2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 23-27, 2022, Kyoto, Japan

Automatic Co-Design of Aerial Robots Using a Graph Grammar

Allan Zhao[1], Tao Du[1], Jie Xu[1], Josie Hughes[2], Juan Salazar[1], Pingchuan Ma[1], Wei Wang[1], Daniela Rus[1], and Wojciech Matusik[1]

Abstract— **Unmanned aerial vehicles (UAVs) have broad applications including disaster response, transportation, photography, and mapping. A significant bottleneck in the development of UAVs is the limited availability of automatic tools for task-specific co-design of a UAV's shape and controller. The development of such tools is particularly challenging as UAVs can take many forms, including fixed-wing planes, radial copters, and hybrid topologies, with each class of topology showing different advantages. In this work, we present a computational design pipeline for UAVs based on a graph grammar that can search across a wide range of topologies. Graphs generated by the grammar encode different topologies and component selections, while continuous parameters encode the dimensions and properties of each component. We further augment the shape representation with deformation cages, which allow expressing a variety of wing shapes. Each UAV design is associated with an LQR controller with tunable continuous parameters. To search over this complex discrete and continuous design space, we develop a hybrid algorithm that combines discrete graph search strategies and gradient-based continuous optimization methods using a differentiable UAV simulator. We evaluate our pipeline on a set of simulated flight tasks requiring dynamic motions, showing that it discovers novel UAV designs that outperform canonical UAVs typically made by engineers.**

I. Introduction

Improvements in hardware capabilities, perception, and control have led to a rapid uptake of aerial robots across many application domains [1], including remote sensing [2], search and rescue [3], delivery of goods [4] and precision farming [5]. In this work, we focus on designing unmanned aerial vehicles (UAVs), which show significant variation in topology for different applications due to their operational requirements, such as load carrying capabilities, maneuverability, and operational range. For example, quadcopters are preferable [6] for applications requiring hovering, fixed-wing gliders offer high efficiency [7] for long-distance tasks, and hybrid systems, combining both rotor and fixed wings, offer a compromise [8]. The design space for UAVs is therefore expansive, highly varied, and application-specific. It is also particularly challenging to parameterize as it must capture both discrete parameters (which describe the topology) and continuous parameters (which describe, for example, the size of the frame or the coefficients of a controller). Currently, engineers typically perform the design of a UAV manually, utilizing domain-specific knowledge, design instinct, and experience [9].

[1] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology (email to Allan Zhao: azhao@csail.mit.edu).

[2] Computational Robot Design and Fabrication Lab, École polytechnique fédérale de Lausanne (EPFL).

There have been several computational design tools for automating the UAV design process [10]–[12]. While these tools make significant advances, they explore only a limited shape space describing a fixed category of UAVs with the same topology, e.g., multirotors with continuously varying sizes [10]. A significant bottleneck in existing methods is the lack of a proper UAV shape representation that allows for both continuous variations and topological changes. Inspired by previous work on graph grammars [13]–[17] and shape deformation [18]–[20], we present a novel, hybrid UAV shape representation combining a new graph grammar and cage-based deformation. Our proposed graph grammar consists of a set of carefully chosen rules that encode various UAV topologies, including quadrotors, fixed-wing planes, and hybrid UAVs. Moreover, the cage-based deformation allows for continuous variations of individual UAV components, e.g., changing the thickness of a frame shell or the wingspan of a UAV. Similar to previous work [10], we further equip a UAV shape with an LQR controller whose coefficients are trainable continuous parameters. The shape design space and the LQR controller design space define our co-design space of UAVs' shapes and controllers with both discrete and continuous parameters.

Unlike existing papers which only optimize continuous parameters in UAV designs [10], [11], our work presents a hybrid design space. To find an optimized UAV design for a given task, we need a novel and effective search algorithm to explore both the discrete topologies and the continuous parameters. We present a hybrid algorithm that combines the benefits of discrete search and gradient-based continuous optimization. Specifically, we run gradient-based continuous optimization algorithms for a fixed UAV topology to tune the continuous shape and control parameters, with the gradient information from a differentiable UAV simulator we developed in this work. With the continuous parameters optimized, we then run discrete search strategies to explore various UAV topologies. We demonstrate the efficacy of our algorithm on a few flight tasks, and our experimental results show that it can automatically re-discover canonical UAV designs typically handcrafted by engineers. In addition, our algorithm reveals novel UAV designs with performances comparable to or better than these canonical designs, highlighting the advantage of our computational UAV co-design pipeline.

Our work makes the following contributions in developing and implementing this computational UAV design pipeline: First, we present a novel UAV co-design space enabling both discrete and continuous variations in UAV shapes. Second, we provide an effective, hybrid search algorithm that co-

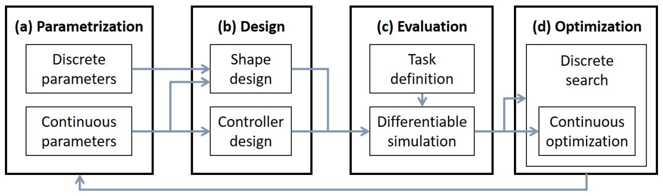

Fig. 1: An overview of our computational UAV design pipeline. (a): Our pipeline starts with a set of discrete and continuous parameters. (b): The design parameters determine the UAV shape (Sec. III) and controller (Sec. IV). (c): We evaluate our design in a differentiable simulator for given tasks (Sec. V). (d): The evaluation is then fed into our optimization algorithm (Sec. VI), which runs both discrete search and continuous optimization to update the design parameters.

optimizes the discrete and continuous design parameters, together with a differentiable UAV simulator we developed. Finally, we evaluate our computational design pipeline on dynamic flight tasks to demonstrate its value in both automatically re-discovering canonical UAV designs and revealing new higher-performance designs.

II. System Overview

We present an overview of our computational UAV design pipeline in Fig. 1. Our design consists of discrete and continuous parameters (Fig. 1 (a)). The discrete parameters refer to the choices of the graph grammar rules, which determine the topology of the UAV shape (Fig. 1 (b) top). The continuous parameters consist of the cage variables and the LQR coefficients, which define the shape of individual UAV components and the controller (Fig. 1 (b) bottom), respectively. Once we define the design, we evaluate its performance on a given task (Fig. 1 (c) top) using a differentiable simulator (Fig. 1 (c) bottom). Finally, we use the evaluation result to guide the discrete search and perform gradient-based optimization (Fig. 1 (d)) to update the design parameters (Fig. 1 (a)).

III. Shape Design

Our shape design space can represent many existing UAV designs of various sizes and topologies by incorporating both continuous and discrete design parameters. Below, we explain them in detail.

A. Continuous Parametrization

We first define a library of UAV components, e.g., rotors, wings, rods, and batteries. The location of each component in a UAV design is parametrized by continuous parameters, e.g., position and orientation of a motor or installation angle of a wing. Furthermore, we utilize a deformation-cage representation [18]–[20] that enables continuous variations of the geometry of certain components. This is especially useful for expressing various wing designs with different cross sections. More concretely, we immerse a default wing shape in an axis-aligned bounding box and store the relative

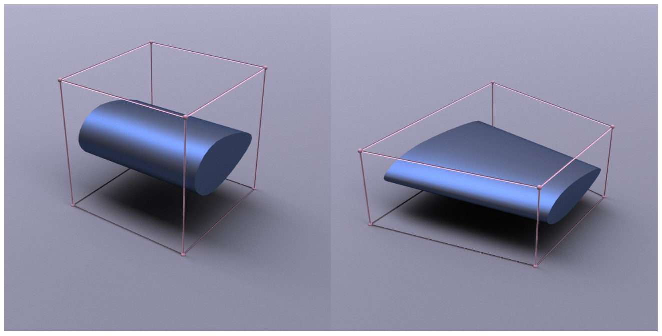

Fig. 2: Illustration of a wing segment immersed in a deformation cage (left) and its new shape after the handles of the cage are moved (right).

position of each surface point from the wing (Fig. 2, left). By varying the eight corners of the bounding box, we recalculate the position of each surface point by a trilinear interpolation using its stored relative position as the weight (Fig. 2, right).

B. Graph Grammar

The above continuous parametrization allows us to change smoothly the position, orientation, or geometry of an individual UAV component. To assemble these components into a UAV, we now introduce a graph-based grammar that allows us to express various UAV topologies. The grammar, illustrated in Fig. 3, includes a starting symbol and other non-terminal and terminal symbols. It also includes a number of rules which define the growth of a graph G from the starting symbol. These rules are of the form of $A \rightarrow B$ where a non-terminal symbol A is replaced by B, which contains terminal or non-terminal symbols enabling the recursive growth. We repeatedly apply rules to grow a graph from the starting symbol and terminate when no more non-terminal symbols exist in the graph. In our UAV designs, each terminal symbol is a component from the UAV component library with a continuous parametrization described before, and the rules define the permissive connections between these components.

As an example, we present the progressive update of a graph that leads to a standard quadrotor design and highlight the final design graph in Fig. 4.

To summarize, we represent a UAV shape with a graph G and the union of the continuous parameters in its terminal symbols. We stack these continuous parameters into a column vector θ and represent a UAV shape as a (G, θ) pair. The graph G determines the topology of a UAV, and the vector θ controls the size and shape of each individual component. The (G, θ) parametrization defines a mixed continuous-discrete shape space which grows combinatorially with the number of grammar rules.

IV. Controller Design

For any flight task, a high-performing UAV requires not only a proper shape design but also a suitable controller. In this work, we focus on controlling the dynamic motion of a UAV using the Linear-Quadratic Regulator (LQR), a classical control method that stabilizes a UAV around its steady states,

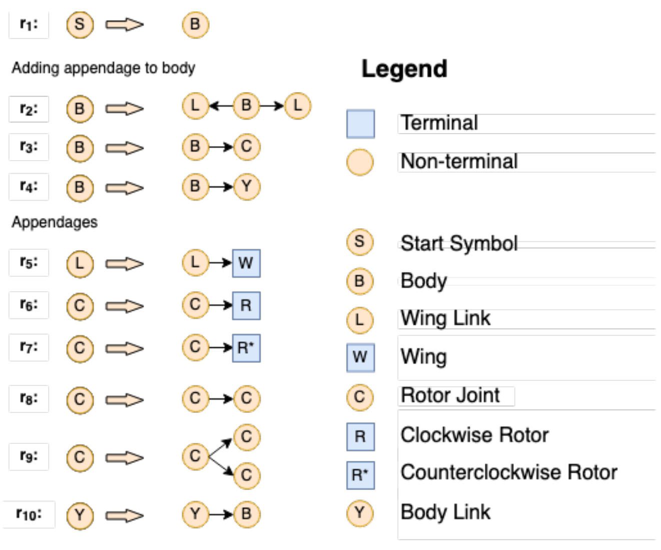

Fig. 3: Rules of our UAV grammar. S, B, L, C, Y are non-terminal symbols. Rule r_1 initializes the body structure, while $r_2 \sim r_4$ can be used to extend the body. Note that each body segment B can have at most one pair of wings attached to it. Rule r_5 connects the body with the wing. Rules r_6 and r_7 produce clockwise and counterclockwise rotors. Rule r_8 and r_9 extend rotor joints with one or two additional rotor joints. Rule r_{10} extends the body.

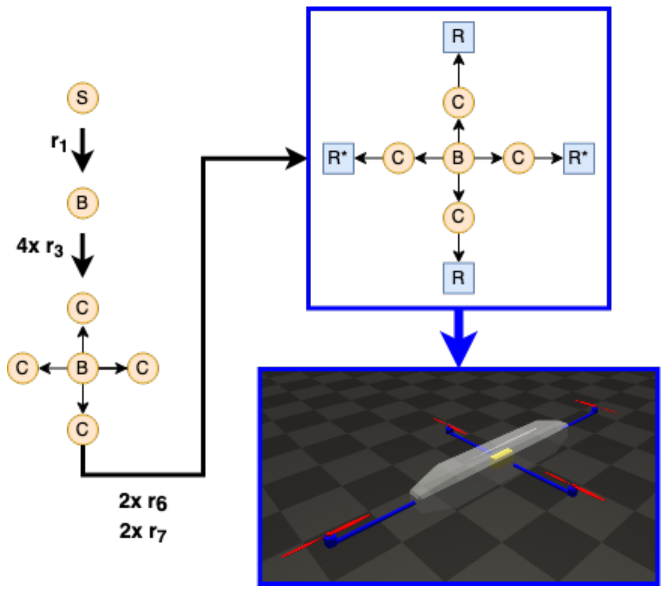

Fig. 4: The derivation sequence for a quadrotor using our graph grammar. This design uses 9 rules in total. We visualize the partial design graphs in between rules and the final quadrotor design after applying all rules. The non-terminal and terminal nodes in the graphs are indicated by circles and squares, respectively.

also known as trim states. In this section, we first describe the dynamic model of our UAV designs. Next, we explain our method for finding a trim state of a given UAV and associating it with an LQR controller.

A. Dynamic Model

We first define the world and body frames for describing a UAV's motion. We use the standard North-East-Down (NED) frame system in aerodynamics as the world frame. The body frame defined on a UAV is a 3D coordinate system rigidly attached at its center of mass with the three axes pointing forward (x), right (y), and down (z). We define a UAV's state by its linear and angular positions and velocities [21], which form a 13-dimensional vector $\mathbf{s}$:

$$\mathbf{s} = (\mathbf{x}, \mathbf{q}, \mathbf{v}_B, \boldsymbol{\omega}). \tag{1}$$

Here, $\mathbf{x}$ is the world position of the UAV's center of mass, $\mathbf{q}$ is the orientation expressed as a quaternion, $\mathbf{v}_B$ denotes the linear velocity of $\mathbf{x}$ in the body frame, and $\boldsymbol{\omega}$ is the angular velocity in the world frame. The time derivative of $\mathbf{s}$ is defined by Newton's second law and Euler's equations stated below:

$$m\ddot{\mathbf{x}} = m\mathbf{g} + \mathbf{f}, \tag{2}$$
$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} = \boldsymbol{\tau}, \tag{3}$$

where $\mathbf{g}$ is gravitational acceleration, m the mass, $\mathbf{I}$ the moment of inertia in the world frame, and $\mathbf{f}$ and $\boldsymbol{\tau}$ the net forces and torques exerted from all rotors, wings, and the fuselage in the world frame. Computing m is straightforward: we simply sum the component masses scaled by their change in volume. Computing the moment of inertia $\mathbf{I}$ is more involved as components may be scaled, translated, and rotated. To compute $\mathbf{I}$, we first find each component's moment of inertia in its body frame followed by applying the corresponding transformation. Next, we use the parallel axis theorem to update the moment of inertia after shifting the transformed component to its location in the body frame. Accurate moments of inertia are important, as the orientation of a UAV is usually the state component that is most sensitive to control inputs.

To compute $\mathbf{f}$ and $\boldsymbol{\tau}$, we take into account each rotor's orientation and relative position with respect to the center of mass to find the direction of its thrust and induced torque. The magnitude of the thrust is given by the controller to be described shortly. For the fuselage and the wings, we sum the lift and drag forces from each triangle in their deformed meshes as described in [22], [23]:

$$\boldsymbol{f}_{\text{drag}} = \frac{1}{2}\rho A C_d(\phi)\|\boldsymbol{v}_{\text{rel}}\|^2 \boldsymbol{d} \tag{4}$$
$$\boldsymbol{f}_{\text{lift}} = -\frac{1}{2}\rho A C_l(\phi)\|\boldsymbol{v}_{\text{rel}}\|^2 \boldsymbol{n} \tag{5}$$

where ρ is the air density, A is the triangle's area, C_d and C_l are polynomial functions of the angle of attack ϕ given in [24], $\boldsymbol{v}_{\text{rel}}$ is the relative air velocity, $\boldsymbol{d} = \frac{\boldsymbol{v}_{\text{rel}}}{\|\boldsymbol{v}_{\text{rel}}\|^2}$, and $\boldsymbol{n}$ is the triangle's normal. We add the resulting net force and torque to $\mathbf{f}$ and $\boldsymbol{\tau}$, respectively. Inertial and gyroscopic effects of rotor motion are omitted since the rotors comprise a small fraction of the overall mass.

Eqns. (2, 3) give us enough information to compute the time derivatives of $\mathbf{s}$, which we compactly represent as the dynamic model M, whose definitions are given in previous work [21]:

$$\dot{\mathbf{s}} = \mathrm{M}(\mathbf{s}, \mathbf{a}, \mathrm{G}, \theta). \tag{6}$$

Here, $\mathbf{a}$ is the action vector consisting of the magnitude of thrust sent to each rotor. As stated above, the shape parameters (G, θ) influence the dynamic model, e.g., by determining a rotor's direction or a wing's size. In short, given the current design (G, θ), the current state $\mathbf{s}$, and the current action $\mathbf{a}$, the dynamic model M computes $\dot{\mathbf{s}}$ that evolves the dynamic system.

B. LQR Control

After building the dynamic model for a given UAV sampled from our shape space in Sec. III, we associate it with an LQR controller parametrized by its trim state $(\bar{\mathbf{s}}, \bar{\mathbf{a}})$ and its $\mathbf{Q}$ and $\mathbf{R}$ matrices. A trim state is a state with zero linear and angular accelerations:

$$(\dot{\bar{\mathbf{x}}}, \dot{\bar{\mathbf{q}}}, \dot{\bar{\mathbf{v}}}_B, \dot{\bar{\omega}}) = \mathrm{M}(\bar{\mathbf{s}}, \bar{\mathbf{a}}, \mathrm{G}, \theta), \tag{7}$$

$$\dot{\bar{\mathbf{v}}}_B, \dot{\bar{\omega}} = \mathbf{0}, \mathbf{0}. \tag{8}$$

Once a trim state is found, we linearize the dynamic model M at the trim state and use the $\mathbf{Q}$ and $\mathbf{R}$ matrices to compute the LQR gain matrix $\mathbf{K}$ from the Continuous Algebraic Riccati Equation (CARE). The action $\mathbf{a}$ is then determined by the control policy $\mathbf{a} = -\mathbf{K}(\mathbf{s} - \bar{\mathbf{s}}) + \bar{\mathbf{a}}$.

Our controller design introduces a number of continuous parameters $(\bar{\mathbf{s}}, \bar{\mathbf{a}}, \mathbf{Q}, \mathbf{R})$. These control parameters and the shape parameters (G, θ) are the decision variables we consider in the problem of co-designing a UAV. Additionally, the controller design introduces to the co-design problem the trim-state constraints described in Eqns. (7, 8).

V. Simulation

To evaluate the performance of a given UAV shape and controller, we developed a differentiable UAV simulator based on the dynamic model described in Sec. IV-A. Given an initial state $\mathbf{s}_0$ of the UAV design, we compute the action $\mathbf{a}_0$ from its LQR controller and integrate Eqn. (6) using the fourth-order Runge Kutta method (RK4). The chosen time step of $\Delta t = 1/60$ s was sufficiently small to avoid instability while maintaining performance. With a loss function defined on the sequence of states at each time step, we backpropagate through each time step to compute its gradients with respect to all the continuous parameters: θ, $\bar{\mathbf{s}}$, $\bar{\mathbf{a}}$, $\mathbf{Q}$, and $\mathbf{R}$. This allows us to optimize them with classical gradient-based numerical optimization methods. Our simulator also provides the option to add random wind forces at each time step, which helps us evaluate the robustness of UAV designs and controllers.

VI. Optimization

Given a specific flight task, we describe the problem of finding an optimal UAV design as the following numerical optimization problem:

$$\min_{\mathrm{G}, \theta, \bar{\mathbf{s}}, \bar{\mathbf{a}}, \mathbf{Q}, \mathbf{R}} \quad L(\{\mathbf{s}_i\}, \{\mathbf{a}_i\}, \bar{\mathbf{s}}, \bar{\mathbf{a}}), \tag{9}$$

$$s.t. \quad \mathbf{s}_0 = \bar{\mathbf{s}}, \tag{10}$$

$$\mathbf{a}_i = \bar{\mathbf{a}} - \mathbf{K}(\mathbf{s}_i - \bar{\mathbf{s}}), \tag{11}$$

$$\mathbf{s}_{i+1} = \mathbf{s}_i + \Delta t \mathrm{M}(\mathbf{s}_i, \mathbf{a}_i, \mathrm{G}, \theta), \tag{12}$$

$$\mathbf{K} = \mathrm{CARE}(\frac{\partial \mathrm{M}}{\partial \mathbf{s}}\Big|_{\bar{\mathbf{s}}}, \frac{\partial \mathrm{M}}{\mathbf{a}}\Big|_{\bar{\mathbf{a}}}, \mathbf{Q}, \mathbf{R}), \tag{13}$$

$$\text{Trim conditions from Eqns. (7,8)}, \tag{14}$$

$$\text{Bound constraints on } \theta, \bar{\mathbf{s}}, \bar{\mathbf{a}}, \mathbf{Q}, \mathbf{R}. \tag{15}$$

We now explain the above definition in detail. The objective of the flight task is described as a loss function L defined on the sequences of states $\{\mathbf{s}_i\}$ and actions $\{\mathbf{a}_i\}$ where $i = 0, 1, \cdots$ indicates the indices of time steps. We use the trim state $\bar{\mathbf{s}}$ as the initial state $\mathbf{s}_0$ (Eqn. (10)). At the i-th time step, we call the simulator with the current state $\mathbf{s}_i$ and compute the action $\mathbf{a}_i$ from the LQR controller (Eqns. (11) and (13)). $\mathbf{a}_i$ is clipped to respect the same bounds as $\bar{\mathbf{a}}$. It is worth mentioning that the trim state $\bar{\mathbf{s}}$ in Eqn. (11) may be updated from time to time depending on the task definition. For example, in the task of waypoint navigation, $\bar{\mathbf{s}}$ is occasionally updated to the next waypoint on the flight trajectory. The new state $\mathbf{s}_{i+1}$ at the next time step is then computed from $\mathbf{s}_i$, $\mathbf{a}_i$, and the dynamic model M (Eqn. (12)). Note that we use forward Euler time integration to write Eqn. (12) for simplicity, and our implementation uses a more sophisticated RK4 integrator as mentioned in Sec. V. Finally, we include trim state constraints and bound constraints on the continuous parameters (Eqn. (15)).

The above optimization problem is challenging to solve because it mixes both discrete and continuous parameters. We propose a hierarchical algorithm to separate the discrete and continuous parameters in the optimization problem. At a high level, our algorithm repeatedly samples new graphs G from the graph grammar, evaluates the loss of each G, and returns the UAV design with the lowest loss. For each sampled graph G, we fix the graph parameter in the optimization problem and reduce it to a purely continuous optimization problem. We then run Sequential Quadratic Programming (SQP), a gradient-based numerical optimization method, to optimize all continuous parameters until SQP reaches a local minimum or exhausts the computation budget. This numerical optimization step associates a given G with the best loss found for that graph, allowing us to employ a discrete search strategy on top of that to improve the choice of G. We provide two discrete search strategies in our algorithm: a random and greedy search which randomly samples the graph G and only stores the G with the lowest loss, and a graph heuristic search (GHS) strategy [14], [15] which trains a graph neural network to predict the loss of each G to guide the search process. We summarize the whole optimization algorithm in Alg. 1.

Algorithm 1 Optimization algorithm

```
Inputs: a loss function $L$, a maximum number of graph proposals $N$.
Output: UAV shape and controller design parameters: G, $\theta$, $\bar{\mathbf{s}}$, $\bar{\mathbf{a}}$, $\mathbf{Q}$, $\mathbf{R}$.
$L_{\min} \leftarrow +\infty$
for $i \leftarrow 1$ to $N$ do
    ▷ Run discrete search to propose a graph
    Sample a graph $\mathrm{G}_i$ from the grammar
    ▷ Run continuous optimization
    $\theta_i, \bar{\mathbf{s}}_i, \bar{\mathbf{a}}_i, \mathbf{Q}_i, \mathbf{R}_i \leftarrow \mathrm{SQP}(\mathrm{G}_i)$
    ▷ Update the best loss for G
    $L_i \leftarrow L(\mathrm{G}_i, \theta_i, \bar{\mathbf{s}}_i, \bar{\mathbf{a}}_i, \mathbf{Q}_i, \mathbf{R}_i)$
    if $L_i < L_{\min}$ then
        $L_{\min} \leftarrow L_i$
        $\mathrm{G}, \theta, \bar{\mathbf{s}}, \bar{\mathbf{a}}, \mathbf{Q}, \mathbf{R} \leftarrow \mathrm{G}_i, \theta_i, \bar{\mathbf{s}}_i, \bar{\mathbf{a}}_i, \mathbf{Q}_i, \mathbf{R}_i$
    end if
end for
```

VII. Experiments

A. Implementation Details

The differentiable simulator is written in C++ with a Python wrapper, and the remainder of our pipeline is implemented in Python. Specifically, we use PyTorch Geometric [25] to implement the graph neural networks and the SLSQP solver from NLopt [26] to optimize the continuous design parameters. We run all of our experiments on Google Cloud N2 instances with 80 cores and 64G or 80G of memory. Our discrete search is parallelized across all CPU cores.

B. Task Specification

We consider four flight tasks requiring a UAV to perform dynamic motions: StraightLineTask, TurningTask, TakeoffTask, and TransitionTask. Each task is described by its loss function defined on a one-second-long flight, which we state in detail below:

a) StraightLineTask: This task requires following a horizontal and straight trajectory at a predefined speed while maximizing energy efficiency. 1 kg is added to the UAV's center of mass to represent a payload requirement, which we found necessary to encourage the use of wings. Specifically, we consider the following objective:

$$L = \underbrace{I_b(\{\mathbf{s}_i\}, \{\mathbf{a}_i\})}_{L_{\text{energy}}} + \gamma \underbrace{\max_i \|\mathbf{x}_i - \mathbf{x}_i^{\text{ref}}\|_2}_{L_{\text{pos}}}. \tag{16}$$

The loss function consists of two terms. The first term L_{energy} estimates the battery current I_b, where V_k is the effective, PWM voltage applied to motor k, n_k is the motor RPM, K_v is the motor velocity constant, R_w is the motor winding resistance, and $e = 0.95$ is ESC efficiency:

$$I_b = \sum_k \frac{V_k - n_k/K_v}{R_w e} \tag{17}$$

Minimizing L_{energy} encourages energy-efficient flight. The second term in the loss function penalizes the discrepancy between the desired and actual location of the UAV at each time step. In this task, the reference position $\mathbf{x}_i^{\text{ref}} = (i\Delta t \dot{\bar{\mathbf{x}}}(0), 0, 0)$, i.e., the UAV is expected to fly along the x-axis at a constant speed equal to the trim speed. Putting them together, L prefers a UAV design that can fly steadily along the x-axis while maximizing its energy efficiency. The coefficient γ represents our preferences over the two objectives, and we use $\gamma = 10$ in all tasks.

b) TurningTask: Our TurningTask has an objective similar to the StraightLineTask above but with two differences: the payload requirement is removed, and the reference position $\mathbf{x}_i^{\text{ref}}$ is now evenly distributed on an arc whose radius is 5 meters. This task prefers a UAV design that can closely track the arc without consuming much battery power.

c) TakeoffTask: This task focuses on optimizing the takeoff behavior of a UAV. Its objective is identical to the TurningTask except that the reference point $\mathbf{x}_i^{\text{ref}}$ is now evenly distributed on a two-meter-long vertical trajectory. Additionally, we force the initial velocity to be $\mathbf{0}$ so that the UAV must accelerate from a standstill and catch up to the reference points above.

d) TransitionTask: Our last task is a concatenation of takeoff and horizontal flight tasks. The objective function is defined the same as in the TurningTask except that the reference point $\mathbf{x}_i^{\text{ref}}$ is evenly distributed along a two-meter-long vertical trajectory followed by a two-meter-long horizontal trajectory. We design this task to encourage a UAV design that can combine the benefits from quadrotors and fixed-wing planes: quadrotors are suitable for vertical takeoff and precise tracking but consume more battery power, while fixed-wing planes may lower L_{energy} but lead to a much larger L_{pos} due to reduced maneuverability.

C. Our Results

We now present the UAV designs optimized by our computational pipeline. For each task, we run our pipeline described in Alg. 1 with both random search and GHS and visualize the optimized UAV shape designs they find in Fig. 5. We refer interested readers to our supplemental material for the flight videos of these UAV designs in simulation. Furthermore, we report their final losses for each task in Table I and the optimization progress of both random search and GHS in Fig. 6. For both random search and GHS, we run 4000 iterations in Alg. 1, i.e., $N = 4000$. For all four tasks, the overall search algorithm finishes within 15 to 20 hours.

Our first observation from Fig. 5 is that our pipeline discovers high-performing UAV designs with topologies that are quite different from any traditional designs. The novelty and variety of these UAV topologies highlight the benefits of incorporating our new graph grammar to express and explore the discrete shape design space. In particular, our graph grammar induces a hybrid shape space that flexibly mixes rotors and wings in a single design, as can be seen in the optimized design for our StraightLineTask. It is also interesting that the optimized designs often exhibit a topology

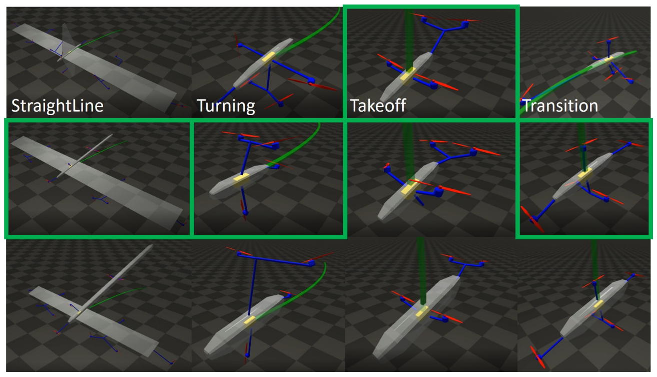

Fig. 5: Optimized UAV shapes discovered by our computational pipeline using random search (top row) and GHS (middle row) for each of the four tasks (left to right). For each task, the bottom row shows the better UAV from the two rows above (measured by their final loss in Table I and highlighted in green) with its deformation cages reset to their default positions.

that matches an engineer's instinct. For example, wings tend to be favored in the StraightLineTask due to the emphasis on horizontal flight. In the TakeoffTask and TransitionTask however, the high-performing designs use multiple rotors without wings, as wings hinder vertical takeoff.

A second observation is that although we run our algorithm for up to 4000 iterations, we notice that both random search and GHS search strategies reduce the loss drastically in the first few hundreds of iterations, after which the loss plateaus. This observation reveals specific properties of the design space induced by our graph grammar: it is not so difficult to find a UAV design with a near-optimal performance for these tasks when searching in this design space. However, a lot more effort is needed to push the performance limit to its extreme. Furthermore, users of our pipeline can use this observation as an empirical rule for choosing the number of iterations N when running Alg. 1.

D. Discussion

a) Classic designs: To show the advantage of searching over topologies using our computational UAV design pipeline, we compare the performances of the best designs discovered by our pipeline with a number of classic UAV designs (Fig. 7) that we handcraft using our grammar rules. For each classic design, only the continuous parameters are optimized while the topology remains fixed. We report their performance on each task in Table I. The performances of these classic designs can serve as a sanity check for the validity of our task definitions. For example, in our TakeoffTask, we notice that the two quadrotors have similar losses (2.48 and 2.30) while the octacopter has a much higher loss (3.47). This is consistent with the definition of the loss function: as copters are suitable for precise position tracking, the loss largely comes from its power consumption, and an octacopter should consume much more battery power than quadrotors because it has more rotors (leading to greater

TABLE I: Performance of classic UAVs and the optimized UAVs discovered by our method on each task. Each column reports the loss values on a given task with the best number highlighted in bold (lower is better). For the fixed-wing UAV in our TransitionTask, we report N/A because no trim states are found after trying 10 random seeds in the continuous optimization problem.

UAVs	StraightLine	Turning	Takeoff	Transition
X-quadrotor	4.26	1.98	2.48	3.82
Plus-quadrotor	4.23	8.87	**2.30**	3.63
Octacopter	4.16	2.90	3.47	4.86
Fixed-wing	3.98	2.28	7.18	N/A
Double fixed-wing	3.78	2.47	5.46	5.01
Ours (random)	1.68	1.63	2.43	3.20
Ours (GHS)	**0.95**	**1.55**	2.60	**2.80**

mass). Similarly, in our StraightLineTask, both fixed-wing planes perform better than the rotor-based designs because we expect fixed-wing planes to be more efficient at carrying heavy payloads.

From the table, we can conclude that both random search and GHS manage to reveal UAV designs that outperform the classic UAVs in three tasks, as indicated by their lower losses, and achieve comparable performance on TakeoffTask. Only upward thrust is required to solve the TakeoffTask, which may favor simpler designs like the quadrotors. The traditional, handcrafted designs appear to be suboptimal for the other three standard flight tasks, suggesting that our automatic, computational UAV design method is justified in exploring non-traditional UAV topologies. In fact, as can be seen in Fig. 5, most of the high-performing UAV designs discovered by our method have quite unusual topologies that are not covered by any of the classic designs in Fig. 7. These novel topologies would be difficult to find using previous UAV design methods without our graph grammar.

b) Discrete search: To compare the performances of the random strategy and our GHS strategy on discrete topology search, we test both strategies on all four tasks with three individual runs. Each individual run searches for 4000 UAV designs. We report the objective loss of the best designs discovered by each algorithm in Table I and plot the optimization progress curves in Fig. 6. From Table I, we can see GHS is able to find better designs than random search within the same design budget in three out of four tasks. This might be due to the additional learning phase in each GHS iteration that progressively trains a prediction network (i.e., heuristic function) from the searched designs [14]. In this way, the learned prediction network can guide the search towards the high-performing designs in our large, combinatorial search space. Fig. 6 shows the advantage of GHS more clearly. On all four tasks, the random strategy nearly converges to its best performance in a very early stage. In contrast, GHS keeps improving the best design as it learns a more accurate prediction function as the algorithm proceeds.

It is worth mentioning that random search also performs

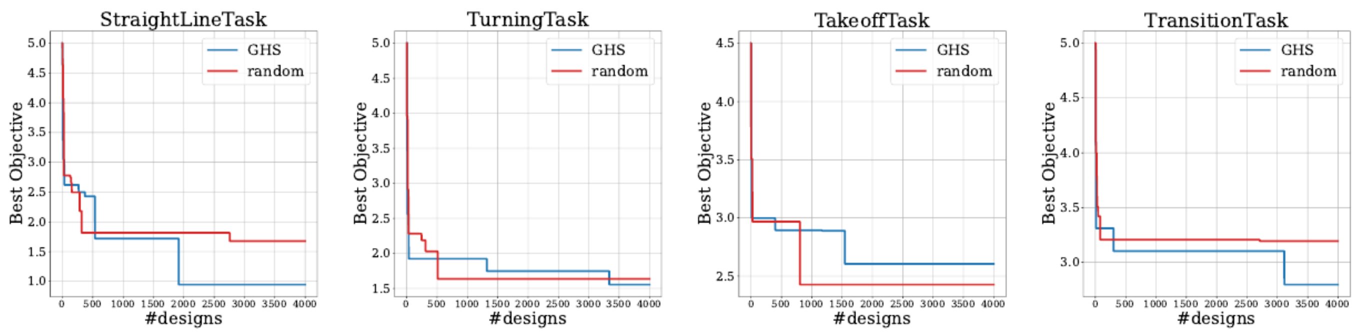

Fig. 6: The optimization progress plots of our two search strategies: random search and GHS. The horizontal axis is the number of designs sampled by the corresponding discrete search algorithm and optimized by SQP, and the vertical axis is the objective of the best design as the optimization proceeds. We generate the curves from three individual runs for each algorithm on each task.

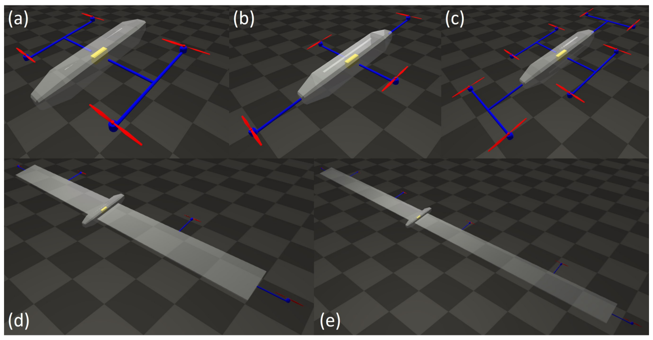

Fig. 7: The five classic designs we use as baselines in Table I. Top: (a) X-quadrotor, (b) Plus-quadrotor, (c) Octacopter. Bottom: (d) Fixed-wing plane. (e) Double fixed-wing plane.

considerably well in our tasks because most high-performing designs (as shown in Fig. 5) have relatively simple structures which can be easily sampled by a random strategy. We still keep GHS as an alternative option for discrete search since as demonstrated in previous work [14], [15], GHS shows a huge performance gain over random search when the optimal designs tend to have more complicated structures.

c) Deformation cages: To see the benefits of incorporating deformation cages in our design space, we restrict the deformation cages in the optimized UAVs found by our pipeline to their default positions during the search process (Fig. 5, bottom). We report the performances of the optimized UAVs with and without deformation cage optimization in Table II.

It is apparent that optimizing the deformation cages allows for greater performance compared to the case where cage parameter optimization is not allowed. The improvement is especially profound in the StraightLine task, which is understandable because the best design found leverages a pair of large wings whose shapes are entirely controlled by deformation cages. This observation confirms the benefit of incorporating deformation cages in the shape design space.

d) Controller robustness: Finally, to validate the robustness of our controller design, we simulate the best design found for each task with random wind gusts of a

TABLE II: Performances of the best UAVs discovered by our method with and without optimization of their deformation cages. For each task, we choose the better of the two designs found by GHS and random search. Each column reports the value of the loss function in a given task with the better case highlighted in bold (lower is better). Allowing optimization of cage parameters results in substantially better performance.

UAVs	StraightLine	Turning	Takeoff	Transition
Ours	**0.95**	**1.55**	**2.43**	**2.80**
Ours (no cages)	3.03	2.22	3.26	3.23

TABLE III: The position tracking error L_{pos} of the optimized UAVs discovered by our method when subjected to simulated wind conditions. The numbers are rescaled so that $L_{\text{pos}} = 1$ is the position tracking error in a clean simulation environment (wind speed = 0).

Wind speed (m/s)	StraightLine	Turning	Takeoff	Transition
0.5	2.40	1.04	1.00	1.15
1.0	2.93	1.07	1.01	1.28
2.0	3.86	1.21	1.02	1.65

fixed speed. The direction of the wind is sampled uniformly from the unit sphere, and changes every 0.25 seconds. We report the maximum position tracking error L_{pos} for each best-performing design in this noisy simulation environment (Table III). We normalize the position tracking error by dividing it by L_{pos} in the clean environment. In other words, $L_{\text{pos}} = 1$ indicates the UAV has comparable performance in the noisy and clean environments. The copter-like designs for the TurningTask, TakeoffTask, and TransitionTask are relatively insensitive to wind gusts due to their minimal surface area. Their rotor configuration and low mass allows them to counteract disturbances quickly. In comparison, the winged design optimized for the StraightLineTask (and its cargo carrying requirement) is sent off course much more easily. All controllers manage to recover however, and allow only a finite amount of deviation.

VIII. Conclusions and Future Work

Designing UAVs is challenging as the design space is large and difficult to parameterize. We present a computational UAV design pipeline that tackles this problem with a novel graph grammar and a mixed continuous-discrete optimization strategy. Our pipeline manages to discover unusual UAV designs that substantially outperform traditional UAV designs in three out of four evaluation tasks, while matching their performance in the fourth. We verify the robustness of the UAV shapes and LQR controllers co-designed by our pipeline in noisy simulation environments.

While the approach presented in this work produces high-performing aerial robot designs, there are several limitations that we leave as future research directions. First and foremost, our pipeline evaluates all UAV designs in simulation only and does not consider the details of fabrication. Although previous works show that some topologies discovered by our pipeline are indeed plausible to build [10], [11], we can consider how to compile resulting designs to fabrication plans that allow efficient manufacturing. A second limitation in our pipeline is the restriction of the design space to only rigid body structures, while many state-of-the-art aerial robots use articulated structures [27] or even soft or bio-inspired materials and components [28]. Expanding the shape design space to incorporate such features could help us find even higher-performing designs. A third limitation is the assumption that all aerodynamic components such as rotors and wings are independent of each other. Although neglecting interactions between these components can still result in successful sim-to-real transfer [11], a more detailed treatment could produce designs that are even more likely to perform well when fabricated. Finally, the flight tasks we consider in this work have a short time horizon. In future work, we plan to consider tasks with longer time horizons requiring multiple distinct motions or more dynamic maneuvers.

Acknowledgments

This work is supported by Defense Advanced Research Projects Agency (DARPA) grant No. FA8750-20-C-0075.

References

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *Ieee Access*, vol. 7, pp. 48 572–48 634, 2019.

[2] G. Pajares, "Overview and current status of remote sensing applications based on unmanned aerial vehicles (uavs)," *Photogrammetric Engineering & Remote Sensing*, vol. 81, no. 4, pp. 281–330, 2015.

[3] P. Doherty and P. Rudol, "A uav search and rescue scenario with human body detection and geolocalization," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2007, pp. 1–13.

[4] K. Kuru, D. Ansell, W. Khan, and H. Yetgin, "Analysis and optimization of unmanned aerial vehicle swarms in logistics: An intelligent delivery platform," *Ieee Access*, vol. 7, pp. 15 804–15 831, 2019.

[5] D. C. Tsouros, S. Bibi, and P. G. Sarigiannidis, "A review on uav-based applications for precision agriculture," *Information*, vol. 10, no. 11, p. 349, 2019.

[6] J. Kim, M.-S. Kang, and S. Park, "Accurate modeling and robust hovering control for a quad-rotor vtol aircraft," in *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*. Springer, 2009, pp. 9–26.

[7] B. Uragun, "Energy efficiency for unmanned aerial vehicles," in *2011 10th International Conference on Machine Learning and Applications and Workshops*, vol. 2. IEEE, 2011, pp. 316–320.

[8] U. Ozdemir, Y. O. Aktas, A. Vuruskan, Y. Dereli, A. F. Tarhan, K. Demirbag, A. Erdem, G. D. Kalaycioglu, I. Ozkol, and G. Inalhan, "Design of a commercial hybrid vtol uav system," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 1, pp. 371–393, 2014.

[9] A. Sobester and A. Keane, "Multidisciplinary design optimization of uav airframes," in *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference 14th AIAA/ASME/AHS Adaptive Structures Conference 7th*, 2006, p. 1612.

[10] T. Du, A. Schulz, B. Zhu, B. Bickel, and W. Matusik, "Computational multicopter design," 2016.

[11] J. Xu, T. Du, M. Foshey, B. Li, B. Zhu, A. Schulz, and W. Matusik, "Learning to fly: computational controller design for hybrid uavs with reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[12] N. Umetani, Y. Koyama, R. Schmidt, and T. Igarashi, "Pteromys: Interactive design and optimization of free-formed free-flight model airplanes," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–10, 2014.

[13] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, pp. 15–22.

[14] A. Zhao, J. Xu, M. Konaković-Luković, J. Hughes, A. Spielberg, D. Rus, and W. Matusik, "Robogrammar: graph grammar for terrain-optimized robot design," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–16, 2020.

[15] J. Xu, A. Speilberg, A. Zhao, D. Rus, and W. Matusik, "Multi-objective graph heuristic search for terrestrial robot design," in *2021 International conference on robotics and automation (ICRA)*. IEEE, 2021.

[16] F. R. Stöckli and K. Shea, "A simulation-driven graph grammar method for the automated synthesis of passive dynamic brachiating robots," in *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers Digital Collection, 2015.

[17] D. Pathak, C. Lu, T. Darrell, P. Isola, and A. A. Efros, "Learning to control self-assembling morphologies: a study of generalization via modularity," *arXiv preprint arXiv:1902.05546*, 2019.

[18] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal, "An End-to-End Differentiable Framework for Contact-Aware Robot Design," in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[19] A. Jacobson, I. Baran, J. Popović, and O. Sorkine, "Bounded biharmonic weights for real-time deformation," *ACM Trans. Graph.*, vol. 30, no. 4, July 2011. [Online]. Available: https://doi.org/10.1145/2010324.1964973

[20] T. Ju, S. Schaefer, and J. Warren, "Mean value coordinates for closed triangular meshes," *ACM Trans. Graph.*, vol. 24, no. 3, p. 561–566, July 2005. [Online]. Available: https://doi.org/10.1145/1073204.1073229

[21] A. J. Barry, P. R. Florence, and R. Tedrake, "High-speed autonomous obstacle avoidance with pushbroom stereo," *Journal of Field Robotics*, vol. 35, no. 1, pp. 52–68, 2018.

[22] S. Min, J. Won, S. Lee, J. Park, and J. Lee, "Softcon: Simulation and control of soft-bodied animals with biomimetic actuators," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–12, 2019.

[23] P. Ma, T. Du, J. Z. Zhang, K. Wu, A. Spielberg, R. K. Katzschmann, and W. Matusik, "Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, p. 132, 2021.

[24] A. Zhao, J. Xu, J. Salazar, W. Wang, P. Ma, D. Rus, and W. Matusik, "Graph grammar-based automatic design for heterogeneous fleets of underwater robots," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 3143–3149.

[25] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[26] S. G. Johnson, "The nlopt nonlinear-optimization package," 2014.

[27] J. C. Gomez and E. Garcia, "Morphing unmanned aerial vehicles," *Smart Materials and Structures*, vol. 20, no. 10, p. 103001, 2011.

[28] E. Ajanic, M. Feroskhan, S. Mintchev, F. Noca, and D. Floreano, "Bioinspired wing a and tail morphing extends drone flight capabilities," *Sci. Robot.*, vol. 5, p. eabc2897, 2020.