Soft Multicopter Control Using Neural Dynamics Identification

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]Dartmouth College, Computer Science Department
[2]Lawrenceville School
* `yitong.deng.gr@dartmouth.edu`

Abstract: We propose a data-driven method to automatically generate feedback controllers for soft multicopters featuring deformable materials, non-conventional geometries, and asymmetric rotor layouts, to deliver compliant deformation and agile locomotion. Our approach coordinates two sub-systems: a physics-inspired network ensemble that simulates the soft drone dynamics and a custom LQR control loop enhanced by a novel online-relinearization scheme to control the neural dynamics. Harnessing the insights from deformation mechanics, we design a decomposed state formulation whose modularity and compactness facilitate the dynamics learning while its measurability readies it for real-world adaptation. Our method is painless to implement, and requires only conventional, low-cost gadgets for fabrication. In a high-fidelity simulation environment, we demonstrate the efficacy of our approach by controlling a variety of customized soft multicopters to perform hovering, target reaching, velocity tracking, and active deformation.

Keywords: Soft Robotics, Deformation Mechanics, Data-driven control, LQR, Physics-informed Machine Learning

1 Introduction

Making a drone's body soft opens up brand new horizons to advance its maneuverability, safety, and functionalities. The intrinsic property of soft materials to deform and absorb energy during collision allows safe human-machine interactions [1, 2, 3]. In circumscribed environments, soft drones can naturally deform their bodies to travel through gaps and holes, making them effective for emergency rescues. Moreover, the ability to perform controlled deformation enables soft drones to perform secondary functionalities apart from aerial locomotion, such as flapping wings, grasping objects, and even operating machines, any additional mechanical parts.

Despite the various advantages that this promises, to date a reliable and practical algorithm that controls soft drones to fly and deform has been lacking, due to multifaceted challenges. Unlike how it is for rigid drones, which are fully defined by 12-dimensional state vectors, describing the state of soft drones is far from trivial. Since a continuum body deforms in infinite DOFs, one needs to design discrete representations both compact —— so that the underactuated control problem is feasible, and measurable —— so that the controller can adjust to unmodelled errors. Even if such discretizations are obtained, the dynamic interplay of these state variables cannot be derived analytically in closed-form, as the dynamics in the full space is governed by complex PDEs. Finally, if one is to adopt machine-learning methods to model the dynamics, a problem is posed by the complexity of soft body simulation, which inevitably leads to scarce data, making brute-force learning an ill-fated avenue.

Bridging the deformation mechanics, deep learning, and optimal control, our method is so designed to successfully overcome the abovementioned challenges. First, we adopt the polar decomposition theorem to build a low-dimensional decomposed state space consisting of three geometric, latent variables each representing rotation, translation, and deformation, which can be synthesized from the readings of a set of onboard Inertial Measurement Units (IMUs). We define the interdependencies of these three variables and learn them with lightweight neural networks, thereby learning a

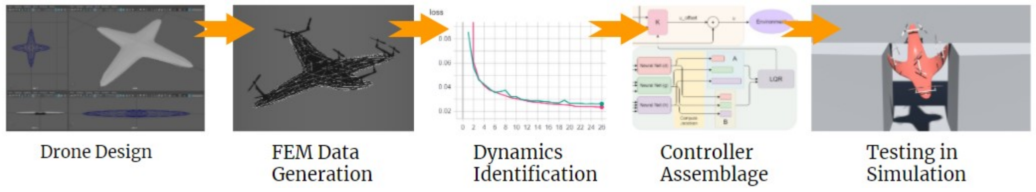

Figure 1: The 5-stage pipeline of our contorller generation system

neural simulator in a latent space on which the controller will be based. With automatic differentiation, we then extract the numeric gradients of the learned system to be controlled with a Linear Quadratic Regulator (LQR). Due to the fact that LQR requires the system to be linearized around fixed points that are inaccessible, we extend it with a novel online-relinearization scheme that iteratively converges to the desired target, bringing robustness and convenience for human piloting.

As shown in Figure. 1, our system takes soft drone geometries as input, and returns functionals that compute control matrices based on the drone's current state. To the best of our knowledge, the proposed approach is the first to control soft drones that are meant to deform significantly in flight; we show that we can not only regulate such deformation for balanced locomotion, but also capitalize on the deforming ability to perform various feats in the air.

2 Related Work

Multicopter Over the last few years, quadcopters have virtually dominated the commercial UAV industry, thanks to their simple mechanical structures, optimized efficiency for hovering, and easy-to-control dynamics that has been extensively studied by [4, 5] and many more. Various methods have been successfully developed to control quadcopters, including PD/PID [6], LQR [7, 8], differential flatness [9], sliding mode [10], and MPC [11] methods. Recent research explores the control of non-conventional multicopter designs, including ones with extra rotors [12], asymmetric structures [7], articulated linkage [13], or with gliding wings [14]. The problem of controlling drones fabricated with soft materials is still understudied.

Aerial Deformation Recent works have explored the potential of drones to actively deform in flight [15]. [13, 16] achieve impressive results with their multi-linked drones in passing through small openings or grasping objects, but the added mechanical components in their designs imply additional cost, fabrication complexity, energy consumption, and maladroitness. [17] proposes a lightweight, planar folding mechanism actuated by servo motors that is effective in controlling the drones to travel through confined spaces, but the simplified mechanism limits the ability to perform extra functionalities such as grasping. [18] propose the incorporation of a cable-actuated soft gripper with a rigid quadcopter to achieve load manipulation. While previous works add additional actuators to control deformation, we control deformation jointly with locomotion using rotors only.

Data-driven Soft Robot Control The modeling and control of soft robots is a challenging problem due to the high DOFs and the non-linear dynamics [1], which together make closed-form solutions unfeasible to be derived [19], and therefore making data-driven approaches favorable. [20, 21, 22], and many more, use deep reinforcement learning to train neural network controllers for soft robots. [23, 24] marries machine learning with control theory, applying MPC or PD control methods on models learned from data. [19] propose the possibility of end-to-end supervised learning with a differentiable soft-body simulator, although their current design does not facilitate feedback mechanism in real-life due to the assumption of full state measurement.

3 Methodology

Overview The workflow of our proposed system is given in Figure. 2. We start with the IMU sensor readings, which are position, orientation and their rates of change at multiple locations of the drone's body, which will be processed to form the current state vector $\mathbf{x}$ as the concatenation of the three decomposed latent variables $\mathbf{s}$, $\mathbf{e}$, and $\mathbf{p}$. This vector, along with the previously applied actuation $\mathbf{u}$ will be passed into the three trained network modules as inputs. Then we extract the numeric gradients to form the Jacobian matrices, which will be used to assemble the matrices $\mathbf{A}$ and $\mathbf{B}$ representing the linearized dynamics. Then, $\mathbf{A}$ and $\mathbf{B}$ will be passed into the LQR algorithm to form the control matrix $\mathbf{K}$, given which we simply need to pass in our current state $\mathbf{x}$, our next waypoint state $\mathbf{x}_{wp}$, and the current actuation $\mathbf{u}$, to obtain the new actuation $\mathbf{u}_{next}$, which will be applied to the drone's rotors to complete the control loop. The $\mathbf{x}_{wp}$ will be computed by a Proportional-Derivative (PD) controller given final goal state $\mathbf{x}_{goal}$. The control loop will operate at $100Hz$ while the re-computation of $\mathbf{K}$ takes place at $10Hz$.

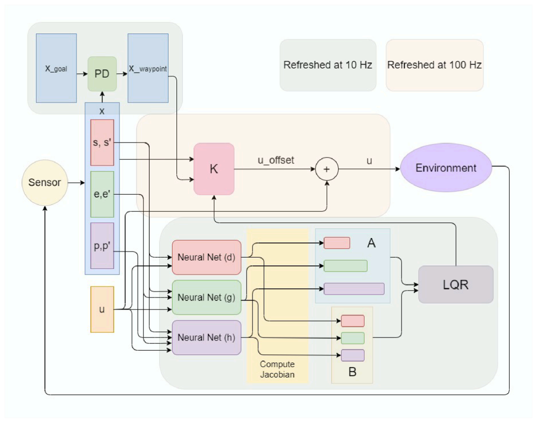

Figure 2: Method overview

Geometric State Decomposition Let $\hat{\Omega}$, Ω denote the volume of the drone's undeformed and deformed geometry, and let $\hat{\mathbf{z}}$, $\mathbf{z}$ be their respective finite discretizations as particles forming the solid body, with $\mathbf{z} = \Phi(\hat{\mathbf{z}})$. We can always write:

$$\mathbf{z} = \mathbf{R}S(\hat{\mathbf{z}}) + \mathbf{p}, \tag{1}$$

for some rotation matrix $\mathbf{R}$, some position $\mathbf{p} \in \mathbf{R}^3$, and some non-linear function S which reduces to Φ when $\mathbf{R} = \mathbf{I}$ and $\mathbf{p} = \mathbf{0}$. In other words, we first define a local reference frame by the $SO3$ transformation resulting from $\mathbf{R}$ and $\mathbf{p}$, and express the deformation in such a frame with S, thereby decomposing the total transformation into a rigid component and a deformable component. In this way, the rigid, $SO3$ component can be simulated side by side with the deformable component S, an approach proposed by [25] and adopted by various works in the deformation simulation community [26, 27, 28], taking advantage of the fact that the local-frame deformation S is much nicer to work with than the world-frame deformation Φ. In this work, we adopt the same philosophy and learn evolution rules for $\mathbf{R}$, $\mathbf{p}$, and S in juxtaposition.

Measurability Since our goal is to build feedback controllers for $\mathbf{z}$, *i.e*, $(\mathbf{R}, \mathbf{p}, S)$, we would need to define $\mathbf{R}$, $\mathbf{p}$, and S in such a way that they can be measured by sensors. In this work, we adopt the following strategy. As shown in Figure. 3 we plant Inertial Measuring Units (IMUs), which are compact, low-cost MEMS outputting rotational and translational status, at the drone's geometric center and around its periphery, next to the propellers. For the local frame rotation $\mathbf{R}$, we use the Euler angles $\mathbf{e} = (e1, e2, e3)$ measured at the geometric center. For local frame origin $\mathbf{p}$, we average the positional measurements of all the IMUs to approximate the center of mass. For the local frame deformation S, we obtain an approximate representation $\mathbf{s}$ by computing a scalar angle difference s_i between the Y-axis measured by each peripheral IMU and the Y-axis defined by $\mathbf{R}$, and concatenate these scalars together. In the shown case, each entry of $\mathbf{s}$ represents how its associated wing is folded, where the sign indicates inward or outward and the magnitude indicates the angle. Such a design of $\mathbf{s}$ facilitates interactive piloting since it is intuitive to describe one's desired deformation in terms of the bent angles. Since we have parameterized $\mathbf{z}$ with $\mathbf{s}$, $\mathbf{e}$ and $\mathbf{p}$, controlling the dynamics of $\mathbf{z}$ now relegates to controlling the dynamics of $(\mathbf{s}, \mathbf{e}, \mathbf{p})$.

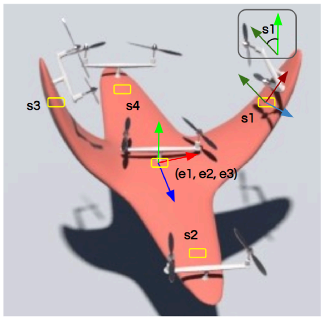

Figure 3: Sensor scheme

Dynamic Coupling The drone's body will be actuated by a set of m propellers, each providing a scalar thrust along the normal of the surface it is planted on. Let $\mathbf{u} \in \mathbf{R}^m$ denote the thrusts. The second-order dynamics of $\mathbf{z}$ is given by the momentum conservation:

$$m\ddot{\mathbf{z}} = f_{actuation}(\mathbf{z}, \mathbf{u}) + f_{stress}(\mathbf{z}) + f_{damping}(\dot{\mathbf{z}}) + m\mathbf{g} \tag{2}$$

where m represents the particle mass. We note here the translational and rotational symmetries of the experienced forces. Since $f_{actuation}$ depends on the rotors' normal directions which are computed locally with neighboring particles, it is unchanged when the whole of $\mathbf{z}$ is rotated or translated.

The translational and rotational invariance of f_{stress} and $f_{damping}$ is based on the principle of material frame-indifference in continuum mechanics, which states that the behavior of a material is independent of the reference frame [29]. Combining the symmetry with the equality $\mathbf{z} = \mathbf{R}S(\hat{\mathbf{z}})+\mathbf{p}$, and ignoring the Coriolis forces resulting from the evolving $\mathbf{R}$, we get that:

$$m\ddot{\mathbf{z}} \approx \mathbf{R}f_{actuation}(S(\hat{\mathbf{z}}), \mathbf{u}) + \mathbf{R}f_{stress}(S(\hat{\mathbf{z}})) + \mathbf{R}f_{damping}(\dot{S(\hat{\mathbf{z}})}) + m\mathbf{g}. \tag{3}$$

Since $\hat{\mathbf{z}}$ is constant, $S(\hat{\mathbf{z}})$ is a function of $\mathbf{s}$, and we know $\mathbf{R}$ is a function of $\mathbf{e}$, then the dynamics coupling of $\mathbf{s}$, $\mathbf{e}$, and $\mathbf{p}$ are as follows. Since $\ddot{\mathbf{p}}$ is the average of $\ddot{\mathbf{z}}$, it depends on $\mathbf{u}$, $\mathbf{e}$, $\dot{\mathbf{e}}$, $\mathbf{s}$ and $\dot{\mathbf{s}}$. Since $\mathbf{R}$ is measured in the local geometric center of $\mathbf{z}$, $\ddot{\mathbf{e}}$ depends on $\mathbf{u}$, $\mathbf{e}$, $\dot{\mathbf{e}}$, $\mathbf{s}$ and $\dot{\mathbf{s}}$ as well. For $\ddot{\mathbf{s}}$, since it is measured by projecting $\mathbf{z}$ onto the local frame by left-multiplying $\mathbf{R}^{-1}$, the $\mathbf{R}$ component in $\ddot{\mathbf{z}}$ cancels out for $\ddot{\mathbf{s}}$ and $\ddot{\mathbf{s}}$ depends on $\mathbf{u}$, $\mathbf{s}$ and $\dot{\mathbf{s}}$ only.

These interdependencies will be modelled by learned neural networks, in particular, we train networks $\{\mathbf{d}, \mathbf{g}, \mathbf{h}\}$ such that $\dot{\mathbf{s}}_{next}= \mathbf{d}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{u})$, $\dot{\mathbf{e}}_{next}= \mathbf{g}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \mathbf{u})$, and $\dot{\mathbf{p}}_{next}= \mathbf{h}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \dot{\mathbf{p}}, \mathbf{u})$.

Learning the Dynamics Training the networks $\mathbf{d}$, $\mathbf{g}$ and $\mathbf{h}$ can be done in a relatively straightforward fashion. The three networks share the same lightweight architecture consisting of four residual blocks [30] featuring linear layers as previously explored by [31, 32]. The three networks will be trained separately using the same reservoir of data samples of the form $\{\mathbf{s}, \mathbf{e}, \mathbf{p}, \dot{\mathbf{s}}, \dot{\mathbf{e}}, \dot{\mathbf{p}}, \mathbf{u}, \dot{\mathbf{s}}_{next}, \dot{\mathbf{e}}_{next}, \dot{\mathbf{p}}_{next}\}$ generated from Finite Element Method (FEM) simulation. We use Adam optimizer and L1 loss for optimization, with hyperparameter details given in the supplement. There are two techniques that we adopt that are worth reporting. First, for data generation, we would apply a constant random thrust uniformly sampled in the range $[-T\_max, T\_max]$, where T_max denotes a rotor's max output thrust, to the drone's rotors for $1s$, before a new random thrust is applied. The soft drone would be dancing and twisting in the air, but despite that, this method leads to successful trainings, while generating data from a guiding controller, or switching random control signals at every instant, would fail. Secondly, we've found that trainings converge much better when we predict the next frame's velocity using the current one, instead of directly predicting the acceleration. Although this approach can lead to local minima for settling at the input, current velocity, that does not happen in our experiments and the acceleration can be convincingly reconstructed from the two velocities, as we shall elaborate in the results section.

Controlling the Learned Dynamics Once the networks $\mathbf{d}$, $\mathbf{g}$ and $\mathbf{h}$ are trained, the dynamics can be expressed as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{\mathbf{s}} \\ \ddot{\mathbf{s}} \\ \dot{\mathbf{e}} \\ \ddot{\mathbf{e}} \\ \dot{\mathbf{p}} \\ \ddot{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{s}} \\ \frac{1}{\alpha}(\mathbf{d}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{u}) - \dot{\mathbf{s}}) \\ \dot{\mathbf{e}} \\ \frac{1}{\alpha}(\mathbf{g}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \mathbf{u}) - \dot{\mathbf{e}}) \\ \dot{\mathbf{p}} \\ \frac{1}{\alpha}(\mathbf{h}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \dot{\mathbf{p}}, \mathbf{u}) - \dot{\mathbf{p}}) \end{bmatrix}, \tag{4}$$

To control such learned dynamics, we summon the Linear Quadratic Regulator (LQR), a well-proven method for controlling rigid drones. Since LQR requires a linear system, we will perform first-order Taylor expansion around an operating point $(\mathbf{x}^*, \mathbf{u}^*)$ with Jacobian matrices from automatic differentiation which can be done with PyTorch. In particular, we write:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \approx \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) + \mathbf{A}(\mathbf{x} - \mathbf{x}^*) + \mathbf{B}(\mathbf{u} - \mathbf{u}^*), \tag{5}$$

where $\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\big|_{\mathbf{x}^*, \mathbf{u}^*}$, and $\mathbf{B} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\big|_{\mathbf{x}^*, \mathbf{u}^*}$, as shown in Equation 6:

$$\mathbf{A} = \begin{bmatrix} \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \frac{1}{\alpha}\frac{\partial \mathbf{d}}{\partial \mathbf{s}} & \frac{1}{\alpha}(\frac{\partial \mathbf{d}}{\partial \dot{\mathbf{s}}} - \mathbf{I}) & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} \\ \frac{1}{\alpha}\frac{\partial \mathbf{g}}{\partial \mathbf{s}} & \frac{1}{\alpha}\frac{\partial \mathbf{g}}{\partial \dot{\mathbf{s}}} & \frac{1}{\alpha}\frac{\partial \mathbf{g}}{\partial \mathbf{e}} & \frac{1}{\alpha}(\frac{\partial \mathbf{g}}{\partial \dot{\mathbf{e}}} - \mathbf{I}) & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} \\ \frac{1}{\alpha}\frac{\partial \mathbf{h}}{\partial \mathbf{s}} & \frac{1}{\alpha}\frac{\partial \mathbf{h}}{\partial \dot{\mathbf{s}}} & \frac{1}{\alpha}\frac{\partial \mathbf{h}}{\partial \mathbf{e}} & \frac{1}{\alpha}\frac{\partial \mathbf{h}}{\partial \dot{\mathbf{e}}} & \mathbf{O} & \frac{1}{\alpha}(\frac{\partial \mathbf{h}}{\partial \dot{\mathbf{p}}} - \mathbf{I}) \end{bmatrix}_{\mathbf{x}^*, \mathbf{u}^*} \quad \mathbf{B} = \begin{bmatrix} \mathbf{O} \\ \frac{1}{\alpha}\frac{\partial \mathbf{d}}{\partial \mathbf{u}} \\ \mathbf{O} \\ \frac{1}{\alpha}\frac{\partial \mathbf{g}}{\partial \mathbf{u}} \\ \mathbf{O} \\ \frac{1}{\alpha}\frac{\partial \mathbf{h}}{\partial \mathbf{u}} \end{bmatrix}_{\mathbf{x}^*, \mathbf{u}^*}. \tag{6}$$

If we make the assumption that for $(\mathbf{x}^*, \mathbf{u}^*)$ and $(\mathbf{x}, \mathbf{u})$ close enough to each other, $\mathbf{f}(\mathbf{x}, \mathbf{u}) - \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) \approx \mathbf{f}(\mathbf{x} - \mathbf{x}^*, \mathbf{u} - \mathbf{u}^*)$, then we have:

$$(\dot{\mathbf{x} - \mathbf{x}^*}) = \mathbf{f}(\mathbf{x} - \mathbf{x}^*, \mathbf{u} - \mathbf{u}^*) \approx \mathbf{A}(\mathbf{x} - \mathbf{x}^*) + \mathbf{B}(\mathbf{u} - \mathbf{u}^*). \tag{7}$$

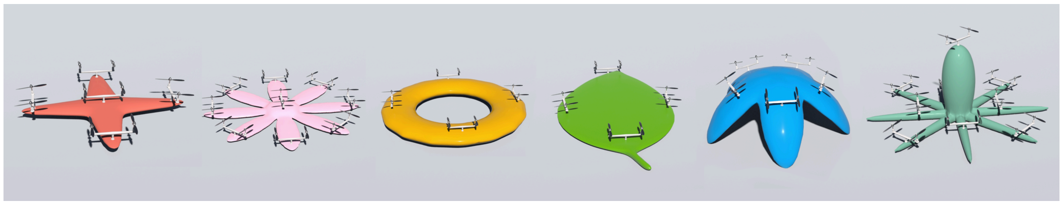
Figure 4: 3D models used in our experiments

Once the linear system is obtained, LQR outputs the control matrix $\mathbf{K}$ and the control policy $\mathbf{u}-\mathbf{u}^*=-\mathbf{K}(\mathbf{x}-\mathbf{x}^*)$ that drives $\mathbf{x}$ to $\mathbf{x}^*$ while keeping $\mathbf{u}$ close to $\mathbf{u}^*$ by minimizing the cost function $\int_0^\infty (\mathbf{x}-\mathbf{x}^*)^{\mathbf{T}}\mathbf{Q}(\mathbf{x}-\mathbf{x}^*)+(\mathbf{u}-\mathbf{u}^*)^{\mathbf{T}}\mathbf{R}(\mathbf{u}-\mathbf{u}^*)dt$. The $\mathbf{Q}$ and $\mathbf{R}$ matrices are cost matrices used to manage the tradeoff between the two objectives. The optimization is done by solving the Continuous-time Algebraic Riccati Equation with SciPy.

Online Relinearization Traditionally, the operating point, which is the state-actuation pair $(\mathbf{x}^*,\mathbf{u}^*)$ is chosen to be a fixed point such that $\mathbf{f}(\mathbf{x}^*,\mathbf{u}^*)=\mathbf{0}$. For typical rigid drones, such fixed points can be obtained trivially, and yet, for deformable drones, without iteratively testing and optimizing with the neural dynamic system, it is generally not possible to know beforehand which set of rotor input would exactly balance the internal stress, viscous damping, and external gravity, or if such a balance exists. We see the fixed-point assumption as being overly strict and seek to circumvent it. The purpose of assuming $\mathbf{f}(\mathbf{x}^*,\mathbf{u}^*)=\mathbf{0}$ is to turn the affine Equation. 5 into a linear one which LQR recognizes. In that case, $(\mathbf{x}^*,\mathbf{u}^*)$ is time-invariant and indeed $\dot{\mathbf{x}}=(\mathbf{x}\dot{-}\mathbf{x}^*)\approx\mathbf{0}+\mathbf{A}(\mathbf{x}-\mathbf{x}^*)+\mathbf{B}(\mathbf{u}-\mathbf{u}^*)$. Since in our case we cannot guarantee $\mathbf{f}(\mathbf{x}^*,\mathbf{u}^*)=\mathbf{0}$, then $(\mathbf{x}\dot{-}\mathbf{x}^*)\neq\mathbf{A}(\mathbf{x}-\mathbf{x}^*)+\mathbf{B}(\mathbf{u}-\mathbf{u}^*)$, so directly running LQR with $\mathbf{A}$ and $\mathbf{B}$ would typically fail. However, by making the proximity assumption as in Equation. 7, we can obtain the linear form locally. In other words, with $(\mathbf{x}^*,\mathbf{u}^*)$ not being fixed points, we can still regulate close-enough neighbor states to it.

Algorithm 1 Online Relinearizing LQR

```
Input: $\mathbf{x}_{curr}$, $\mathbf{x}_{goal}$, $kp$, $kd$, $n$, $\mathbf{Q}$, $\mathbf{R}$
1: $\mathbf{u}_{curr} \leftarrow \mathbf{0}$
2: $iter \leftarrow 0$
3: while running do
4:     update $\mathbf{x}_{curr}$, $\dot{\mathbf{x}}_{curr}$
5:     $iter \leftarrow iter + 1$
6:     if $iter \bmod n = 0$ then
7:         $\mathbf{x}_{wp} \leftarrow kp\cdot(\mathbf{x}_{goal}-\mathbf{x}_{curr})+kd\cdot\dot{\mathbf{x}}_{curr}$
8:         $\mathbf{A},\mathbf{B} \leftarrow \mathcal{J}(\mathbf{d},\mathbf{g},\mathbf{h},\mathbf{x}_{curr},\mathbf{u}_{curr})$
9:         $\mathbf{K} = LQR(\mathbf{A},\mathbf{B},\mathbf{Q},\mathbf{R})$
10:        $\mathbf{u}_{oper} \leftarrow \mathbf{u}_{curr}$
11:    end if
12:    $\mathbf{u}_{curr} \leftarrow -\mathbf{K}(\mathbf{x}_{curr}-\mathbf{x}_{wp})+\mathbf{u}_{oper}$
13: end while
```

As in Algorithm. 1, our solution is built upon this observation. We initialize our drone with arbitrary $\mathbf{x}$, and $\mathbf{u}=\mathbf{0}$. At each instant, we will linearize around the current $(\mathbf{x},\mathbf{u})$. We run LQR with the linear system to obtain $\mathbf{K}$. In practice, we don't want to attract neighboring state to current state, but rather drive the current state to our goal state. Our strategy is that if we want to reach a state $\mathbf{x}_{wp}$ from $\mathbf{x}_{curr}$, then we will pretend to be at $(\mathbf{x}_{curr}-\mathbf{x}_{wp})$ trying to reach $\mathbf{x}_{curr}$, and compute $\mathbf{u}=-\mathbf{K}((\mathbf{x}_{curr}-\mathbf{x}_{wp})-\mathbf{x}_{curr})=-\mathbf{K}(-\mathbf{x}_{wp})$. Given the current state $\mathbf{x}_{curr}$ and the goal state $\mathbf{x}_{goal}$, we calculate $\mathbf{x}_{wp}$ using a PD control: $\mathbf{x}_{wp}=kp\cdot(\mathbf{x}_{goal}-\mathbf{x}_{curr})+kd\cdot\dot{\mathbf{x}}_{curr}$. The control matrix will be used for n timesteps before updated again.

4 Experiments and Evaluation

To test the efficacy of our method, we design a number of soft drone models in 2D and 3D featuring asymmetrical structures and odd numbers of rotors, as depicted in Figure. 4. We conduct training on each model individually and use the generated controllers to direct the models to perform hovering, target reaching, velocity tracking, and active deformation.

Testing the Linearized Networks To verify the quality of our learned dynamics, we focus on two aspects. First, since our model predicts the next frame's velocity with the current velocity, rather than the acceleration (which is what we ultimately want), we need to make sure that the network actually learns how the velocity evolves, instead of reproducing the current velocity. As seen in the left

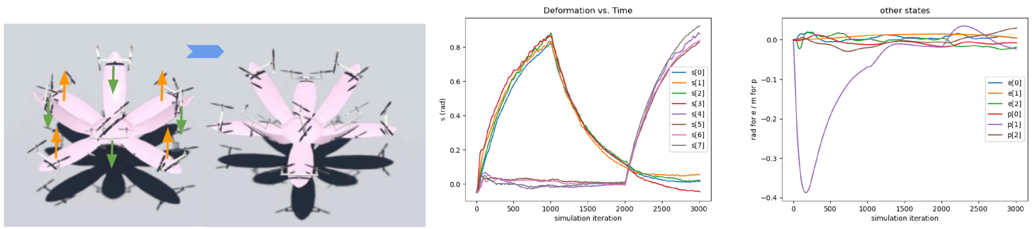

Figure 6: Active deformation while hovering

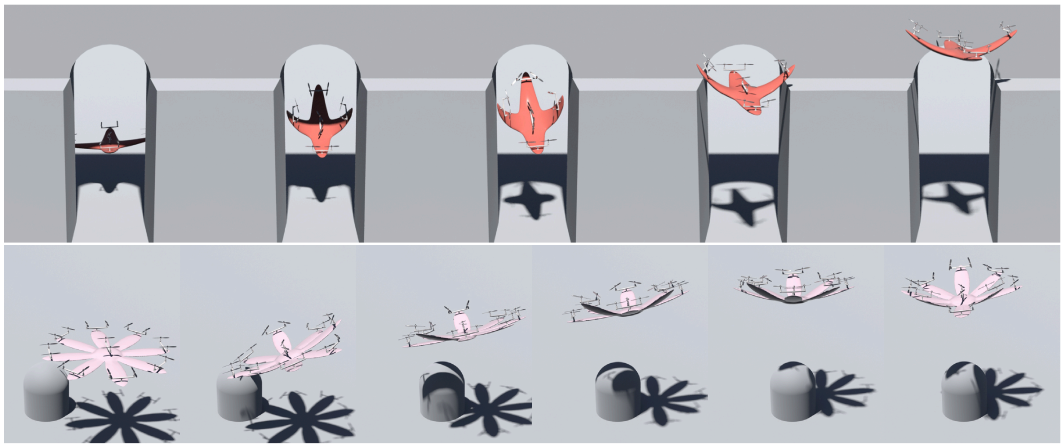

Figure 7: Top: obstacle avoidance; bottom: Target reaching

subfigure of Figure. 5, which overlays the actual acceleration and the predicted acceleration reconstructed from the predicted velocity, the networks learn the evolution rules of velocity successfully. Secondly, since the LQR controller relies on the Taylor-expanded version of the learned networks, it is necessary to verify how well the networks' gradient matches that of the actual dynamics. Since the ground-truth gradient is difficult to obtain, we test this by reproducing temporal sequences using the linearized version as in Equation. 5 relinearized at $20Hz$. As shown in the right subfigure, even with error accumulation, the linearized evolution sequence (*Jacobian Matrix Update*) keeps up with the ground truth for over a minute, showing the linearized neural network can reliably approximate the real dynamics locally.

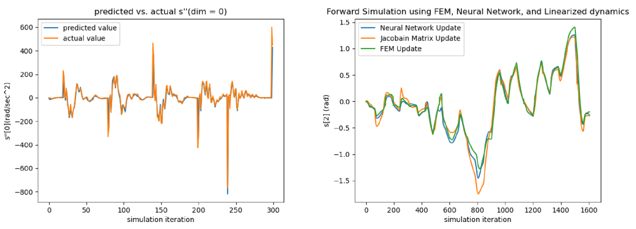

Figure 5: Testing results of the trained networks

Aerial Deformation Our method successfully controls the drones to deform into specified configurations while hovering or tracking a velocity. The first experiment instructs the *flower* drone to maintain its location and orientation while deforming into two different configurations: first with lateral petals raised and axial petals flat, then with lateral petals flat and axial petals raised, as shown in Figure. 6. In the middle subfigure, the pitch of all eight petals are controlled as expected in a smooth manner: petals $1,\ 3,\ 5,\ 7$ will rise first while petals 2 , 4 , 6 , 8 stay still, before the roles are gradually switched. The right subfigure shows that this is done while maintaining precise control of balance and position. Ever since the initial drop in the $-Y$ direction immediately after release, other rotational and translational movements are contained within $\pm 3°$ and $\pm 5cm$.

The second experiment combines velocity tracking with aerial deformation in an **obstacle avoidance** scenario. As shown in the top of Figure. 7, three concrete blocks form a gap that is narrower than the *star* drone's body along the Z direction. Here, we instruct the drone to fold up two wings by increasing their associated angles, and at the same time maintaining a forward and upward velocity in the $(+X, +Y)$ direction. It is also crucial to limit the deviation along the Z direction in order to not crush into the blocks. As one sees in the first subfigure

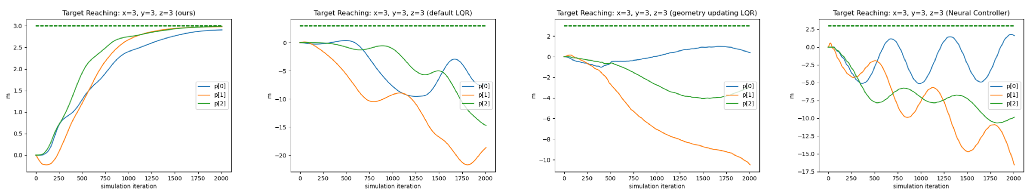

Figure 9: Left to right: ours, LQR, geometry-updating LQR, neural controller drive the drone to reach coordinate $(3, 3, 3)$ from $(0, 0, 0)$, plotted are the time-varying (x, y, z) coordinates

of Figure. 8, our controller increases the wing pitch by over 73°, reducing its width for over 30%. As seen in the second subfigure, it does so while maintaining a steady velocity in the $(+X, +Y)$ direction, and deviating for less than $5cm$ in the Z-axis throughout the entire sequence, thus leading to the successful object avoidance as depicted in the bottom of Figure. 7.

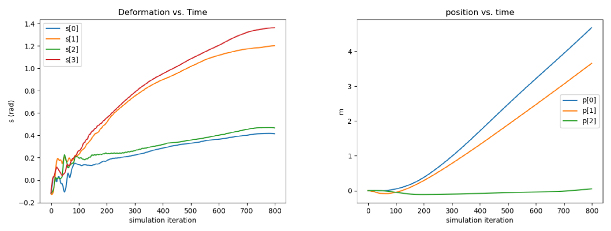

Figure 8: Obstacle avoidance performance

SOTA Comparison We compare our method with three different controllers: traditional LQR, geometry-updating LQR, and a neural network controller trained end-to-end using the strategy proposed by [19]. Our metrics will base on locomotion, since existing controllers for soft drone deformation is unavailable. The first candidate computes an LQR control matrix using the relevant quantities —— fixed point, rotor position, rotor orientation, and rotational inertia, calculated from the drone's rest shape. The second recomputes the control matrix at every timestep with the relevant quantities updated based on the deformation, with further details given in the supplement. The third takes our trained network dynamic system as a differentiable simulator to train another network controller. We instruct all controllers to direct the *flower* drone to travel from position $(0, 0, 0)$ to $(3, 3, 3)$. As depicted in Figure. 9, our method (left) successfully drives the (x, y, z) coordinates of the drone to the target within 7% error, while the other controllers fail due to the fragility of the soft drone dynamics. As shown in the table below, we compare their performance numerically with three metrics: final error, thrust usage, and the survival time before illegal configurations are encountered, which shows that our method excels the other candidates by far.

Target Reaching				
metrics	ours	LQR	Geometry-updating LQR	Neural Controller
survival time (s)	**20.0**	0.67	0.69	0.19
final error (m)	**0.099**	29.744	14.989	23.454
thrust usage (N)	**26740**	90596	61947	179996

Ablation Testing: State Decomposition The state decomposition is helpful for defining a local frame in which deformation can be expressed compactly as scalars. In this experiment, we train without decomposing and feed the network with the concatenated vector of the position, center orientation, and peripheral orientations. As displayed in Figure. 10, the validation loss is unable to converge successfully, signifying that the network fails to learn the correct patterns. This can be due to the fact that such unprocessed states double the size of the decomposed version, and also that the evolution rules of the position and orientation are highly unalike, thus demanding the network to evolve into multi-modal behaviors.

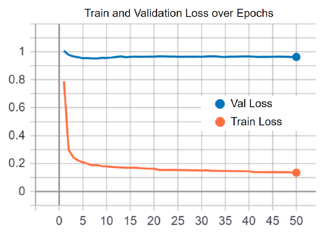

Figure 10: Convergence without decomposition

Ablation Testing: Online Relinearization We show the importance of the online relinearing mechanism by testing the controller performance where the input targets $(\mathbf{x}_{goal}, \mathbf{u}_{goal})$ deviates from a fixed point $(\mathbf{x}^*, \mathbf{u}^*)$ by different margins. We first obtain a fixed point by trial-and-error and then fine-tune it with gradient descent using our learned networks. Then, we keep the goal location untouched, and pollute the rest of the fixed point by adding uniform random noise proportional to

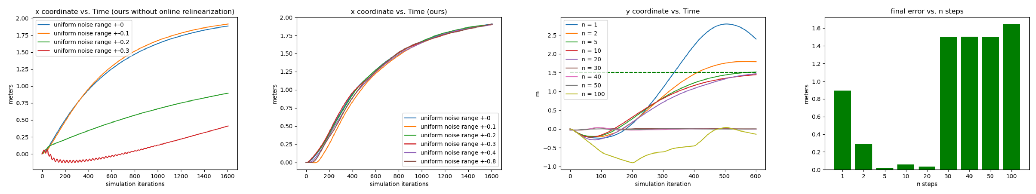

Figure 11: Left 2: The online relinearization scheme provides excelling robustness to non-fixed targets; Right 2: Control performance and final error under different relinearization timestep n

each state variable by 10%, 20%, 30%, 40%, and 80%. We direct the *rod* drone to translate for 2 units along the $+X$ direction. The results are depicted in the left 2 subfigures of Figure. 11. The first subfigure shows the results without online relinearization, where the control quality deteriorates significantly when over 20% noise is added, and start to generate NANs after 40% noise is added. The second subfigure shows the results with online relinearization, in which the controller performs steadily even with 80% noise added. The significance is that during deployment the pilot only needs to input the target configuration without worrying about the hard-to-compute fixedness of such configuration, thereby making our control system feasible for human piloting.

Effect of Relinearization Frequency The parameter n controls the relinearization frequency $\frac{100}{n}Hz$. The more often the system is linearized the more accurate the linear approximation is. And also, since we set a new waypoint at each linearization, whose distance away is inversely proportional to n, frequent linearzation sets many adjacent waypoints while infrequent linearization sets long-term, sparse waypoints. In this experiment, we test the performance of 9 different values of n: $\{1, 2, 5, 10, 20, 30, 40, 50, 100\}$, and show that it is not the case that larger n implies better performance. As shown in the right two subfigures of Figure. 11, $[5,\ 20]$ is clearly the sweet-spot of this parameter, where smaller n leads to overshoots, and larger n insufficient actuation.

5 Discussion and Conclusion

We propose a computational system to generate controllers for soft multicopters that jointly controls the locomotion and active deformation, without relying on extra mechanical parts. Our method takes advantage of a physics-inspired decomposed state space, and train neural networks to represent the dynamics. We control the neural dynamics system using an LQR controller enhanced with a novel online relinearization scheme. We use our method to successfully generate controllers for a variety of soft multicopters to perform hovering, target reaching, velocity tracking, and active deformation.

Sim2Real Transfer Our method is well-suited for real-world adaptations. First, we limit the interfacing between the simulator and the learning system strictly to sensor readings, so we make sure that no unrealistic benefit is gained from experimenting virtually. Secondly, we form a straightforward guideline for sensor deployment, featuring accessible gadgets with easy installation. Thirdly, the computation of LQR optimization and neural network evaluation can be realistically carried out in real-time by onboard computers, as previously explored by [33, 34]. Finally, being data-driven, our method betters the analytic approaches for Sim2Real adaptation, since it learns from data which contain the unmodelled nuances of the real world. The main challenge for Sim2Real transfer would be the experimental designs to generate meaningful data using the fabricated drones.

Limitations With our approach there are several limitations. First, the sensor placement requires human design, and there lacks a mechanism to tell, before training, if a sensing scheme would work. Secondly, every drone requires a separate training with no knowledge transfer. Thirdly, we use dual rotors with counter-rotation to cancel the spinning torque effects, which complicates the manufacturing process. Finally, the current control scheme is ineffective for aggressive maneuvers.

In the future, we plan to build an automated system for optimizing sensor locations as well as a unified neural dynamics platform that facilitates knowledge transfer, and to evaluate our approach on real-world soft multicopters.

Acknowledgments

We thank all the reviewers for their valuable comments. We acknowledges the funding support from Neukom Institute CompX Faculty Grant and Neukom Scholar Program, Burke Research Initiation Award, Toyota TEMA North America Inc, and NSF 1919647. We credit the Houdini Education licenses for the video generations.

References

[1] D. Rus and M. T. Tolley. Design, fabrication and control of soft robots. *Nature*, 521(7553): 467–475, 2015.

[2] C. Lee, M. Kim, Y. J. Kim, N. Hong, S. Ryu, H. J. Kim, and S. Kim. Soft robot review. *International Journal of Control, Automation and Systems*, 15(1):3–15, 2017.

[3] C. M. Best, M. T. Gillespie, P. Hyatt, L. Rupert, V. Sherrod, and M. D. Killpack. A new soft robot control method: Using model predictive control for a pneumatically actuated humanoid. *IEEE Robotics Automation Magazine*, 23(3):75–84, 2016.

[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] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *AIAA guidance, navigation and control conference and exhibit*, page 6461, 2007.

[6] A. Tayebi and S. McGilvray. Attitude stabilization of a four-rotor aerial robot. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, volume 2, pages 1216–1221. Ieee, 2004.

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

[8] S. Bouabdallah, A. Noth, and R. Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2451–2456. IEEE, 2004.

[9] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[10] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin. Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3712–3717. IEEE, 2005.

[11] Z. Wang, K. Akiyama, K. Nonaka, and K. Sekiguchi. Experimental verification of the model predictive control with disturbance rejection for quadrotors. In *2015 54th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 778–783. IEEE, 2015.

[12] R. Baranek and F. Šolc. Modelling and control of a hexa-copter. In *Proceedings of the 13th International Carpathian Control Conference (ICCC)*, pages 19–23. IEEE, 2012.

[13] M. Zhao, F. Shi, T. Anzai, K. Chaudhary, X. Chen, K. Okada, and M. Inaba. Flight motion of passing through small opening by dragon: Transformable multilinked aerial robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4735–4742. IEEE, 2018.

[14] 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)*, 38(4):1–12, 2019.

[15] D. Floreano, S. Mintchev, and J. Shintake. Foldable drones: from biology to technology. In *Bioinspiration, Biomimetics, and Bioreplication 2017*, volume 10162, page 1016203. International Society for Optics and Photonics, 2017.

[16] T. Anzai, M. Zhao, S. Nozawa, F. Shi, K. Okada, and M. Inaba. Aerial grasping based on shape adaptive transformation by halo: Horizontal plane transformable aerial robot with closed-loop multilinks structure. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6990–6996. IEEE, 2018.

[17] D. Falanga, K. Kleber, S. Mintchev, D. Floreano, and D. Scaramuzza. The foldable drone: A morphing quadrotor that can squeeze and fly. *IEEE Robotics and Automation Letters*, 4(2): 209–216, 2018.

[18] J. Fishman and L. Carlone. Control and trajectory optimization for soft aerial manipulation. *arXiv preprint arXiv:2004.04238*, 2020.

[19] 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.

[20] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi. Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators. *IEEE Transactions on Robotics*, 35 (1):124–134, 2018.

[21] S. Satheeshbabu, N. K. Uppalapati, T. Fu, and G. Krishnan. Continuous control of a soft continuum arm using deep reinforcement learning. In *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, pages 497–503. IEEE, 2020.

[22] X. You, Y. Zhang, X. Chen, X. Liu, Z. Wang, H. Jiang, and X. Chen. Model-free control for soft manipulators based on reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2909–2915. IEEE, 2017.

[23] G. Zheng, Y. Zhou, and M. Ju. Robust control of a silicone soft robot using neural networks. *ISA transactions*, 100:38–45, 2020.

[24] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan. Modeling and control of soft robots using the koopman operator and model predictive control. *arXiv preprint arXiv:1902.02827*, 2019.

[25] D. Terzopoulos and A. Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, 8(6):41–51, 1988.

[26] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 215–222, 1989.

[27] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 109–116, 2007.

[28] W. Lu, N. Jin, and R. Fedkiw. Two-way coupling of fluids to reduced deformable bodies. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 67–76, 2016.

[29] C. G. Speziale. A review of material frame-indifference in mechanics. 1998.

[30] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[31] E. Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[32] Y. Lu, A. Zhong, Q. Li, and B. Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285, 2018.

[33] 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.

[34] 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.