A few lessons learned in reinforcement learning for quadcopter attitude control

Nicola Bernini, Mikhail Bessa, Rémi Delmas, Arthur Gold, Eric Goubault, Romain Pennec, Sylvie Putot, François Sillion

A Few Lessons Learned in Reinforcement Learning for Quadcopter Attitude Control

Nicola Bernini
Uber ATCP, Paris, France

Mikhail Bessa
Uber ATCP, Paris, France

Rémi Delmas
Uber ATCP, Paris, France

Arthur Gold
Uber ATCP, Paris, France
Paris, France

Eric Goubault
LIX, Ecole polytechnique, CNRS,
IP-Paris, Palaiseau, France

Romain Pennec
Uber ATCP, Paris, France

Sylvie Putot
LIX, Ecole polytechnique, CNRS,
IP-Paris, Palaiseau, France

François Sillion
Uber ATCP, Paris, France

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. The end goal is to train provably safe controllers, by improving both training and 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.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; **Robotic control**; • **Software and its engineering** → *Formal methods*; • **Theory of computation** → *Modal and temporal logics*; • **Computing methodologies** → **Computational control theory**; **Reinforcement learning**.

ACM Reference Format:
Nicola Bernini, Mikhail Bessa, Rémi Delmas, Arthur Gold, Eric Goubault, Romain Pennec, Sylvie Putot, and François Sillion. 2021. A Few Lessons Learned in Reinforcement Learning for Quadcopter Attitude Control. In *24th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '21), May 19–21, 2021, Nashville, TN, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447928.3456707

1 INTRODUCTION

Neural net based methods have had numerous successes in drone control, over the last few years, using privileged learning [27] or reinforcement learning [26]. 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 guarantees we can reach.

We concentrate here on low-level controls, and more specifically attitude control for quadcopters. These controllers 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 years, with continuous state and action spaces training algorithms such as SAC [23] and TD3 [19].

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 an attitude controller for the crazyflie 2.0 [18] quadcopter. We first present a non-linear ODE model for simulating the dynamics of a quadcopter, Section 2, 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, Section 3, discuss performance evaluation using a robust signal temporal logic, Section 4, and describe our experimental setup, Section 5. Finally we discuss experimental results, Section 6 and Section 7.

Claims. In this paper,

(1) we develop a neural-net based control study case, after modeling a quadcopter's dynamics, including aerodynamic effects and partial power loss on motors, Section 2

(2) we discuss the effect of the chosen training algorithm, neural net architecture, reduced observable state spaces and hyper-parameters on the performance of the controller, and on the RL training process, Section 3 and 6
(3) we present our experimental platform, which allowed us to compare more than 16,000 potential architectures, training algorithms and parameter choices, Section 5
(4) we develop Signal Temporal Logic observers to assess controller performance in a precise manner, Section 4
(5) we demonstrate high-quality attitude control using RL, for a relevant set of queries, making sense for a real aircraft, Section 6
(6) we show these controllers have a certain built-in robustness in non-nominal cases, with respect to partial failures of actuators and perturbations such as wind gusts, Section 6.3. Trying to train networks in non-nominal and nominal cases does not bring better performance so far, Section 6.4

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

RL in control. Reinforcement learning in control has been advertised, since [45], 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. [8].

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 [30], a learning-based model predictive control algorithm has been used to synthesize a low level controller. In [47], a hybrid approach is proposed, combining the model based algorithm PILCO [12] 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 [18]. More specifically we concentrate on actor-critic learning which has undergone massive improvements over the last few years with DDPG [32], SAC [23], TD3 [19], and compare it with the popular PPO method [42].

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. [20].

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 [38], 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 [28], which serves as the basis of our work. We improve the results of [28] 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 [37], [16], [29] and [10].

In [37], the goal is to stabilize a quadcopter in hover mode, from various initial conditions. The authors also consider perturbations to the dynamics, which are different from ours: motor lag and noise on sensors. In [16], the objective is to control a quadcopter under cyber-attacks targeting its localization sensors 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 [29] 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 by discussing the simplification of the observed states, then by more rigorously defining observation metrics for offsets and overshoots. Finally, in [10], 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 a controller that 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.

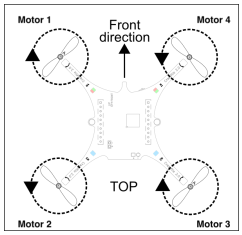
Signal Temporal Logics. Temporal logics with quantitative semantics such as Metric Interval Temporal Logic (MITL) [15], Signal Temporal Logic (STL) [13] ..., have been recently studied in relation with reinforcement learning. Robust interpretation yields a real number indicative of the distance to the falsification boundary. STL has seen numerous extensions improving expressiveness and signal classes [1, 2, 5, 11] as well as smooth differentiable semantics [21, 24, 36]. STL usages are varied: In [3], Q-learning is used to train a policy maximizing both the probability of satisfaction and the expected robustness of a given STL specification; In [31] the authors derive barrier functions from robust temporal logic specifications, either to modulate rewards during training or to control the switch from an optimal and potentially unsafe controller to a safe backup controller [33]. Considering our goal is to study a large hyper-parameter space for training controllers and we need to quantify controller performance rigorously, we used an expressive yet tractable variant of STL [5] to specify properties and assess trained controllers offline, separately after training.

2 MODELLING A CRAZYFLIE 2 QUADROTOR

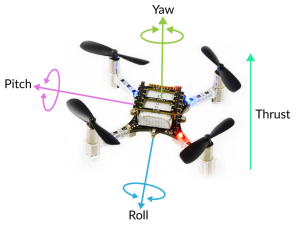
In this section, we detail the dynamical model of the crazyflie quadrotor [22, 35] and we augment it with partial motor failures and wind gusts modelling.

2.1 Nominal model

The Crazyflie 2.0 linear velocities (hence, positions) are controlled through the angular velocities and the angular velocities are controlled through rotor thrust differential. For instance, to increase the pitch rate q (see Figure 1b), $Motor_2$ and $Motor_3$ rotor speeds (see Figure 1a) should be higher than $Motor_1$ and $Motor_4$.



(a) Motors' controls



(b) Principal axes

Figure 1: Crazyflie 2.0 – source: http://www.bitcraze.io [9] CC BY-SA 3.0

As there is symmetry, it works similarly for the roll rate p (with $Motor_4$ and $Motors_3$ vs. $Motor_1$ and $Motor_2$ instead). However, the yaw rate r is controlled through the gyroscopic effect. To make the quadcopter rotate clockwise in the x-y plane, the rotor speeds of the clockwise rotating motors ($Motor_2$ and $Motor_4$) should be higher than those of the counterclockwise rotating ones ($Motor_1$ and $Motor_3$).

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 [35] lead to a 10-dimensional non-linear dynamical system:

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

where z is the vertical position in the world frame, the linear velocity of the center of gravity in the body-fixed frame with respect to the inertial frame is (u, v, w), the angular orientation is represented by the Euler angles (ϕ, θ, ψ) (roll, pitch, yaw), (p, q, r) (roll, pitch and yaw rate) is the attitude or angular velocity with respect to the body frame. We write c_x as a short for $cos(x)$, s_x for $sin(x)$ and t_x for $tan(x)$. Finally, 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 (PWM), the four commands $thrust$, cmd_ϕ, cmd_ψ and cmd_θ are used. They are deduced by a simple linear transformation from the PWM values to apply to each motor. PWMs are linked to rotation rates Ω: $\Omega = [\omega_1\ \omega_2\ \omega_3\ \omega_4]^\top = C_1 PWM + C_2$. and 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:

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

The parameters are those of [22, 35].

2.2 Motor failure

We suppose here that the quadcopter may experience a power loss on motor 1, 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 [18] 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 [6] 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), and Ω_i is the absolute value of the rotation rates (angular velocity, defined in Section 2.1) of the i-th rotor. 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 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 $F^a = (F^a_x, F^a_y, F^a_z)$ divided by m, and to moments of Equation (2), the aerodynamic moments $M^a = (M^a_x, M^a_y, M^a_z)$ 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:

$$\begin{cases} \dot{u} = rv - qw + s_\theta g + \frac{F^a_x}{m} & \dot{p} = \frac{I_y - I_z}{I_x} qr + \frac{1}{I_x}(M_x + M^a_x) \\ \dot{v} = -ru + pw - c_\theta s_\phi g + \frac{F^a_y}{m} & \dot{q} = \frac{I_z - I_x}{I_y} pr + \frac{1}{I_y}(M_y + M^a_y) \\ \dot{w} = qu - pv - c_\theta c_\phi g + \frac{F + F^a_z}{m} & \dot{r} = \frac{I_x - I_y}{I_z} pq + \frac{1}{I_z}(M_z + M^a_z) \end{cases} \quad (3)$$

2.3.2 Wind models. There are two main types of models in the litterature, represented by e.g. discrete wind gusts models and the stochastic von Kármán Gust or Dryden gust models. The discrete wind gusts models consists in a explicit and deterministic representation of wind gusts as half period cosine perturbations ([39], 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; its absolute velocity is given 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 [28], 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 [22].

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, with in particular higher proportional gains, Equation 4 :

$$\begin{cases} thrust = 3000(z_{sp} - z) \\ \qquad + 300 \int (z_{sp} - z)\,\mathrm{d}t - 500\dot{z} + 48500 \\ cmd_\phi = 1000(p_{sp} - p) + 400 \int (p_{sp} - p)\,\mathrm{d}t - 40\dot{p} \\ cmd_\theta = 1000(q_{sp} - q) + 400 \int (q_{sp} - q)\,\mathrm{d}t - 40\dot{q} \\ cmd_\psi = 2000(r_{sp} - r) + 1000 \int (r_{sp} - r)\,\mathrm{d}t - 100\dot{r} \end{cases} \quad (4)$$

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 rewards.

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 [29]). 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 (2), and also, p, q, r that are second order in the formulation of the angular rates, again in Equation (2). In all cases no simplification is performed on physical model presented in section 2, only observations of the learning agent are restricted. We do not consider here adding past information, classical in non Markovian environments [20], that has been used for attitude control in e.g. [28], 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 [28]): $r(s) = -max\left(0, min\left(1, \frac{1}{3\Omega_{max}} \|\Omega^* - \Omega\|\right)\right)$, 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 [4]. It is also known [7] 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 [17]. 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 $\gamma = 0.99$[1]) distance to the target angular rates, until the end of training.

Architectures that have been reported in the literature for similar problems are generally alike. In [37], the neural net is a Multi-Layer Perceptron (MLP) with two layers of 64 neurons each, and with tanh activation function. In [16], 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 [29], 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 [10], the resulting architecture is a two layers MLP with 128 neurons on each layer, and RELU activation function. A few authors argue that deeper networks should behave better, see e.g. [34]. 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 [32], SAC [23] and TD3 [19] are all off-policy, actor-critic methods, and more importantly, with continuous states and actions, which are generally considered to be better suited for control applications in robotics [45] (DDPG is used for instance in [16]). Because of its effectiveness in practice, observed by many authors, e.g. [28] for attitude control, we also compare with the on-policy Proximal Policy Optimisation [42], also used for similar applications in [37] and in [10].

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).

[1]All other parameters, learning rates in particular are the standard ones of Stable Baselines 2.7.0.

Query signals on such longer time windows could also be considered for training : [28] 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 [5], 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), together with the corresponding step size:

$$stable(q) = (On_{[0,T]}\ Max\ q) - (On_{[0,T]}\ Min\ q) < d \quad (5)$$

$$stableup(q) = (D^{\perp}_{-\epsilon}\ \neg stable(q)) \wedge stable(q) \quad (6)$$

$$up(q) = q - (D^{0}_{-\epsilon}\ q) > 0 \quad (7)$$

$$down(q) = q - (D^{0}_{-\epsilon}\ q) \leq 0 \quad (8)$$

$$step(q) = ite(stableup(q), q - D^{0}_{-\epsilon}q, 0) \quad (9)$$

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 $[T_1, T]$:

$$stableup(q) \wedge up(q) \implies On_{[0,T_1]}\ Max\ (x - q) < \alpha step(q) \tag{10}$$

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

$$stableup(q) \implies On_{[T_1,T]}\ Max\ \|x - q\| < \beta step(q) \tag{12}$$

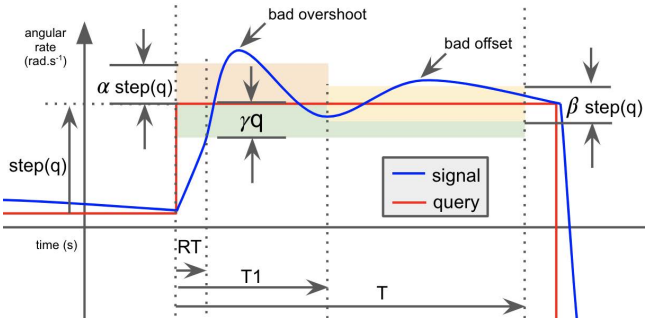


Figure 2: Property parameters T_1, T, RT, α, β, γ.

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

$$ite(stableup(q), t - (t\ U^{+\infty}_{[0,T]}\ \|(x - q)\| < \gamma q), +\infty) \tag{13}$$

Figure 2 illustrates the formalised notions and parameters.

Using observer code generated from these specifications, we compute statistics on 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 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

We have developed a platform[2] 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 itself based on Tensorflow, all of our code is in Python and we used Bazel 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.

[2]The full code is available as open source at https://github.com/uber-research/rl-controller-verification, the repeatability package associated with this paper being only a subset, available under https://github.com/uber-research/rl-controller-verification/tree/main/papers/hscc_2021/rep.

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 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 and the Bazel Kubernetes Rules 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. We used Hiplot [25] for browsing through the enormous number of parameters and data generated, for 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, and for understanding what correlations we have between controller performance and the way it has been trained.

6 EXPERIMENTS RESULTS

6.1 Performance metrics

Each controller is evaluated on a hundred episodes using the STL observers defined in Equations (10) to (13), 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.

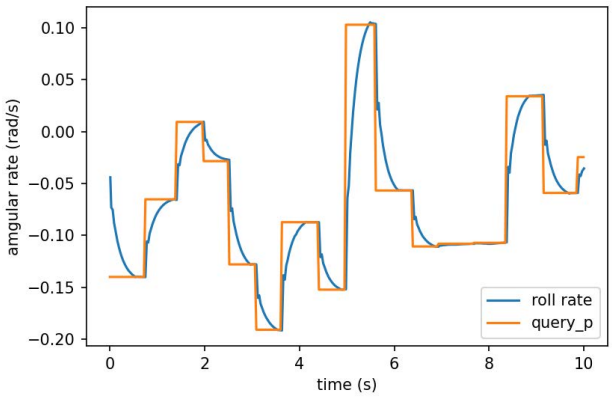


Figure 3: PID2 controller query tracking

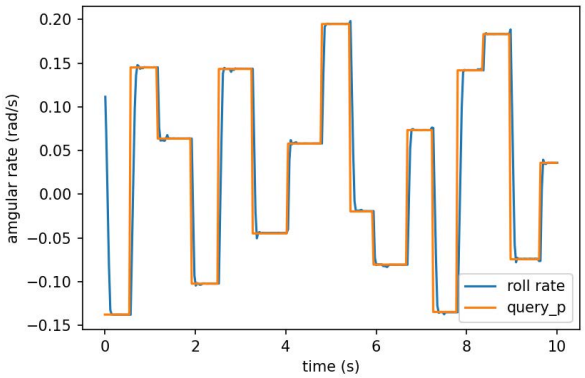


Figure 4: Neural controller behaviour (sac, 2 layers, 16 neurons per layer, 3M iterations)

Then, episode-level statistics are further averaged, yielding results presented in the tables, 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 lesser 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.

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

- 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 [28], 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 [28]) 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.

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, 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 to identify the subtle second-order effects of some of these states on angular rates.

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)

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)

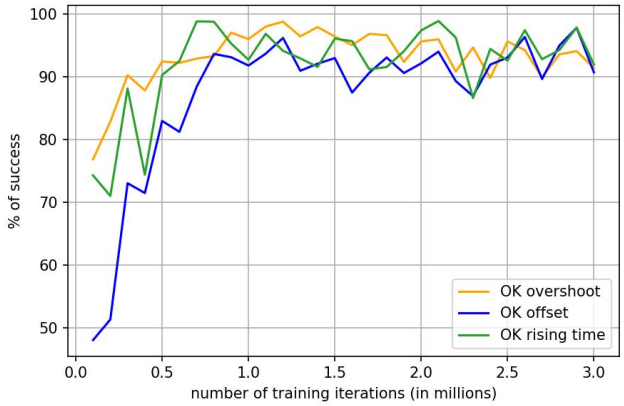


Figure 5: Performance of SAC 32x32 on dim 3 observation space trained neural nets w.r.t. the number of iterations

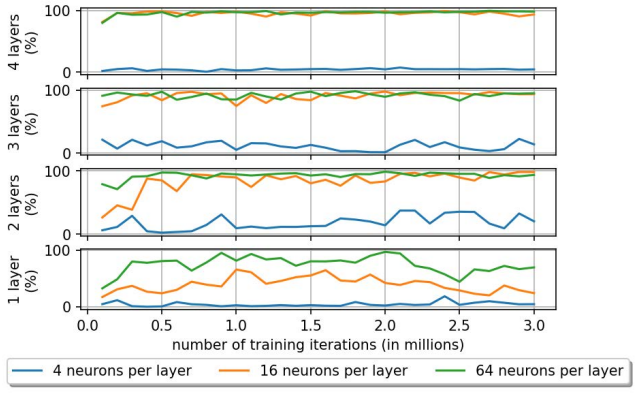


Figure 6: OK rising t. for our best SAC network wrt number of training iterations for different architectures

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 [44] 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 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 again the neural nets nor changing the gains of PIDs.

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 about 94% of the time, whereas even the best PID goes down to a success rate of less than 60%. Our network is even better when it comes to satisfying offset constraints (82%

of the time) whereas the PIDs almost never comply. Performance concerning overshoot is comparable, even though the PIDs are very slightly better, but only in cases where PIDs actually reach the target state, which is the case much less often. 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, Section 6.2) to wind gusts (see Section 2.3.1), without training again the neural nets nor changing the gains of PIDs. We present in Table 3 the same performance measures as the ones used in the nominal case, with random wind gusts of magnitude up to 10 $m.s^{-1}$, 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 $m.s^{-1}$ 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 LESSONS LEARNED

First, we observed that we should restrict to a "good" subspace of the (full quadcopter) states that is sufficiently low dimensional for efficient sampling and such that it avoids potentially spurious correlations, while still bringing sufficient information for learning. For instance, in the nominal case, the observation space (err_p, err_q, err_r) was found to be the optimal choice. Training depends of course on sampling data, that has to be done on representative data, and on sampling initial states in a large enough space. In order to do this, for better results, we developed a specific query generator, and we sampled initial states in quite large spaces.

SAC gives very good results as expected. It is most probably more efficient due to entropy regularization that partially cancels spurious correlations, but it still has to be confirmed in more general situations. A lesson for us was that TD3 was not behaving as well as expected. Our current guess is that TD3 suffers from too much bias on the Q-function estimation at some point in our training environment, or that TD3 needs much more iterations to converge in our case due to bad exploration performance. Recent papers have suggested that action clipping in TD3 can result in poor exploration performance on problems with bounded action spaces (actions on the boundary are too frequently sampled) which has been shown to be remedied by the entropy regularization of SAC or other output scaling and replay buffer sampling approaches that simulate entropy regularization, [40, 46]. Another newly documented [43] undesired behavior of TD3 is to have all Critics converge to a same point in parameter space and degenerate into single-Q-network performance. Without further experiments we cannot say if poor performance is due to action clipping, to critic diversity collapse, or both. Considering SAC works a lot better and also uses dual Q-networks like TD3, it seems more likely that clipping and bad exploration are to blame than diversity collapse.

It is actually hard to find good attitude controllers by RL, probably explaining why papers in this area generally only discuss one neural net controller: we had only 9 out of about 5000 controllers which comply with our specifications. The very last 5% performance seems to be very hard to get because of "spikes" we observed, due to spurious correlations in the fully connected neural net controllers we have been considering. We also note that small and rather shallow (two or three hidden layers) networks were observed to be best trained and to be behaving best for attitude control.

We also observed that there is some amount of robustness built in neural net controllers, suitably trained in nominal conditions, to certain non-nominal situations. We believe this is due to the fact that the controllers which are trained in the nominal case, are actually trained in many different states that appear in non-nominal situations, for the same neural net inputs (e.g. angular rate errors), by using a very wide distribution of initial states during training. Similar observations on robustness by training from wide initial state distributions were made in [41]. Finally, training neural net controllers to both nominal and non-nominal situations is not an easy endeavor and should be further studied. The difficulty lies in training on sufficiently many non-nominal data, as well as avoiding overfitting to non-nominal cases: reward distributions can become multi-modal and expectation maximization could be bad in such cases.

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 w.r.t. wind gusts and motor saturation (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 (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, tested nominally (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)

8 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.

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 [14] and [22].

Acknowledgements. Work supported by DGA AID project "Validation de drones et essaims de drones autonomes" and the academic chair "Complex Systems Engineering" hosted by Ecole Polytechnique, ENSTA Paris, Télécom Paris and sponsored by Thalès, Dassault Aviation, Naval Group, DGA, with the support of Fondation de l'Ecole Polytechnique and Fondation ParisTech.

REFERENCES

[1] Houssam Abbas, Yash Vardhan Pant, and Rahul Mangharam. Temporal logic robustness for general signal classes. In *HSCC*. ACM, 2019.
[2] Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In *CAV*, volume 9207 of *LNCS*. Springer, 2015.
[3] Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *CDC*.

IEEE, 2016.
[4] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *CoRR*, abs/1611.01491, 2016.
[5] Alexey Bakhirkin and Nicolas Basset. Specification and efficient monitoring beyond STL. In *TACAS*, volume 11428 of *LNCS*. Springer, 2019.
[6] Moses Bangura and Robert Mahony. Nonlinear dynamic modeling for high performance control of a quadrotor. In *ACRA*, 2012.
[7] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1), 2002.
[8] D.P. Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific optimization and computation series. 2019.
[9] Bitcraze. https://store.bitcraze.io/.
[10] Lukas Bjarre. Learning for quadcopter control, 2019.
[11] Lubos Brim, Petr Dluhos, David Safránek, and Tomas Vejpustek. STL: Extending signal temporal logic with signal-value freezing operator. *Inf. Comput.*, 236, 2014.
[12] M. Deisenroth and C. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *ICML*, 2011.
[13] Alexandre Donzé. On signal temporal logic. In *RV*, volume 8174 of *LNCS*. Springer, 2013.
[14] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *HSCC*. ACM, 2019.
[15] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42), 2009.
[16] Fan Fei, Zhan Tu, and Xinyan Deng. Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyberphysical attacks. In *ICRA*, 2020.
[17] James Ferlez, Xiaowu Sun, and Yasser Shoukry. Two-level lattice neural network architectures for control of nonlinear systems. *CoRR*, abs/2004.09628, 2020.
[18] Förster, Julian. System Identification of the Crazyflie 2.0 Nano Quadrocopter. B.S. Thesis, ETH Zurich, 2015.
[19] S Fujimoto, H van Hoof, D Meger, et al. Addressing function approximation error in actor-critic methods. *Proceedings of Machine Learning Research*, 80, 2018.
[20] Maor Gaon and Ronen I. Brafman. Reinforcement learning with non-markovian rewards. In *AAAI*, 2020.
[21] 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.
[22] Eric Goubault and Sylvie Putot. Inner and Outer Reachability for the Verification of Control Systems. *HSCC*, 2019.
[23] 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.
[24] Iman Haghighi, Noushin Mehdipour, Ezio Bartocci, and Calin Belta. Control from signal temporal logic specifications with smooth cumulative quantitative semantics. In *CDC*. IEEE, 2019.
[25] D. Haziza, J. Rapin, and G. Synnaeve. Hiplot, interactive high-dimensionality plots. https://github.com/facebookresearch/hiplot, 2020.
[26] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2017.
[27] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. *CoRR*, abs/2006.05768, 2020.
[28] William Koch, Rcihard West Renato Mancuso, and Azer Bestavros. Reinforcement Learning for UAV Attitude Control. *ACM Trans. Cyber-Phys. Syst.*, 3(2), 2019.
[29] Tim Koning. Developing a self-learning drone, 2020.
[30] 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.
[31] Xiao Li and Calin Belta. Temporal logic guided safe reinforcement learning using control barrier functions. 2019.
[32] 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.
[33] Lars Lindemann and Dimos V. Dimarogonas. Control barrier functions for signal temporal logic tasks. *IEEE Control. Syst. Lett.*, 3(1), 2019.
[34] Sergio Lucia and Benjamin Karg. A deep learning-based approach to robust nonlinear model predictive control. *IFAC-PapersOnLine*, 51(20), 2018. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018.
[35] 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, 2016.
[36] Noushin Mehdipour, Cristian Ioan Vasile, and Calin Belta. Arithmetic-geometric mean robustness for control from signal temporal logic specifications. In *ACC*. IEEE, 2019.
[37] 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.
[38] Arnab Nilim and Laurent El Ghaoui. *Robust Markov Decision Processes with Uncertain Transition Matrices*. PhD thesis, 2004.
[39] 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), 2017.
[40] Nirnai Rao, Elie Aljalbout, Axel Sauer, and Sami Haddadin. How to make Deep RL work in practice. 2020.
[41] Daniele Reda, Tianxin Tao, and Michiel van de Panne. Learning to locomote: Understanding how environment design matters for deep reinforcement learning. In *MIG*. ACM, 2020.
[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.
[43] Hassam Ullah Sheikh and Ladislau Bölöni. Reducing overestimation bias by increasing representation dissimilarity in ensemble based deep Q-learning. *arXiv cs.LG 2006.13823*, 2020.
[44] Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2rl: Deep dense architectures in reinforcement learning, 2020.
[45] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2), 1992.
[46] Che Wang, Yanqiu Wu, Quan Vuong, and Keith Ross. Striving for simplicity and performance in off-policy DRL: Output normalization and non-uniform sampling. In *ICML*, volume 119, 2020.
[47] 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), 2020.