Soft Multicopter Control Using Neural Dynamics Identification Supplementary

Yitong Deng[1*], **Yaorui Zhang**[1], **Xingzhe He**[1], **Shuqi Yang**[1], **Yunjin Tong**[1]
Michael Zhang[2], **Daniel DiPietro**[1], **Bo Zhu**[1]
[1]Department of Computer Science, Dartmouth College
[2]Lawrenceville School
[*] yitong.deng.gr@dartmouth.edu

1 Overview

In this document, we present the supplementary materials to our published paper. In Section 2, we describe the specifications of the 2D and 3D drone models used in our training and testing, including the soft material properties, the rotor designs, and the sensor placements. In Section 3 we introduce the details of how the state vector $\mathbf{s}$ and $\mathbf{e}$ are obtained from sensor readings. In Section 4 we propose a general guideline for deploying IMU sensors for arbitrary drone shapes. In Section 5 we describe the simulation environment, the simulation model used, and the noise treatment. In Section 6, we specify the details of the learning module, including the network structure used, the data generation scheme, as well as the techniques and hyperparameters used along the training procedure. In Section 7, we specify the parameters used in our control module and the mathematical derivation of the benchmark LQR controller. In Section 8, we present a discussion about the system design choices, the assumptions we have made, and the potential challenges for the fabrication and control of real-world soft multicopters.

2 Drone Designs

Sensor Layouts For the 3D examples, the sensing scheme is depicted in Figure. 1. Each IMU measures the local X, Y, and Z axes, which are coded by Red, Green, and Blue respectively. The Y axis will point out of the plane. For the peripheral measurements, we will only make use of the measured Y axis. Since rotation in 2D can be represented by one scalar only, for 2D drones the IMU will only output the angle between the measured vector and the horizontal. The measured vectors are depicted in Figure. 2. Please also refer to Figure. 1 and Figure. 2 for the nicknames of these drone models. For 2D drones, we only train controllers of the *rod* model since it is the most deformable 2D geometry among all and therefore the one that displays the most interesting behaviors.

Drone Specifications The specifications of our tested models' size and material properties are presented in Table. 1.

Drone Design Procedure To customize 2D drones, we develop a web-based painting tool , as shown in Figure. 3, to sketch the contours, and use TetGen[1] to create triangle meshes from the contours. The interface also allows users to set rotor positions and assign materials to triangle elements of the mesh interactively. 3D drones are modeled in Maya and then converted to tetrahedron meshes using TetGen.

Dual-Propeller Rotor A rotor mounted on a soft drone will influence the drones' body with

1. the thrust from accelerating the air and creating a low-pressure region in front of it, a force which will act in the normal direction of the surface on which the rotor is mounted;
2. the torque that acts on the drone's body in the opposite direction of the rotor's rotation to conserve angular momentum;

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

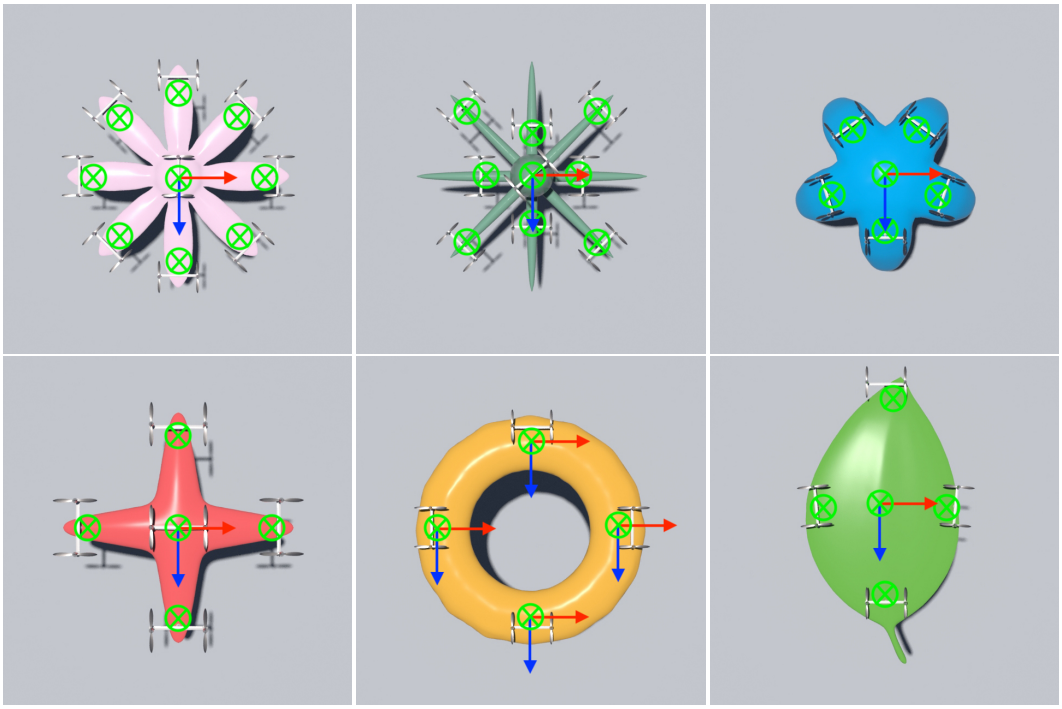

Figure 1: Sensor placement for 3D drone designs. Top row: Flower, Octopus, Orange Peel; Bottom row: Star, Donut, Leaf.

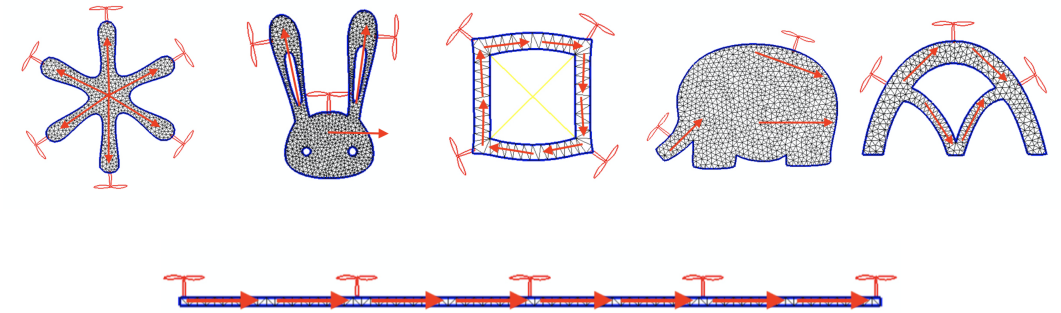

Figure 2: Sensor placement for 2D drone designs (illustrated by red arrows). Top row: Engine, Bunny, Diamond, Elephant, Rainbow; Bottom row: Rod.

3. the gyroscopic torque that will act in the direction perpendicular to the gravity and the rotor's spinning direction, which happens when the rotor is tilted.

In this work, each of the m actuators will be implemented by a dual-rotor with counter-rotation, and the actuation will be split in half for each of the two rotors. With the two sub-rotors spinning in countering directions, the second term will be canceled out. The two sub-rotors will cancel the gyroscopic moments of their counterparts as well. Under this setting, in our simulation, only the normal force is modeled.

3 Computation of the State Vectors

Computation of e In the common case where the drone's body contains no hole in the middle, an IMU will be placed at the geometric center, and the measured rotation of the IMU's rigid frame will be used as the definition of the drone's body frame. For cases like the *donut*, where there is a hole in the middle, the strategy is to insert a few IMU at the circumferential locations, and average these

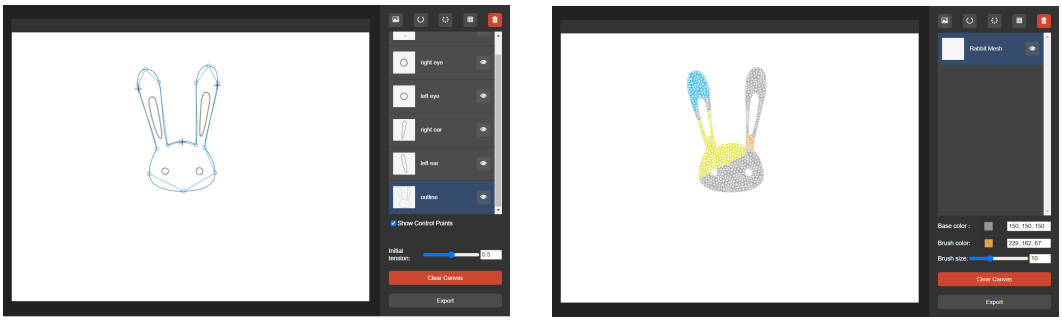

Figure 3: Left: sketching the contour; Right: painting the mesh.

3D models						
specs	Donut	Starfish	Flower	Leaf	Octopus	Orange peel
mass(kg)	1	1	1	1	1	1
modulus(N/m^2)	1e4	3e3	6e3	3e3	1e4	5e2
length-x(m)	3.5	3.6	3.6	2.4	3.6	3
length-y(m)	0.36	0.375	0.225	0.075	1.5	1.3
length-z(m)	3.5	3.6	3.6	4.5	3.6	2.9
num sensors	4	4	8	4	8	5
num rotors	4	5	9	4	9	5
max thrust(N)	10	10	10	10	10	10
2D models						
specs	Engine	Bunny	Diamond	Elephant	Rainbow	Long Rod
mass(kg)	1	1	1	1	1	1
modulus(N/m^2)	6e3	6e3	6e3	6e3	6e3	6e3
length-x(m)	1.90	1.12	1.47	2.46	2.08	0.1
length-y(m)	2.16	1.75	1.42	1.69	1.30	8.0
num sensors(m)	6	3	8	3	4	8
num rotors(m)	6	3	4	2	2	5
max thrust(N)	10	10	10	10	10	10

Table 1: Design Specifications

obtained rotations. In our case where the 4 inserted IMUs are center-symmetric, the average rotation is obtained by averaging the body-frame X-direction of IMU 1, 3, the body-frame Y-direction of IMU 2, 4, and use cross products to obtain the combined body frame. For the general case, this operation can be done by converting these measurements into quaternions and apply the averaging methods described in [2] to obtain the body frame.

Computation of s The deformation vector $\mathbf{s}$ will constitute measurements from IMUs inserted at peripheral points. For measuring these local deformations, we will measure the normal vector of the local body surface, which is the direction of the Y-axis of IMU's body frame. Given the IMU's measured rotation matrix (body-to-world) $\mathbf{R}_{peripheral}$, we will first calculate its Y-axis in the world frame by

$$\mathbf{y}_w = \mathbf{R}_{peripheral} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}. \tag{1}$$

Then given the body-to-world rotation matrix $\mathbf{R}_{central}$ defined by $\mathbf{e}$, we will map the $\mathbf{y}_w$ on to the drone's body frame:

$$\mathbf{y}_b = \mathbf{R}_{central}^T \mathbf{y}_w \tag{2}$$

Then, an axis-angle will be calculated for how to rotate the Y-axis in the drone's body frame to $\mathbf{y}_b$. The axis will be calculated by:

$$\mathbf{v} = \mathbf{y}_b \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3}$$

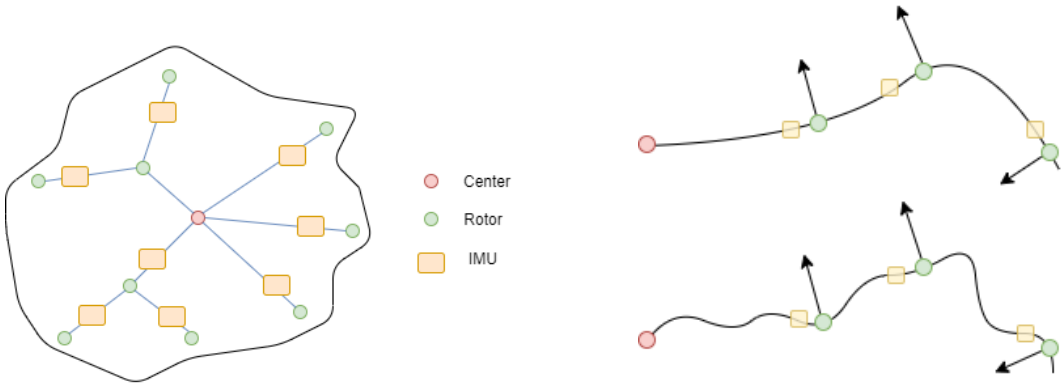

Figure 4: IMU placement. The left figure describes the proposed scheme for inserting IMUs for drones with arbitrary, irregular geometries. The right illustrates the different level of adequacy of this scheme at two different levels of softness.

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{||\mathbf{v}||} \tag{4}$$

The angle will be calculated by:

$$\alpha = \beta \cdot arccos(\frac{\mathbf{y}_b}{||\mathbf{y}_b||} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}) \tag{5}$$

where

$$\beta = \begin{cases} 1 & \text{if } \hat{v} \times \mathbf{r} \text{ has positive x-entry} \\ -1 & \text{if } \hat{v} \times \mathbf{r} \text{ has negative x-entry} \end{cases}, \tag{6}$$

with $\mathbf{r}$ representing the body frame location of the inserted IMU when undeformed.

In this way, the deformation is converted into a scalar, and by the construction of β, the magnitude of the scalar will represent the magnitude of the deformation, while the sign represents whether the deformation is inward (positive) or outward (negative).

4 Guidelines for Sensor Placement

We present a general guideline for selecting where IMUs are deployed in the left part of Figure. 4. Given an arbitrary drone shape in 3D (pressed onto the X-Z plane), we build a tree with the root node being the geometric center of the drone, and the child nodes being the rotors. The IMUs will be inserted at the edges of the tree near the outer rotor. The effectiveness of this approach is contingent on the simple modality of the soft drone's deformation. For instance, if you take a look at the right part of Figure. 4, for the above case, the deformed shape of the drone's arm can be approximately reconstructed from the three measurements, whereas in the lower case, the three measurements are far from enough to describe the deformed shape, as the deformation is highly multi-modal, while these higher-order deformation modes are effectively beyond the controlling capacity of the drone's rotors. As a result, it is the task in the design of these drones (mostly selecting the modulus and thickness) so that the drone is soft enough to perform significant deformation, while the deformation mode of the drone is simple. In practice, this IMU insertion guideline works for our various examples.

5 Simulation Description

Soft Body Model In our simulation environment, the deformation of a soft body is simulated using an explicit co-rotated elastic finite element model [3]. A mass-proportional damping term is used to model the damped elastic behavior. We use tetrahedron (3D) and triangle (2D) meshes for discretization. An OpenMP-based parallel implementation of the elastic solver was employed to

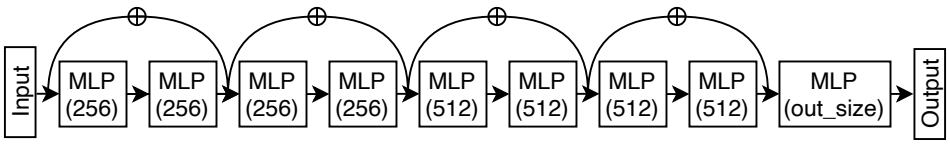

Figure 5: Network architecture: This architecture is similar to ResNet. Here each MLP consists of a fully connected layer and a ReLU except the last layer. The number in the parentheses means output dimension of each layer.

boost the simulation performance. Each rotor is rigidly bound to a local set of surface vertices on the finite element mesh in the course of the simulation, with the rotor direction aligned to the averaged normal direction of the local surface triangle primitives in 3D (surface segments in 2D).

In our simulation environment, an IMU is implemented by binding a number of nearby vertices and use their positions to define a reference frame via cross products.

Noise Treatment In the simulation environment, in order to emulate the perturbations and uncertainties in the real world, noise is added to the sensor readings, and a time delay is added to the rotor output. The details of these noises are given in the table below.

Category	Noise type	Level
angle measurements	Gaussian	$\mu = 0, \sigma = 0.573^\circ$
position measurements	Gaussian	$\mu = 0, \sigma = 0.01m$
rotor perturbation	Gaussian	$\mu = 0, \sigma = 0.1N$
output delay	constant	$0.03s$

6 Learning of Neural Networks

Dataset Generation The training data are generated with our implementation of a Finite Element simulator. Given a drone geometry, we initialize the drone as undeformed, lying at the origin, and apply a random thrust to each rotor and observe the drone's position, rotation, and deformation at $100Hz$. Each set of random thrust is applied for $1s$. Other data generation schemes we tried also consist of using a rigid LQR controller to generate the thrusts, or apply a different random thrust each frame, but the former yields poor test loss due to the confined distribution of LQR control outputs, while the latter generates data too noisy to train on. The insight is that we need to give the system enough time to respond to a signal and display meaningful behaviors.

Network Architecture and Training As shown in Figure. 5, All the three neural networks to learn $\{\mathbf{d}, \mathbf{g}, \mathbf{h}\}$ adopts the same architecture. The architecture is similar to ResNet except that the convolution layers are replaced by fully connected layers. Note that there are no normalization techniques used in our networks. We use Adam optimizer with initial learning rate 0.001 and decay rate 0.8 for each 20 steps. The batch size is 512. We train for 50 epoches. For loss function we found out L1 loss provides superior result to L2 loss due to the robustness of the L1 loss.

Testing of the Networks Section 4 of the paper presents the testing results of our trained neural dynamic systems. More testings are depicted in Figure. 6 and Figure. 7 with the same experimental setups.

7 Control

LQR Overview The Linear Quadratic Controller is a kind of full-state feedback controller, where the control of the system is based on the current state. Given a linear system in state-space form:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{7}$$

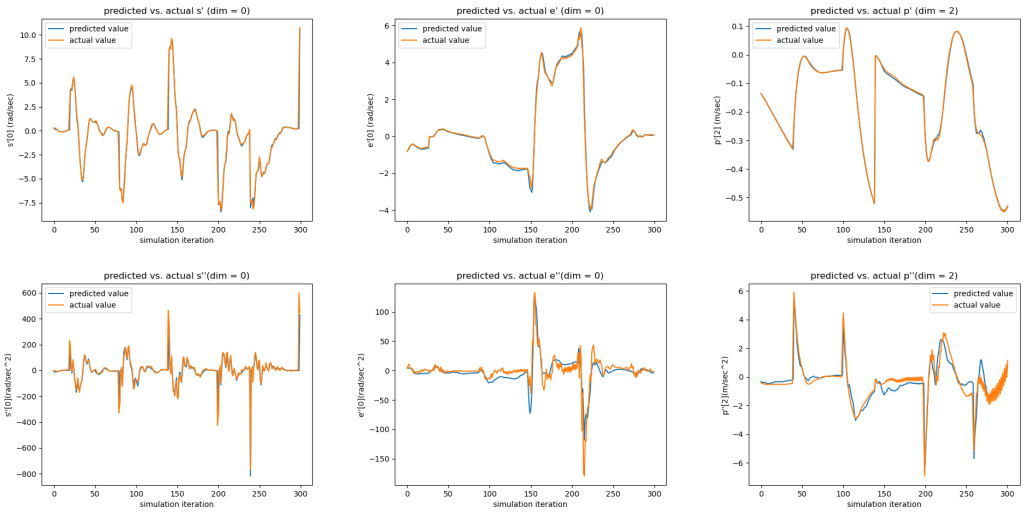

Figure 6: Testing results of the networks

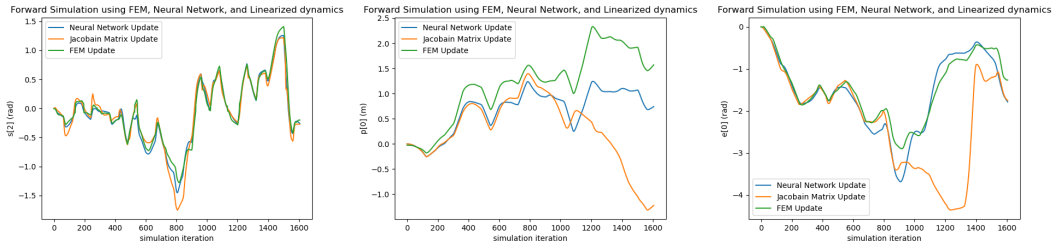

Figure 7: Testing results of the Network Linearization

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the control vector (in our case the thrusts for individual propellers), and $\mathbf{A}$ and $\mathbf{B}$ are matrices, we compute a control matrix $\mathbf{K}$ and combine that with the state by:

$$\mathbf{u} = -\mathbf{K}\mathbf{x} \tag{8}$$

The way we obtain $\mathbf{K}$ is as follows: suppose we want to set both the state and the control to be $\mathbf{0}$. We define cost matrices $\mathbf{Q}$ and $\mathbf{R}$ that penalizes $\mathbf{x}$ squared and $\mathbf{u}$ squared respectively, we desire to minimize the infinite horizon cost:

$$\int_{\mathbf{0}}^{\infty} [\mathbf{x}^{\mathbf{T}}\mathbf{Q}\mathbf{x} + \mathbf{u}^{\mathbf{T}}\mathbf{R}\mathbf{u}]\mathbf{dt} \tag{9}$$

which means our goal is to find the optimal cost-to-go function $\mathbf{J}^* = \mathbf{x}^{\mathbf{T}}\mathbf{S}\mathbf{x}$ that satisfies the Hamilton–Jacobi–Bellman (HJB) equation. Utilizing the convex nature of the problem, we know that the minimum occurs when the gradient is zero, so we have:

$$\frac{\partial}{\partial \mathbf{u}} = \mathbf{2}\mathbf{u}^{\mathbf{T}}\mathbf{R} + \mathbf{2}\mathbf{x}^{\mathbf{T}}\mathbf{S}\mathbf{B} = \mathbf{0} \tag{10}$$

which yields the control policy:

$$\mathbf{u} = -[\mathbf{R}^{-\mathbf{1}}\mathbf{B}^{\mathbf{T}}\mathbf{S}]\mathbf{x} = -\mathbf{K}\mathbf{x} \tag{11}$$

After transformation, we can find the value of $\mathbf{S}$ by solving the equation:

$$\mathbf{0} = \mathbf{S}\mathbf{A} + \mathbf{A}^{\mathbf{T}}\mathbf{S} - \mathbf{S}\mathbf{B}\mathbf{R}^{-}\mathbf{1}\mathbf{B}^{\mathbf{T}}\mathbf{S} + \mathbf{Q} \tag{12}$$

This is known as the **Algebraic Riccati Equation**, which can be solved by iterating backward in time.

Online Reinitialization Parameters Although our network eliminates the necessity for the extensive, empirical parameter tuning process, there are a few hyper-parameters that needs to be tuned for effective performance. We will present the exact value or the value range for these parameters in the table below.

Parameter type	Value/Value range
$\mathbf{Q}$ gain (related to $\mathbf{s}$)	100 to 200
$\mathbf{Q}$ gain (related to $\mathbf{e}$)	50 to 200
$\mathbf{Q}$ gain (related to $\mathbf{p}$)	100 to 200
$\mathbf{R}$ gain	2
kp	0.03
kd	0.0001
n	10

Rigid LQR State Definition The state of a rigid object can be described by its position and rotation. Let $\mathbf{p}$ be the vector describing position, and let $\mathbf{e}$ be the vector describing rotation. And let $\mathbf{q} = \begin{pmatrix} \mathbf{p} \\ \mathbf{e} \end{pmatrix}$. Since the dynamics is second order, the state $\mathbf{x}$ will be defined as $\mathbf{x} = \begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{pmatrix}$. For the 3D case, $\mathbf{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, $\mathbf{e} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix}$, where x, y, z are the spacial coordinates, ϕ, θ, ψ are the Euler angles. For the 2D case, $\mathbf{p} = \begin{pmatrix} x \\ y \end{pmatrix}$, $\mathbf{e} = (\phi)$, as the rotation in 2D can be described by a sole parameter.

$\mathbf{R}$	SO(3)	Body-to-world rotation matrix
$\mathbf{r}$	R^3	Motor position in body frame
$\mathbf{d}$	unit sphere	Motor orientation in body frame
$\mathbf{M_f}$	$R^{3\times n}$	Mapping from thrusts to net force. The i-th column is $\mathbf{di}$.
$\mathbf{M_t}$	$R^{3\times n}$	Mapping from thrusts to net torque. The i-th column is $bi\lambda i\mathbf{di} + ri \times \mathbf{di}$
$\mathbf{J}$	R^3	Inertia Tensor in Body Frame. Value is $\begin{bmatrix} Ixx & Ixy & Ixz \\ Iyx & Iyy & Iyz \\ Izx & Izy & Izz \end{bmatrix}$
Ixx	R	$\sum_i mi * (yi^2 + zi^2)$
Iyy	R	$\sum_i mi * (zi^2 + xi^2)$
Izz	R	$\sum_i mi * (xi^2 + yi^2)$
Ixy	R	$-\sum_i mi * xi * yi$
Ixz	R	$-\sum_i mi * xi * zi$
Iyz	R	$-\sum_i mi * yi * zi$
$\mathbf{L}$	R^3	Mapping from world frame angular velocity to body frame angular velocity, such that $\omega = \mathbf{L}\dot{\mathbf{e}}$. Value is $\begin{bmatrix} 1 & 0 & -s(\theta) \\ 1 & c(\theta) & s(\phi)c(\theta) \\ 1 & -s(\phi) & c(\phi)c(\theta) \end{bmatrix}$
$\dot{\mathbf{L}}$	R^3	Derivative of $\mathbf{L}$. Value is $\begin{bmatrix} 0 & 0 & -c(\theta)\dot{\theta} \\ 1 & -s(\phi)\dot{\phi} & c(\phi)c(\theta)\dot{\phi} - s(\phi)s(\theta)\dot{\theta} \\ 1 & -c(\phi)\dot{\phi} & s(\phi)c(\theta)\dot{\phi} - c(\phi)s(\theta)\dot{\theta} \end{bmatrix}$

Dynamic Model Let $\mathbf{u}$ denote the drone's actuation, and $\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix}$, where m is the number of propellers and u_i represents the thrust provided by each propeller. The dynamic model is a function $\mathbf{f}$ such that $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. In 3D, the dynamics of the drone will be directly derived from the Newton-Euler equations:

$$m\ddot{\mathbf{p}} = m\mathbf{g} + \mathbf{R}\mathbf{M_f}\mathbf{u} \tag{13}$$

$$\mathbf{J}(\dot{\mathbf{L}}\dot{\mathbf{e}} + \mathbf{L}\ddot{\mathbf{e}}) + (\mathbf{L}\dot{\mathbf{e}}) \times \mathbf{J}\mathbf{L}\dot{\mathbf{e}} = \mathbf{M_t}\mathbf{u} \tag{14}$$

with the variable definitions given in the table below. For the 2D case, these equations simplify to

$$m\ddot{\mathbf{p}} = m\mathbf{g} + \mathbf{R}\mathbf{M_f}\mathbf{u} \tag{15}$$

$$\mathbf{J}\ddot{\mathbf{e}} = \mathbf{M_t}\mathbf{u} \tag{16}$$

Manipulator Form Follow the formulation purposed in [4], we will reorganize these equations into the Manipulator Form, whose template is as follows:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{B}(\mathbf{q})\mathbf{u}, \tag{17}$$

Consequently,

$$\ddot{\mathbf{q}} = \mathbf{H^{-1}}(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})) \tag{18}$$

This allows us to write:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{H^{-1}}(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})) \end{bmatrix} \tag{19}$$

For the 3D case, reorganizing the dynamics equations yields $\mathbf{H} = \begin{bmatrix} m\mathbf{I_3} & \mathbf{O} \\ \mathbf{O} & \mathbf{JL} \end{bmatrix}$, $\mathbf{C} = \begin{bmatrix} \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{J}\dot{\mathbf{L}} + \mathbf{L}\dot{\mathbf{e}} \times \mathbf{JL} \end{bmatrix}$, $\mathbf{G} = \begin{bmatrix} -m\mathbf{g} \\ \mathbf{O} \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} \mathbf{R}\mathbf{M_f} \\ \mathbf{M_t} \end{bmatrix}$.
For the 2D case, we have $\mathbf{H} = \begin{bmatrix} m\mathbf{I_2} & \mathbf{O} \\ \mathbf{O} & \mathbf{J} \end{bmatrix}$, $\mathbf{C} = [\mathbf{O}]$, $\mathbf{G} = \begin{bmatrix} -m\mathbf{g} \\ \mathbf{O} \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} \mathbf{R}\mathbf{M_f} \\ \mathbf{M_t} \end{bmatrix}$.

Linearization via Taylor Expansion Since the function $\mathbf{f}(\mathbf{x}, \mathbf{u})$ described above is a non-linear model, we will linearize it by taking the first order Taylor Expansion around an operating point $(\mathbf{x}^*, \mathbf{u}^*)$ such that $\mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) = \mathbf{0}$. For $\mathbf{x}$ close enough to $\mathbf{x}^*$, we have:

$$\begin{aligned} \mathbf{f}(\mathbf{x} - \mathbf{x}^*) &\approx (\frac{\partial f}{\partial x}|_{\mathbf{x}=\mathbf{x}^*, \mathbf{u}=\mathbf{u}^*})(\mathbf{x} - \mathbf{x}^*) + (\frac{\partial \mathbf{f}}{\partial \mathbf{u}}|_{\mathbf{x}=\mathbf{x}^*, \mathbf{u}=\mathbf{u}^*})(\mathbf{u} - \mathbf{u}^*) \\ &= \mathbf{A_{lin}}(\mathbf{x} - \mathbf{x}^*) + \mathbf{B_{lin}}(\mathbf{u} - \mathbf{u}^*) \end{aligned} \tag{20}$$

Since we know that:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{H^{-1}}(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})) \end{bmatrix}, \mathbf{x} = \begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{pmatrix}, \tag{21}$$

$\mathbf{A_{lin}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ can be represented by the block matrix:

$$\begin{bmatrix} \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{q}} & \frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{q}}} \\ \frac{\partial \mathbf{H^{-1}}(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}))}{\partial \mathbf{q}} & \frac{\partial \mathbf{H^{-1}}(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}))}{\partial \dot{\mathbf{q}}} \end{bmatrix} = \begin{bmatrix} \mathbf{T1} & \mathbf{T2} \\ \mathbf{T3} & \mathbf{T4} \end{bmatrix} \tag{22}$$

It can be seen trivially that $\mathbf{T1} = \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{q}} = \mathbf{O}$ and $\mathbf{T2} = \frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{q}}} = \mathbf{I_3}$.

For $\mathbf{T_3} = \frac{\partial \mathbf{H^{-1}}(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}))}{\partial \mathbf{q}}$, by the Product Rule we know,

$$\mathbf{T_3} = \frac{\partial \mathbf{H^{-1}}}{\partial \mathbf{q}}(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})) + \mathbf{H^{-1}}\frac{\partial(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}))}{\partial \mathbf{q}} \tag{23}$$

Since we defined $\mathbf{x}^*, \mathbf{u}^*$ to be such that $\mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) = \mathbf{0}$, then $\mathbf{H^{-1}}(\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})) = \mathbf{0}$ at $(\mathbf{x}^*, \mathbf{u}^*)$. Since we know $\mathbf{H^{-1}}$ is non-zero, then $\mathbf{B}(\mathbf{q})\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) = \mathbf{0}$. Besides, since we have $\mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) = \mathbf{0}$, we have $\dot{\mathbf{q}} = \mathbf{0}$, then $\frac{\partial \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}}{\partial \mathbf{q}} = \frac{\partial \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \mathbf{q}}\dot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{q}} = \mathbf{0} + \mathbf{0} = \mathbf{0}$. Also, since $\mathbf{G}(\mathbf{q}) = \begin{bmatrix} -m\mathbf{g} \\ \mathbf{O} \end{bmatrix}$, and has nothing to do with $\mathbf{q}$, $\frac{\partial \mathbf{G}(\mathbf{q})}{\partial \mathbf{q}} = \mathbf{0}$. So we can conclude that:

$$\mathbf{T_3} = \mathbf{H^{-1}}\frac{\partial \mathbf{B}(\mathbf{q})\mathbf{u}}{\partial \mathbf{q}} = \mathbf{H^{-1}}(\frac{\partial(\mathbf{B}(\mathbf{q})}{\partial \mathbf{q}}\mathbf{u} + \mathbf{B}(\mathbf{q})\frac{\partial \mathbf{u}}{\partial \mathbf{q}}) = \mathbf{H^{-1}}\frac{\partial(\mathbf{B}(\mathbf{q})}{\partial \mathbf{q}}\mathbf{u}. \tag{24}$$

Since

$$\mathbf{B} = \begin{bmatrix} \mathbf{RM_f} \\ \mathbf{M_t} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} (\mathbf{RM_f})_1 \\ (\mathbf{M_t})_1 \end{pmatrix} & \begin{pmatrix} (\mathbf{RM_f})_2 \\ (\mathbf{M_t})_2 \end{pmatrix} & \begin{pmatrix} (\mathbf{RM_f})_3 \\ (\mathbf{M_t})_3 \end{pmatrix} & \begin{pmatrix} (\mathbf{RM_f})_4 \\ (\mathbf{M_t})_4 \end{pmatrix} \end{bmatrix} \tag{25}$$

$$\frac{\partial \mathbf{B}}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial \begin{pmatrix} (\mathbf{RM_f})_1 \\ (\mathbf{M_t})_1 \end{pmatrix}}{\partial \mathbf{q}} & \frac{\partial \begin{pmatrix} (\mathbf{RM_f})_2 \\ (\mathbf{M_t})_2 \end{pmatrix}}{\partial \mathbf{q}} & \frac{\partial \begin{pmatrix} (\mathbf{RM_f})_3 \\ (\mathbf{M_t})_3 \end{pmatrix}}{\partial \mathbf{q}} & \frac{\partial \begin{pmatrix} (\mathbf{RM_f})_4 \\ (\mathbf{M_t})_4 \end{pmatrix}}{\partial \mathbf{q}} \end{bmatrix}. \tag{26}$$

So,

$$\frac{\partial \mathbf{B}}{\partial \mathbf{q}} \mathbf{u} = \sum_i \frac{\partial \begin{pmatrix} (\mathbf{RM_f})_i \\ (\mathbf{M_t})_i \end{pmatrix}}{\partial \mathbf{q}} * u_i, \tag{27}$$

where:

$$\begin{aligned} \frac{\partial \begin{pmatrix} (\mathbf{RM_f})_i \\ (\mathbf{M_t})_i \end{pmatrix}}{\partial \mathbf{q}} &= \begin{bmatrix} \frac{\partial (\mathbf{RM_f})_i}{\partial \mathbf{p}} & \frac{\partial (\mathbf{RM_f})_i}{\partial \mathbf{e}} \\ \frac{\partial (\mathbf{M_t})_i}{\partial \mathbf{p}} & \frac{\partial (\mathbf{M_t})_i}{\partial \mathbf{e}} \end{bmatrix} = \begin{bmatrix} \mathbf{O_{3\times 3}} & \frac{\partial \mathbf{R}}{\partial \mathbf{e}} * (\mathbf{M_f})_i \\ \mathbf{O_{3\times 3}} & \mathbf{O_{3\times 3}} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{O_{3\times 3}} & \begin{bmatrix} \frac{\partial \mathbf{R}}{\partial \phi} * (\mathbf{M_f})_i & \frac{\partial \mathbf{R}}{\partial \theta} * (\mathbf{M_f})_i & \frac{\partial \mathbf{R}}{\partial \psi} * (\mathbf{M_f})_i \end{bmatrix} \\ \mathbf{O_{3\times 3}} & \mathbf{O_{3\times 3}} \end{bmatrix}. \end{aligned} \tag{28}$$

Finally,

$$\mathbf{T_4} = \frac{\partial \mathbf{H^{-1}}}{\partial \dot{\mathbf{q}}} (\mathbf{B(q)u - C(q,\dot{q})\dot{q} - G(q)}) + \mathbf{H^{-1}} \frac{\partial (\mathbf{B(q)u - C(q,\dot{q})\dot{q} - G(q)})}{\partial \dot{\mathbf{q}}} = -\mathbf{H^{-1}C},$$

since $\mathbf{B(q)u - C(q,\dot{q})\dot{q} - G(q)} = \mathbf{0}$, with $\dot{\mathbf{q}} = \mathbf{0}$, $\frac{\partial (\mathbf{B(q)u - C(q,\dot{q})\dot{q} - G(q)})}{\partial \dot{\mathbf{q}}} = \frac{\partial (-\mathbf{C(q,\dot{q})\dot{q}})}{\partial \dot{\mathbf{q}}} = -\mathbf{C(q,\dot{q})}) \frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{q}}} = -\mathbf{C(q,\dot{q})}$.

So to sum up:

$$\mathbf{A_{lin}} = \begin{bmatrix} \mathbf{O_{6\times 6}} & \mathbf{I_{6\times 6}} \\ \mathbf{H^{-1}} * \sum_i \begin{bmatrix} \mathbf{O_{3\times 3}} & \begin{bmatrix} \frac{\partial \mathbf{R}}{\partial \phi} * (\mathbf{M_f})_i & \frac{\partial \mathbf{R}}{\partial \theta} * (\mathbf{M_f})_i & \frac{\partial \mathbf{R}}{\partial \psi} * (\mathbf{M_f})_i \end{bmatrix} \\ \mathbf{O_{3\times 3}} & \mathbf{O_{3\times 3}} \end{bmatrix} * u_i & \mathbf{-H^{-1}C} \end{bmatrix} \tag{29}$$

For $\mathbf{B_{lin}}$, we have:

$$\mathbf{B_{lin}} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}} = \begin{bmatrix} \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{H^{-1}(B(q)u - C(q,\dot{q})\dot{q} - G(q))}}{\partial \mathbf{u}} \end{bmatrix} \tag{30}$$

.

It is clear to see that $\frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{u}} = \mathbf{0}$, and $\frac{\partial \mathbf{H^{-1}(B(q)u - C(q,\dot{q})\dot{q} - G(q))}}{\partial \mathbf{u}} = \frac{\partial \mathbf{H^{-1}(B(q)u)}}{\partial \mathbf{u}} = \mathbf{H^{-1}B(q)} \frac{\partial \mathbf{u}}{\partial \mathbf{u}} = \mathbf{H^{-1}B(q)}$, (remember than none of **G, B, C, H** is related to **u**). So we have

$$\mathbf{B_{lin}} = \begin{bmatrix} \mathbf{O_{6\times k}} \\ \mathbf{H^{-1}B} \end{bmatrix} \tag{31}$$

For the 2D case $\mathbf{A_{lin}}$ and $\mathbf{B_{lin}}$ are simplified to become:

$$\begin{aligned} \mathbf{A_{lin}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} &= \begin{bmatrix} \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{q}} & \frac{\partial \dot{\mathbf{q}}}{\partial \dot{\mathbf{q}}} \\ \frac{\partial \mathbf{H^{-1}(B(q)u - C(q,\dot{q})\dot{q} - G(q))}}{\partial \mathbf{q}} & \frac{\partial \mathbf{H^{-1}(B(q)u - C(q,\dot{q})\dot{q} - G(q))}}{\partial \dot{\mathbf{q}}} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{O_{3\times 3}} & \mathbf{I_{3\times 3}} \\ \mathbf{H^{-1}} * \sum_i \begin{bmatrix} \mathbf{O_{2\times 2}} & \frac{\partial \mathbf{R}}{\partial \phi} * (\mathbf{M_f})_i \\ \mathbf{O_{1\times 2}} & \mathbf{O_{1\times 1}} \end{bmatrix} * u_i & \mathbf{O_{3\times 3}} \end{bmatrix} \end{aligned} \tag{32}$$

$$\mathbf{B_{lin}} = \begin{bmatrix} \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{H^{-1}(B(q)u - C(q,\dot{q})\dot{q} - G(q))}}{\partial \mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{O_{3\times k}} \\ \mathbf{H^{-1}B} \end{bmatrix} \tag{33}$$

The matrices $\mathbf{A_{lin}}$ and $\mathbf{B_{lin}}$ will then be optimized by the LQR to yield the control matrix $\mathbf{K}$. For the geometry-updating LQR, the quantities that are updated each time are $\mathbf{M_f}$, $\mathbf{M_t}$ and $\mathbf{J}$, which are all the time-varying values in the above derivation. Besides, the fixed-point is also recalculated.

Fixed Point Assuming $(\mathbf{x}^*, \mathbf{u}^*)$ satisfies $\mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) = \mathbf{0}$, we have:

$$\mathbf{R}\mathbf{M_f}\mathbf{u}^* = -m\mathbf{g} \tag{34}$$

$$\mathbf{M_t}\mathbf{u}^* = \mathbf{0} \tag{35}$$

for torque and force balance respectively. Given certain $\mathbf{M_f}$, $\mathbf{M_t}$, we will first solve the equation:

$$\begin{bmatrix} \mathbf{1} \\ \mathbf{M_t} \end{bmatrix} \mathbf{u} = \begin{pmatrix} -||m\mathbf{g}|| \\ \mathbf{0} \end{pmatrix} \tag{36}$$

to satisfy the torque balance. Then we will rotate the reference frame so that the direction of the combined thrust aligns with the Y-axis. The rotation axis is calculated by:

$$\mathbf{v} = \mathbf{M_f}\mathbf{u}^* \times (-m\mathbf{g}) \tag{37}$$

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{||\mathbf{v}||} \tag{38}$$

The angle is calculated by:

$$\alpha = \frac{-m\mathbf{g}^T\mathbf{M_f}\mathbf{u}^*}{||\mathbf{M_f}\mathbf{u}^*||||m\mathbf{g}||} \tag{39}$$

After the rotation in axis-angle form is calculated, the fixed-point Euler angles would be extracted to form the $\mathbf{e}^*$ part of $\mathbf{x}^*$.

8 Toward a Real Soft Drone

Although we carried out the experiments purely in numerical simulation environments, we designed our approach with its real-life feasibility in mind, and our method is intrinsically suitable for real-world deployment. First, our perception of the soft drone is explicitly sensor-based. Unlike many other works that deal with soft-robot controls like [5] [6] which observe the full state (particle positions) and apply model reduction techniques to synthesize the state, we resist this unrealistic assumption, and throughout our pipeline, the interfacing between the simulator and the training/controlling modules is strictly limited to the sensor measurements. In this sense, we observe the simulation environment in the same limited fashion as we observe the real world, so that no unfair advantage is taken. We expect the rest of the pipeline to work exactly the same if we swap the simulator with the real-world environment, since the interfacing will not be changed. Secondly, as we have mentioned in the paper, in designing the sensing scheme, the only sensors we used are Inertial Measurement Units (IMU), which are basic and accessible tools used everywhere for rigid drones. No other sensor types, such as bending, thermal or fluidic sensors are used. This simplistic approach allows us to conveniently fabricate these soft drones by implanting the IMU microprocessors at the surface, without having to cut open the drone's body or insert extra measurement devices. Basically, to fabricate an actual soft drone, we just need to cut out the desired shape from solid materials (if not with 3D printing techniques), implant the IMUs at the surface, set up their connection to a central onboard processor using WIFI, and flash the trained neural network and controlling script onto the hardware. Thirdly, the computational efficiency of our algorithm allows it to be handled by on-board processors. In the testing case, our relinearization is done at $10Hz$, and can be relaxed to $20Hz$ for the more stable geometries. There have been previous works conducted that performs LQR recalculation [7] and network-based control loop [8] at $10Hz$ using onboard computers. With further code optimization, we believe that our current system can be implemented fully onboard. Lastly, we simulate the soft body using a co-rotated elastic Finite Element simulator, which is known for providing physically realistic behaviors and is commonly used in engineering design, with noise and time delay applied. As a result, our success in this simulation testing environment is meaningful for its future adaptation to real-world scenarios.

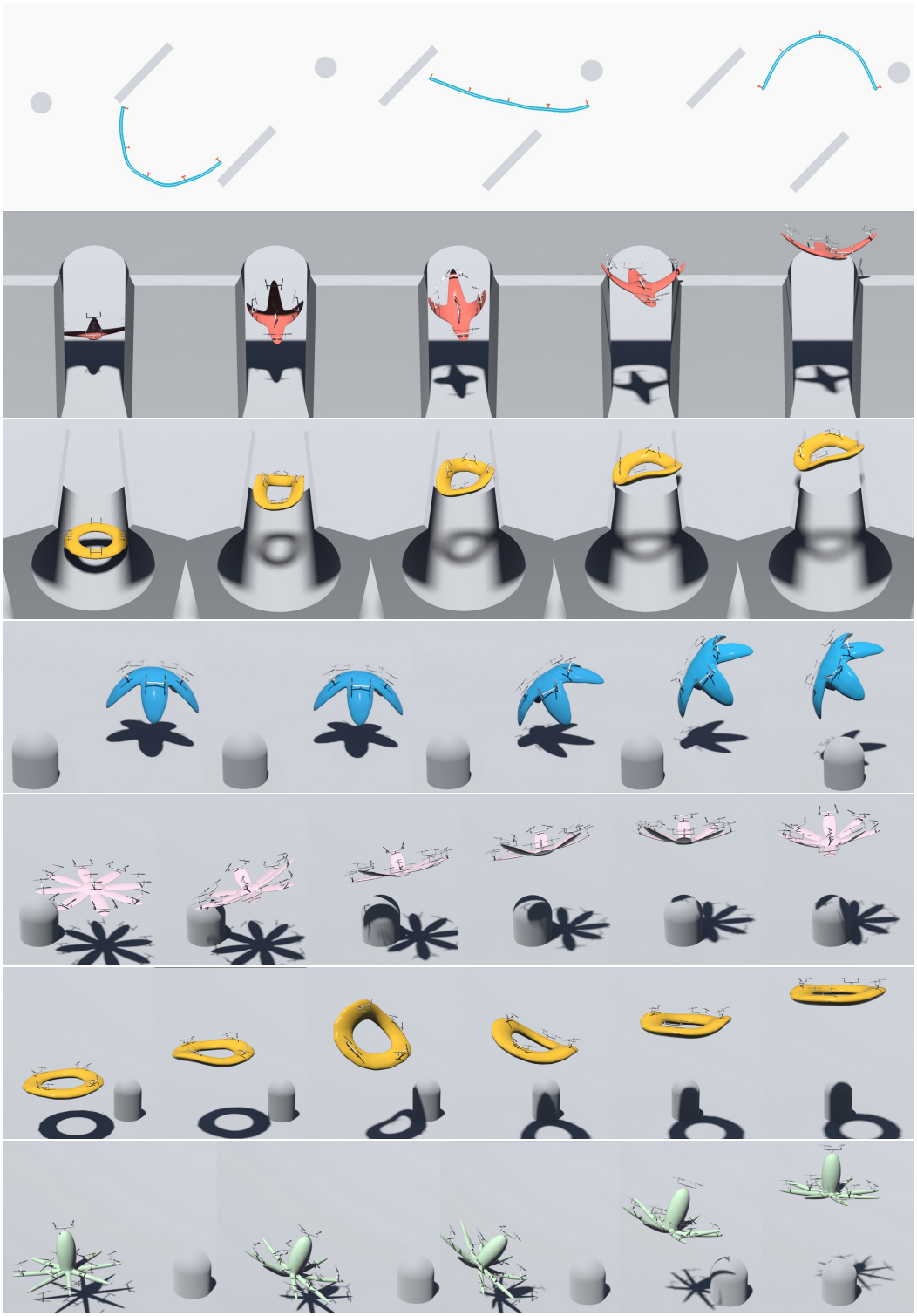

Figure 8: Visualization of more test results; Top 3: Obstacle avoidance animation; Bottom: Locomotion animation

References

[1] H. Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41 (2), Feb. 2015. ISSN 0098-3500. doi:10.1145/2629697. URL https://doi.org/10.1145/

2629697.

[2] F. L. Markley, Y. Cheng, J. L. Crassidis, and Y. Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.

[3] M. Müller and M. H. Gross. Interactive virtual materials. In *Graphics interface*, volume 2004, pages 239–246, 2004.

[4] R. Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832), downloaded on 7, 19, 2020 from http://underactuated.mit.edu/.

[5] J. Barbič and J. Popović. Real-time control of physically based simulations using gentle forces. *ACM transactions on graphics (TOG)*, 27(5):1–10, 2008.

[6] A. Spielberg, A. Zhao, Y. Hu, T. Du, W. Matusik, and D. Rus. Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. In *Advances in Neural Information Processing Systems*, pages 8284–8294, 2019.

[7] P. Foehn and D. Scaramuzza. Onboard state dependent lqr for agile quadrotors. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6566–6572. IEEE, 2018.

[8] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. *arXiv preprint arXiv:1806.08548*, 2018.