# Constrained Imitation Learning for a Flapping Wing Unmanned Aerial Vehicle

Tejaswi K. C. and Taeyoung Lee

***Abstract*—This paper presents a data-driven optimal control policy for a micro flapping wing unmanned aerial vehicle. First, a set of optimal trajectories are computed off-line based on a geometric formulation of dynamics that captures the nonlinear coupling between the large angle flapping motion and the quasi-steady aerodynamics. Then, it is transformed into a feedback control system according to the framework of imitation learning. In particular, an additional constraint is incorporated through the learning process to enhance the stability properties of the resulting controlled dynamics. Compared with conventional methods, the proposed constrained imitation learning eliminates the need to generate additional optimal trajectories on-line, without sacrificing stability. As such, the computational efficiency is substantially improved. Furthermore, this establishes the first nonlinear control system that stabilizes the coupled longitudinal and lateral dynamics of flapping wing aerial vehicle without relying on averaging or linearization. These are illustrated by numerical examples for a simulated model inspired by Monarch butterflies.**

## I. Introduction

Flapping wing aerial vehicles exhibit substantial advantages in energy efficiency and agility, compared against the conventional fixed or rotary wing types whose lift-to-drag ratio deteriorates rapidly as its size is reduced. As such, the flapping wing mechanism has been envisaged to be a critical component for micro autonomous drones of the next generation [1]. However, the corresponding development for control is limited, especially compared with the recent progress in autonomous unmanned aerial vehicles such as quadrotors. This is mainly because the flapping wing aerial vehicles correspond to infinite-dimensional nonlinear time-varying systems, where unsteady aerodynamics is coupled with structural deformation of wings and body dynamics in a sophisticated manner.

Most of the existing control systems for flapping wing aerial vehicles bypass these challenges by averaging the linearized dynamics over a flapping period [2], [3], [4]. These works exploit the large disparity in time scales between wingbeat frequency and flight dynamics after exploiting high frequency oscillations of small wings. Consequently, they are not suitable for flapping wing aerial vehicles or insect flight with a relatively large wing flapping at a low frequency, such as in Monarch butterflies [5], which exhibit remarkable flight characteristics migrating over three thousands miles disproportionate to their size.

Recently, the authors have proposed a geometric model for flapping wing aerial vehicles inspired by Monarch butterfly [6]. This is formulated in an intrinsic fashion such that any arbitrary maneuver involving large-angle flapping is described globally without singularities caused by local parameterization. Based on this, a feedback controller is developed for just the longitudinal dynamics, where controller parameters are optimized to guarantee stability according to the Lyapunov-Floquet theory [7]. Next, optimal control trajectories are constructed via direct optimization for the coupled dynamics of the longitudinal mode and the lateral mode [8]. While this approach deals with arbitrary maneuvers in the three-dimensional space, it is computationally expensive due to the extensive number of iterations involved, and therefore, it can not be implemented in real time. In short, due to the complexities of the dynamics, it is challenging to design a control system from Lyapunov stability analysis or model-predictive online optimization.

In aerial robotics for multirotor UAVs, these have been addressed by model predictive control integrated with reinforcement learning [9]. There has been noticeable progress even in real world implementations of such sensorimotor controls. Directly mapping states to control values, [10] proposes offline training using trajectories generated from optimal controls. More recently, policies trained in simulation have been utilized in autonomous flight for challenging tasks [11], [12].

A common theme here is the idea of imitation learning which is to emulate the behavior of an expert by learning from the demonstrated actions. A naive implementation where a neural network is trained to model expert demonstrations, is referred to as behavior cloning [13]. However, its performance is not satisfactory in practice, as the mismatch of distributions between offline training data and online trajectories accumulates over time. To address this, the DAgger algorithm [14] repeatedly augments the training data with the actual state encountered during online implementation and the corresponding optimal control input of the expert. Recently, a robust imitation learning referred to as DART has been proposed, which injects noise into the expert itself to improve robustness [15]. However, these require that the expert is continuously queried online, and also, the neural network should be retrained for training data that are constantly renewed and enlarged.

In this paper, to address these challenges of imitation learning for flapping wing aerial vehicles, we propose a new scheme referred to as constrained imitation learning (COIL). The fundamental idea is that we impose an equality constraint on the neural network to guarantee zero control for zero state error, instead of augmenting the training data with additional trajectories. This injection of domain knowledge in the form of constraints has been illustrated to improve the performance of learned networks in supervised learning [16], and various end-to-end constrained optimization learning methods have been reviewed [17]. To alleviate some of the computational

Tejaswi K. C. and Taeyoung Lee, Mechanical and Aerospace Engineering, The George Washington University, Washington DC 20052 kctejaswi999@gmail.com,tylee@gwu.edu

*This research has been supported in part by NSF under the grants NSF CMMI-1761618 and CMMI-1760928.

impediments, the target values are adjusted strategically [18], whose convergence properties are analyzed in [19].

The proposed COIL is inspired by these ideas of exploiting domain knowledge to enhance the computational properties of imitation learning. The unique advantages of COIL are summarized as follows:

1) Imitation learning is tailored for control of dynamic systems to improve stability properties without causing computational burden;
2) The issue of instability in behavior cloning is addressed without enlarging the training data set;
3) There is no need to communicate with the expert online during the learning process.

This scheme is applied to construct a data-driven feedback control system for a flapping wing aerial vehicle, where the optimal control trajectories constructed offline are considered as expert demonstration. The above third property of COIL is particularly useful as there is no need to solve time-consuming, trajectory optimization repeatedly. Nor does it require Lyapunov stability analysis for sophisticated equations of motion involving aerodynamic forces and moments.

This paper is organized as follows. The dynamics of a flapping wing aerial vehicle and the offline optimal control framework are presented in Section II. The COIL scheme is proposed in Section III and it is verified numerically in Section IV. An open-source MATLAB implementation of the dynamical model, optimal controller, and learning schemes is available at https://github.com/fdcl-gwu/FWUAV.

## II. Preliminaries

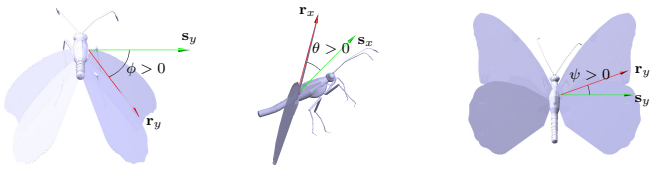
In this section, we outline the background materials required to understand the imitation learning problem formulation for the control of a flapping wing unmanned aerial vehicle (FWUAV). First, a dynamical model of FWUAV formulated in [20], [6] is introduced. Then a geometric numerical integration scheme is presented which is utilized to implement an optimal controller [8].

Throughout this paper, the three-dimensional special orthogonal group is denoted by $\mathsf{SO(3)} = \{R \in \mathbb{R}^{3\times3} \mid R^TR = I, \det(R) = 1\}$, and its Lie algebra is $\mathfrak{so}(3) = \{A \in \mathbb{R}^{3\times3} \mid A = -A^T\}$. For all numerical examples, the units are in kg, m, s, and rad, unless specified otherwise.

### A. Equations of motion

We model FWUAV as a composition of three rigid bodies, namely thorax and two wings, interconnected by spherical joints. The detailed definition of these elements are presented in [6], with an additional component of abdomen. The key components essential to the development of this paper are listed as follows.

- *Body:* Its position is denoted by $x \in \mathbb{R}^3$ and the attitude is given by $R \in \mathsf{SO(3)}$ with the angular velocity $\Omega \in \mathbb{R}^3$.
- *Right wing:* Attitude, $Q_R \in \mathsf{SO(3)}$, is characterized by 1–3–2 Euler angles $(\phi_R(t), \psi_R(t), \theta_R(t))$ (see Figure 1) along with the stroke angle, $\beta$, and the angular velocity is $\Omega_R \in \mathbb{R}^3$.




(a) flapping angle $\phi_R$ (b) pitch angle $\theta_R$ (c) deviation angle $\psi_R$

Fig. 1. Euler angles corresponding to right wing attitude [20]

- *Left Wing:* $Q_L$ is defined symmetrically to the right wing, and the angular velocity is $\Omega_L \in \mathbb{R}^3$.

In this paper, we are interested in influencing the motion of FWUAV by controlling the wings, assuming that flapping motion of the wing, represented by $Q_R(t), Q_L(t)$, is given as a function of time. This motion is in fact further parameterized by a specific model of wing kinematic parameters inspired by insect flapping [6]. Thus the whole configuration can be decomposed into a freely varying part, $g_1 = (x, R)$, $\xi_1 = [\dot{x}, \Omega]$ and a prespecified part, $g_2 = (Q_R, Q_L)$, $\xi_2 = [\Omega_R, \Omega_L]$. For given $(g_2(t), \xi_2(t))$, the governing equations for $(g_1, \xi_1)$ can be rearranged into,

$$\dot{g}_1 = g_1\xi_1, \qquad \dot{\xi}_1 = F(g_1, \xi_1, g_2, \xi_2). \tag{1}$$

As such, it corresponds to a time-varying ordinary differential equation on $\mathbb{R}^3\times\mathsf{SO(3)}\times\mathbb{R}^6$. The detailed developments of the above equations with explicit formulation are available in [8].

### B. Numerical Integrator

Since we utilize the optimal trajectories of (1) later for imitation learning, it is critical to discretize them properly. One particular challenge is that the rotation matrices evolve on a nonlinear Lie group and general purpose numerical integrators do not preserve their orthogonality [21]. For example, it is observed that the common Runge-Kutta methods are not suitable for the presented problem involving multiple rigid bodies undergoing large rotations. More specifically, it was challenging to achieve any consistency or robustness in the subsequent numerical optimization and neural network training.

To address this, we utilize Lie group methods where the group elements are updated by the group operation to preserve group structures. In particular, a Crouch-Grossman method [21, Chapter IV.8] is adopted: (1) is discretized from $t = [t_0, t_1]$ with a fixed step size of $h$ into

$$g_1(t_1) = g_1(t_0)\prod_{i=1}^{s}\exp(hb_i\xi_1^i), \tag{2}$$

$$\xi_1(t_1) = \xi_1(t_0) + h\sum_{i=1}^{s} b_i\zeta^i, \tag{3}$$

where $g_1^i \in \mathbb{R}^3\times\mathsf{SO(3)}, \xi_1^i \in \mathbb{R}^6, \zeta^i \in \mathbb{R}^6$ for $i = 1, \ldots, s$ are given by

$$g_1^i = g_1(t_0)\prod_{j=1}^{i-1}\exp(ha_{ij}\xi_1^j), \quad \xi_1^i = \xi_1(t_0) + h\sum_{j=1}^{i-1}a_{ij}\zeta^i,$$
$$t^i = t_0 + c_ih, \qquad \zeta^i = F(g_1^i, \xi_1^i, g_2(t^i), \xi_2(t^i)).$$

Here, $a \in \mathbb{R}^{s\times s}$ and $b, c \in \mathbb{R}^s$ are real valued constants satisfying certain conditions to ensure an order of accuracy. In particular, we use an explicit method of order 4 with $s = 5$ stages with the constant values in [22]. For the numerical examples considered in this paper, the error in satisfaction of the orthogonality of rotation matrices is approximately $10^{-7}$ for a 4-th order Runge-Kutta method, and it is $10^{-12}$ for the presented Crouch Grossman method.

### C. Optimal Control of FWUAV

The objective of the control system presented in this paper is to adjust the wing kinematics such that the motion of FWUAV asymptotically converges to a desired flight maneuver. This is addressed by generating a set of optimal trajectories for varying initial conditions, as presented in [8] and summarized here, which are utilized for imitation learning in the subsequent section. Let $\mathbf{x}(t) = (x(t), R(t), \dot{x}(t), \Omega(t))$ denote the state of FWUAV, and let $\mathbf{x}_d(t) = (x_d(t), R_d(t), \dot{x}_d(t), \Omega_d(t))$ be the desired trajectory. The control objective is to ensure $\mathbf{x}(t) \to \mathbf{x}_d(t)$, i.e., the trajectory asymptotically converges to the periodic orbit when $\mathbf{x}(0) \neq \mathbf{x}_d(t)$.

First, a desired trajectory has to be constructed through a constrained optimization problem while enforcing periodicity of flapping. In this paper, we consider a hovering reference flight where the position and the attitude at the end of the flapping period match with the values at the beginning of the period. Next, instead of controlling all wing kinematics parameters, to improve numerical properties and convergence of optimization, the following $N_\Delta = 6$ parameters are selected:

$$\Delta = [\Delta\phi_{m_s}, \Delta\theta_{0_s}, \Delta\phi_{m_a}, \Delta\phi_{0_s}, \Delta\theta_{0_a}, \Delta\psi_{0_a}]. \tag{4}$$

The effects of these parameters on each component of the aerodynamic force and moment are summarized in [8, Table 2].

These values of $\Delta$ are interpolated in a piecewise linear manner by dividing a flapping period into $N_s = 10$ steps, and an additional constraint $\Delta(0) = \Delta(T) = 0$ is imposed to ensure periodicity. The objective function for optimization is taken to be the discrepancy between the desired trajectory and the controlled trajectory, measured by

$$J = \sum_{i=1}^{N_p} W_i \sqrt{\sum_j (W_{\mathbf{x}_j}(\mathbf{x}_j(t_i) - \mathbf{x}_{d_j}(t_i)))^2}. \tag{5}$$

Here, the weights $W$ are chosen such that components of states are scaled by their own physical characteristics, and also to ensure that final state errors have more importance.

While solving this problem we adopt the procedure of model predictive control (MPC) using the `fmincon` solver from MATLAB. In other words, we obtain control parameters over a prediction horizon of two time periods ($N_p = 20$) while the actual controller is implemented only for the first period ($N_s = 10$). At the end of this period, the process of optimization is repeated. The detailed procedure for optimization is presented in [8].

## III. Constrained Imitation Learning

The control procedure illustrated in the previous section is not ideal for real-time implementation, as the time required for optimization is greater than the flapping period by several orders: a few minutes are required for optimization while a single flapping period is on the scale of 0.1 seconds. In this section, we adopt the framework of imitation learning, where a feedback control scheme is constructed from a set of optimal trajectories obtained offline. Popular imitation learning schemes, namely behavior cloning, DAgger and DART are introduced, and a new constrained imitation learning, referred to as COIL, is proposed.

### A. Behavior Cloning

The most straightforward approach is to utilize supervised learning and construct a neural network controller that is trained to reproduce the above optimal control trajectories for a given state. Exploiting the periodicity of the desired maneuver, the input of the neural network will be the state of FWUAV at the beginning of a flapping period and the output is wing kinematics parameters over the entire flapping period. This implies that the feedback mechanism of comparing the actual state to the desired state is activated periodically at the beginning of each period. This is sensible as the same state errors at two distinct flapping phases imply different dynamic characteristics, and it reduces the dimension of the input for the neural network substantially. Numerical examples considered in this paper indicate that stabilization is achievable with this approach, but it can be readily extended to incorporate state errors in other phases of the flapping period.

To generate the training data set, the above optimization problem is solved repeatedly for varying initial conditions. We select random initial errors in the form of $\Delta\mathbf{x}(0) = [\Delta x(0), \Delta R(0), \Delta\dot{x}(0), \Delta\Omega(0)] \in \mathbb{R}^{12}$, defined by

$$\Delta x = x - x_d, \quad \Delta R = \frac{1}{2}(R_d^T R - R^T R_d)^\vee,$$
$$\Delta\dot{x} = \dot{x} - \dot{x}_d, \quad \Delta\Omega = \Omega - R^T R_d \Omega_d.$$

Each part of this error vector is sampled from the uniform sphere in $\mathbb{R}^3$, and scaled to ensure that all states are varied in a similar physical level. For each initial error, the preceding optimization problem is solved to construct the optimal control parameters $u \in \mathbb{R}^{60}$, which is the values of $\Delta \in \mathbb{R}^6$ defined in (4) over $N_s = 10$ points in a flapping period. The training data are composed of $N$ pairs of the corresponding $(\Delta\mathbf{x}(0), u)$, which also include $N_0$ pairs of $(\Delta\mathbf{x}(0) = 0, u = 0)$ to promote that zero state error results in zero control.

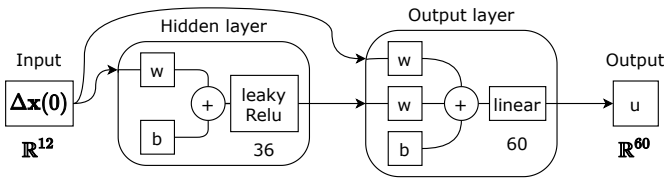



Fig. 2. Neural network architecture

Next a neural network is formulated as a simple cascade-forward architecture with a single hidden layer composed of 36 leakyRelu neurons (see Figure 2). Let its input matrix be $X \in \mathbb{R}^{12\times N}$, whose columns $\{X_k, k = 1 \dots N\}$ correspond to the values of $\Delta\mathbf{x}(0)$. Similarly the output matrix $Y \in \mathbb{R}^{60\times N}$ is constructed by the optimal control parameters. The input-output relation of the neural network is represented by $\hat{Y} = f(X, \theta)$, where $\theta \in \mathbb{R}^{N_\theta}$ are its parameters composed of weights and biases, and $\hat{Y}$ is the predicted output. The parameters $\theta$ are trained to minimize a loss function, $L(\hat{Y}, Y)$ which is often chosen as the mean squared error in regression.

The detailed implementation and the corresponding controlled trajectories are presented in Section IV. It turns out that the resulting performance is not satisfactory. Following an initial convergence of the state errors with respect to the reference trajectory, the controlled trajectories start to diverge as the number of flapping increases. This is not surprising as the presented approach based on supervised learning, referred to as behavior cloning, does not perform well in practice. It is because the particular states encountered in the controlled trajectory may not be well represented by any of the optimal trajectories in the training data. The mismatch of the distributions between the training data set and the actual controlled trajectory accumulates over time inevitably.

### B. DAgger

The issues of behavior cloning can be mitigated by increasing the size of the training data so that any state in the controlled trajectory is sufficiently close to an optimal trajectory in the data set. However this will increase the computational cost substantially in terms of both trajectory optimization and neural network training. Alternatively, a popular benchmark algorithm, referred to as DAgger addresses the problems of behavior cloning by integrating online learning and iterative training [14]. It involves interaction with the expert during training as summarized below.

**Algorithm 1** DAgger

**Input:** $(X, Y), N_i \in \mathbb{Z}$

1: **for** $i = 1$ **to** $N_i$ **do**
2: $\quad\theta^i = \arg\min_\theta\{L(\hat{Y}, Y) \mid \hat{Y} = f(X, \theta)\}$
3: $\quad$Execute the trained policy to construct on-policy trajectories
4: $\quad$Add these new states, $X_k$ into the input set $X$
5: $\quad$Obtain optimal control for new states through optimization, $Y_k$, and add them to the output set $Y$
6: **end for**

**Output:** $\theta^{N_i}$

Algorithm 1 describes the DAgger approach as applied to our control problem. With the given data set, the neural network model is trained at Step 2 as in behavior cloning. Then in Step 3, the trained neural network is executed over several periods to get new controlled trajectories. The offline optimal control presented in Section II-C is utilized for these new states to obtain the corresponding optimal control inputs, which are added to the training dataset. This procedure is repeated for $N_i$ iterations to improve the performance.

In short, training is repeated with the new data set obtained from on-policy, online trajectories. As the particular states encountered in the controlled trajectories are constantly added to training, the aforementioned issue of the distribution mismatch is directly addressed. However, this approach is computationally expensive as the size of the training data increases and the time-consuming trajectory optimization should be solved repeatedly. In the perspective of imitation learning, the expert (represented by trajectory optimization in this paper) should be available for interaction during the learning process.

### C. DART

Next, the robustness of imitation learning has been addressed in DART, by injecting noise into the expert's policy [15]. This is completed off-policy, thereby avoiding undesirable visits to potentially unsafe states during training. However it can provide corrective guidance to the controller since it enlarges the boundary of the expert's distribution. Specifically, the actual value of noise is determined by minimizing the difference between the distribution of the export policy and that of the trained policy.

The particular implementation of DART for the presented control problem is summarized in Algorithm 2. First, Step 1 corresponds to the initial training with the data obtained without noise. Next, let the current policy and the expert policy be denoted by $\pi_{\theta^i}$ and $\pi_{\theta^*}$, respectively. Then in Step 3, the second moment of $\pi_{\theta^{i-1}} - \pi_{\theta^*}$ is estimated by $\hat{\mathbf{\Sigma}}$. Later this value is scaled in Step 4 to account the anticipated final policy error through a parameter, $\alpha_d$. The expert policy realized by the optimal control Section II-C is augmented with the additive noise. Finally, the neural network model is trained at Step 7 with the aggregated dataset. These iterations are repeated.

**Algorithm 2** DART

**Input:** $(X, Y), N_i \in \mathbb{Z}, \alpha_d \in \mathbb{R}$

1. $\theta^0 = \arg\min_\theta\{L(\hat{Y}, Y) \mid \hat{Y} = f(X, \theta)\}$
2. **for** $i = 1$ **to** $N_i$ **do**
3. $\quad\hat{\mathbf{\Sigma}}^i \approx \frac{1}{N}\sum_k(\pi_{\theta^{i-1}}(X_k) - \pi_{\theta^*}(X_k))(\pi_{\theta^{i-1}}(X_k) - \pi_{\theta^*}(X_k))^T$
4. $\quad\mathbf{\Sigma}^i = \dfrac{\alpha_d}{N\mathrm{tr}[\hat{\mathbf{\Sigma}}^i]}\hat{\mathbf{\Sigma}}^i$
5. $\quad$Collect data using the optimal control expert with injected noise : $\mathcal{N}(\pi_{\theta^*}, \mathbf{\Sigma}^i)$
6. $\quad$Add these input output pairs to the dataset $(X, Y)$
7. $\quad\theta^i = \arg\min_\theta\{L(\hat{Y}, Y) \mid \hat{Y} = f(X, \theta)\}$
8. **end for**

**Output:** $\theta^{N_i}$

In short, DART addresses the drawbacks of DAgger through an off-policy procedure of adding noise to the expert, which is carefully chosen to minimize the mismatch between the expert and the trained policy. However, DART still requires interacting with the expert online in Step 5, which causes substantial computational load.

### D. Constrained Imitation Learning

Now we propose constrained imitation learning to address these issues. It turns out that due to the limited number of optimal trajectories in the training data, the controller does not perform properly especially near the desired trajectory. Even though the controller performs reasonably well during the initial transient period with large errors, it is unable to maintain the boundedness of the small error over a longer period of time.

This has been partially mitigated by augmenting the training data set with the additional pairs of $(\Delta \mathbf{x}(0) = 0, u = 0)$. But, it does not resolve the issue completely. Another alternative workaround would be enforcing the constraint of zero input to zero output directly on the neural network structure, by eliminating all of the bias terms in Figure 2. However, this method drastically reduces the learning capabilities of the neural network and it results in poor generalization.

In this paper, we propose a new imitation learning scheme inspired by the recent studies on supervised learning with constraints [18], [19]. To improve long-term convergence, the zero input, zero output constraint is incorporated in the training process. Instead of varying the neural network parameters to learn the target and to satisfy the constraint simultaneously, the learning target is adjusted to mitigate the computational challenges in constrained supervised learning. The foremost advantage is that it does not require the solution of trajectory optimization for new states online, and therefore, the size of the training data is fixed.

More specifically, the neural network training for the proposed constrained imitation learning is formulated into the following optimization problem:

$$\theta^* = \arg\min_{\theta} \left\{ L(\hat{Y}, Y) \mid \hat{Y} = f(X, \theta), \text{ and } f(0, \theta) = 0 \right\}.$$

The proposed procedure to address this optimization is summarized at Algorithm 3.

**Algorithm 3** Constrained Imitation Learning (COIL)

**Input:** $(X, Y), \alpha \in \mathbb{R}, N_i \in \mathbb{Z}$

1. $\theta^0 = \arg\min_\theta \{L(\hat{Y}, Y) \mid \hat{Y} = f(X, \theta)\}$ # initial training
2. $\hat{Y}^0 = f(X, \theta^0)$
3. **for** $i = 1$ **to** $N_i$ **do**
4. $\quad \theta_{z_0} = \arg\min_\theta \{L(Z_0, (1-\alpha)Y + \alpha \hat{Y}^{i-1}) \mid Z_0 = f(X, \theta)\}$
5. $\quad \mathbf{F} \approx \frac{1}{N} \sum_k \nabla_\theta \log p_{\theta_{z_0}}(Y_k|X_k) \nabla_\theta \log p_{\theta_{z_0}}(Y_k|X_k)^T$
6. $\quad \theta_z = \arg\min_\theta \frac{1}{2} \|\theta - \theta_{z_0}\|^2_{\mathbf{F}}, \quad f(0, \theta) = 0$ # constrained optimization
7. $\quad Z^i = f(X, \theta_z)$ # target adjustment
8. $\quad \theta^i = \arg\min_\theta \{L(\hat{Y}, Z^i) \mid \hat{Y} = f(X, \theta)\}$ # unconstrained training
9. $\quad \hat{Y}^i = f(X, \theta^i)$
10. **end for**

**Output:** $\theta^{N_i}$

First, there is the initial training step which is implemented without considering any constraints. This part is composed of Steps 1 and 2, and it is equivalent to behavior cloning. Next, the iterative parts of Steps 3 through 10, are motivated by the main algorithm in [19] and consist of the combination of target adjustment and unconstrained training. Essentially the target $Y$ is adjusted into a new target $Z$ such that unconstrained training with respect to $Z$ automatically satisfies the given constraint.

Steps 4-7 correspond to the process of finding the alternative target $Z$ that is more feasible for the constraint. Initially in Step 4, the neural network is trained such that its output becomes closest to $(1-\alpha)Y + \alpha\hat{Y}$ for $\alpha \in [0, 1]$. This is on the line connecting the ideal output $Y$ and the output $\hat{Y}^{i-1}$ adjusted for the constraint in the prior iteration. As such, this has the net effect of adjusting the neural network parameters to find the compromise between emulating the ideal output (smaller $\alpha$) and satisfying the constraint (larger $\alpha$).

The next part of Steps 5 and 6 directly addresses the constraint via nonlinear constrained optimization. In Step 6, the neural network parameters $\theta_z$ are optimized to enforce the zero input, zero output constraint while minimizing the parameter deviation from the value $\theta_{z_0}$ obtained in Step 4. The motivation for the objective function $\|\theta - \theta_{z_0}\|^2_{\mathbf{F}}$ is to ensure the consistency of the resulting control policy. In other words, we wish that the optimal control system represented by the neural network is not drastically altered while enforcing the constraint of $f(0, \theta) = 0$.

More specifically, let $\pi_\theta$ represent the Gaussian policy obtained by the current value of neural network parameters $\theta$, i.e., $\pi_\theta(Y|X) \sim \mathcal{N}(f(X, \theta), \mathbf{\Sigma})$ for a covariance $\mathbf{\Sigma} \in \mathbb{R}^{60\times 60}$. Similarly, $\pi_{\theta_z}$ is constructed from $\theta_{z_0}$. The difference between those two control policies can be measured by the KL-divergence approximated by

$$D_{KL}(\pi_\theta || \pi_{\theta_{z_0}}) \approx \frac{1}{2}(\theta - \theta_{z_0})^T \mathbf{F} (\theta - \theta_{z_0}), \tag{6}$$

where $\mathbf{F} \in \mathbb{R}^{N_\theta \times N_\theta}$ is the Fisher information matrix (FIM) [23] defined by

$$\mathbf{F} = \mathrm{E}_{\pi_\theta} \left[ \nabla_\theta \log \pi_\theta(Y|X) \nabla_\theta \log \pi_\theta(Y|X)^T \right]. \tag{7}$$

For the Gaussian distribution, the gradient term in (7) can be evaluated by

$$\begin{aligned} \nabla_\theta \log \pi_\theta(Y|X) &= \nabla_\theta \left( -\frac{1}{2} \|Y - f(X, \theta)\|^2_{\mathbf{\Sigma}^{-1}} \right) \\ &= (\nabla_\theta f(X, \theta)) \mathbf{\Sigma}^{-1} (Y - f(X, \theta)). \end{aligned} \tag{8}$$

The gradient $\nabla_\theta f$ in (8) is computed through backpropagation, and it is readily available for any deep learning software library. During numerical implementation, the arithmetic mean in (7) is replaced by the sample mean with each element of the training data, as shown at Step 5, and the covariance matrix can be chosen as $\mathbf{\Sigma} = I$ for our purposes [23].

The role of the Fisher information matrix is critical in Step 6. If we directly optimize a simple square error term, $\|\theta - \theta_{z_0}\|^2$, then it takes a lot of iterations to converge. This is because the sensitivity of $f(X, \theta)$ with respect to $\theta$ varies greatly. In other words, some parameters change the output of the neural network drastically while others are not strongly correlated to the output. The matrix $\mathbf{F}$ properly scales the neural network parameters to have more uniform effects

on the output, thereby improving computational properties of optimization.

Once we obtain the adjusted parameter $\theta_z$, the corresponding target adjusted for the constraint is computed in Step 7. In the remaining steps, the neural network is retrained for the adjusted target, and these iterations are repeated $N_i$ times. The desirable feature is that the neural network training in Step 4 and Step 8 do not consider the constraint explicitly. As such, these steps can be performed with any supervised learning technique.

The key difference of this procedure from [19, Algorithm 1] (which we will refer to as $RC$) is the explicit form of the constraint considered here. More specifically, the zero input, zero output constraint that is enforced here is a function of the parameters of the neural network with respect to a specific input, i.e., $f(0,\theta) = 0$. But, $RC$ requires that the output of the network for any input, namely $\hat{Y}$ belongs to a prescribed feasible set. The infeasible adjustment step of $RC$ does not involve any neural network training, and it has to be modified into multiple steps leading to the constrained optimization in Algorithm 3. That is, first in Step 4 the network parameters are estimated after training on $(1-\alpha)Y+\alpha\hat{Y}$. Next, the constraint is enforced in Step 6 after the Fisher information matrix $\mathbf{F}$ is estimated using these obtained parameters. Moreover, there is no feasible adjustment step of $RC$ since the constraint here will not be exactly satisfied.

This Algorithm 3 is referred to as constrained imitation learning (COIL). Compared with DAgger and DART, the most distinctive benefit is that we do not have to solve the cumbersome trajectory optimization during the iteration. In other words, there is no need to communicate with the expert to acquire an additional optimal trajectory. This also does not increase the size of the training data constantly. While there are two neural network training steps per an iteration, compared with a single training in the other two algorithms, each training corresponds to minor adjustments for the adjusted targets within the existing data. Thus, it can be solved more efficiently compared with the training in DAgger and DART involving new input and output pairs. In short, the zero input, zero output constraint is carefully incorporated into the neural network training without increasing the computational load drastically. This is illustrated by numerical examples in the next section.

## IV. Numerical Simulation

In this section, we present numerical properties of all of behavior cloning, DAgger, DART and COIL applied to a FWUAV model presented in [6], [8]. We focus on evaluating the performance through the initial convergence rate and the ultimate boundedness, compared against the required computation time. As it is not feasible to investigate stability properties with Lyapunov analysis, we perform an exhaustive numerical study where the controlled trajectories are computed for numerous initial conditions within a prescribed domain. This is to ensure reasonable performance for any new initial condition that is not close to training data.

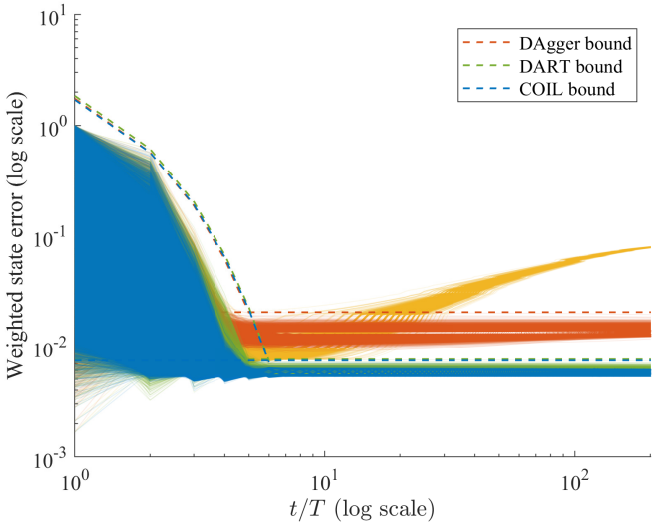



Fig. 3. Comparison of controller performance : Behavior Cloning (yellow), DAgger (orange), DART (green), COIL (blue); dashed lines represent the ultimate bounds while the initial decay is characterized by a bounding exponential function

More specifically, motivated by the definition of ultimate uniform boundedness in [24], we identify positive constants $\gamma, t_T, b$ satisfying

$$\begin{aligned} \|\mathbf{x}(t)-\mathbf{x}_d(t)\|_{W_\mathbf{x}} &\le \|\mathbf{x}(0)-\mathbf{x}_d(0)\|_{W_\mathbf{x}} e^{-\gamma t}, \forall\ 0 \le t \le t_T, \\ \|\mathbf{x}(t)-\mathbf{x}_d(t)\|_{W_\mathbf{x}} &\le b, \quad \forall\ t \ge t_T \end{aligned} \tag{9}$$

where $\|\mathbf{x}\|_{W_\mathbf{x}} = \sqrt{\sum_i (W_{\mathbf{x}_i}\mathbf{x}_i)^2}$ is a weighted two norm. Here $\gamma$ corresponds to the initial exponential convergence rate, and $b$ is the ultimate bound. Next, $t_T$ is the first time to reach the ultimate bound.

### A. Controller Performance

The above values are determined by simulating 12291 trajectories initiated in the set of $\|\mathbf{x}(0)-\mathbf{x}_d(0)\|_{W_\mathbf{x}} \le 1$ for each method.

*a) Behavior Cloning:* Behavior cloning is implemented with a neural network that has $N_\theta = 3408$ parameters (Figure 2), and the size of the training dataset is $N = 1587$ including $N_0 = 300$ pairs of zero points. The corresponding convergence of the tracking error is illustrated in Figure 3 for all initial conditions. It is observed that there is a non-negligible increase of the error after about 50 flapping periods, implying an ultimate bound can not be characterized for the given simulation period.

*b) DAgger:* This is implemented according to Algorithm 1 with $N_i = 5$ iterations. At Step 3, the trained neural network is executed over 5 time periods to obtain around 150 new controlled trajectories. However the controller failed to make the error converge. Therefore, we had to increase the number of neurons in the hidden layer from 36 to 60 (resulting $N_\theta = 5160$) and also the number of initial training data to $N = 3049$. The resulting tracking error is presented in Figure 3. The issue of behavior cloning is addressed to attain the ultimate bound of $b = 0.0204$. However, this is achieved with more training data and a larger neural network.

TABLE I
NUMERICAL COMPARISON

| Algorithm | BC | DAgger | DART | COIL |
|---|---|---|---|---|
| Computation time (min) | 3.8 | 132.22 | 131.08 | 44.53 |
| Ultimate bound, $b$ | N/A | 0.0204 | 0.0076 | 0.0074 |
| Initial decay rate | 1.1296 | 1.1168 | 1.1046 | 1.0944 |
| Size of network, $N_\theta$ | 3408 | 5160 | 3408 | 3408 |
| Initial training data size, $N$ | 1587 | 3049 | 2012 | 1587 |

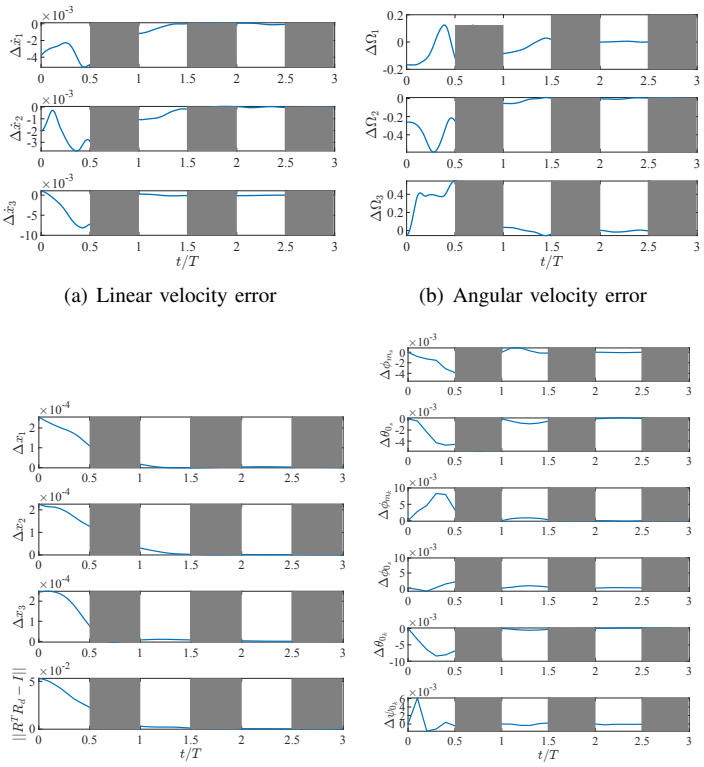
*c) DART:* Next, Algorithm 2 corresponding to DART is implemented with $N_i = 5$ iterations again. The scaling factor in Step 4 is taken to be $\frac{\alpha_d}{N\mathrm{tr}[\hat{\mathbf{\Sigma}}^i]} = 10^{-4}$. The value of $\alpha_d$ has to be relatively small since in our case, even a minor change in the control value obtained from expert leads to a large perturbation in the state due to the complexity of the dynamics. Then in Step 5, the data are collected using the noisy optimal controller over 5 time periods resulting in 150 new state and control values. Unlike DAgger, the number of network parameters did not have to be increased from $N_\theta = 3408$, but the size of initial training data in Step 1 still had to be slightly increased to $N = 2012$. It can be observed from Figure 3 that DART yields a smaller ultimate bound of $b = 0.0076$.

*d) Constrained Imitation Learning:* Finally, the proposed COIL is implemented by Algorithm 3 with the same number of iterations, $N_i = 5$. The trade-off parameter between the ideal and current labels is taken to be $\alpha = 0.75$ which gives a little more weight to constraint adjustment. The network structure and the training data are identical to those of behavior cloning, i.e., $N_\theta = 3408$ and $N = 1587$. As presented in Figure 3, COIL successfully achieved the ultimate bound of $b = 0.0074$, which is lower than DAggger and comparable to DART.

## B. Results

These results are summarized in Table I, with computation time when implemented in MATLAB with Intel(R) Xeon(R) Silver 4216 CPU. While behavior cloning requires the shortest computation time, its performance is not desirable as presented above. DAgger achieves ultimate boundedness, but it requires the most computation time, which is caused by time-consuming trajectory optimization solved in every iteration (Step 5 in Algorithm 1), and the more complex network architecture. Meanwhile, DART performs better compared to DAgger in terms of the ultimate bound. On the other hand, the proposed COIL achieves the smallest ultimate bound and the least computation time with a similar initial decay rate. This directly illustrates the desirable features of COIL addressing the issue of distribution mismatch of behavior cloning without excessive additional computation required in DAgger and DART.

All of these imitation learning approaches can be improved by providing more data or by adopting more sophisticated network model, according to the universal approximation property of neural networks. For instance, if sufficient data are available, even behavior cloning may perform in a satisfactory manner. The proposed approach improves sampling efficiency by imposing a certain structure, and further, it eliminates the need to interact with the expert online.




(a) Linear velocity error (b) Angular velocity error

(c) Position and attitude error (d) Control inputs

Fig. 4. Optimal trajectory errors and control parameters

The value of the zero input-zero output error measured as $\|f(0,\theta)\|_2$ for each of DAgger, DART and COIL are $0.1849, 0.0618$, and $0.0007$ respectively. On the other hand, the MSE in training the network are $0.0062, 0.0063$, $0.0083$ respectively. The COIL exhibits the lowest error in satisfaction of the constraint since it has been explicitly considered in the algorithm, which also causes a trade off in the training error. The intriguing fact is that the trade off included by the constraints actually improves the stability properties.

Next, the average time taken by the MPC expert is over $100$ seconds. On the other hand, the inference time of the control policy for a given state error is less than $10^{-4}$ seconds. In the presented problem, it would be required to obtain new control values at the start of each trajectory time period which is around $0.085$ seconds (flapping frequency of $11.75$ Hz). This justifies the proposed imitation learning compared with other techniques based on online optimization.

As an illustration, the tracking error in each of position, velocity, and attitude for a single particular trajectory is presented in Figure 4, along with COIL control inputs. This illustrates that the hovering flight of FWUAV is successfully stabilized without the common averaging or linearization for the coupled longitudinal and lateral dynamics.

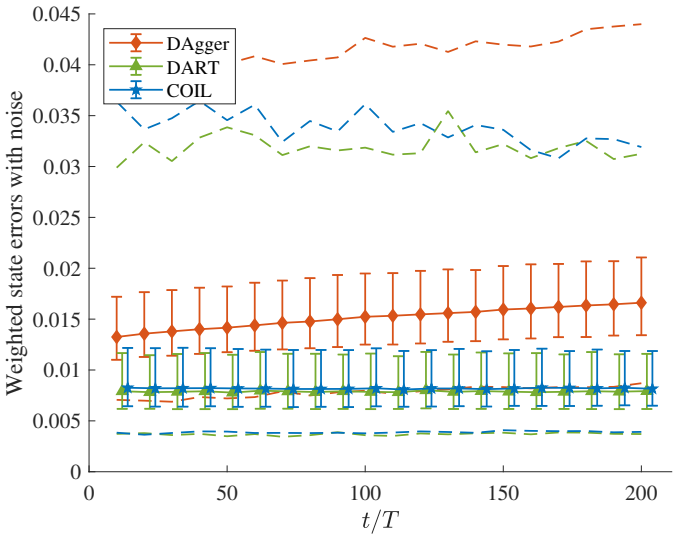



Fig. 5. Comparison of controller performance with measurement noise : box plots represent 0.25, 0.5 (median), 0.75 quartiles of errors; dashed lines correspond to maximum and minimum values

### C. Robustness to Noise

Next, we analyze the robustness properties. In particular, the state input to the control policy is corrupted by an additive noise, which can be regarded as errors in the estimate of the FWUAV state. Random noise is generated from uniform spheres for each part of the state, similar to the initial errors for training data, and scaled with the weights of the states multiplied by the noise level, taken as $10^{-3}$. Then it is added to the actual state value before being passed to the trained neural network. We perform numerical simulations over multiple random initial conditions, as completed in Figure 3. The corresponding tracking errors after 10 periods are presented in the form of summary statistics in Figure 5. The results for the first 10 periods are omitted as they are close to Figure 3 when the state errors are large. It is observed that DAgger is most susceptible to the noise: it yields the largest maximum error and the mean error increases over time. Meanwhile, DART and COIL exhibit comparable performance, where the mean error remains within an acceptable reason, and it decreases slightly over time. In short, the proposed COIL has reasonable robustness properties against DART, one of the recent imitation learning schemes that are specifically designed to improve robustness.

## V. Conclusions

This paper presents imitation learning for a flapping wing aerial vehicle inspired by Monarch butterflies, where a set of optimal trajectories computed offline is considered as an expert demonstration to be emulated. A new constrained imitation learning is proposed to achieve exponential convergence and ultimate boundedness without relying on additional trajectory optimization, thereby improving computational efficiency substantially. This is successfully applied to the nonlinear dynamics of a flapping wing aerial vehicle. A potential future direction is utilizing visual input directly in sensorimotor control since it is challenging to estimate the complete state of flapping wing aerial vehicle in practice.

## References


[1] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, 2015.

[2] C. T. Orlowski and A. R. Girard, "Dynamics, stability, and control analyses of flapping wing micro-air vehicles," *Progress in Aerospace Sciences*, vol. 51, pp. 18–30, may 2012.

[3] M. Sun, "Insect flight dynamics: stability and control," *Reviews of Modern Physics*, vol. 86, no. 2, p. 615, 2014.

[4] W. Shyy, C.-k. Kang, P. Chirarattananon, S. Ravi, and H. Liu, "Aerodynamics, sensing and control of insect-scale flapping-wing flight," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 472, no. 2186, p. 20150712, 2016.

[5] M. K. Sridhar, C. Kang, D. B. Landrum, H. Aono, S. L. Mathis, and T. Lee, "Effects of flight altitude on the lift generation of Monarch butterflies: from sea level to overwintering mountain," *Bioinspiration & Biomimetics*, vol. 16, no. 3, p. 034002, mar 2021.

[6] T. C, M. Sridhar, C. Kang, and T. Lee, "Effects of abdomen undulation in energy consumption and stability for the flights of Monarch butterfly," *Bioinspiration & Biomemetics*, vol. 16, no. 4, p. 046003, 5 2021.

[7] T. C, C. Kang, and T. Lee, "Dynamics and control of a flapping wing uav with abdomen undulation inspired by Monarch butterfly," in *Proceedings of the American Control Conference*, 2021.

[8] T. C and T. Lee, "Geometric optimal controls for flapping wing UAV on a Lie group," in *IFAC Workshop on Lagrangian and Hamiltonian Methods for Nonlinear Control*, 2021.

[9] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 528–535.

[10] S. Li, E. Öztürk, C. De Wagter, G. C. De Croon, and D. Izzo, "Aggressive online control of a quadrotor via deep network representations of optimality principles," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6282–6287.

[11] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," *arXiv preprint arXiv:2006.05768*, 2020.

[12] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.

[13] P. Batavia, D. Pomerleau, and C. Thorpe, "Applying advanced learning algorithms to ALVINN," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-96-31, October 1996.

[14] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[15] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," in *Conference on robot learning*. PMLR, 2017, pp. 143–156.

[16] A. Borghesi, F. Baldo, and M. Milano, "Improving deep learning models via constraint-based domain knowledge: a brief survey," *arXiv preprint arXiv:2005.10691*, 2020.

[17] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder, "End-to-end constrained optimization learning: A survey," *arXiv preprint arXiv:2103.16378*, 2021.

[18] F. Detassis, M. Lombardi, and M. Milano, "Teaching the old dog new tricks: supervised learning with constraints." in *NeHuAI@ ECAI*, 2020, pp. 44–51.

[19] T. C and T. Lee, "Iterative supervised learning for regression with constraints," *arXiv preprint arXiv:2201.06529*, 2022.

[20] M. Sridhar, C.-K. Kang, and T. Lee, "Geometric formulation for the dynamics of Monarch butterfly with the effects of abdomen undulation," in *AIAA Scitech Forum*, 2020, AIAA-2020-1962.

[21] E. Haier, C. Lubich, and G. Wanner, *Geometric Numerical integration: structure-preserving algorithms for ordinary differential equations*. Springer, 2006.

[22] Z. Jackiewicz, A. Marthinsen, and B. Owren, "Construction of Runge–Kutta methods of Crouch–Grossman type of high order," *Advances in Computational Mathematics*, vol. 13, no. 4, pp. 405–415, 2000.

[23] R. Pascanu and Y. Bengio, "Revisiting natural gradient for deep networks," *arXiv preprint arXiv:1301.3584*, 2013.
[24] H. K. Khalil, *Nonlinear systems*. Prentice-Hall, 2002.