A Few Lessons Learned in Reinforcement Learning for Quadcopter Attitude Control

Anonymous Author(s)

ABSTRACT

In the context of developing safe air transportation, our work is focused on understanding how *Reinforcement Learning* methods can improve the state of the art in traditional control, in nominal as well as non-nominal cases (motor and sensor failures, weather conditions etc.). The end goal is to train provably safe controllers, by improving both the training and the verification methods. In this paper, we explore this path for controlling the attitude of a quadcopter: we discuss theoretical as well as practical aspects of training neural nets for controlling a *crazyflie 2.0* drone. In particular we describe thoroughly the choices in training algorithms, neural net architecture, hyperparameters, observation space etc. We also discuss the robustness of the obtained controllers, both to partial loss of power for one rotor and to wind gusts. Finally, we measure the performance of the approach by using a robust form of a signal temporal logic to quantitatively evaluate the vehicle's behavior.

ACM Reference Format:
Anonymous Author(s). 2021. A Few Lessons Learned in Reinforcement Learning for Quadcopter Attitude Control. In *Proceedings of Hybrid Systems: Computation and Control (HSCC).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Neural net based methods have had numerous successes in drone control, over the last few years, using privileged learning [39] or reinforcement learning [38]. However impressive actual experiments look like, we are still in need of fully understanding what advantages and performances we can gain with learning-based control, and what level of formal guarantees we can reach, either at design or at verification time.

We concentrate here on low-level controls, and more specifically attitude control for quadcopters. These have the advantage of being understandable - performances being easily measurable -, well studied in the literature, and essential to all higher levels controls and path tracking algorithms. We also focus on reinforcement learning (RL) methods, which are close to control and more particularly optimal control. Furthermore, RL has known tremendous progress over the past few years, with modern continuous state and action spaces training algorithms such as Soft Actor Critic (SAC) [32] and Twin-Delayed Deep Deterministic Policy Gradients (TD3) [27].

HSCC, 2021, Nashville, TN, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

A common belief is that learning-based control would be more robust to perturbations than e.g. PIDs, or at least could be trained to be more robust. Indeed, even a rather small neural net can encode a much more complex feedback control function than a simple PID, but this is commonly believed to be at the expense of formal guarantees. Also, the current zoology of training methods and architecture choices makes it difficult to fully understand the range of possible results.

This paper studies some of these aspects on the fundamental case of an attitude controller for the crazyflie 2.0 [26] quadcopter. We first present in Section 2 a non-linear ODE model for simulating the dynamics of a quadcopter, and extend it to account for partial motor failures, aerodynamic effects and wind gusts. We then present a flexible training platform with various neural net architectures and algorithms in Section 3, discuss performance evaluation using a robust signal temporal logic in Section 4, and describe our experimental setup in Section 5. Finally we discuss experimental results in Section 6.

Claims. In this paper,

(1) we develop a neural-net based control study case, based on careful modeling of a quadcopter's dynamics, including aerodynamic effects and partial power loss on motors,
(2) we fully discuss the effect of the chosen training algorithm, neural net architecture and hyperparameters on the performance of the controller, and investigate the impact of reduced observable state spaces in the RL training process
(3) we present in detail our experimental platform, which allowed us to compare more than 16,000 potential architectures, training algorithms and parameter choices,
(4) we develop Signal Temporal Logic observers to assess controller performance in a precise manner,
(5) we demonstrate high-quality attitude control using RL, for a relevant set of queries making sense for a real aircraft,
(6) we show these neural net controllers have a certain built-in robustness in non-nominal cases, with respect to (a) partial failures of actuators (b) perturbations such as wind gusts. Trying to train networks in non-nominal as well as nominal cases does not seem to bring better performance so far.

This paper is based on, and compared with, the following work:

RL in control. Reinforcement learning in control has been advertised, since [54], for the possibility to be more adaptative than classical methods in control such as PIDs. RL's close relationship with optimal control (the reward function is dual to the objective function) also makes it particularly appealing for applications to control, see e.g. [16].

Recently model-based reinforcement learning has been successfully used to train controllers without any initial knowledge of the dynamics and in a data-efficient way. For instance, in [42], a learning-based model predictive control algorithm has been used

to synthesize a low level controller. In [56], a hybrid approach is proposed, combining the model based algorithm PILCO [19] and a classic controller like a PD or a LQR controller.

In this paper, we focus on model-free algorithms because of their generality and because we have high fidelity models available for quadrotors, such as the crazyflie 2.0 [26]. More specifically we concentrate on actor-critic learning which has undergone massive improvements over the last few years with DDPG [44], SAC [32], TD3 [27], and compare it with the popular PPO method [52].

The high dimensionality of the full Markovian observation space is a challenge for training, prompting for a study of different choices for the sets of states observed by RL: we consider sub-spaces of the full Markovian observation space, where we leave out the states which have the least effect on the dynamics of the quadcopter. This is linked to partially observed Markov Decision Processes and Non Markovian learning, see e.g. [29].

We also study the robustness of our neural nets, as well as the specific training of the neural net controller to be able to handle disturbances (wind gusts, partial motor failures). These issues are linked to robust MDPs [50], although we use the classical (PO)MDP approach here.

RL for quadcopters, and attitude control. Most papers have been focusing on higher control loops, with the notable exception of [40], which serves as the basis of our work. We improve the results of [40] by considering more recent training algorithms (SAC and TD3), finer performance measures, and refined physical models (in particular perturbations due to partial motor failures and wind gusts). The closest other works related to attitude control for quadcopter are [49], [23], [41] and [17].

In [49], the goal is to stabilize a quadcopter in hover mode, from various initial conditions (including initial angular rates). The authors also consider perturbations to the dynamics, which are different from ours: motor lag and noise on sensors.

In [23], the objective is to control a quadcopter under cyber-attacks targeting its localization sensors (gyroscope and GPS) and motors. The authors consider (partial) motor failure (a limit on its maximal power, just like we do), but not wind gusts. Contrarily to most approaches including ours, their controller combines a classical controller and a neural net.

In [41] the authors discuss the training of a neural net controller for both attitude and position. They observe that it is difficult to train both aspects at the same time, whereas separating control in hover mode (acting mostly on the attitude) and control in position seems to work better. The learning process is based on a full state observation plus the difference with the target state. We extend this work first in discussing the simplification of the observed states, then in more rigorously defining observation metrics for offsets and overshoots.

In [17], the author considers neural nets for controlling roll, pitch, yaw rate and thrust, which is similar to the problem we are studying here, and attempts to train the controller such that it can accommodate motor and mass uncertainties within given bounds. In contrast, we deal with uncertainties such as wind gusts and motor failures, following known parametric models.

Temporal logics and safe learning. The study of reinforcement learning under temporal logic specifications has gained a lot of interest in recent years. In a discrete and finite state setting, in [55] an LTL property observer automaton is composed with the system MDP to allow blocking unsafe actions during training. In [28, 34] rewards are modulated depending on the observer state, and a model-free approach is proposed in [35] using Limit Deterministic Büchi Automata. *Shielding* [9] simultaneously trains an optimal controller and a *shield* that corrects the LTL-formula violating actions. The method requires a fully explicit model of the environment MDP and builds the product of the orignal MDP with the property monitor. Later works extend shielding to the continuous [14, 57] and online [13] cases, assuming an embeddable predictive environment model is available, but only handle simple state invariants.

In the continuous state setting, temporal logics with quantitative semantics such as Metric Interval Temporal Logic (MITL) [22], Signal Temporal Logic (STL) [20] ..., have been studied in relation with reinforcement learning. Robust interpretation yields a real number indicative of the distance to the falsifying states boundary. STL has seen numerous extensions improving expressiveness and signal classes [6, 7, 11, 18] as well as smooth semantics [30, 33, 48]. In [8], Q-learning is used to train a policy maximizing both the probability of satisfaction and the expected robustness of a given STL specification. The approach requires storing previously visited states in the MDP in addition to the original MDP state, yielding a high dimensional system and limiting learning efficiency. Similarly, recent work proposes deriving barrier functions from robust temporal logic specifications, either to modulate rewards during training [43] or to control the switch from an optimal and potentially unsafe controller to a safe backup controller [45].

In summary, existing methods focused on the training phase either suffer from dimensionality and combinatorial explosion, require expected robustness approximations, and are strongly tied to the Q-learning algorithms. Solutions to long-standing dimension and magnitude mismatch in robust STL interpretation were proposed in [58] but do not translate naturally to the RL framework.

Considering our goal is to study a large hyper-parameter space for training controllers, we used an expressive yet tractable variant of STL to specify properties and assess trained controllers offline, separately after training.

The next steps will be to start using STL-derived reward signals during training on the most promising architectures.

2 MODELLING AND CONTROL OF A CRAZYFLIE 2 QUADROTOR

2.1 Modelling

In this section, we detail the dynamical model of the crazyflie quadrotor [31, 47] and we augment it with partial motor failures and wind gusts modelling.

We study the dynamics on the vertical axis and the pitch rate, roll rate and yaw rate control (4 degrees of freedom), with the following state variables: the vertical position in the world frame z, the linear velocity of the center of gravity in the body-fixed frame with respect to the inertial frame (u, v, w), the angular orientation represented by the Euler angles (ϕ, θ, ψ) where ϕ is the roll angle θ is the pitch angle and ψ is the yaw angle, the attitude or angular

velocity with respect to the body frame (p, q, r) with p the roll rate, q the pitch rate and r the yaw rate.

Using Newton's equations given a *thrust* force and moments M_x, M_y and M_z exerted along the three axes of the quadcopter, and using the rotation matrix R from the body frame to the inertial frame, the Translation-Rotation kinematics and dynamics [47] lead to a 10-dimensional non-linear dynamical system:

$$\begin{cases} \dot{z} = -s_\theta u + c_\theta s_\phi v + c_\theta c_\phi w & \dot{\theta} = c_\phi q - s_\phi r \\ \dot{u} = rv - qw + s_\theta g & \dot{\psi} = \frac{c_\phi}{c_\theta} r + \frac{s_\phi}{c_\theta} q \\ \dot{v} = -ru + pw - c_\theta s_\phi g & \dot{p} = \frac{I_y - I_z}{I_x} qr + \frac{1}{I_x} M_x \\ \dot{w} = qu - pv - c_\theta c_\phi g + \frac{F}{m} & \dot{q} = \frac{I_z - I_x}{I_y} pr + \frac{1}{I_y} M_y \\ \dot{\phi} = p + c_\phi t_\theta r + t_\theta s_\phi q & \dot{r} = \frac{I_x - I_y}{I_z} pq + \frac{1}{I_z} M_z \end{cases} \quad (1)$$

writing c_x as a short for $cos(x)$, s_x for $sin(x)$ and t_x for $tan(x)$. F is the sum of the individual motor thrusts, I_x, I_y, I_z are the quadcopter's moments of inertial around the x, y and z axes, respectively.

Instead of controlling directly each rotor speed, the four commands $thrust$, cmd_ϕ, cmd_ψ and cmd_θ are used to deduce the PWM values to apply to each motor, Equation (2):

$$PWM = \begin{bmatrix} PWM_1 \\ PWM_2 \\ PWM_3 \\ PWM_4 \end{bmatrix} = \begin{bmatrix} 1 & -1/2 & -1/2 & -1 \\ 1 & -1/2 & 1/2 & 1 \\ 1 & 1/2 & 1/2 & -1 \\ 1 & 1/2 & -1/2 & 1 \end{bmatrix} \begin{bmatrix} thrust \\ cmd_\phi \\ cmd_\theta \\ cmd_\psi \end{bmatrix} \quad (2)$$

PWMs are linked to rotation rates Ω: $\Omega = [\omega_1\ \omega_2\ \omega_3\ \omega_4]^\top = C_1 PWM + C_2$. Finally, we deduce the input force and moments from the squared rotation rates, Equation (1), with force and momentum equations $[F\ M_x\ M_y\ M_z]^\top$ equal to:

$$\begin{bmatrix} C_T\left(C_1^2(cmd_\theta^2 + cmd_\phi^2 + 4cmd_\psi^2 + 4thrust^2) + 8C_1C_2 thrust + 4C_2^2\right) \\ 4C_T d\left(C_1^2(cmd_\phi thrust - cmd_\theta cmd_\psi) + C_1C_2 cmd_\phi\right) \\ 4C_T d\left(C_1^2(cmd_\theta thrust - cmd_\phi cmd_\psi) + C_1C_2 cmd_\theta\right) \\ 2C_D\left(C_1^2(4cmd_\psi thrust - cmd_\phi cmd_\theta) + 4C_1C_2 cmd_\psi\right) \end{bmatrix} \quad (3)$$

The parameters are given in Appendix A.

Given a series of controls (at time steps multiple of $dt\_commands$, 0.03 seconds here), the ODE of Equation (1) can now be solved numerically (using Runge-Kutta of order 4 here, with a time step of 0.01 seconds) to obtain full quadcopter trajectories with high precision.

2.2 Motor failure

We suppose, for the sake of simplicity, that the quadcopter may experience a power loss on motor 1. This partial failure is modeled as a saturation of the maximum PWM, with a factor between 0.8 and 1.

Since quadcopter controls rely on differential thrust between motors, motor failures are very difficult to cope with. In order to keep a constant yaw when one motor is failing, the gyroscopic effect must be made equal to zero, for instance by having the two motors rotating in the opposite direction match the saturation of the faulty motor. The same idea applies to pitch and roll axes.

Therefore, if the failure is not too harsh, and the target states are not too demanding, it is *a priori* feasible to recover some control of the faulty quadrotor by saturating all four motors in the same way.

In this paper, we will look at two potential solutions to control in the presence of partial motor failure. The first one is to look at how robust a controller that has been designed for nominal cases (i.e. without partial motor failures) is. The other one is to train, using reinforcement learning, a controller optimized for a variety of situations, from no failure to maximal failure rate.

2.3 Wind gusts

2.3.1 Aerodynamic effects. In Equation (1), we neglected all aerodynamic effects. When we take into account aerodynamic forces, an extra force F^a is exerted on the quadcopter that depends on the wind speed and direction relative to the quadcopter, the angular velocities of the rotors and extra moments M_x^a, M_y^a and M_z^a. We follow the full aerodynamic model of [26] with the coefficients measured for a crazyflie 2.0, where the effect of the wind on the structure is neglected with respect to the effect on the rotors, and the blade flipping effect (due to elasticity of the rotor) is also neglected.

The extra force F^a can be decomposed as the sum of the four extra aerodynamic forces on rotor i ($i = 1, \ldots, 4$), that can be modelled as depending linearly on the rotors angular velocities, and linearly on the wind relative speed with respect to rotors. Other models [12] include blade flipping and other drag effects, but the induced drag we are modelling is the most important one for small quadrotors with rigid blades. We use $f^i = \Omega_i K W_i^r$ for the aerodynamic force exerted on rotor i in the inertial frame, where K is the drag coefficients matrix, W_i^r is the relative wind speed as seen from rotor i, in the body frame, i.e. $W_i^r = (u_i, v_i, w_i) - R^T W_a$ with W_a the absolute wind speed in the inertial frame, (u_i, v_i, w_i) being the linear velocities of the rotors in the body frame, R is the rotation matrix from the body frame to the inertial frame (R^T is its inverse, i.e. its transpose here), and Ω_i is the absolute value of the angular velocity of the i-th rotor.

On the crazyflie, $\Omega_i = C_1 PWM_i + C_2$, where PWM_i are given using $thrust$, cmd_ϕ, cmd_θ and cmd_ψ using Equation (2).

The linear velocities of rotors can be computed as follows:

$$\begin{pmatrix} u_j \\ v_j \\ w_j \end{pmatrix} = \begin{pmatrix} p \\ q \\ r \end{pmatrix} \times \begin{pmatrix} dc_j \\ ds_j \\ h \end{pmatrix} + \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} qh - rds_j + u \\ -ph + rdc_j + v \\ pds_j - qdc_j + w \end{pmatrix}$$

where (u, v, w) are the linear velocities of the center of mass of the quadrotor in the body frame, (p, q, r) are the angular velocities of the quadrotor, d is the length of the arm linking the center of the drone to any of the four motors, and for $j \in \{1, 2, 3, 4\}$, $c_j = sin\left(\frac{\pi}{2}(j-1) + \frac{3\pi}{4}\right)$ and $s_j = cos\left(\frac{\pi}{2}(j-1) + \frac{3\pi}{4}\right)$ are such that (c_j, s_j, h) is the coordinate of rotor j in the body frame, with the center of mass being the origin.

Now, we add to the second term of Equation (1) for $\dot{u}$, $\dot{v}$, $\dot{w}$ the aerodynamic force $F^a = (F_x^a, F_y^a, F_z^a)$ divided by m, and to moments of Equation (3), the aerodynamic moments $M^a = (M_x^a, M_y^a, M_z^a)$ with $F^a = f_1 + f_2 + f_3 + f_4$ and $M^a = (dc_1, ds_1, h) \wedge f_1 + (dc_2, ds_2, h) \wedge f_2 + (dc_3, ds_3, h) \wedge f_3 + (dc_4, ds_4, h) \wedge f_4$.

We derive the full dynamics of the quadcopter considering aerodynamic effects, and only write below the modified equations:

$$\begin{cases} \dot{u} = rv - qw + s_\theta g + \frac{F_x^a}{m} & \dot{p} = \frac{I_y - I_z}{I_x} qr + \frac{1}{I_x}(M_x + M_x^a) \\ \dot{v} = -ru + pw - c_\theta s_\phi g + \frac{F_y^a}{m} & \dot{q} = \frac{I_z - I_x}{I_y} pr + \frac{1}{I_y}(M_y + M_y^a) \\ \dot{w} = qu - pv - c_\theta c_\phi g + \frac{F + F_z^a}{m} & \dot{r} = \frac{I_x - I_y}{I_z} pq + \frac{1}{I_z}(M_z + M_z^a) \end{cases} \tag{4}$$

2.3.2 Wind models. There are two main types of models in the litterature, represented by e.g. Discrete Wind Gust and von Kármán Gust or Dryden Gust models. Von Kármán gusts and Dryden gusts are stochastic gust models (homogeneous and stationary gaussian processes) characterized by their power spectral densities for the wind's three components, Dryden gusts being an approximation of Von Kármán gusts.

The Discrete Wind Gusts model consists in a explicit and deterministic representation of wind gusts as half period cosine perturbations ([51], eq. (45)) detailed below. We focus on this model because it is widely used for aircraft certification (using dozens of discrete wind gusts with different magnitudes and scales).

A discrete wind gust is characterized by its fixed direction, magnitude and scale, and lasts for a half period during which wind speed increases until it reaches its maximum intensity. The absolute wind velocity is given as a function of time as, using the same notations as in Section 2.3.1: $W_a(t) = \frac{A_g}{2}\left(1 - cos\left(\frac{\pi(t-t_0)}{\delta}\right)\right) V_d$ if $t_0 \leq t \leq t_0 + 2\delta$, 0 otherwise, where A_g is the maximal magnitude of the wind gust, δ is the half life of the gust, and V_d is a normalized vector in R^3, which is the wind (absolute) direction.

2.4 PID Control

As in [40], the objective is to train only the attitude controller, and not the altitude one. We therefore use a PID for controlling z. We will also need some idea of what a standard PID controller may achieve in terms of performance, and robustness to wind gusts and failures. For this, we will primarily use one of the altitude and attitude PID controller implemented in the crazyflie 2.0. Given setpoints z_{sp}, p_{sp}, q_{sp} and r_{sp}, the quadrotor is controlled using a PID controller (called PID1 in the sequel) which is the one of [31] (see Appendix A).

But as we will see, the attitude controller implemented in the crazyflie 2.0 is not very reactive, most probably for ensuring that the altitude is very securely controllable (since too much reactivity in pitch and roll means sudden loss of vertical speed). In order to give an idea of what we could observe as best performance, we also designed a specific PID for attitude, that we call PID2, which is much more reactive:

$$\begin{cases} thrust = 3000(z_{sp} - z) \\ \qquad\qquad + 300 \int (z_{sp} - z)\,\mathrm{d}t - 500\dot{z} + 48500 \\ cmd_\phi = 1000(p_{sp} - p) + 400 \int (p_{sp} - p)\,\mathrm{d}t - 40\dot{p} \\ cmd_\theta = 1000(q_{sp} - q) + 400 \int (q_{sp} - q)\,\mathrm{d}t - 40\dot{q} \\ cmd_\psi = 2000(r_{sp} - r) + 1000 \int (r_{sp} - r)\,\mathrm{d}t - 100\dot{r} \end{cases} \tag{5}$$

3 TRAINING

3.1 Underlying Markov decision process

Reinforcement learning is designed to solve Markov decision problems. At each discrete time step $k = 1, 2, \ldots$, the controller observes the state x_k of the Markov process, selects action a_k, receives a reward r_k, and observes the next state x_{k+1}. As we are dealing with Markov processes, the probability distributions for r_k and x_{k+1} depend only on x_k and a_k. Reinforcement learning tries to find a control policy, i.e. a mapping from states to actions, in the form of a neural net, that maximizes at each time step the expected discounted sum of future reward.

For the attitude control problem at hand, the set of Markovian states is $thrust, p, q, r, err_p = p_{sp} - p, err_q = q_{sp} - q, err_r = r_{sp} - r$ (where (p_{sp}, q_{sp}, r_{sp}) is the target state, or "plateau" we want to reach), in the nominal case (similarly to what is done in e.g. [41]). We will also consider partially observed Markov processes, with only subsets of states for improving sampling over smaller dimensional states, by leaving out those states which should have less influence on the dynamics: our first candidate is to leave out thrust, which appears only as second order terms in the moments calculation, Equation (3), and also, p, q, r that are second order in the formulation of the angular rates, again in Equation (3). We do not consider here adding past information, classical in non Markovian environments [29], that has been used for attitude control in e.g. [40], but increases the dimension by a large amount.

In the case of partial motor failure, we add the knowledge of the maximum thrust for faulty motor 1, as a continuous variable between 80% and 100%. In the case of aerodynamic effect and wind gusts, we add the knowledge of the maximal magnitude and direction (in the inertial frame) of the incoming wind. In both cases, it can effectively be argued that it is possible to detect failures in almost real time, and to measure (or be given from ground stations) maximum winds and corresponding directions, in almost real time as well. In the case of wind-gusts, Markovian states include also the linear velocities u, v and w, since wind gusts are only defined in the inertial frame, and the induced aerodynamic effects depend on relative wind speed.

With a view to solving optimal control problems (or Model-Predictive like control), we choose to use a reward function which is a measure of the distance between the current attitude (p, q, r) with the target attitude (p_{sp}, q_{sp}, r_{sp}) (similar to the one used in [40]): $r(s) = -max\left(0, min\left(1, \frac{1}{3\Omega_{max}} \|\Omega^* - \Omega\|\right)\right)$, Ω_{max} being the maximal angular rate that we want to reach for the quadcopter, and Ω is the angular rate vector (p, q, r) which is part of the full state s of the quadcopter.

3.2 Neural net architecture

Neural nets, such as multiple layer perceptrons (MLP) with RELU activation, can efficiently encode all piecewise-affine functions [10]. It is also known [15] that the solution to a quadratic optimal control (MPC) problem for linear-time invariant system is piecewise-affine. Furthermore, there are good indications that this applies more generally, in particular for non-linear systems [25]. This naturally leads to thinking that MLPs with RELU networks are the prime candidates for controlling the attitude with distance to the objective as cost

(or reward). In some ways, the resulting piecewise-affine function encodes various proportional gains that should be best adapted to different subdomains of states, so as to reach an optimal cumulated (and discounted, here with discount rate $\gamma = 0.99$) distance to the target angular rates, until the end of training.

In theory [24], one could find a good indication of the architecture of the neural net in such situations, but the bounds that are derived in [24] are not convenient for such a highly complex system. It is by no means obvious what architecture will behave best, both for training and for actual controller performance, although a few authors argue that deeper networks should be better, see e.g. [46].

Architectures that have been reported in the literature for similar problems are generally alike. In [49], the neural net is a Multi-Layer Perceptron (MLP) with two layers of 64 neurons each, and with tanh activation function. In [23], the part of the controller which is a neural net is a MLP with two layers of 96 neurons each and tanh activation function, whose input states (observation space) are all states plus the control. In [41], the hover mode neural net controller, which is the most comparable to our work, is a MLP with two layers of 400 and 300 neurons respectively, with RELU activation for hidden layers and tanh for the last layer. In [17], the resulting architecture is a two layers MLP with 128 neurons on each layer, and RELU activation function.

We will report experiments with one to four layers, and with 4, 8, 16, 32 or 64 neurons per layer, with RELU activation function (except for the rescaling of the output, using tanh). We limit the reporting of our experiments to these values since we observed that these were enough to find best (and worst) behaviours.

3.3 Training algorithms

The first three algorithms we are discussing in Section 5, DDPG [44], SAC [32] and TD3 [27] are all off-policy, actor-critic methods, which are generally considered to be better suited for control applications in robotics [54] (DDPG is used for instance in [23]). Because of its effectiveness in practice, observed by many authors, e.g. [40] for attitude control, we also compare with the on-policy Proximal Policy Optimisation [52], also used for similar applications in [49] and in [17].

DDPG is the historical method for continuous observation and action space applications to control, SAC and TD3 being later improvements of DDPG. For instance, SAC regularizes the reward with an entropy term that is supposed to reduce the need to fine hyper-parameter tuning.

Let us now describe the training mechanism: we call *query signal* the function describing the prescribed angular rates at any given time. We model this signal by a constant plateau, of magnitude chosen randomly between -0.6 and 0.6 radians per second, and duration chosen randomly between 0.1 and 1 second. We are training over a time window of 1 second (a training episode) during which the query signal is a constant plateau followed by a value of 0 until the end of the episode. We chose to report on training where these query signals are used independently on pitch, roll and yaw. Having joint queries on pitch, roll and yaw does not seem to change the outcome of our experiments.

Controls are updated every 0.03 seconds, and we simulate the full state of the quadrotor, using a Runge Kutta of order 4 on Equation (1) with a time step of 0.01 seconds.

The evaluation of the controller is made on similar query signals, but on time windows that last 20 seconds, with a query signal generated according to a more general class of queries (see below). Query signals on such longer time windows could also be considered for training : [40] refers to this approach as "continuous mode" and reports much poorer performance compared to the "episodic mode" with 1 second queries. We therefore decided to report only on episodic mode training.

Such query classes are characterised by three distributions A, D and S for respectively the amplitude and duration of stable plateaus, and the step amplitude between successive stable plateaus. These distributions are the same for each axis. We define three different classes of queries (where U(a,b) denotes the Uniform distribution of support [a,b]): easy (A = U(-0.2, 0.2), D = U(0.5, 0.8), S = U(0, 0.3)), medium (A = U(-0.4, 0.4), D = U(0.2, 0.5), S = U(0, 0.6)) and hard (A = U(-0.6, 0.6), D = U(0.1, 0.2), S = U(0, 0.9)). Our query generator actually changes the joint distribution of amplitude and duration of stable plateaus by filtering out those queries which would make the roll, pitch and yaw go through singular values in the Euler angles description of the dynamics.

4 FORMAL PERFORMANCE CRITERIA

Designing a controller for a specific application requires balancing multiple criteria such as rising time, overshoot, steady error, etc. In order to quantify rigorously the performance of the learned controller, we formalized requirements using a recent extension of Signal Temporal Logic (STL) published in [11], interpreted over piecewise constant signals. In addition to the usual *Globally*, *Finally* and *Until* modalities, this logic offers:

- $On_{[a,b]}\ Agg\ \tau(x)$, with $Agg \in \{Min, Max\}$, which computes a Min/Max aggregate of a real-valued term $\tau(x)$ on time interval $[a, b]$;
- $\tau(x)\ U^{d}_{[a,b]}\ P(x)$, which samples the term $\tau(x)$ when $P(x)$ first becomes true within $[a, b]$, or returns the default value d if $P(x)$ does not become true in $[a, b]$;
- $D^{d}_{a}\tau(x)$ which samples the value of term $\tau(x)$ (or formula $P(x)$) at time a if it is defined, or returns d otherwise;
- $ite(P(x), \tau_1(x), \tau_2(x))$ which is equal to $\tau_1(x)$ when $P(x)$ is true and to $\tau_2(x)$ otherwise.

A first set of formulae allows to identify instants when a query signal q becomes stable for T time units, and whether q goes up or down at any instant (with ϵ and d two small constants), and the step size:

$$stable(q) = (On_{[0,T]}\ Max\ q) - (On_{[0,T]}\ Min\ q) < d \tag{6}$$

$$stableup(q) = (D^{\perp}_{-\epsilon}\ \neg stable(q)) \wedge stable(q) \tag{7}$$

$$up(q) = q - (D^{0}_{-\epsilon}\ q) > 0 \tag{8}$$

$$down(q) = q - (D^{0}_{-\epsilon}\ q) \leq 0 \tag{9}$$

$$step(q) = ite(stableup(q), q - D^{0}_{-\epsilon}q, 0) \tag{10}$$

We consider an angular rate signal x as acceptable if it does not overshoot a stable query q by more than α% of the step size on

$[0, T_1]$, and does not stray away from a stable query q by more than $\beta\%$ of the step size on $[T_1, T]$:

$$stableup(q) \wedge up(q) \implies On_{[0,T_1]}\ Max\ (x - q) < \alpha step(q) \tag{11}$$

$$stableup(q) \wedge down(q) \implies On_{[0,T_1]}\ Max\ (q - x) < \alpha step(q) \tag{12}$$

$$stableup(q) \implies On_{[T_1,T]}\ Max\ \|x - q\| < \beta step(q) \tag{13}$$

We define the rising time RT as the time it takes for x to first reach q within $\gamma\%$:

$$ite(stableup(q), t - (t\ U^{+\infty}_{[0,T]}\ \|(x - q)\| < \gamma q), +\infty) \tag{14}$$

Figure 1 illustrates the formalised notions and parameters.

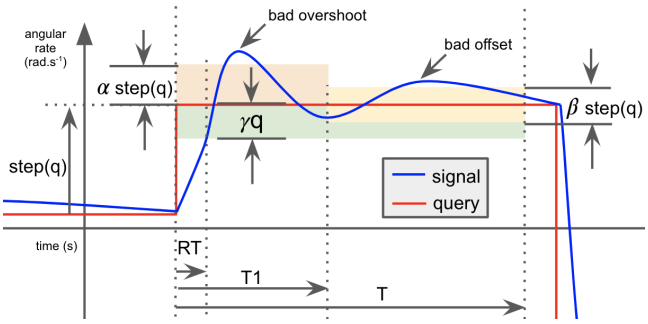

Figure 1: Property parameters T_1, T, RT, α, β, γ.

Using observers code generated from these specifications, we compute statistics about property violations and associated robustness margins on angular rate signals and queries on pitch, yaw and roll axis of the system, acquired at regular intervals during the training of the controller. For evaluation each property $P(x, q)$ is wrapped in a *globally* modality over the episode length yielding $G_{[0,episode\_length]}\ P(x, q)$. Automating the computation of these behavioral metrics is essential in allowing to scale up the hyper-parameter space exploration and identify the best controller according to objective measurements.

5 EXPERIMENTAL SETUP

5.1 Implementation

We have developed a platform with the purpose of running experiments in a reproducible and scalable way, becoming an integration layer between the different moving parts in both training and testing. From a technological standpoint the platform is based on the Stable Baselines 2.7.0 reinforcement learning library [37] itself based on Tensorflow [5], all of our code is in Python and we used Bazel [2] as build system. We used Tensorboard to monitor losses and the internal dynamic of the neural networks during the training.

One intermediate goal was to explore the large combinatorial hyperparameter space efficiently to be able to identify the best hyperparameters values with respect to the STL metrics we defined and to get a better understanding of their impact.

With 4 different algorithms, 20 possible configurations for the network architecture and 3 sets of observed states, our hyperparameters space contains a total of 240 points that need to be trained and tested. The corresponding jobs are dispatched on our Kubernetes cluster [4] where they can run in parallel. Disposing of 1 vCPU on the Cascade Lake platform (base frequency of 2.8 GHz), the 3 millions iterations of a single training job take between 3 and 8 hours to complete. The cluster autoscales with the workload and allowed us to run 1200 hours worth of training in half a day.

The container images that end up running on the cluster are created, uploaded and finally dispatched in a reproducible manner thanks to the Bazel rules of our Research Platform. Those rules are built on top of the Bazel Image Container Rules [1] and the Bazel Kubernetes Rules [3] and specially designed to generate all the experiment jobs of the hyperparameters analysis.

The training and testing results are automatically uploaded on our cloud storage where they can be browsed for quick inspections, or fed as input for the next pipeline stage. We saved 30 checkpoints per experiment (each file containing 100k training iterations weights between 10KB and 100KB). Including the TensorFlow logs, the training results amount to >100GB of data.

Each of the 30 × 240 checkpoints was then evaluated on 100 queries computed by the Query Generator, producing the same number of concrete traces representing the commands and the states over the whole episode. Each set of such traces is about 600k hence it yields total of 60MB per checkpoint. Finally each of the 30 × 240 × 100 traces was evaluated with STL properties observer to compute synthetic metrics: aggregating the 100 traces of a single checkpoint produced a 150KB file and required approximately 45 minutes. The checkpoint specific CSV files were further aggregated in experiment specific and round specific checkpoints for the final visual inspection.

5.2 Interactive browsing of the experiments database

We want to understand what correlations we have between controller performance and the way it has been trained, and for this, we used Hiplot [36] for browsing through the enormous number of parameters and data generated. We show in Figure 2 how we used Hiplot in an interactive manner for verifying our hypotheses. Each parameter and performance measure is represented by a column in the graph generated by Hiplot from our database. For each parameter, either fixed or free, choosing intervals of values for each performance measure creates lines that link parameter values to performance values within the chosen intervals. The number of entries in the database (i.e. the number of controllers) that satisfy the constraints is also shown, as well as the table of all their corresponding parameters and performance values.

For instance, we used Hiplot to select the "best" networks, filtering the data set of controllers, only retaining the ones with better success in offset, overshoot and rising times altogether, with respect to the best PIDs. This resulted in two neural nets with much better performances than the PIDs on offset and on rising time as we will discuss in Section 6.

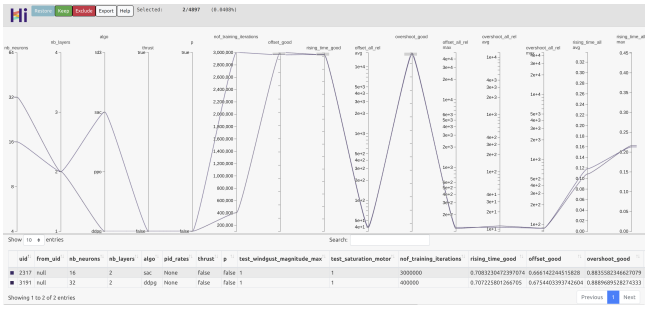

Figure 2: Hiplot interactive session

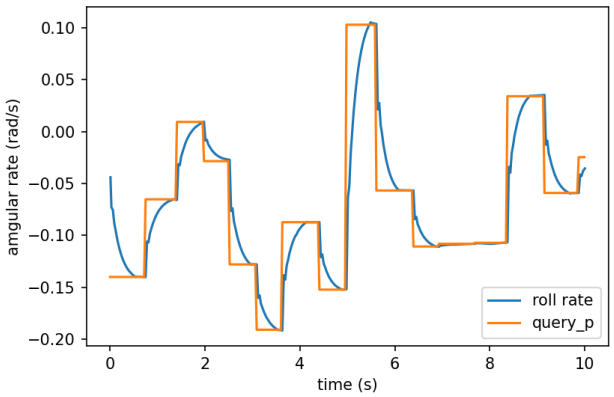

Figure 3: PID2 controller query tracking

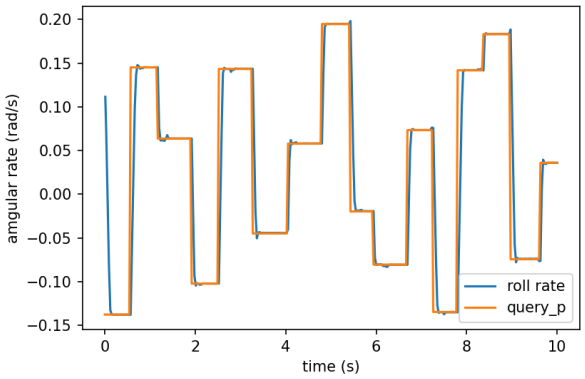

Figure 4: Neural controller behaviour (sac, 2 layers, 16 neurons per layer, 3M iterations)

6 EXPERIMENTS RESULTS

6.1 Performance metrics

Each controller is evaluated on a hundred evaluation episodes using STL observers defined in Equations (11) to (14), where parameters are set to $\alpha = 10\%$, $\beta = 5\%$ and $\gamma = 5\%$, $T = 0.5s$, $T_1 = 0.25s$, $\epsilon = 0.01s$, $d = 0.005$. For each evaluation episode the following statistics are computed over all stable query plateaus:

- average and maximum overshoot percentage relative to the query step size,
- average and maximum offset percentage relative to the query step size,
- average and maximum rising time values in seconds (only for plateaus where the signal actually reaches $\gamma\%$ of the query within $[0, T]$).

For each metric (overshoot, offset, rising time), we compute the *success percentage* % OK, i.e. the percentage of stable plateaus of the episode for which the controller behaviour satisfies the specification.

Then, episode-level statistics are further averaged, yielding results presented in the tables of the following sections, where columns represent:

- avg (resp. max) overshoot: is the per-episode-average of the average (resp. maximum) overshoot values,
- avg (resp. max) offset: is the per-episode-average of the average (resp. maximum) offset values,
- avg (resp. max) rising time: is the per-episode-average of the average (resp. max) rising time,
- % Ok offset (resp. overshoot, rising time): is the per-episode-average of the success percentage for the offset (resp overshoot, rising time) metric.

6.2 Performance of nominal-trained networks in nominal test case

6.2.1 Overall best performance comparison. The PID performance metrics in the nominal case are reported in the first two lines of Table 1 to serve as a reference point for neural controller evaluation. Examples of query tracking behavior are given in Figure 3 for reference.

PID2 reaches within 5% of the target state for about 70% of the queries, and is relatively slow with an average rising time of 0.44s. PID1 in comparison reaches within 5% of the target state for only 8% of the queries, with a better rising time, but that is not statistically meaningful. Overshoot success rates are really good for both PIDs (95-100% OK). Offset success rates are bad (1-3% OK), due to their slow convergence. We will hence use PID2 as a reference for discussing neural controller performance.

The comparison between the best networks and the PIDs is reported in Table 1. We see that our neural nets give much quicker controls, with an average rising time of about a fourth to a fifth of the rising time for the two PIDs, although with a negligible offset. This is at the expense of a slightly less good performance on the maximum overshoot at least for SAC and DDPG trained networks, with respect to PID2 (our neural nets are still much better than PID1). Results are far less good, in particular concerning overshoots, with PPO and TD3 trained networks. This is also visible when comparing signals between Figure 4 and Figure 3. Somehow, neural nets exhibit extreme reactivity as well as good asymptotic convergence, but show some very short-lived "spikes", as in the sample trajectory shown in Figure 4.

algo	OK rising t.	OK off.	OK overshoot	avg rising t.	avg off.	avg overshoot	max rising t.	max off.	max overshoot
pid2	70.52	0.52	100.00	0.41	20.22	0.00	0.48	25.26	0.00
pid1	7.73	3.44	94.88	0.33	58.93	3.91	0.38	138.44	50.56
sac	97.35	99.54	96.55	0.08	0.12	0.50	0.21	2.44	6.21
ddpg	99.05	98.59	98.21	0.08	0.21	0.23	0.17	3.97	3.91
ppo	96.96	97.26	91.98	0.08	0.43	1.41	0.19	7.08	11.93
td3	96.70	86.80	88.16	0.09	2.40	2.13	0.22	18.08	20.09

Table 1: PIDs and overall best networks performance (all in % except rising t. in seconds)

When we filter the neural nets meeting or exceeding the performances of PID2, many networks remain, among which the best are:

- DDPG 64×64×64×64 trained for 1500000 iterations (and also DDPG 32 × 32, 400000 iterations) on the three-dimensional observation space $(p - p_{sp}, q - q_{sp}, r - r_{sp})$
- SAC 32×32×32×32 (and SAC 32×32 and 16×16 trained for 3000000 iterations coming very close) trained for 2900000 iterations on the same three-dimensional observation space

6.2.2 Training algorithm influence. We observe in Table 1 that PPO and TD3 do not show as good performance as SAC (and even DDPG), moderating the conclusion of [40], and the common belief that TD3 should improve performance of neural net control. We have for now no explanation for this, largely because we have not been able (which is also the case in [40]) to get rid of the overshoot spikes, even using SAC which does some amount of regularization, or TD3 which should lead to more stable solutions at, potentially, the expense of a slower convergence rate. In terms of optimal control, if the neural net controller were trained with correctness objectives[1], these spikes would certainly be much smaller and appear only at the very beginning of plateaus.

6.2.3 Convergence of the training algorithms. We show in Figure 5 the evolution of the three main performance measures, the OK overshoot, OK offset and OK rising time, for one of the best network architecture and training algorithm, SAC 32×32 neurons. The three metrics improve quickly and almost stabilize in the first 1000000 iterations.

6.2.4 Observation state influence. Of course, the smaller dimension the observation state is, the better the quality of the sampling is, for the same number of iterations. Still, we observe that using a Markovian state or the simpler three-dimensional state space (err_p, err_q, err_r) does not change significantly the performance of the best neural nets obtained, see Table 2, although the 3 dimension observation space gives slightly better performance overall. In fact, we even get a worse performance with the dimension 7 full state, mostly because of the difficulty to sample this higher dimensional space, and identify the subtle second-order effects of some of these states on angular rates.

6.2.5 Neural net architecture influence. First, we observe that almost none of the single-layer neural nets seem to converge to a correct controller (see e.g. Figure 6). At 64 neurons, 1 hidden layer networks seem to exhibit some good behaviour, but still far from any of the e.g. two-layers neural nets.

Still, 3-layers and even 4-layers networks do not seem to exhibit much better behaviour than the "best" 2-layers networks, with 16 or 32 neurons each, although they converge in a faster manner. Recently Sinha et al. in [53] empirically observed the performance of SAC have a peak using 2 layers MLP and their explanation for this result relies on the Data Processing Inequality hence the fact

[1]Future work to cope with this phenomenon includes improving the reward function using our STL observers, and adding some more regularization during training.

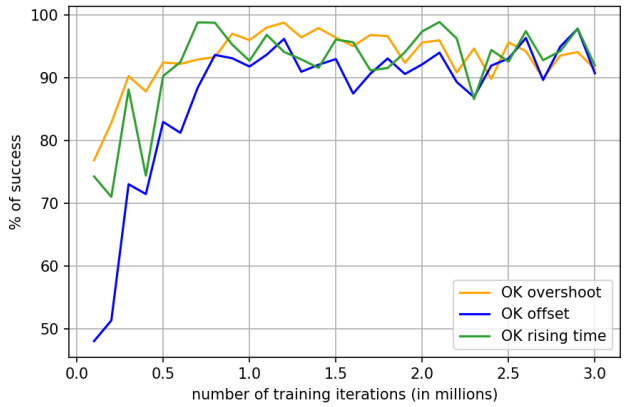

Figure 5: Performance of SAC 32x32 on dim 3 observation space trained neural nets w.r.t. the number of iterations

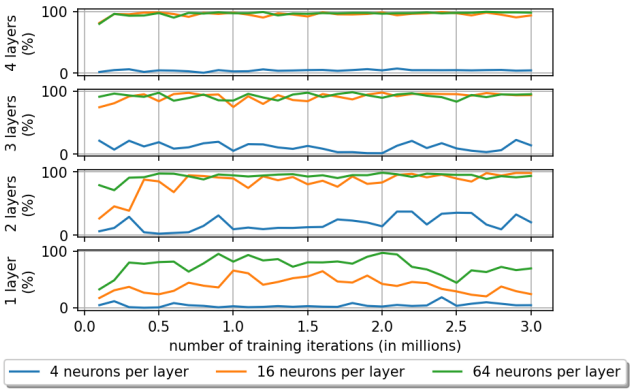

Figure 6: OK rising t. for our best SAC network wrt number of training iterations for different architectures

algo	dim	OK rising t.	OK off.	OK overshoot	avg rising t.	avg off.	avg overshoot	max rising t.	max off.	max overshoot
sac	3	98.88	98.01	98.58	0.09	0.28	0.18	0.19	5.01	3.28
sac	6	97.24	98.70	94.82	0.09	0.25	0.81	0.19	4.72	8.07
sac	7	94.26	94.57	97.79	0.09	0.80	0.33	0.25	11.56	5.75

Table 2: Influence of the observable space dimension (all in % except rising t. in seconds)

that mutual information between layers decreases with depth. This will have to be further investigated in our framework.

6.3 Performance of nominal-trained networks in non-nominal test cases

6.3.1 Robustness to motor failures. We now assess the robustness of our PIDs and "best" neural nets (trained in nominal situations as discussed in Section 6.2) to partial motor failures, without training the neural nets nor changing the gains of PIDs to cope specifically for this situation.

We report in Table 3 the same performance measures as the one used in the nominal case, in the test cases where motor 1 can experience a partial power loss, down to 80% of its maximal power, at the start of any new plateau along the 20 second episodes that we are observing (which can contain about 30 different target angular states, or plateaus, to reach within a short time). We take maxima and averages of these measures on 100 such queries as before.

In case of partial motor failure, our best SAC trained neural net behaves much better than our two PIDs: it keeps on reaching plateaus within 0.5 seconds for about 94% of the time, whereas even the best PID goes down to less than 60% success rate. Our network is even better when it comes to satisfying offset constraints (82% of the time) whereas the PIDs almost never comply. Performances concerning overshoot are comparable, even though the PIDs are very slightly better, but this only concerns cases where PIDs actually reach the target state, which is the case much less often. Basically, the best neural nets that have been trained under nominal conditions show very little degradation of performance when a partial failure occurs.

6.3.2 Robustness to wind gusts. We then assess the robustness of our PIDs and "best" neural nets (trained in nominal situations as discussed in Section 6.2) to wind gusts (described in Section 2.3.1), without training the neural nets nor changing the gains of PIDs to cope specifically for this situation.

We present in Table 3 the same performance measures than the ones used in the nominal case, when random wind gusts of magnitude up to 10 $m.s^{-1}$ can occur, from any fixed direction in the inertial frame, randomly chosen at the start of any new target plateau along the 20 second episodes that we are observing. We take maxima and averages of these measures on 100 such queries as before.

The PIDs and the neural nets exhibit the same kind of minor loss of performance, and the nominal trained neural nets are still far superior to the two PIDs.

6.4 Performance of non-nominal-trained networks

6.4.1 Training under partial motor failures. In what follows, we train the attitude controller to sustain partial motor failures adding the magnitude of the power loss (1 extra dimension) to the observation states discussed in Section 3.1. We report the performance measures obtained in the non-nominal case in Table 4. The concern one may have is that, training the neural net in more various conditions (nominal and non-nominal), the resulting controller may exhibit lower performance. We thus report the same performance measures for neural nets trained with potential motor failures, in nominal situations, e.g. when no power loss happens, see Table 5

We see that we still achieve much better performance than PIDs, but that we are only similar and even slightly worse than the neural nets trained in nominal conditions, both in nominal conditions, compare Table 5 to Table 1 and in non-nominal conditions, compare Table 4 to Table 3. Understanding this non intuitive behaviour and improving the training in this case is left for future work.

6.4.2 Training under wind gusts. In what follows, we train the attitude controller to sustain wind gusts up to 10 $\mathrm{m.s}^{-1}$ in any direction, adding to the observation states we discussed in Section 3.1 the wind gust magnitude and directions (4 dimensions more) plus the linear velocities of the quadcopter (u, v and w, 3 dimensions more) since they are necessary for determining the relative wind velocity.

We report the performance measures that we get in the non-nominal case in Table 6 and in the nominal case in Table 7.

We see that, the SAC and DDPG controller trained with potential wind gusts still behave about as well as the nominal controller, compare Table 7 to Table 1. Surprisingly, the best (SAC) network behaves slightly worse than the nominal-trained SAC network under wind gusts, compare Table 6 to Table 3, where we can see a slight drop of performance in e.g. *OK off.* and *OK overshoot*: it does not seem to be able to learn correctly how to stay close enough to the target plateau, in some cases.

7 CONCLUSION

We have presented a complete study of learned attitude controls for a quadcopter using reinforcement learning. In particular we extend previous results by modeling partial motor failure as well as wind gusts, and generating extensive tests of various network architectures, training algorithms and hyperparameters using a flexible and robust experimental platform. We also present a precise evaluation mechanism based on robust signal temporal logic observers, which allows us to characterize the best options for training attitude controllers. Results show that learned controllers exhibit high quality over a range of query signals, and are more robust to perturbations than PID controllers.

mode	algo	OK rising t.	OK off.	OK overshoot	avg rising t.	avg off.	avg overshoot	max rising t.	max off.	max overshoot
windgust	sac	98.50	97.40	98.48	0.09	0.41	0.22	0.17	6.78	3.81
windgust	pid2	70.32	0.65	100.00	0.41	20.16	0.00	0.48	24.25	0.00
windgust	pid1	6.55	2.97	94.28	0.33	60.23	4.43	0.38	146.00	57.65
saturation	sac	94.20	81.58	92.85	0.12	17.63	6.04	0.33	145.91	81.25
saturation	pid2	58.24	2.94	93.98	0.39	34.33	4.66	0.48	129.36	58.46
saturation	pid1	14.50	3.71	92.13	0.30	64.83	5.67	0.40	166.82	70.45

Table 3: Robustness of the best networks and PIDs in case of wind gusts and motor saturation (all in % except rising t. in seconds)

algo	OK rising t.	OK off.	OK overshoot	avg rising t.	avg off.	avg overshoot	max rising t.	max off.	max overshoot
ddpg	89.40	73.82	90.53	0.13	20.61	6.84	0.37	156.87	87.98
sac	90.93	79.00	90.45	0.12	20.36	7.18	0.35	164.34	95.79

Table 4: Best networks trained for partial motor failures, tested under potential motor failures situations (all in % except rising t. in seconds)

algo	OK rising t.	OK off.	OK overshoot	avg rising t.	avg off.	avg overshoot	max rising t.	max off.	max overshoot
sac	95.35	95.42	97.47	0.09	0.66	0.34	0.22	9.35	5.30
ddpg	94.21	95.17	94.69	0.09	0.83	0.80	0.21	11.29	10.51

Table 5: Performance of best networks trained with potential motor failures, and tested in nominal situations (all in % except rising t. in seconds)

algo	OK rising t.	OK off.	OK overshoot	avg rising t.	avg off.	avg overshoot	max rising t.	max off.	max overshoot
ddpg	99.13	90.04	95.28	0.09	1.75	0.79	0.21	17.46	11.40
sac	97.67	89.58	92.97	0.10	1.52	1.01	0.28	13.91	11.65

Table 6: Best networks trained for wind gusts conditions, tested under wind gusts conditions (all in % except rising t. in seconds)

algo	OK rising t.	OK off.	OK overshoot	avg rising t.	avg off.	avg overshoot	max rising t.	max off.	max overshoot
ddpg	96.53	91.43	94.83	0.09	1.76	0.96	0.23	17.47	12.41
sac	95.96	97.09	96.77	0.09	0.42	0.45	0.21	5.57	6.34

Table 7: Best networks trained for wind gusts conditions, tested in nominal conditions (all in % except rising t. in seconds)

The immediate next step will be to start using STL-derived reward signals during training on the most promising architectures, and try to improve training under non-nominal situations.

Finally, because we use an explicit ODE model, we can hope to discuss formal reachability properties of the complete controlled system, using or elaborating on approaches such as [21] and [31].

REFERENCES

[1] Bazel Container Image Rules. https://github.com/bazelbuild/rules_docker.
[2] Bazel Documentation. https://docs.bazel.build/.
[3] Bazel Kubernetes Rules. https://github.com/bazelbuild/rules_k8s.
[4] Kubernetes Documentation. https://kubernetes.io/docs/reference/.
[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.
[6] Houssam Abbas, Yash Vardhan Pant, and Rahul Mangharam. Temporal logic robustness for general signal classes. In Necmiye Ozay and Pavithra Prabhakar, editors, *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 45–56. ACM, 2019.
[7] Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015,*

San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II, volume 9207 of *Lecture Notes in Computer Science*, pages 356–374. Springer, 2015.

[8] Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12-14, 2016*, pages 6565–6570. IEEE, 2016.

[9] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2669–2678. AAAI Press, 2018.

[10] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *CoRR*, abs/1611.01491, 2016.

[11] Alexey Bakhirkin and Nicolas Basset. Specification and efficient monitoring beyond STL. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 11428 of *Lecture Notes in Computer Science*, pages 79–97. Springer, 2019.

[12] Moses Bangura and Robert Mahony. Nonlinear dynamic modeling for high performance control of a quadrotor. In *Australasian Conference on Robotics and Automation*, 2012.

[13] Osbert Bastani. Safe reinforcement learning via online shielding. *CoRR*, abs/1905.10691, 2019.

[14] Osbert Bastani. Safe reinforcement learning with nonlinear dynamics via model predictive shielding, 2020.

[15] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, January 2002.

[16] D.P. Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific optimization and computation series. Athena Scientific, 2019.

[17] Lukas Bjarre. Learning for quadcopter control, December 2019.

[18] Lubos Brim, Petr Dluhos, David Safránek, and Tomas Vejpustek. Stl: Extending signal temporal logic with signal-value freezing operator. *Inf. Comput.*, 236:52–67, 2014.

[19] M. Deisenroth and C. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[20] Alexandre Donzé. On signal temporal logic. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings*, volume 8174 of *Lecture Notes in Computer Science*, pages 382–383. Springer, 2013.

[21] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, Montreal, QC, Canada, April 16-18, 2019. ACM, New York, NY, USA. https://doi.org/10.1145/33025043313351.

[22] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.

[23] Fan Fei, Zhan Tu, and Xinyan Deng. Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyberphysical attacks. In *ICRA*, 2020.

[24] James Ferlez and Yasser Shoukry. Aren: assured relu NN architecture for model predictive control of LTI systems. In *HSCC*, pages 6:1–6:11. ACM, 2020.

[25] James Ferlez, Xiaowu Sun, and Yasser Shoukry. Two-level lattice neural network architectures for control of nonlinear systems. *CoRR*, abs/2004.09628, 2020.

[26] Förster, Julian. System Identification of the Crazyflie 2.0 Nano Quadrocopter. B.S. Thesis, Institute for Dynamic Systems and Control, Swiss Federal Institute of Technology (ETH) Zurich, August 2015.

[27] S Fujimoto, H van Hoof, D Meger, et al. Addressing function approximation error in actor-critic methods. *Proceedings of Machine Learning Research*, 80, 2018.

[28] Qitong Gao, Davood Hajinezhad, Yan Zhang, Yiannis Kantaros, and Michael M. Zavlanos. Reduced variance deep reinforcement learning with temporal logic specifications. In Xue Liu, Paulo Tabuada, Miroslav Pajic, and Linda Bushnell, editors, *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 237–248. ACM, 2019.

[29] Maor Gaon and Ronen I. Brafman. Reinforcement learning with non-markovian rewards. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3980–3987. AAAI Press, 2020.

[30] Yann Gilpin, Vince Kurtz, and Hai Lin. A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control. Syst. Lett.*, 5(1):241–246, 2021.

[31] Eric Goubault and Sylvie Putot. Inner and Outer Reachability for the Verification of Control Systems. *HSCC*, April 2019.

[32] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018.

[33] Iman Haghighi, Noushin Mehdipour, Ezio Bartocci, and Calin Belta. Control from signal temporal logic specifications with smooth cumulative quantitative semantics. In *58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019*, pages 4361–4366. IEEE, 2019.

[34] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J. Pappas, and Insup Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019*, pages 5338–5343. IEEE, 2019.

[35] Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Towards verifiable and safe model-free reinforcement learning. In Nicola Gigante, Federico Mari, and Andrea Orlandini, editors, *Proceedings of the 1st Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, co-located with the 18th International Conference of the Italian Association for Artificial Intelligence, OVERLAY@AI*IA 2019, Rende, Italy, November 19-20, 2019*, volume 2509 of *CEUR Workshop Proceedings*, page 1. CEUR-WS.org, 2019.

[36] D. Haziza, J. Rapin, and G. Synnaeve. Hiplot, interactive high-dimensionality plots. https://github.com/facebookresearch/hiplot, 2020.

[37] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable Baselines. https://github.com/hill-a/stable-baselines, 2018.

[38] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, pages 1–1, 06 2017.

[39] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. *CoRR*, abs/2006.05768, 2020.

[40] William Koch, Rcihard West Renato Mancuso, and Azer Bestavros. Reinforcement Learning for UAV Attitude Control. *ACM Trans. Cyber-Phys. Syst.*, 3, 2, Article 22, (February 2019).

[41] Tim Koning. Developing a self-learning drone, april 2020.

[42] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

[43] Xiao Li and Calin Belta. Temporal logic guided safe reinforcement learning using control barrier functions, 2019.

[44] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016.

[45] Lars Lindemann and Dimos V. Dimarogonas. Control barrier functions for signal temporal logic tasks. *IEEE Control. Syst. Lett.*, 3(1):96–101, 2019.

[46] Sergio Lucia and Benjamin Karg. A deep learning-based approach to robust nonlinear model predictive control. *IFAC-PapersOnLine*, 51(20):511 – 516, 2018. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018.

[47] Carlos Luis and Jérôme Le Ny. Design of a Trajectory Tracking Controller for a Nanoquadcopter. Technical report, Mobile Robotics and Autonomous Systems Laboratory, Polytechnique Montreal, August 2016.

[48] Noushin Mehdipour, Cristian Ioan Vasile, and Calin Belta. Arithmetic-geometric mean robustness for control from signal temporal logic specifications. In *2019 American Control Conference, ACC 2019, Philadelphia, PA, USA, July 10-12, 2019*, pages 1690–1695. IEEE, 2019.

[49] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A. Preiss, Nora Ayanian, and Gaurav S. Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. *CoRR*, abs/1903.04628, 2019.

[50] Arnab Nilim and Laurent El Ghaoui. *Robust Markov Decision Processes with Uncertain Transition Matrices*. PhD thesis, USA, 2004. AAI3165509.

[51] C. Poussot-Vassal, F. Demourant, A. Lepage, and D. Le Bihan. Gust load alleviation: Identification, control, and wind tunnel testing of a 2-d aeroelastic airfoil. *IEEE Transactions on Control Systems Technology*, 25(5):1736–1749, Sep. 2017.

[52] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[53] Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2rl: Deep dense architectures in reinforcement learning, 2020.

[54] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, 1992.

[55] Min Wen, Ruediger Ehlers, and Ufuk Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 4983–4990. IEEE, 2015.

[56] Jaehyun Yoo, Dohyun Jang, H Jin Kim, and Karl H Johansson. Hybrid reinforcement learning control for a micro quadrotor flight. *IEEE Control Systems Letters*, 5(2):505–510, 2020.

[57] Wenbo Zhang and Osbert Bastani. MAMPS: safe multi-agent reinforcement learning via model predictive shielding. *CoRR*, abs/1910.12639, 2019.

[58] Zhenya Zhang, Ichiro Hasuo, and Paolo Arcaini. Multi-armed bandits for boolean connectives in hybrid system falsification. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 401–420. Springer, 2019.

A PARAMETERS

The physical and constant parameters for the crazyflie are obtained by merging data from [47] and [26].

Param	Description	Value	Unit
I_x	Inertia about x-axis	$1.657\,171 \times 10^{-5}$	$\mathrm{kg \cdot m^2}$
I_y	Inertia about y-axis	$1.665\,560\,2 \times 10^{-5}$	$\mathrm{kg \cdot m^2}$
I_z	Inertia about z-axis	$2.926\,165\,2 \times 10^{-5}$	$\mathrm{kg \cdot m^2}$
m	Mass	0.028	kg
g	Gravity	9.81	$\mathrm{m \cdot s^{-2}}$
C_T	Thrust Coefficient	1.285×10^{-8}	$\mathrm{N \cdot rad^{-2} \cdot s^2}$
C_D	Torque Coefficient	7.645×10^{-11}	$\mathrm{N \cdot rad^{-2} \cdot s^2}$
C_1	PWM to Ω factor	0.040 765 21	-
C_2	PWM to Ω bias	380.8359	-
h	z rotor wrt CoG	0.005	m
d	Arm length	$0.046/\sqrt{2}$	m
p_{max}	Maximum motor PWM	65 535	-

Table 8: Parameters for the model

The drag coefficients are [26]:

$$K = \begin{pmatrix} -9.1785 & 0 & 0 \\ 0 & -9.1785 & 0 \\ 0 & 0 & -10.311 \end{pmatrix} 10^{-7} kg.rad^{-1}$$

The maximal magnitudes of states are given in Table 9:

Variable	Unit	Lower Bound	Higher Bound
z	m	-1000	+inf
u	$\mathrm{m \cdot s^{-1}}$	-30	30
v	$\mathrm{m \cdot s^{-1}}$	-30	30
w	$\mathrm{m \cdot s^{-1}}$	-30	30
ϕ	rad	$-\pi$	π
θ	rad	$-\pi$	π
ψ	rad	$-\pi$	π
p	$\mathrm{rad \cdot s^{-1}}$	-5π	5π
q	$\mathrm{rad \cdot s^{-1}}$	-5π	5π
r	$\mathrm{rad \cdot s^{-1}}$	-5π	5π
cmd_ϕ	PWM	-400	400
cmd_θ	PWM	-400	400
cmd_ψ	PWM	-1000	1000
F	N	0	52428

Table 9: State and action space bounds

The base PID1 we have been comparing our controller to is given in [31]:

$$\begin{cases} thrust = 1000\big(25(2(z_{sp} - z) - w) \\ \qquad\qquad + 15\int(2(z_{sp} - z) - w)\,\mathrm{d}t\big) + 36000 \\ cmd_\phi = 250(p_{sp} - p) + 500\int(p_{sp} - p)\,\mathrm{d}t \\ cmd_\theta = 250(q_{sp} - q) + 500\int(q_{sp} - q)\,\mathrm{d}t \\ cmd_\psi = 120(r_{sp} - r) + 16.7\int(r_{sp} - r)\,\mathrm{d}t \end{cases} \tag{15}$$