Neural Network Verification for Gliding Drone Control: A Case Study

Colin Kessler[1,2] ✉, Ekaterina Komendantskaya[1], Marco Casadio[1], Ignazio Maria Viola[2], Thomas Flinkow[3], Albaraa Ammar Othman[1], Alistair Malhotra[1], and Robbie McPherson[1]

[1] Heriot-Watt University and Edinburgh Centre for Robotics, UK
ck2049@hw.ac.uk
[2] School of Engineering, University of Edinburgh, UK
[3] Maynooth University, Maynooth, Ireland

Abstract. As machine learning is increasingly deployed in autonomous systems, verification of neural network controllers is becoming an active research domain. Existing tools and annual verification competitions suggest that soon this technology will become effective for real-world applications. Our application comes from the emerging field of microflyers that are passively transported by the wind, which may have various uses in weather or pollution monitoring. Specifically, we investigate centimetre-scale bio-inspired gliding drones that resemble *Alsomitra macrocarpa* diaspores. In this paper, we propose a new case study on verifying *Alsomitra*-inspired drones with neural network controllers, with the aim of adhering closely to a target trajectory. We show that our system differs substantially from existing VNN and ARCH competition benchmarks, and show that a combination of tools holds promise for verifying such systems in the future, if certain shortcomings can be overcome. We propose a novel method for robust training of regression networks, and investigate formalisations of this case study in Vehicle and CORA. Our verification results suggest that the investigated training methods do improve performance and robustness of neural network controllers in this application, but are limited in scope and usefulness. This is due to systematic limitations of both Vehicle and CORA, and the complexity of our system reducing the scale of reachability, which we investigate in detail. If these limitations can be overcome, it will enable engineers to develop safe and robust technologies that improve people's lives and reduce our impact on the environment.

Keywords: Neural Network Control · Bioinspired Robots · Verification of Cyber-Physical Systems · Machine Learning.

1 Introduction

A recent research trend in drone design concerns the development of gliding microdrones, which could serve a function as airborne sensors and remain aloft for

extended periods of time [15, 17, 20, 34]. Current research focuses on the aerodynamics of seeds that have exceptional wind dispersal mechanisms: for example, *Taraxacum* (dandelion) [9] and *Alsomitra* (Javan cucumber). This case study focuses specifically on *Alsomitra*-inspired drones (Figs. 1,4) as the aerodynamics underlying the flight of this diaspore is unique in the plant kingdom, enhancing the dispersal mechanism provided by the wind by an efficient gliding flight. This allows one of the heaviest seeds (314 mg) [4] to reach a similar descent velocity as some of the lightest seeds such as the dandelion (0.6 mg) [9]. Because of this unique feature, several authors have considered this diaspore as a bioinspiration for microdrones [25, 34]. Such drones could function as distributed sensors in the atmosphere, for weather monitoring or detecting pollutants [15, 17, 20, 25, 34]. This could be particularly useful for environmental monitoring and meteorology, with research and regulations moving towards incorporating drone observations to improve weather predictions [11, 35]. It has been demonstrated that such systems are capable of sustained flight with active control and internal electronics [17], although more work on effective actuation and control methods is needed in the future.

Neural networks (NNs) have been widely investigated for drone control, for both quadcopters [3, 24] and fixed-wing designs [30, 32]. The control of small passive gliders is a relatively unexplored field, with the most relevant works involving larger aircraft [1, 32] or without continuous control [17]. For our application, we consider NN control since it has been shown to achieve accurate and robust control for systems with uncertain dynamics [16], it is particularly applicable to controlling swarms [29], and improvements to low-order aerodynamic modelling [23] facilitates easier simulations of such drones. This approach could facilitate particularly lightweight and low-cost drones; such as with analogue network circuits printed on flexible substrates acting as the body of the gliding drone [27, 31]. One could alternatively consider uncontrolled flying sensors [15, 20, 34] or traditional approaches such as state-space or model predictive control. However, one should consider that such systems will need to be verifiably safe with regard to people, other air users, and the environment [35]. These drones could collide with each other, veer into unsafe airspace, fall into an endangered ecosystem, or otherwise cause harm. The utility of uncontrolled flyers would be hampered by such issues, unless they can be made biodegradable. Traditional control methods may be applied, but NN methods have advantages in that they can be made data-driven and adaptive, and printed NN circuits could lead to lighter designs than digital microcontrollers.

1.1 Contributions

Our first aim is the introduction of a novel case study (outlined in Sect. 4) in the verification of *Alsomitra*-inspired drone controllers (our modelling methods are explained in Sect. 3), that differs significantly from existing benchmarks. Unlike VNN-COMP benchmarks such as ACAS Xu, our study involves regression control and continuous dynamic equations. Compared to ARCH-COMP benchmarks such as QUAD, our system involves differential equations that are far

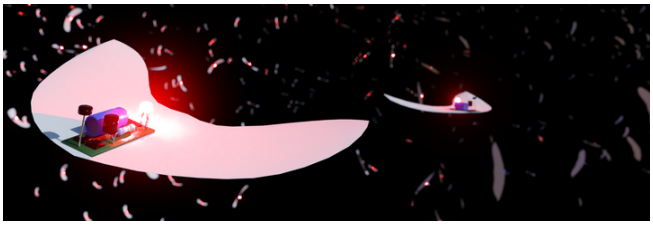

Fig. 1. An artist's impression of a swarm of gliding drones inspired by *Alsomitra* seeds [7].

more complex in terms of the number of non-linear terms. Moreover, unlike the majority of ARCH-COMP cases, this problem does not have as natural a notion of the start, goal, safe, and unsafe states; and thus requires an out-of-the-box approach to property specification.

We propose our ideal formalisation of the problem in Sect. 4.1, and distil the formalisation down to properties that can be handled with available tools (Marabou [19] implemented with Vehicle [10], and CORA [2]) in Sects. 6 and 7. The choice is motivated by the fact that each can be seen as a representative of a set of tools that come from the research communities of VNN-COMP [5] and ARCH-COMP [14], respectively. We present a new implementation of adversarial training for Lipschitz robustness applied to regression training for our controllers in Sect. 5, and present the results of verifying those properties with our robust networks in Sects. 6.3 and 7.2.

Our second aim is to present the lessons learnt from investigating this case study, to help inform the development of relevant tools for similar real-life cyber-physical projects in the future (Sect. 8). The main lesson learnt is that no single existing tool ticks all the desirable boxes. Moreover, each individual tool we chose would benefit from further development in several aspects that are crucial for real-life models. Concretely:

- On the Vehicle side, the verification properties that arise in the presented study are more complex than the usual VNN-COMP benchmarks in at least three ways.
  - Firstly, the constraints on the input vector are more complex: instead of constraining individual vector elements by constants (as e.g. in $a \leq x_i \leq b$), as is the case in the majority of benchmarks including ACAS XU [18], the constraints establish relation between different vector elements, as e.g. in $x_i \leq cx_j$. This changes mathematical interpretation of the verification problem: it no longer boils down to defining a hyper-rectangle (or other constant shape) on the input space and propagating it through the network layers, but gives a more general case of linear programming that works on arbitrary input space constraints. Not every VNN-COMP [5] verifier will be able to deal with such verification properties: Marabou is one of the most general tools in this family of tools and this case study suggests this generality may play a bigger role in the future.

- Secondly, for verification of Lipschitz robustness, we implemented *relational properties*, i.e. properties that compare different outputs of a neural network. These properties are not natively supported by Marabou or Vehicle yet, and required some additional plumbing. On-going implementation of support for relational verification in Marabou will be useful for cases such as this.
- Finally, some novelty of our verification approach is derived from the fact that, unlike most benchmarks in VNN-COMP, our models are regression models, rather than verification models. Some of the methods for training and verification are specialised to classification tasks only, and we predict that this has to change with occurrence of new engineering-inspired benchmarks.

– On the CORA side, the system outlined in this study required several workarounds in order to compute reachability:

  - The complexity of our system of equations [23] far exceeds that of all ARCH-COMP [14] benchmarks, in the number of non-linear terms. This would cause the Jacobian and Hessian matrices to far exceed the maximum number of terms supported by MATLAB , and fail to run. The equations were simplified (Sect. 3.2) by constraining the pitch angle and using an angle-of-attack definition, solving the complexity issue, but (for any reasonably large initial set) the reachable set still tended to expand exponentially after relatively few timesteps. This was solved by dividing the initial set into smaller subsets, computing reachable sets for each, and combining the results.
  - CORA expects a NN controller that takes the system variables as inputs, with relatively few layer types supported [2]. Certain parameters occupy wider ranges than others (for example, $\theta \in [-0.93, -0.07]$, $x \in [0.48, 41.7]$) but unlike Vehicle, input normalisation (keeping all inputs between 0 and 1) is not supported. This is problematic since we intend to observe the effect of adversarial training, for which the input ranges should to be normalised, such that Projected Gradient Descent (PGD) attacks occur in ϵ ranges that are not imbalanced between input dimensions. A workaround was found by training an adversarial network on normalised data, then implementing normalisation layers to the start and end of the network.

In summary, our study aims to provide insights into the challenges and successes in applying neural network verification to *Alsomitra*-inspired drones for environmental monitoring, with a focus on safety and robustness evaluation of controllers. Although this study considers only one type of gliding drones, most of the paper's conclusions will be common between their different modifications, such as e.g. dandelion-inspired drones, and the lessons learnt can be broadly applied to other continuous control tasks. All relevant files are publicly available here.

2 Background

2.1 Neural Network Control

For our control method, we will use the common closed-loop negative feedback method, an overview of which can be seen in Fig. 2. In simple terms, the controller in a drone is given information about its current state (such as position, relative to some desired state) as input, and outputs a command to an actuator which affects how the drone flies. The controller can be considered as an equation linking the system states to an actuation force that changes the states over time according to the system dynamics, where the controller design affects how the drone behaves. If a traditional control theory approach is difficult (such as if the dynamics are highly complex) or a data-driven approach is desirable (if collecting data is easier than modelling the system, or if adaption based on new data is required), an engineer might consider implementing a NN controller.

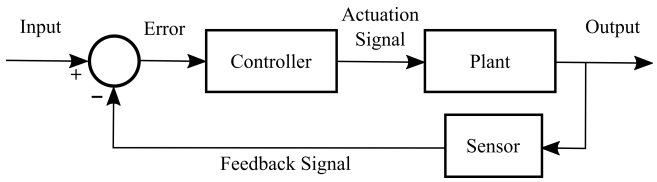

Fig. 2. Overview of a negative feedback control system. For each control iteration, an error signal is calculated by subtracting the current system state (feedback) from the desired system state (input). A controller computes an actuation based on this error, which is applied to a simulated or real system (plant), resulting in some new output state.

2.2 Verification Tools

The case study will rely on the following three groups of neural network verification (NNV) tools. The first group concerns verification of infinite time-horizon properties of controllers in isolation from verification of the overall system dynamics. The most famous benchmark in the domain is ACASXu, and the representative verifier is Marabou [19]; other tools, such as ERAN [26], Pyrat [22] or $\alpha\beta$-CROWN [36] could be interchangeably used for the verification tasks in which Marabou is deployed in this paper; we refer the reader to VNN-Comp [5] for an in depth discussion of existing tools in this category. In addition, we use Vehicle [10], a higher-level interface on top of Marabou, and take advantage of its facility in bridging the *embedding gap* [8] between the physical domains and vector representation of data.

The second group of methods considers the neural controller together with the overall system dynamics to ensure that the entire system avoids unsafe states, see Fig. 3. This class of problems is also known under the umbrella term reachability verification and representative examples include e.g. POLAR-Express [33],

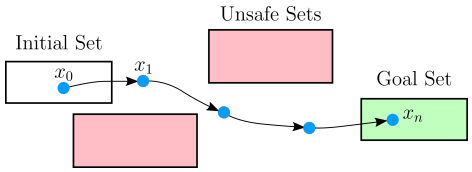

Fig. 3. General form of reachability specifications - dots represent the system at successive control time steps, and arrows represent the continuous trajectory of the system. Any trajectory starting in the initial set should never intersect an unsafe set, and always finish in the goal set.

and CORA [2], see [24] for an exhaustive overview of the mainstream tools in this category. Representative benchmarks include simple dynamic problems such as the inverted pendulum, and more complex problems such as the quadcopter, space docking, and 2-wheeled obstacle avoidance. Each benchmark has a predetermined set of dynamic equations and a NN controller, with a mix of supervised learning (through behaviour-cloning) and reinforcement learning. The limitations of these benchmarks are in the complexity of the networks (large networks require reduction methods), complexity of the equations (systems are either linear, or relatively simple non-linear differential equations), and verification of complex properties (no tools can successfully verify the Spacecraft Docking benchmark as of the most recent results [14]).

Finally, an important group of methods for practical NNV cases comes from machine learning domain, under the umbrella term of *property-driven training (PDT)*. These methods allow to optimise a given neural network for satisfying a desired verification property, with a view of improving the verification success [6, 12, 13]. Although methods in this group vary, they usually deploy a form of training with PGD [21]. PGD methods involve finding the worst-case perturbed example in a region around a data point, which can then be implemented as a loss function during training:

$$\min_{\theta} \; E_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\Delta} \; \mathcal{L}(f_\theta(x+\delta), y) \right]$$

where θ represents the parameters of the NN; $(x, y) \sim \mathcal{D}$ are input-label pairs sampled from the data distribution $\mathcal{D}$; E is the expected value, averaging the loss over all samples in the data distribution $\mathcal{D}$; $\delta \in \Delta$ is the adversarial perturbation constrained within a feasible set Δ (e.g., $\|\delta\|_p \leq \epsilon$) and $\mathcal{L}$ is the loss function (e.g., RMSE, MAE) measuring the discrepancy between the predicted output $f_\theta(x + \delta)$ and the true label y.

The inner maximisation, which identifies the *worst-case* adversarial perturbation $\delta \in \Delta$, is performed using PGD that iteratively adjusts δ by ascending the gradient of the loss function with respect to the input, followed by projection back onto the feasible set Δ (e.g., ensuring $\|\delta\|_p \leq \epsilon$).

The outer minimisation, aimed at optimising the neural network parameters θ to minimise the adversarial loss, is achieved using gradient descent.

3 Modelling Methodology

3.1 *Alsomitra macrocarpa*

A dynamics model (Fig. 4) was derived from [23] resulting in a system of equations for falling plates with displaced centre of mass (CoM), as defined in Sect. 3.2. Based on experimental measurements, our model accurately describes the falling trajectories of *Alsomitra* seeds by inferring aerodynamic forces from the angle of attack [23]. The flight characteristics are highly dependent on the CoM displacement (e_x, Fig. 4), providing us with a convenient actuation method for an *Alsomitra*-inspired drone.

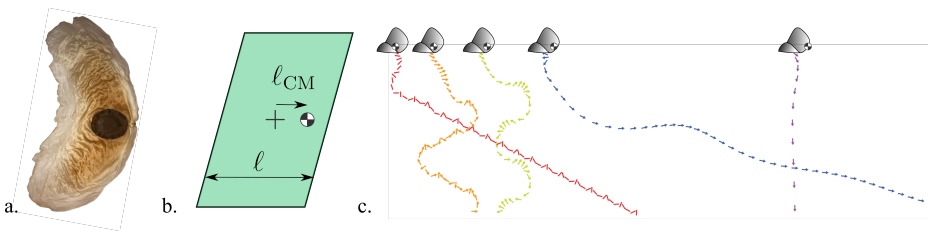

Fig. 4. (a) An *Alsomitra* seed [7]. (b) A two-dimensional approximation of an *Alsomitra* seed, with centre of mass (CoM) displaced by ℓ_{CM} (nondimensional form $e_x = \ell_{\mathrm{CM}}/\ell$). (c) Effect of various e_x on gliding trajectories; according to a quasi-steady aerodynamic model ([23], Sect. 3.2). As the CoM is displaced the trajectory behaviour is affected significantly.

3.2 Equations

The following equations describe falling plates with a displaced centre of mass [23], with six system variables ($x_{1\ldots6}$, Equations 11 ... 16), involving mechanical and aerodynamic constants chosen to match that of *Alsomitra* seeds [7]. The main

Constant(s)	**Definition(s)**	**Value(s)**
ℓ, m, I	Plate length [m], mass [kg], inertia [kgm^2]	0.07, 3.175e-04, 0.0796
ρ_f, g	Fluid Density [kg/m^3], gravity [m/s^2]	1.225, 9.81
α_0, δ	Critical α at stall, transition smoothness [°]	14, 6
C_L^1, C_L^2	Lift coefficient components	5.1822, 0.80751
$C_D^0, C_D^1, C_{\mathrm{D}}^{\pi/2}$	Drag Coefficient components	0.10598, 4.9368, 1.4996
$C_{\mathrm{CP}}^0, C_{\mathrm{CP}}^1, C_{\mathrm{CP}}^2$	CoP components	0.2386, 2.8529, 0.3689
C_{R}	Rotational lift	1.73

Table 1. *Alsomitra* model constants defined according to [7, 23]

assumptions of the model are that the aerodynamic behaviour of the plate is quasi-steady and independent of previous motion. The three most important aerodynamic force coefficients (lift 3, drag 4, and the location of the center of pressure 5) are defined as functions of the instantaneous angle of attack 1. This approximation has been found to capture the interesting aspects of *Alsomitra* flight (especially the effects of the CoM position on flight modes), without requiring expensive numerical simulations of the flow field around the seed. A more detailed overview can be seen in Appendix. 9.1

$$\tan\alpha = (x_2 - x_3 y_1 \ell)/x_1 \approx x_2/x_1 \tag{1}$$

$$f = (1 - \tanh((\alpha - \alpha_0)/\delta))/2 \tag{2}$$

$$-C_{\mathrm{L}} = f(|\alpha|)C_{\mathrm{L}}^1 \sin(|\alpha|) + (1 - f(|\alpha|))C_{\mathrm{L}}^2 \sin(2\,|\alpha|) \tag{3}$$

$$C_{\mathrm{D}} = f(|\alpha|)(C_{\mathrm{D}}^0 + C_{\mathrm{D}}^1 \sin^2(|\alpha|)) + (1 - f(|\alpha|))C_{\mathrm{D}}^{\pi/2} \sin^2(|\alpha|) \tag{4}$$

$$\ell_{\mathrm{CP}}/\ell = f(|\alpha|)(C_{\mathrm{CP}}^0 - C_{\mathrm{CP}}^1 \alpha^2) + C_{\mathrm{CP}}^2[1 - f(|\alpha|)](1 - |\alpha|\,/(\pi/2)) \tag{5}$$

$$L_{\mathrm{T}} = \frac{1}{2}\rho_f \ell C_{\mathrm{L}} \sqrt{{x_1}^2 + (x_2 - x_3 y_1 \ell)^2}\,(x_2 - x_3 y_1 \ell, x_1) \tag{6}$$

$$L_{\mathrm{R}} = -\frac{1}{2}\rho_f \ell^2 C_{\mathrm{R}} x_3\,(x_2 - x_3 y_1 \ell, x_1) \tag{7}$$

$$D = -\frac{1}{2}\rho_f \ell C_{\mathrm{D}} \sqrt{{x_1}^2 + (x_2 - x_3 y_1 \ell)^2}\,(x_1, x_2 - x_3 y_1 \ell) \tag{8}$$

$$\tau_{\mathrm{T}} = -\frac{1}{2}\rho_f \ell \sqrt{{x_1}^2 + (x_2 - x_3 y_1 \ell)}\,[C_{\mathrm{L}} x_1 + C_{\mathrm{D}}(x_2 - x_3 y_1 \ell)]\,(\ell_{\mathrm{CP}} - \ell_{\mathrm{CM}}) \tag{9}$$

$$\tau_{\mathrm{R}} = -\frac{1}{128}\rho_f \ell^4 C_{\mathrm{D}}^{\pi/2} x_3\,|x_3| \left[(2y_1 + 1)^4 \pm (2y_1 + 1)^4\right] \tag{10}$$

$$m\dot{x_1} = \left(m + \pi\rho_f \ell^2/4\right) x_3 x_2 - (\pi\rho_f \ell^2/4) x_3^2 \ell_{CM} + L_T^{x'} + L_R^{x'} + D^{x'} - m' g \sin x_4 \tag{11}$$

$$\left(m + \pi\rho_f \ell^2/4\right) \dot{x_2} = -m x_3 x_1 + (\pi\rho_f \ell^2/4)\dot{x_3}\ell_{CM} + L_T^{y'} + L_R^{y'} + D^{y'} - m' g \cos x_4 \tag{12}$$

$$I\dot{x_3} = \tau_T + \tau_R \tag{13}$$

$$\dot{x_4} = x_3 \tag{14}$$

$$\dot{x_5} = x_1 \cos x_4 - x_2 \sin x_4 \tag{15}$$

$$\dot{x_6} = x_1 \sin x_4 + x_2 \cos x_4 \tag{16}$$

4 Verification Task

Our *Alsomitra* model from Sect. 3.2 is used as the basis of a feedback control system with a NN controller, as described in Sect. 2.1. In our case, the plant is the aerodynamic model, and the desired input is a linear reference trajectory in x_5 and x_6 (translational x and y):

$$x_6 = -x_5 \tag{17}$$

The feedback signal consists of the six system states, and the CoM displacement is actuated by a controller aiming to follow the target trajectory (Fig. 5).

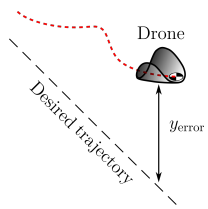

Fig. 5. As a control problem, we consider an *Alsomitra*-inspired microdrone and attempt to follow a linear trajectory in two dimensions.

As per the ARCH-COMP airplane and pendulum benchmarks [14], the neural network controller is trained using behaviour cloning. All simulations ran for a total of 20 s, with a model timestep of 0.01 s and a control timestep of 0.5 s. A PID controller actuates y_1 (e_x) based on an error in x_6, and the gains are tuned manually until the control system performs well for a range of starting x_6 positions. For each controller actuation (24 per simulation, for nine simulations), the system states, x_6 error, and PID actuation are recorded for use as training data. This data is imported to Python for standard regression learning, and networks are exported in .onnx format for evaluation (Fig. 6) and verification. All networks have 6 inputs, 3 hidden layers with 6, 4, and 1 nodes respectively with ReLU activation functions, and 1 output.

4.1 Formalisation

The core of this case study lies in examining the challenges in adopting the existing NNV methods in this new domain. Our ideal formalisation of the problem is as follows. We consider a hybrid program where the six system states $x_1, ..., x_6$ change over continuous time t according the dynamics model shown in Sect. 3.2, and a NN controller acts to change the system state discretely every 0.5 s. For any starting state $x_1, ..., x_6(0)$, after some time t^* the trajectory of the drone will always be within some small distance y^* of the target trajectory (ideally, $x_6 = -x_5$). This boils down to the following ideal verification property:

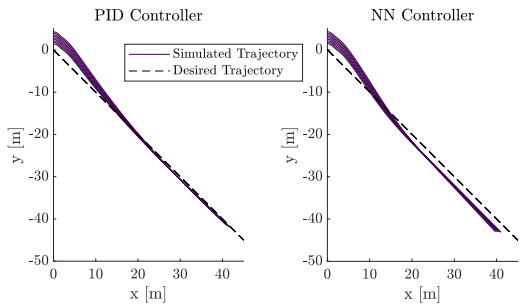

Fig. 6. PID and basic NN controller performance on an *Alsomitra*-inspired drone. The naive network is trained on regression data obtained from simulations with the PID controller, and the resulting performance is similar but not perfect.

$$\forall t \geq t^*, \forall x_1, ..., x_6(0) \in \mathbb{R} : |x_6(t) + x_5(t)| \leq y^* \tag{18}$$

There are several features that distinguishes this system from standard NNV benchmarks, and we aim to explain the technical implications of these challenges for existing verification technologies, and propose ways in which these challenges can be overcome:

1. The system dynamics are continuous, therefore unlike standard control verification benchmarks (such as ACAS Xu [5]), control is modelled as a regression task as oppose to classification.
2. Unlike the ARCH-COMP benchmarks [24] that have a pre-defined notion of safe and unsafe state, these gliding drones do not have a notion of safety in the sense of a pre-defined coordinate region. A safe state is instead defined in a relational way, as adhesion to certain safe trajectory.
3. Unlike many ARCH-COMP benchmarks, our verification task requires modelling with an infinite-time horizon. Each drone could stay airborne for an arbitrary duration of time, depending on the surrounding airflow.
4. As defined by our model, the dynamics of gliding drones are more complex than what is currently handled by the ARCH-COMP benchmarks and tools, and in particular it is more complex than the dynamics handled by tools that can verify infinite-time horizon systems, such as KeyMaeraX [28].

The available verification tools (Sect. 2.2) do not allow us to formalise this idealised goal directly, since CORA does not support infinite time, and Marabou does not support differential equations (and in that sense, it does not directly support analysis of the system evolution). As a result, we simplified this general task as two simpler tasks (in the first case sacrificing the analysis of the overall system dynamics, and in the second case the infinite-time horizon and relational definition of the target state):

1. The NN will never command the drone to deviate significantly from the target trajectory. This task was implemented in Marabou, using the Vehicle specification language since it facilitates complex property definition.
2. Given an interval of initial positions in space and a finite-time horizon, the NN-controlled drone will always reach a goal region, defined as a region around the target trajectory within this finite time frame. This task was verified in CORA.

We note that task 1 resembles a robustness property in some ways [6], except for now we deal with a regression NN and robustness relative to a line rather than a given data point.

5 Robustness Training for Regression

Since robustness is critically important for drone safety, it seems reasonable to attempt a form of adversarial training based on PGD methods for our controller. The guiding hypothesis was that a general improvement in NN robustness should lead to improved verification performance. Since our neural network is a regression model, the classification-based training methods surveyed in Sect. 2.2 could not easily be used without modification. We therefore had to modify the PGD algorithm to use an RMSE loss function instead of cross-entropy, and modify other aspects of the adversarial training algorithm that relied on the presence of discrete classes.

We focused on two notion of robustness, *standard* and *Lipschitz* robustness [6]. Given $x^* \in \mathcal{D}$ and constants $\epsilon, \delta, L \in R$,

$$\forall x \in R^n : \|x - x^*\| \leq \epsilon \implies \|f(x) - f(x^*)\| \leq \delta \tag{19}$$

$$\forall x \in R^n : \|x - x^*\| \leq \epsilon \implies \|f(x) - f(x^*)\| \leq L \|x - x^*\| \tag{20}$$

Since the latter has been proven to be strictly stronger than the former in [6], we implemented a form of PGD training with a Lipschitz loss function. During each training epoch, the algorithm finds the worst-case adversarial example (x^*, $f(x^*)$) in an ϵ-ball around each training point (x, $f(x)$).To optimise the regression model for Lipschitz robustness, we dynamically compute the highest value of L from the training and adversarial points (according to Eq. 20), which is summed to the training RMSE loss, penalising the network for large gradients about each data point. This is expected to result in a network with a smoother and therefore more robust output, at the expense of some accuracy.

6 Vehicle Implementation

Task 1 was broken down into five simpler specifications to be implemented in Vehicle (a detailed introduction to which can be found in [10]), to ensure the NNs control the drone as desired in various ways. Global properties relating to the controller's output relative to the target trajectory are introduced in Sect. 6.1,

and a local robustness property is introduced in Sect. 6.2. Global properties are verified for all inputs bounded by the training data (representing the entire parameter space over which our controller is trained), and the local property is evaluated about ϵ-balls from the training data.

6.1 Global Property Specifications

Our first goal is to ensure the controller never causes the drone to deviate from the target trajectory. To establish a performance criteria, properties 1-4 include y^* (a threshold distance from the target trajectory, Eqs. 17, 18), such that a critical y^* can be found per network per property where verification succeeds. For example, for properties 1 and 2, a lower critical y^* would indicate a controller that better adheres to the target trajectory:

1. *If the drone is above the line by some threshold y^*, the NN output will always make the drone pitch down* (Listing 1)

$$x_6 \geq -x_5 + y^* \Rightarrow f(x) \geq 0.187 \tag{21}$$

2. *If the drone is below the line by some threshold y^*, the NN output will always make the drone pitch up*

$$x_6 \leq -x_5 - y^* \Rightarrow f(x) \leq 0.187 \tag{22}$$

Our third property is reversed, where a larger y^* would indicate better adherence to a larger region around the target trajectory:

3. *If the drone is close to the line by some threshold y^*, and at an intermediate pitch angle, the NN output will always be intermediate* (Listing 2)

$$-x_5 - y^* \leq x_6 \leq -x_5 + y^* \wedge -0.786 \leq x_4 \leq -0.747 \Rightarrow 0.184 \leq f(x) \leq 0.19 \tag{23}$$

Our fourth property is more complex, and represents a desireable behaviour not present in the data:

4. *If the drone is above and close to the line, pitching down quickly and moving fast, the NN output will always make the drone pitch up*

$$-x_5 \leq x_6 \leq -x_5 + y^* \;\; \wedge \;\; x_3 \leq -0.12 \;\; \wedge \;\; x_2 \leq -0.3 \Rightarrow f(x) \leq 0.187 \tag{24}$$

In our Vehicle code, *alsomitra* represent the NN, *validInput* represents the input space bounded by the training data, and the parameter *ystar* is defined during runtime.

```
droneFarAboveLine : UnnormalisedInputVector -> Bool
droneFarAboveLine x =
        x ! d_y >= - x ! d_x + ystar

@property
property1 : Bool
property1 = forall x . validInput x and droneFarAboveLine x =>
  alsomitra x ! e_x >= 0.187
```

Listing 1: Property 1 implemented in Vehicle.

```
intermediatePitch : UnnormalisedInputVector -> Bool
intermediatePitch x =
        -0.786 <= x ! d_theta <= -0.747

closeToLine : UnnormalisedInputVector -> Bool
closeToLine x =
        x ! d_y >= -x ! d_x - ystar and
        x ! d_y <= - x ! d_x + ystar

@property
property3 : Bool
property3 = forall x . validInput x and intermediatePitch x
and closeToLine x => 0.184 <= alsomitra x ! e_x <= 0.19
```

Listing 2: Property 3 implemented in Vehicle.

6.2 Local Robustness Specification

Our fifth property is an evaluation of robustness around ϵ-balls with respect to the training dataset, as defined in Sect. 5. A detailed introduction to ϵ-ball robustness for image classification implemented in Vehicle can be found in [10]. Similarly to properties 1-4, we are interested in finding at what threshold L value (L^*) does each network pass verification. Similarly to properties 1-4, we expect the verification results for this property to depend on the strictness of L, which we consider as the parameter L^*. However, due to Marabou limitations, this was evaluated with respect to the training dataset, where for each network Property 5 was evaluated for each training point, given fixed $L*$ and ϵ values. Additionally, the distance between points was computed with L^∞ and the input distance could not be included in the formula, leading use to use a different robustness definition:

5. *For any given input point x, the network output $f(x^*)$ of any perturbed point x^* within an ϵ-ball around x, will have a distance less than or equal to $L^*\epsilon$ to $f(x)$* (Listing 3)

$$\forall x \in R^n : \|x - x^*\| \leq \epsilon \implies \|f(x) - f(x^*)\| \leq L^*\epsilon \tag{25}$$

This definition is less strict than our definition of Lipschitz robustness (Eq. 20), since it is effectively equivalent to standard robustness (Eq. 19) where $L^*\epsilon = \delta$. This means that any counterexample to Property 5 will also violate Lipschitz robustness where $L = L^*$, but not the other way around. Since we train for the

stronger definition (Sect. 5), we expect to see improved robustness with regard to this weaker definition.

In our Vehicle code, parameters *epsilon* and *Lipschitz* are defined during runtime, and n is inferred from the training data (provided in idx format [10]).

```
myList : List Rat
myList = [0, 1, 2, 3, 4, 5]

boundedByEpsilon : InputVector -> Bool
boundedByEpsilon x = forall i in myList . -epsilon <= x ! i - x ! i + 6 <= epsilon

validPerturbation : InputVector -> Bool
validPerturbation x = forall i in myList . x ! i == 0.0

standardRobustness : InputVector -> OutputVector -> Bool
standardRobustness input output = forall pertubation .
          let perturbedInput = input - pertubation in validPerturbation pertubation and
        validInput perturbedInput and boundedByEpsilon perturbedInput =>
        (output ! e_x - alsomitra perturbedInput ! e_x2) <= Lipschitz * epsilon and
        alsomitra perturbedInput ! e_x2 - output ! e_x <= Lipschitz * epsilon

@property
property4 : Vector Bool n
property4 = foreach i . standardRobustness (trainingInputs ! i) (trainingOutputs ! i)
```

Listing 3: Property 5 implemented in Vehicle. In this case, the states are defined in normalised terms to avoid scaling issues. Instead of calling the network twice to evaluate $f(x)$ and $f(x^*)$, the NN is doubled in onnx format so that two sets of inputs and outputs can be evaluated at once.

6.3 Verification Results

Table 2. Critical y^* values for properties 1-4 (Sect. 6.1), for naive and adversarially trained NNs (Sect. 5). For properties 1, 2, and 4, a lower y^* indicates a controller that better adheres to the target trajectory, and the inverse for Property 3.

Property	Naive	Adversarial
1 (21)	46	30
2 (22)	42	42
3 (22)	Failed	Failed
4 (23)	0	0

Table 3. Verification success rates (%) of Property 5 (Sect. 6.2) for naive and adversarial NNs (Sect. 5), per L^* values and ϵ, with respect to the training dataset. A higher success rate means that the NN is robust with respect to more of the training data points. As ϵ increases we are increasing the radius for perturbation around each training point, and a decreasing L^* results in a stricter maximum gradient threshold. Empty cells represent properties that timed out before verifying 100 data points.

Naive		$L^*\epsilon$			
		10^{-3}	10^{-2}	10^{-1}	10^{0}
ϵ	10^{-5}				100
	10^{-4}			96.1	100
	10^{-3}		17.0	98.5	100
	10^{-2}	-	-	-	100

Adversarial		$L^*\epsilon$			
		10^{-3}	10^{-2}	10^{-1}	10^{0}
ϵ	10^{-5}				100
	10^{-4}			99.7	100
	10^{-3}		23.9	99.7	100
	10^{-2}	0	13.6	92.5	100

These results provide interesting insights from an engineering perspective. From Table 2 there is a clear improvement in performance for Property 1 when implementing adversarial training, suggesting that our approach has been successful. However, Properties 1 and 2 only succeed with very large values of y^* - our controllers only adhere to a region around the target trajectory so wide as to

be useless in real applications. Properties 3 and 4 failed and succeeded, for all y^* values, for all the tested networks, suggesting that they are particularly difficult and easy to verify respectively. Table 3 shows a marginal improvement in robustness performance for our adversarial network, suggesting that our approach has been moderately successful.

7 CORA Implementation

7.1 Reachability Specification

Since our case study is based on the QUAD benchmark from the ARCH competition [24], our reachability specification is defined similarly. The initial set is:

$$x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0, x_6 \in [1.43, 4.29] \tag{26}$$

The reachability goal is for the drone to always be within a distance $y^* = 2$ of the target trajectory after 20 s. To compute reachability, CORA uses set representations (such as zonotopes [2]) and set operations to over-approximate the continuous time reachable set in discrete time steps. Additional considerations include the initial set representation, time step size, controller type, and reachability algorithm - in our case we use a zonotope, 0.01 s, a NN controller, and conservative linearization.

7.2 Reachability Results

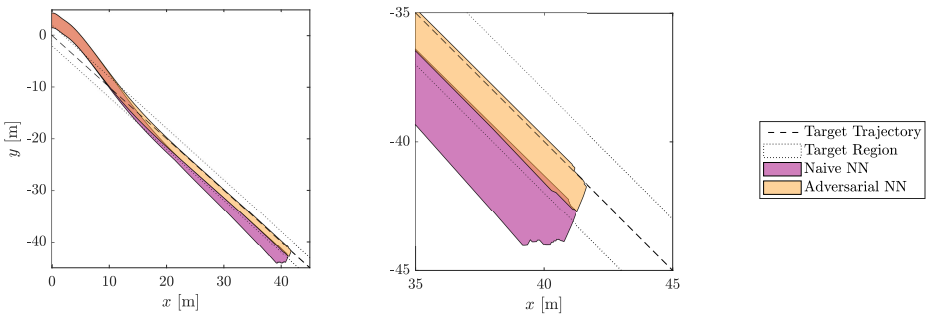

Fig. 7. Reachable regions in x_5 and x_6 for naive and adversarial NNs implemented as controllers, from the initial set defined in Eq. 26. The naive NN fails to reach a region bounded by $y^* = 2$ after 20 s, and the adversarial NN succeeds.

This result shows significant improvement from the adversarial network compared to the naive (these networks are identical to those from Tables 2 and 3), with the adversarial network adhering much closer to the target trajectory. This would suggest that our adversarial training has been successful, but we note that whilst these networks are identical in terms of structure, training data, and training epochs, the discrepancy in this plot could be partly due to differences in regression performance.

8 Discussion

8.1 Limitations and Lessons Learned

Use cases such as ours requires multiple tools to verify, each with benefits and drawbacks. Although standard methods apply, modifications are needed for proper implementation, such as with robustness training (Sect. 5). Vehicle does not have integration with Python, and our dynamics model and CORA are in Matlab, requiring the use of three programming languages for our case study. Marabou does not support multiple network calls, so a workaround involving doubling of the NN (in .onnx format) was required to evaluate robustness in Vehicle (Listing. 3). Additionally, for our local robustness property, certain Marabou queries timed out after after a few data points (Table 3). Correct handling of normalisation was found to be very important since the implemented robustness training and verification methods require normalisation, yet CORA does not support normalisation. To evaluate reachability in CORA with normalised networks required another workaround, incorporating the normalisation arithmetic as extra layers in the network. Furthermore, CORA could not evaluate reachability for the full system of equations described in Sect. 3.2, due to their complexity and the size of the starting set.

Implementing CORA for our application was difficult due to set explosions, where the complexity of the reachability computation would cause an exponential expansion in the set, causing a crash. To reduce the complexity of the equations, we modified the angle of attack definition (1). To solve the initial set size issue, the initial set was divided in x_6, and the resulting subsets combined to a final reachable set. The reachability timesteps still needed to be small (0.01 s) to avoid set explosions; resulting in long computation times for full reachable sets (over 8 hours in some cases). All reachable sets only involve a starting interval in x_6 (starting values are constants in every other dimension), since increasing the starting interval size in multiple dimensions would cause set explosions. These limitations in CORA were found to stem from the complexity of the partial derivative matrices (Jacobian and Hessian), especially due to a large number of nonlinear terms. CORA could be improved significantly by faster approximations of these derivatives, such as with finite differences instead of symbolic methods, but performing such calculations over sets was found to be non-trivial. ReLU activation functions were also found to reduce computation times compared to sigmoid.

Another consideration for future work is the trade-off between robustness and regression performance for such a controller. For our verification implementations we assume that each NN is capable of controlling the drone relatively well, but that may not always have been the case. A better comparison could be made using NNs with equivalent regression performance on a test set of data, using a coefficient such as R^2.

8.2 Conclusion

In this paper, we introduce a novel case study in the verification of gliding drones (Sect. 4), and evaluate the utility and challenges involved in the tools used. We developed a two-dimensional model for *Alsomitra*-inspired drones in Sect. 3, presented an ideal verification formalisation in Sect. 4.1, reduced the formalisation into properties manageable by current tools (Vehicle and CORA, Sects. 6 and 7), and presented verification results in Sects. 6.3 and 7.2. We noted challenges that this class of problems presents, among which are its more complex dynamics, under-defined notion of safe and unsafe states, preference for infinite-time horizon guarantees, and its different learning-as-regression regimes. We have shown that in principle, a combination of existing verification tools (Sect. 2.2) and novel training methods (Sects. 5 could be effectively adopted in order to enable future inclusion of this class of benchmarks into NNV portfolios. We note, however, that verification tasks like this motivate strongly development of tools that cross the boundaries of ARCH-COMP and VNN-COMP on the one hand, and incorporate these tools more smoothly with machine learning toolboxes on the other hand; cohering perhaps with the general agenda of building more complex programming language interfaces for such more complex verification tasks [8]. Technical problems such as insufficient support for normalisation (as reported here) can make a difference between verification success and failure, yet they are often over-looked in papers and tools that are dedicated to implementing NNV algorithms. Whilst we can define and verify for the behaviours that we want to a certain extent, the state of tools makes it difficult to say exactly how well NNs will fulfil their role - even for our relatively simple (two-dimensional, non-turbulent) drone system. For example, Table 2 suggests that our NNs are only guaranteed to adhere to a very large region around the line, and Fig. 7 shows reachable regions but for which the initial set is relatively small (zero-width in 5 dimensions). All of these issues will likely be found on any comparable regression task with complex dynamic equations. If these limitations can be overcome, it will help enable engineers to develop safe and robust control and modelling methods for technologies that improve people's lives and reduce our impact on the environment.

References

1. Abouheaf, M., Mailhot, N., Gueaieb, W.: An online reinforcement learning wing-tracking mechanism for flexible wing aircraft. In: 2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE). pp. 1–7 (2019). https://doi.org/10.1109/ROSE.2019.8790425
2. Althoff, M., Kochdumper, N., Ladner, T., Wetzlinger, M.: Manual v2025 (2024), https://tumcps.github.io/CORA/data/archive/manual/Cora2025Manual.pdf
3. Amer, K., Samy, M., Shaker, M., ElHelw, M.: Deep convolutional neural network based autonomous drone navigation. In: Proceedings of the Thirteenth International Conference on Machine Vision. vol. 11605, p. 1160503 (2021). https://doi.org/10.1117/12.2587105

4. Azuma, A., Okuno, Y.: Flight of a samara, alsomitra macrocarpa. Journal of Theoretical Biology **129**(3), 263–274 (1987). https://doi.org/https://doi.org/10.1016/S0022-5193(87)80001-2
5. Brix, C., Bak, S., Johnson, T.T., Wu, H.: The 5th international verification of neural networks competition (vnn-comp 2024): Summary and results (2024), https://www.arxiv.org/pdf/2412.19985
6. Casadio, M., Komendantskaya, E., Daggitt, M.L., Kokke, W., Katz, G., Amir, G., Refaeli, I.: Neural network robustness as a verification property: A principled case study. Computer Aided Verification pp. 219–231 (2022)
7. Certini, D.: The flight of Alsomitra macrocarpa. Phd thesis, University of Edinburgh (February 2023)
8. Cordeiro, L.C., Daggitt, M.L., Girard-Satabin, J., Isac, O., Johnson, T.T., Katz, G., Komendantskaya, E., Lemesle, A., Manino, E., Šinkarovs, A., Wu, H.: Neural network verification is a programming language challenge (2025), https://arxiv.org/abs/2501.05867
9. Cummins, C., Seale, M., Macente, A., Certini, D., Mastropaolo, E., Viola, I.M., Nakayama, N.: A separated vortex ring underlies the flight of the dandelion. Nature **562**, 414–418 (2018), https://doi.org/10.1038/s41586-018-0604-2
10. Daggitt, M., Kokke, W., Komendantskaya, E., Atkey, B., Arnaboldi, L., Slusarz, N., Casadio, M., Coke, B., Lee, J.: A vehicle tutorial (2024), https://vehicle-lang.github.io/tutorial/
11. ERC: A dandelion-inspired drone for swarm sensing. https://cordis.europa.eu/project/id/101001499
12. Fischer, M., Balunović, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., Vechev, M.: Dl2: Training and querying neural networks with logic. In: International Conference on Machine Learning (2019)
13. Flinkow, T., Pearlmutter, B.A., Monahan, R.: Comparing differentiable logics for learning with logical constraints. Science of Computer Programming **244**, 103280 (Sep 2025). https://doi.org/10.1016/j.scico.2025.103280
14. Frehse, G., Althoff, M.: Arch-comp24: Volume information proceedings of the 11th int. workshop on applied verification for continuous and hybrid systems. EPiC Series in Computing **103** (2024), https://easychair.org/publications/volume/ARCH-COMP24
15. Iyer, V., Gaensbauer, H., Daniel, T.L., Gollakota, S.: Wind dispersal of battery-free wireless devices. Nature **603**, 427–433 (2022)
16. J. Li, Q. Yang, B.F.Y.S.: Robust state/output-feedback control of coaxial-rotor mavs based on adaptive nn approach (2019), https://ieeexplore.ieee.org/document/8715436
17. Johnson, K., Arroyos, V., Ferran, A., Villanueva, R., Yin, D., Elberier, T., Aliseda, A., Fuller, S., Iyer, V., Gollakota, S.: Solar-powered shape-changing origami microfliers. Science Robotics **8**(82) (2023), https://www.science.org/doi/abs/10.1126/scirobotics.adg4276
18. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: An efficient smt solver for verifying deep neural networks (2017), https://arxiv.org/abs/1702.01135
19. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The marabou framework for verification and analysis of deep neural networks pp. 443–452 (2019)

20. Kim, B.H., Li, K., Kim, J.T., Park, Y., Jang, H., Wang, X., Xie, Z., Won, S.M., Yoon, H.J., Lee, G., Jang, W.J., Lee, K.H., Chung, T.S., Jung, Y.H., Heo, S.Y., Lee, Y., Kim, J., Cai, T., Kim, Y., Prasopsukh, P., Yu, Y., Yu, X., Avila, R., Luan, H., Song, H., Zhu, F., Zhao, Y., Chen, L., Han, S.H., Kim, J., Oh, S.J., Lee, H., Lee, C.H., Huang, Y., Chamorro, L.P., Zhang, Y., Rogers, J.A.: Three-dimensional electronic microfliers inspired by wind-dispersed seeds. Nature (2021), https://doi.org/10.1038/s41586-021-03847-y
21. Kolter, Z., Madry, A.: Adversarial robustness—theory and practice. NeurIPS 2018 tutorial (2018), available at https://adversarial-ml-tutorial.org/
22. Lemesle, A., Lehmann, J., Le Gall, T.: Neural network verification with pyrat (2024), https://arxiv.org/abs/2410.23903
23. Li, H., Goodwill, T., Jane Wang, Z., Ristroph, L.: Centre of mass location, flight modes, stability and dynamic modelling of gliders. Journal of Fluid Mechanics **937**, A6 (2022). https://doi.org/10.1017/jfm.2022.89
24. Lopez, D.M., Althoff, M., Benet, L., Blab, C., Forets, M., Jia, Y., Johnson, T.T., Kranzl, M., Ladner, T., Linauer, L., Neubauer, P., Neubauer, S.A., Schilling, C., Zhang, H., Zhong, X.: Arch-comp24 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. EPiC Series in Computing **103**, 64–121 (2024), https://easychair.org/publications/paper/WsgX
25. Lumini, M.: Pherodrone1.0: An innovative inflatable uav's concept, inspired by zanonia macrocarpa's samara flying-wing and to insect's sensillae, designed for the biological control of harmful insects in pa (precision agriculture). Bionics and Sustainable Design (2022)
26. Müller, M.N., Singh, G., Balunovic, M., Makarchuk, G., Ruoss, A., Serre, F., Baader, M., D Cohen, D., Gehr, T., Hoffmann, A., Maurer, J., Mirman, M., Müller, C., Püschel, M., Tsankov, P., Vechev, M.: Eran (2025), https://github.com/eth-sri/eran
27. Oshima, K., Kuribara, K., Sato, T.: Flex-snn: Spiking neural network on flexible substrate. IEEE Sensors Letters **7**(5), 1–4 (2023)
28. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63588-0
29. Qamar, S., Khan, S.H., Arshad, M.A., Qamar, M., Gwak, J., Khan, A.: Autonomous drone swarm navigation and multitarget tracking with island policy-based optimization framework. IEEE Access **10**, 91073–91091 (2022). https://doi.org/10.1109/ACCESS.2022.3202208
30. Richter, D.J., Calix, R.A., Kim, K.: A review of reinforcement learning for fixed-wing aircraft control tasks. IEEE Access **12**, 103026–103048 (2024). https://doi.org/10.1109/ACCESS.2024.3433540
31. Singaraju, S.A., Weller, D.D., Gspann, T.S., Aghassi-Hagmann, J., Tahoori, M.B.: Artificial neurons on flexible substrates: A fully printed approach for neuromorphic sensing. Sensors **22** (2022)
32. Wada, D., Araujo-Estrada, S.A., Windsor, S.: Unmanned aerial vehicle pitch control using deep reinforcement learning with discrete actions in wind tunnel test. Aerospace **8**, 18 (2021). https://doi.org/10.3390/aerospace8010018
33. Wang, Y., Zhou, W., Fan, J., Wang, Z., Li, J., Chen, X., Huang, C., Li, W., Zhu, Q.: Polar-express: Efficient and precise formal reachability analysis of neural-network controlled systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **43**(3), 994–1007 (2024). https://doi.org/10.1109/TCAD.2023.3331215

34. Wiesemüller, F., Meng, Z., Hu, Y., Farinha, A., Govdeli, Y., Nguyen, P.H., Nyström, G., Kovač, M.: Transient bio-inspired gliders with embodied humidity responsive actuators for environmental sensing. Frontiers in Robotics and AI (2023)
35. WMO: Global Demonstration Campaign for Evaluating the Use of Uncrewed Aircraft Systems in Operational Meteorology: White Paper. Tech. rep. (2023)
36. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. Advances in Neural Information Processing Systems **31**, 4939–4948 (2018), `https://arxiv.org/pdf/1811.00866.pdf`

9 Appendix

9.1 Equations

Table 4. 2D quasi-steady equations for falling plates with a displaced centre of mass [23], with six system variables $(x, y, \theta, v_{x'}, v_{y'}, \omega)$. When integrated over time, the force coefficients are derived from the angle of attack (5-7), the forces are calculated (8-12), followed by the equations of motion (13-18). For simplicity the plate is assumed to be infinitesimally thin, and to always have an angle of attack within the region $\alpha \in [-\pi/2, 0]$.

Constant(s)	**Definition(s)**	**Value(s)**
ℓ, m	Plate length [m] and mass [kg]	0.07, 3.175e-04
ρ_f	Fluid Density [kg/m^3]	1.225
α_0, δ	Critical α at stall, stall transition smoothness [°]	14, 6
$C_L^1, C_L^2, C_D^0, C_D^1, C_D^{\pi/2}, C_{CP}^0, C_{CP}^1, C_{CP}^2, C_R$	System-specific aerodynamic coefficients	0.23857, 2.8529, 0.36893, 5.1822, 0.80751, 0.10598, 4.9368, 1.4996, 1.73
a, b	Elliptical semi axes [m]	0.03375, 5e-04

No.	**Variable**	**PDE expression**
1	CoM displacement, ℓ_{CM}/ℓ or e_x	Defined by controller, in the range $[0.181, 0.193]$
2	Moment of Inertia, I [kgm^2]	$I = (m(a^2 + b^2)/(\rho_f \ell^4)) + 1/32 + (\ell_{\mathrm{CM}}/\ell)^2$
3	Angle of attack, α (p15)	$\tan\alpha = (v_{y'} - \omega\ell_{\mathrm{CM}})/v_{x'} \approx v_{y'}/v_{x'}$
4	Selection function, $f(\alpha)$ (5.2)	$f = (1 - \tanh((\alpha - \alpha_0)/\delta))/2$
5	Lift coefficient, $C_{\mathrm{L}}(\alpha)$ (5.1, 5.3)	$-C_{\mathrm{L}} = f(\lvert\alpha\rvert)C_{\mathrm{L}}^1 \sin(\lvert\alpha\rvert) + (1 - f(\lvert\alpha\rvert))C_{\mathrm{L}}^2 \sin(2\lvert\alpha\rvert)$
6	Drag coefficient, $C_{\mathrm{D}}(\alpha)$ (5.4, 5.5)	$C_{\mathrm{D}} = f(\lvert\alpha\rvert)(C_{\mathrm{D}}^0 + C_{\mathrm{D}}^1 \sin^2(\lvert\alpha\rvert)) + (1 - f(\lvert\alpha\rvert))C_{\mathrm{D}}^{\pi/2} \sin^2(\lvert\alpha\rvert)$
7	Center of pressure, $\ell_{\mathrm{CP}}(\alpha)$ (5.6, 5.7)	$\ell_{\mathrm{CP}}/\ell = f(\lvert\alpha\rvert)(C_{\mathrm{CP}}^0 - C_{\mathrm{CP}}^1\alpha^2) + C_{\mathrm{CP}}^2[1 - f(\lvert\alpha\rvert)](1 - \lvert\alpha\rvert/(\pi/2))$
8	Translational lift force, L_{T} (4.10)	$L_{\mathrm{T}} = \frac{1}{2}\rho_f \ell C_{\mathrm{L}} \sqrt{{v_{x'}}^2 + (v_{y'} - \omega\ell_{\mathrm{CM}})^2}\,(v_{y'} - \omega\ell_{\mathrm{CM}}, v_{x'})$
9	Rotational lift force, L_{R} (4.11)	$L_{\mathrm{R}} = -\frac{1}{2}\rho_f \ell^2 C_{\mathrm{R}}\omega\,(v_{y'} - \omega\ell_{\mathrm{CM}}, v_{x'})$
10	Drag force, D (4.13)	$D = -\frac{1}{2}\rho_f \ell C_{\mathrm{D}} \sqrt{{v_{x'}}^2 + (v_{y'} - \omega\ell_{\mathrm{CM}})^2}\,(v_{x'}, v_{y'} - \omega\ell_{\mathrm{CM}})$
11	Torque from transl. forces, τ_{T} (4.14)	$\tau_{\mathrm{T}} = -\frac{1}{2}\rho_f \ell \sqrt{{v_{x'}}^2 + (v_{y'} - \omega\ell_{\mathrm{CM}})}\,[C_{\mathrm{L}} v_{x'} + C_{\mathrm{D}}(v_{y'} - \omega\ell_{\mathrm{CM}})]\,(\ell_{\mathrm{CP}} - \ell_{\mathrm{CM}})$
12	Aerodynamic rot. resistance, τ_{R} (4.15)	$\tau_{\mathrm{R}} = -\frac{1}{128}\rho_f \ell^4 C_{\mathrm{D}}^{\pi/2}\omega\lvert\omega\rvert \left[\left(\frac{2\ell_{\mathrm{CM}}}{\ell} + 1\right)^4 \pm \left(\frac{2\ell_{\mathrm{CM}}}{\ell} - 1\right)^4\right]$
13	Fixed frame x velocity, $\dot{x}$ (4.4)	$\dot{x} = v_{x'}\cos\theta - v_{y'}\sin\theta$
14	Fixed frame y velocity, $\dot{y}$ (4.5)	$\dot{y} = v_{x'}\sin\theta + v_{y'}\cos\theta$
15	Angular velocity, $\dot{\theta}$ (4.6)	$\dot{\theta} = \omega$
16	Platewise x' acceleration, $\dot{v}_{x'}$ (4.7)	$m\dot{v_{x'}} = \left(m + \pi\rho_f \ell^2/4\right)\omega v_{y'} - (\pi\rho_f \ell^2/4)\omega^2 \ell_{CM} + L_T^{x'} + L_R^{x'} + D^{x'} - m'g\sin\theta$
17	Platewise y' acceleration, $\dot{v}_{y'}$ (4.8)	$\left(m + \pi\rho_f \ell^2/4\right)\dot{v_{y'}} = -m\omega v_{x'} + (\pi\rho_f \ell^2/4)\dot{\omega}\ell_{CM} + L_T^{y'} + L_R^{y'} + D_{y'} - m'g\cos\theta$
18	Angular acceleration, $\dot{\omega}$ (4.9)	$I\dot{\omega} = \tau_T + \tau_R$