network Controller

In this work, we consider a controller parameterized by the neural network. That is, HyGIPO optimizes this controller to learn the waypoint tracking policy by optimizing its parameters. The neural network is implemented in PyTorch, with an architecture consisting of two hidden layers, each with 32 neurons and Tanh activation. The parameters are initialized by randomly sampled from distributions based on PyTorch's default settings. The input to the controller includes the full system state $\boldsymbol{x}$ and the waypoint position $\boldsymbol{p}_{\text{ref}}$. To slightly simplify the problem, the controller only outputs the desired linear velocity $\boldsymbol{u} = [v_{\text{cmd},x}, v_{\text{cmd},y}, v_{\text{cmd},z}]$ since our focus is on the positional tracking. The desired yaw rate is set to 0. In total, 1859 parameters need to be optimized.

V. Simulation

In this section we evaluate HyGIPO on the quadcopter waypoint tracking problem in the simulated environment.

A. Simulation Setup

In HyGIPO optimization, the gradient of the quadcopter dynamics is computed by the auto differentiation package CasADi [23]. The quadratic optimization problem (12) is then solved in Python with the OSQP package [24]. The quadcopter simulator is built based on [25], which also provides compatibility with the PPO algorithm that will be described and compared in the later subsection. The simulation and the discretization frequency for the optimization is 200Hz. The dynamical parameters, including the mass m and the inertia $\boldsymbol{I}$ in the model $\boldsymbol{f}_A$, are based on the specifications for the Crazyflie 2.0 [26]. However, for $\boldsymbol{f}_R$ in the simulation, we assume a 20% mismatch in m and a -20% mismatch in $\boldsymbol{I}$. The initial position of the quadcopter is set to $(0, 0, 0.1)$. The domain of the waypoint positions is set to $[0, 1] \times [0, 1] \times [0.1, 1.1]$. The time horizon is set to 2 seconds.

B. Learning and Testing Performance

The learning procedure of HyGIPO is summarized as follows: a waypoint target is sampled from the domain. Then, $\boldsymbol{f}_R$ performs a rollout toward this waypoint. An optimization is solved, and the parameters are updated every four samples to achieve more stable learning results.

The learning costs over 100 samples are shown in Fig. 2. We conduct five trials and plot the results, displaying the mean and the standard deviation. Initially, the controller parameters are initialized with distributions close to 0. Therefore, the resulting control signals are close to 0, and the quadcopter stays near at the starting point, resulting in large costs. After several iterations of parameter updates, the learning costs rapidly decrease.

We also evaluate the testing performance of HyGIPO during learning on progressive amounts of data. In each learning iteration, the quadcopter is tested on 50 randomly sampled waypoints from the same domain using the current parameter values. The testing result is also shown in Fig. 2. The decreasing and consistent testing costs indicate that HyGIPO effectively learns the waypoint tracking policy.

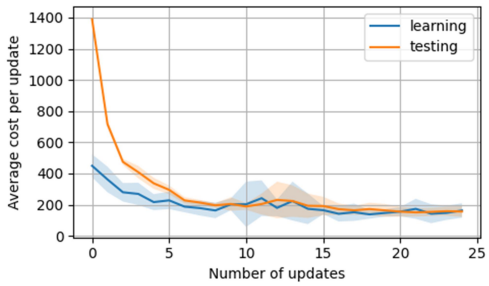

Fig. 2. Learning and testing results in simulation.

C. Comparison to PPO

To evaluate the strength of data efficiency of HyGIPO, we compare the performance of HyGIPO to variants of Proximal Policy Optimization (PPO) [3], a widely used model-free RL algorithm. We utilize the Stable Baselines3 [27] implementation of PPO, and the quadcopter system is wrapped as a gym environment in [25]. After tuning, the learning rate is set to $1e^{-3}$. The network architecture and parameter initialization process are the same as those used in HyGIPO. Other hyperparameters not mentioned are set to default values in Stable Baselines3.

We define two variants of PPO, referred to as Robust PPO and Ideal PPO. Robust PPO is trained with the dynamics randomization technique [17], with the deviation of m and $\boldsymbol{I}$ are sampled from the uniform distributions $\Delta m = \pm 30\%$ and $\Delta \boldsymbol{I} = \pm 30\%$. Ideal PPO, representing the idealized training scenario, is directly trained on $\boldsymbol{f}_R$.

We train the PPO methods with an equivalent of 100, 1000, and 10000 samples of data. We conduct three trials for the PPO methods across all sample length settings. The testing results compared to HyGIPO are shown in Tab. I and Fig. 3. Tab. I presents the terminal positional distance between the real system trajectory and the waypoint. HyGIPO demonstrates the lowest terminal distances after learning for 100 samples, while the PPO methods achieve comparable performance only after training for 10000 samples. Interestingly, the performance of Robust PPO is better than that of Ideal PPO. This may be because Robust PPO trains the policy on various dynamics, resulting in a more generalized policy and helping to avoid local minima. Fig. 3 shows 15 sampled testing trajectories for HyGIPO and PPO methods. The trajectories of HyGIPO steer toward the waypoints after 100 samples of learning, whereas the trajectories of PPO show patterns of moving toward the waypoints after 10000 samples. In this example, HyGIPO is able to learn the policy with orders of magnitude fewer samples than the PPO methods.

VI. Hardware Experiment

The hardware experiments are implemented on the commercially available Crazyflie 2.0 nano quadcopter, as shown

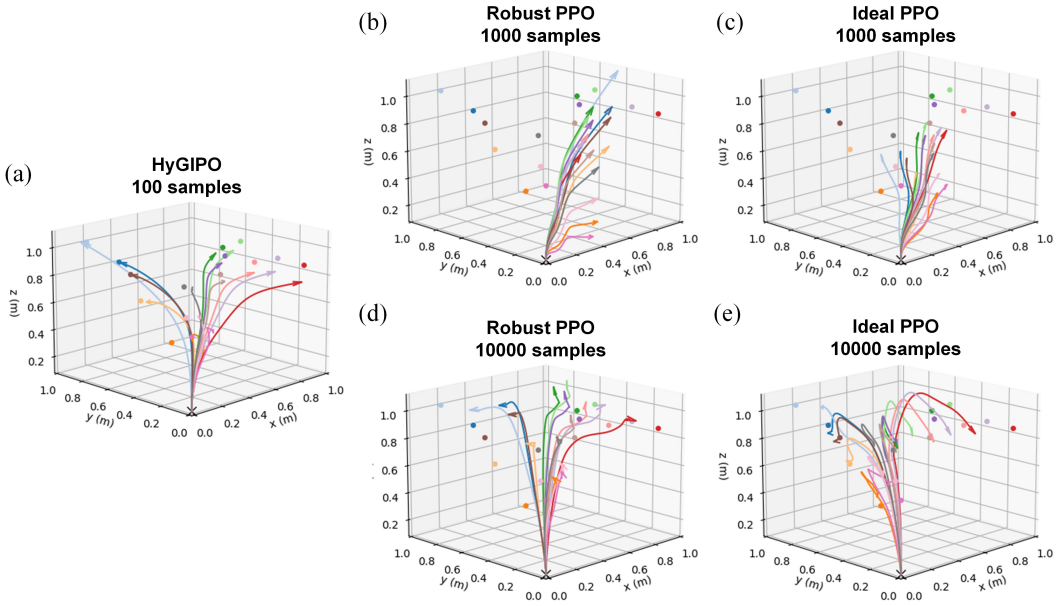

Fig. 3. 15 sampled testing trajectories of (a) HyGIPO trained with 100 samples, (b) Robust PPO trained with 1000 samples, (c) Ideal PPO trained with 1000 samples, (d) Robust PPO trained with 10000 samples, (e) Ideal PPO trained with 10000 samples. The quadcopter starts from the same starting point (cross). The dots represent the waypoints and the curves represent the syetem trajectories in 3D.

TABLE I
TERMINAL DISTANCES OF TESTING DATA

	HyGIPO	Robust PPO	Ideal PPO
100 samples	$\mathbf{0.20 \pm 0.07}$	0.91±0.03	0.97±0.03
1000 samples		0.56±0.05	0.48±0.09
10000 samples		0.22±0.04	0.27±0.09

in Fig. 4(a). For more details of system specifications we refer to [26]. Note that the Vicon motion capture markers are attached to the platform, resulting in different mass and inertia from the specifications used in $\boldsymbol{f}_A$. The experiments are conducted in the flying arena under the Center for Autonomy, University of Illinois, as shown in Fig. 4(b). The arena is equipped with the Vicon motion capture system, which can capture the Crazyflie's pose and velocity. The data is then transmitted to the ground control station and serves as feedback to the onboard controller of the Crazyflie. HyGIPO and the neural network controller are run on the ground control station. Once the state information and a waypoint are provided, the neural network controller computes the command signal, which is sent through radio to the quadcopter at 50Hz. The communication between the control station and the Crazyflie is handled by Crazyswarm [28], which is built on ROS 2 library. The system diagram is shown in Fig. 5.

In the experiment, the time horizon is set to 2 seconds. The Crazyflie starts hovering at $(0, 0, 0.5)$. During the learning phase, we designate four waypoints including $\{(0.3, 0.8, 1.3), (0.5, 0.7, 0.8), (0.7, 0.5, 1.3), (0.8, 0.3, 0.8)\}$ in the space, and HyGIPO updates the policy once the quadcopter completes four rollouts on each waypoint. We conduct five trials of the learning experiment. The cost evolution over iterations is shown in Fig. 6. The costs

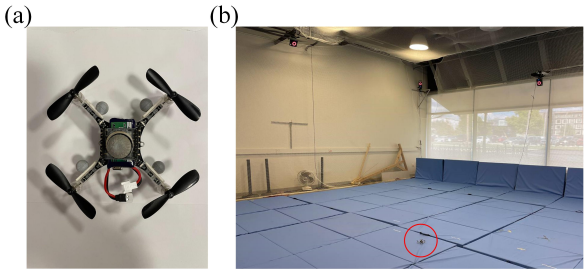

Fig. 4. (a) Crazyflie 2.0. (b) the flying arena. The circle shows the Crazyflie.

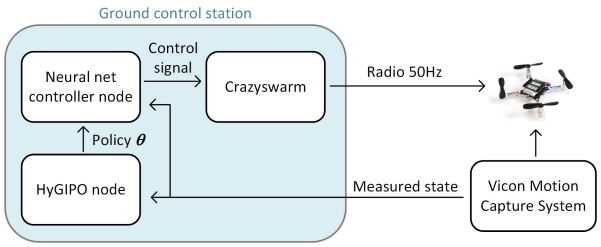

Fig. 5. The architecture of the experiment system.

decrease rapidly and level off after around 5 iterations.

We then evaluate the learning result by testing the Crazyflie and the policy on eight randomly generated waypoints in the lab space. The testing trajectories are shown in Fig. 7, demonstrating that the quadcopter is able to reach near the target waypoints. The mean terminal distance is 0.12. The experiment validates that HyGIPO can successfully learn the waypoint tracking policy on the Crazyflie platform, relying on only a few learning points and iterations. The experiment video: https://youtu.be/n45mqh19C6M.

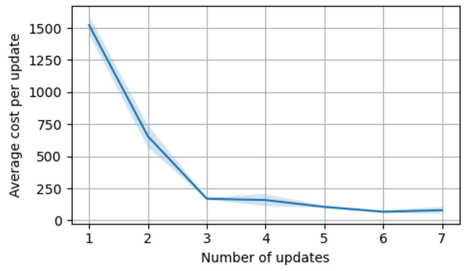

Fig. 6. Cost evolution versus iterations in the experiment.

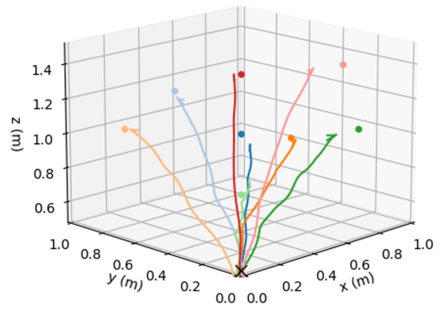

Fig. 7. 8 sampled testing trajectories of HyGIPO in the experiment. The dots represent the waypoints and the curves represent the syetem trajectories in 3D.

VII. CONCLUSIONS

This paper presents HyGIPO, a novel gradient-based iterative policy optimization technique for autonomous systems. By using a hybrid gradient approach, HyGIPO efficiently learns control policies in the presence of mismatches between abstract models and real systems. We demonstrate the efficacy of HyGIPO on a quadcopter waypoint tracking problem, and show that it compares favorably with the well-known PPO method which converges significantly slower in this case study.

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[5] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[6] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2397–2404, 2023.

[7] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, "Data-driven mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, 2021.

[8] 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)*. IEEE, 2017, pp. 1714–1721.

[9] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE control systems magazine*, vol. 26, no. 3, pp. 96–114, 2006.

[10] A. P. Schoellig, F. L. Mueller, and R. D'andrea, "Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Autonomous Robots*, vol. 33, pp. 103–127, 2012.

[11] N. R. Kapania and J. C. Gerdes, "Path tracking of highly dynamic autonomous vehicle trajectories via iterative learning control," in *2015 American control conference (ACC)*. IEEE, 2015, pp. 2753–2758.

[12] K.-Y. Tseng, J. S. Shamma, and G. E. Dullerud, "Low-fidelity gradient updates for high-fidelity reprogrammable iterative learning control," in *2022 American Control Conference (ACC)*, 2022, pp. 4772–4777.

[13] K.-Y. Tseng, M. Zhang, K. Hauser, and G. E. Dullerud, "Adaptive trajectory database learning for nonlinear control with hybrid gradient optimization," *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, (to appear).

[14] E. Chisari, A. Liniger, A. Rupenyan, L. Van Gool, and J. Lygeros, "Learning from simulation, racing in reality," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8046–8052.

[15] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[16] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1205–1212.

[17] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[18] I. Exarchos, Y. Jiang, W. Yu, and C. K. Liu, "Policy transfer via kinematic domain randomization and adaptation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 45–51.

[19] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.

[20] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8973–8979.

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

[22] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 504–10 510.

[23] 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, pp. 1–36, 2019.

[24] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[25] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly—a 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)*, 2021, pp. 7512–7519.

[26] "Crazyflie 2.0 - bitcraze," https://www.bitcraze.io/crazyflie-2/.

[27] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[28] J. A. Preiss*, W. Hönig*, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3299–3304.