2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

How to Model Your Crazyflie Brushless

Alexander Gräfe[1], Christoph Scherer[2], Wolfgang Hönig[2] and Sebastian Trimpe[1]

***Abstract*— The Crazyflie quadcopter is widely recognized as a leading platform for nano-quadcopter research. In early 2025, the Crazyflie Brushless was introduced, featuring brushless motors that provide around 50% more thrust compared to the brushed motors of its predecessor, the Crazyflie 2.1. This advancement has opened new opportunities for research in agile nano-quadcopter control. To support researchers utilizing this new platform, this work presents a dynamics model of the Crazyflie Brushless and identifies its key parameters. Through simulations and hardware analyses, we assess the accuracy of our model. We furthermore demonstrate its suitability for reinforcement learning applications by training an end-to-end neural network position controller and learning a backflip controller capable of executing two complete rotations with a vertical movement of just 1.8 meters. This showcases the model's ability to facilitate the learning of controllers and acrobatic maneuvers that successfully transfer from simulation to hardware. Utilizing this application, we investigate the impact of domain randomization on control performance, offering valuable insights into bridging the sim-to-real gap with the presented model. We have open-sourced the entire project, enabling users of the Crazyflie Brushless to swiftly implement and test their own controllers on an accurate simulation platform.**

I. INTRODUCTION

Over the past decade, Crazyflie (CF) quadcopters have become state-of-the-art in indoor nano-quadcopter research and teaching, both for single quadcopters and quadcopter swarms [1]–[6] . At the beginning of 2025, a new version, the Crazyflie 2.1 Brushless (CFB), was released. In contrast to the original CF 2.1 quadcopter that features brushed motors, the CFB is equipped with brushless motors and electric speed controllers (ESCs). Consequently, the CFB boasts a significantly higher thrust-to-weight ratio of around 3:1 compared to the 2:1 ratio of the CF 2.1. This increase in power positions the CFB as a valuable platform for research and teaching focused on agile control of nano-quadcopters.

To facilitate method development involving the CFB, this work derives a dynamics model of the CFB. We identify key parameters such as thrust curves and motor time constants and provide guidelines for manually re-identifying parameters like moments of inertia, which become necessary when modifications are made to the quadcopter's setup. Furthermore, to demonstrate the model's effectiveness for control design, we employ it to train two end-to-end neural network (NN) controllers entirely in simulation based on the identified model using reinforcement learning (RL). The first controller is capable of performing position control, which is comparable to the controllers embedded in the CFB's firmware. The second controller can execute up to two consecutive backflips with a vertical movement of $1.8\,\mathrm{m}$. We use this application of our model to assess the extent of necessary domain randomization, offering valuable insights into the sim-to-real gap.

In summary, our contributions are:

1) We derive a dynamics model of the CFB, including motor dynamics, and offer practical guidelines for manual parameter identification.
2) We evaluate the model in two ways: (a) by assessing its prediction accuracy and (b) by training end-to-end NN controllers for the CFB, deploying it on real Crazyflie Brushless hardware, and using it to analyze critical choices such as domain randomization.

The codebase, including the model, its parameters and a simulator based on JAX [7] and MuJoCo XLA (MJX) [8] for highly parallelized simulations on GPUs, is available under github.com/Data-Science-in-Mechanical-Engineering/CrazyflieBrushJAX. Additionally, an accompanying video demonstrating the hardware experiments is available at tiny.cc/CFBVideo.

The remainder of this work is organized as follows. First, we review related research on modeling previous versions of the CF in Sec. II and give an overview of the components of the CF Brushless in Sec. III. Next, we derive the model and identify its parameters in Sec. IV. Subsequently, we evaluate the model's predictive performance in Sec. V and conduct experiments using end-to-end RL to explore the sim-to-real gap in Sec. VI.

II. RELATED WORK

This section reviews related work on modeling previous versions of the CF, referred to as CF 2.x. Additionally, it presents application for CF models, emphasizing the relevance of the CF platform in robotics research.

A. Crazyflie Modeling

Quadcopter dynamics are generally well understood and models feature varying levels of detail. These range from

This work has been supported by the German Federal Ministry of Research, Technology and Space (BMFTR) under the Robotics Institute Germany (RIG). The authors gratefully acknowledge the computing time provided to them at the NHR Center NHR4CES at RWTH Aachen University (project number p0021919). This is funded by the Federal Ministry of Education and Research, and the state governments participating on the basis of the resolutions of the GWK for national high performance computing at universities (www.nhr-verein.de/unsere-partner).

[1]Alexander Gräfe and Sebastian Trimpe are with the Institute for Data Science in Mechanical Engineering, RWTH Aachen University, Germany alexander.graefe@dsme.rwth-aachen.de, trimpe@dsme.rwth-aachen.de

[2]Christoph Scherer and Wolfgang Hönig are with the Intelligent Multi-Robot Coordination Lab, TU Berlin, Germany c.scherer@tu-berlin.de, hoenig@tu-berlin.de

simple models that treat the quadcopter as a rigid body subject to external forces and torques to more complex models incorporating learned residual dynamics and even full fidelity fluid simulators [9]–[13].

For the CF 2.x, several studies have derived models all with similar levels of detail [14]–[17]. In these models, the CF is represented as a rigid body influenced by motor forces and reactive torques. The motor forces and torques are either modeled using static functions [15], [16] or combined with a dynamic system [14], [17], where motor commands are processed through a first-order dynamic system to simulate motor dynamics.

Building on these models, several simulators have been developed. We highlight three of the most common and recent ones: PyBullet-drones [16], which utilizes PyBullet [18] to simulate multiple quadcopters for swarming simulations with vision support; CrazySim [19], based on Gazebo [20], offering software-in-the-loop simulations via CF firmware wrappers; and Crazyflow [21], currently under development for highly parallelized simulations of CF swarms using MJX.

The presented models and simulators focus on the CF 2.x and to our knowledge, no comparable dynamics model of the new CF is openly accessible. To bridge this gap, this work introduces both a model and simulator specifically for the CFB. We maintain a similar level of detail as existing models, while basing our simulator on MJX, akin to Crazyflow [21].

B. Crazyflie Model Use Cases

Existing CF models have found extensive application in domains such as control design and multi-robot systems. For instance, Socas et al. [4] used the model to design and evaluate periodic and event-based PID controllers, while Nguyen et al. [22] leveraged a linearized model to develop an onboard model predictive control (MPC) for dynamic flight. For multi-robot systems, Gräfe et al. [6], [23] evaluated a distributed MPC for swarms using PyBullet-drones [16] and Wahba et al. [5], [24] used the model to design collaborative transport algorithms of cable-suspended payloads.

The models have also been widely adopted for RL. Applications use RL to optimize PID gains [25], manage swarms with high-level commands via multi-agent RL [26] and learn end-to-end control policies [17], [27].

To demonstrate the capability of the CFB model proposed in this work, we use it to train an end-to-end NN controller with RL. This application serves as a test of model precision, as end-to-end learning is highly sensitive to model inaccuracies [28], [29]. Furthermore, it enables an evaluation of the domain randomization required for successful sim-to-real transfer, which serves as a metric for the model's accuracy.

III. BACKGROUND: CRAZYFLIE BRUSHLESS

This section provides an overview of the various components of the CFB as depicted in Fig. 1.

Hardware. In the largest parts, the CF 2.x and CFB are identical [31], [32]. They comprise two processors, one dedicated to 2.4 GHz communication and an application processor (STM32F405 ARM Cortex-M4 single core MCU

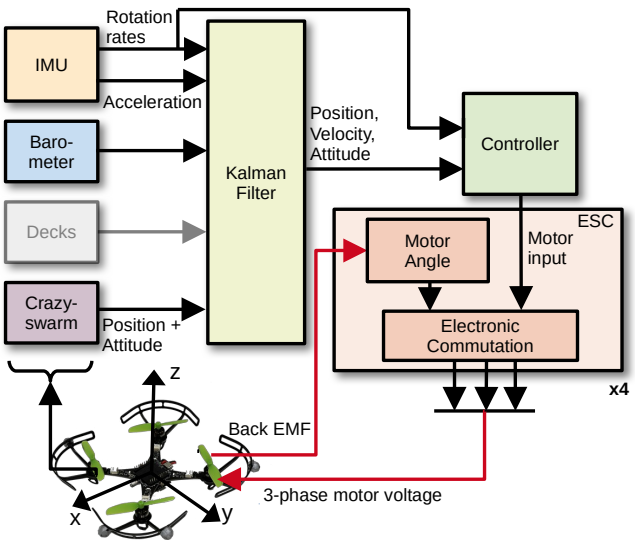


Fig. 1: Overview of the different components of the CFB [30], [31]. *A Kalman filter estimates position, velocities and attitude from the data of different sensors. The controller uses these in combination with rotation rates of the inertial measurement unit (IMU) to give motor inputs to the ESCs, which perform the electronic commutation of the brushless motors using the motor angle measured via the back electromotive force (EMF).*

with 168MHz, 192 kB SRAM and 1 MB flash) for processing tasks. Additionally, they feature an IMU equipped with a 3-axis accelerometer and gyroscope (BMI088), along with a pressure sensor (BMP388). They can be expanded using various decks that provide additional sensors.

The primary difference between the CF 2.x and the CFB lies in their motor systems. The CF 2.x utilizes brushed motors, which are controlled by a transistor. In contrast, the CFB employs four ESCs to control its brushless motors. Each ESC consists of an EFM8BB21 MCU and three half-bridges.

Software. The application processor manages the entire state estimation, control and planning pipeline. It employs an extended Kalman filter [33]–[35], operating at 100 Hz, to fuse the measurements from the IMU, pressure sensor, other decks and, if available, motion capture system information into a state estimate. Subsequently, the controller computes motor commands which are proportional to the motor voltage. These commands are transmitted to four ESCs, which perform the electronic commutation and apply the desired motor voltage. For this, each ESC measures the motor's angle using the induced voltage of the rotating motor. It then computes and applies the three-phase motor voltages correspondingly.

IV. SYSTEM MODEL

In this section, we first present the dynamics model in Sec. IV-A and then its parameters in Sec. IV-B. Finally, in Sec. IV-C, we give an overview of our simulator implementation based on MJX.

A. First-principle model

When modeling the CFB, we preserve the same level of detail as existing models for the CF 2.x [14]–[17], [19] (cf. Sec. II-A). In Sec. V, we enumerate unmodeled effects that could become significant in research exploring the limits of CFB dynamics.

First, we present the comprehensive set of equations that describe the quadcopter's dynamics. Subsequently, we will examine each equation and its parameters in a step-by-step manner. Our model follows common quadcopter models presented, e.g., in works [9]–[11], [14]–[17], [19], [29], [36]. We describe the CFB using a state-space model $\dot{x} = f(x, u)$, where $x \in \mathbb{R}^{17}$ is the state and $u \in [0, 1]^{4\times1}$ are the motor commands. In detail, the model is

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{q} \\ \dot{\omega} \\ \dot{\Omega} \end{bmatrix} = \begin{bmatrix} v \\ \frac{1}{m} F_\mathrm{m}(\Omega, q) - g e_\mathrm{z} \\ \frac{1}{2}\, q \otimes \begin{bmatrix} 0 & \omega \end{bmatrix}^T \\ J^{-1}(\tau_\mathrm{m}(\Omega, u) - \omega \times (J\omega)) \\ -\frac{1}{T}\Omega + \frac{K}{T} u \end{bmatrix} \begin{matrix} (1) \\ (2) \\ (3) \\ (4) \\ (5) \end{matrix}, \tag{}$$

where $v \in \mathbb{R}^3$ is the velocity, $q \in S^3$ are the quaternions representing orientation, $\omega \in \mathbb{R}^3$ is the angular velocity in the body frame, $\Omega \in \mathbb{R}^4$ are the rotation speeds of the motors and $g = 9.81\,\mathrm{m\,s^{-2}}$. The coordinate system is depicted in Fig. 1 and roll, pitch and yaw rates rotate around x, y and z axes respectively. These rotations are deemed positive when counter-clockwise. All parameters can be found in Table I. We now go through the equations explaining their parts.

Rotational and Translational Dynamics (1)–(4). The translational dynamics of the quadcopter are a standard model, summing up the influence of the motor's thrust (F_m, τ_m). The thrust that the motors generate is a nonlinear function of the motor rotational speed

$$F_\mathrm{m}(\Omega, q) = R(q) e_\mathrm{z} \left[1, 1, 1, 1\right] F_\mathrm{thrust}(\Omega), \tag{6}$$

where $R(q) \in \mathbb{R}^{3\times3}$ is the rotation matrix and $F_\mathrm{thrust}(\Omega) : \mathbb{R}^{4\times1} \to \mathbb{R}^{4\times1}$ is an element-wise 3rd order polynomial with units $\mathrm{rad\,s^{-1}} \to \mathrm{N}$ (cf. Sec. IV-B).

The torques from the motors in pitch and roll direction are the thrusts of the motors times the orthogonal distance to the center of mass. The torque around the z-axis is the reactive torque of the motor, which is the sum of the inertia torque $J_\mathrm{m}\dot{\Omega}$ and the aerodynamic/friction torques on the motor τ_f

$$\tau_\mathrm{m}(\Omega, u) = \begin{bmatrix} L & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} F_\mathrm{thrust}(\Omega) \\ J_\mathrm{m}\dot{\Omega} + \tau_\mathrm{f}(\Omega) \end{bmatrix}, \tag{7}$$

where we measured the element-wise 3rd order polynomial $\tau_\mathrm{f}(\Omega) : \mathbb{R}^4 \to \mathbb{R}^4$ with dimension $\mathrm{rad\,s^{-1}} \to \mathrm{N\,m}$ (cf. Sec. IV-B) and

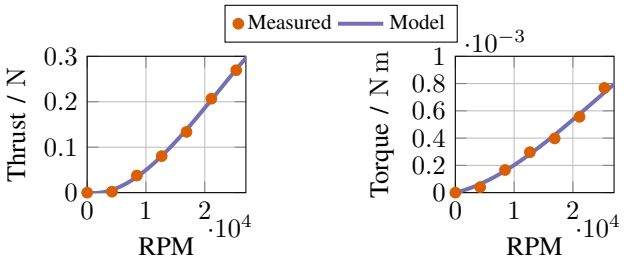
$$L = \begin{bmatrix} -\ell/2 & -\ell/2 & \ell/2 & \ell/2 \\ -\ell/2 & \ell/2 & \ell/2 & -\ell/2 \end{bmatrix}, \tag{8}$$

where ℓ is the width of the quadcopter.

While accounting for aerodynamic forces in the model could further enhance accuracy, accurately modeling these effects is challenging [10], [11], [37]. Our current results indicate that effective performance in low speed scenarios ($\leq 3\,\mathrm{m\,s^{-1}}$) can be achieved by solely modeling the rigid-body dynamics, motor dynamics and motor response.

TABLE I: Symbols, explanations and dimensions for the quadcopter model.

Symbol	Explanation	Dimension
v	Linear velocity	$\mathbb{R}^{3\times1}$ ($\mathrm{m\,s^{-1}}$)
q	Quaternion	S^3 (unitless)
ω	Angular velocity in body frame	$\mathbb{R}^{3\times1}$ ($\mathrm{rad\,s^{-1}}$)
Ω	motor rotation speed	$\mathbb{R}^{4\times1}$ ($\mathrm{rad\,s^{-1}}$)
u	Motor command	$[0, 1]^{4\times1}$ (unitless)
m	Quadcopter mass	$\mathbb{R}$ (kg)
ℓ	Quadcopter width	$\mathbb{R}$ (m)
J	Quadcopter moment of inertia	$\mathbb{R}^{3\times3}$ ($\mathrm{kgm^2}$)
T	motor time constant	$\mathbb{R}$ (s)
K	motor amplification factor	$\mathbb{R}$ ($\mathrm{rad\,s^{-1}}$)
J_m	motor moment of inertia	$\mathbb{R}$ ($\mathrm{kgm^2}$)



(a) Thrust (b) Torque

Fig. 2: Comparison between measured and model-predicted thrust and torque characteristics. *The propeller model accurately captures both thrust and torque behavior across the operating range.*

Motor Dynamics (5). The motor dynamics model summarizes the behavior of the dynamics of the motor rotational mass, the coils and the ESCs. However, developing a model of all this proves challenging, especially given the complicated behavior of the ESC's electronic commutation, rate limitation etc. We hence approximate the dynamics via a first-order linear system with time constant T and amplification K as done in [10], [11], [17], [29], [36].

B. System identification

We identify the parameters of the model in three different experiments. First, we measure the motor parameters with sensors measuring the motor's revolutions per minute (RPM). Then, we measure the thrust and reactive torques $F_\mathrm{a}(\Omega)$ and $F_\mathrm{thrust}(\Omega)$ and finally identify the remaining parameters (moment of inertia matrix) by fitting measured trajectories of the quadcopter to simulated ones.

Motor Dynamics Identification. We identify the motor system dynamics by recording the motor commands and the RPM during flight. While we can directly assess the motor commands in the firmware of the quadcopter, we measure the RPM using a custom deck with a QRD1114 infrared sensor and a reflective band beneath the propeller as described in [38]. We then fit the time constant T and amplification

TABLE II: Symbols, explanations and dimensions for the quadcopter model. *Values in parentheses are for the CFB without propeller guards.*

Symbol	Value
m	$44\,\mathrm{g}$ ($40\,\mathrm{g}$)
J	$\mathrm{diag}([3.3, 3.6, 5.9]) \cdot 10^{-5}\mathrm{kgm}^2$ ($\mathrm{diag}([1.8, 2.4, 3]) \cdot 10^{-5}$ kgm^2)
T	$50\,\mathrm{ms}$
K	$2.9 \cdot 10^3 \mathrm{rad\,s}^{-1}$
J_m	$0.5 \cdot 10^{-7}\mathrm{kgm}^2$
ℓ	$7.07\,\mathrm{cm}$

K to the measured curves leading to $T = 0.05\,\mathrm{s}$ and $K = 2900\,\mathrm{rad\,s}^{-1}$.

Thrust Measurements. We place the quadcopter on a force/torque measurement testbed we built using a scale and a rotating arm. We increase the motor command of only one motor in steps and once it has reached a steady state, we measure the force/torque. Afterwards, we fit a polynomial through the points measured leading to

$$F_\mathrm{thrust}(\Omega) = -0.23(\frac{\Omega}{\sigma})^3 + 0.562(\frac{\Omega}{\sigma})^2 - 0.043(\frac{\Omega}{\sigma}), \quad (9)$$

$$\tau_\mathrm{f}(\Omega) = 10^{-4}\left[-3.4(\frac{\Omega}{\sigma})^3 + 8.7(\frac{\Omega}{\sigma})^2 + 2.9(\frac{\Omega}{\sigma})\right] \quad (10)$$

with $\sigma = 2.9 \cdot 10^3$ for numerical stability.

Fitting Measured Trajectories. All remaining parameters are measured by manually fitting simulated trajectories to measured ones (Table II). We emphasize that especially the moments of inertia are dependent on the additional payload of the quadcopter, e.g., whether equipped with propeller guards, motion capture system markers and decks. The given ones should, however, give a good starting ground to quickly identify the parameters for the CFB in a specific application using the following procedures on a flying quadcopter.

For the inertias around x and y axis, the quadcopter should perform a maneuver where it rotates around its x and y axes. This can be achieved by swapping from a stabilizing controller to a policy that gives each motor different motor commands such that they create torques around the x and y axis. After a short amount of time (around $500\,\mathrm{ms}$), the quadcopter swaps back to the stabilizing controller to avoid crashing. This allows us to fit the moments of inertia around x and y axis by aligning the simulated and measured rotation rates around x and y axis, respectively.

Similarly, for the moment of inertia around z axis and J_m, we perform an experiment, where we keep the motor command of three motors low and give a high command to one, which causes the yaw rotation rate to rise fast. This yaw rotation rate can be used to fit the parameters.

C. MJX Simulator

We implement the derived model inside MJX [8]. Although the differential equation of the drone could also be solved using a suitable solver, MJX allows to reuse the simulator for complex tasks that involve e.g., contacts. MuJoCo itself takes care of the translational and rotational dynamics. We place the motor model outside of MuJoCo and solve it separately at each step. As the dynamics are linear, this can be done exactly. From Ω, we then calculate the thrust reactive torque of the motor and give this to the MuJoCo model as input. We simplify the dynamics of the Kalman filter, the sensor low-pass filters and the delay of communication with the motion capture system via Crazyswarm with a fixed delay of $8\,\mathrm{ms}$.

V. EVALUATION: MODEL ACCURACY

This section evaluates the predictive capabilities of the model by comparing simulated and measured trajectories.

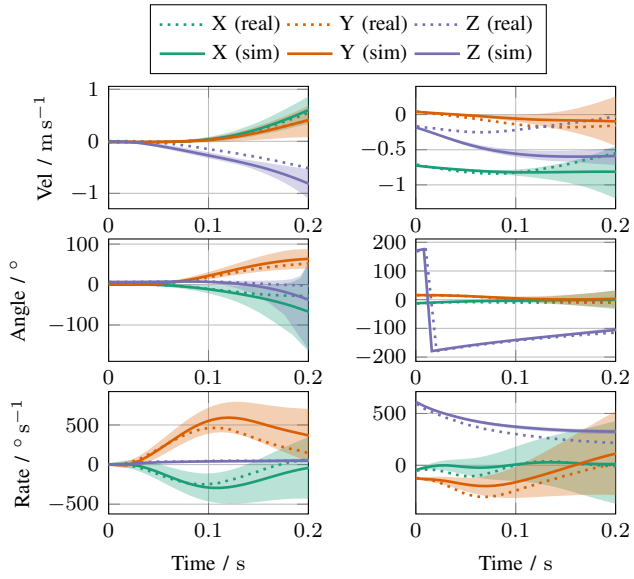
A. Method

We will evaluate the model on two different maneuvers flown by the CFB. First, a maneuver, where the CFB changes from hovering to a horizontal flight. And second, a maneuver, where we disturb the yaw orientation and let the CFB control itself back to the original position. During these maneuvers, we measure the estimated state of the CFB and its motor commands. The simulation then receives these motor commands along with the initial state and then generates the simulated trajectories. We did not use these maneuvers to fit model parameters. To also assess the effect of parameter uncertainties, we simulate the CFB 4096 times with different inertia and amplify/reduce the thrust and torque of the motors. The randomly sampled parameters deviate at maximum $2\,\%$ from the identified ones. Furthermore, we change the center of gravity by at maximum $2\,\mathrm{mm}$, which can be achieved by changing the matrix L and apply velocity and angular noise per simulation step ($4\,\mathrm{ms}$) of $0.001\,\mathrm{m\,s}^{-1}$ and $0.001\,\mathrm{rad\,s}^{-1}$.

B. Results

Fig. 3 demonstrates that our model can capture the behavior of the CFB in a horizon of $200\,\mathrm{ms}$. Over time, the predictions of the model increasingly deviate from the real behavior. We also noticed larger differences between measured and simulated velocities in z-direction especially during the second maneuver.

We suspect multiple sources for the remaining deviation of our model. First, uncertainties have an increasingly large effect over the prediction horizon. Within this context, we observed that inaccuracies in the center of gravity position exert a particularly large influence. Second, the measured values from the CFB are taken from its Kalman filter (cf. Sec. III) and might not necessarily represent the actual ground-truth values, they hence have their own errors and uncertainties. This is especially true for the velocity as there is no direct sensor for it (position, attitude and angular rates are measured by the motion capture system and the IMU). Third, our model neglects aerodynamics, particularly the complex aerodynamics of the motors, whose thrust changes during movement. Modeling these effects is non-trivial [10], [11] and represents an exciting topic for future work. Fourth,



(a) Maneuver 1 (b) Maneuver 2

Fig. 3: Comparison of hardware measurements (dotted lines) versus simulation (solid lines). *The simulation incorporated 4096 quadcopters with different mass, moment of inertia and thrust/torque scalings (differing at maximum 2 % from the nominal ones). The line in the middle denotes the mean, whereas the shaded areas the minimum and maximum. We excluded position in the plots as it is almost constant.*

during flight, we cannot measure the motor speeds and thus estimate their initial value, adding uncertainty. Other potential factors include disturbances, like subtle air movements.

To contextualize the performance of our model, we compared the PyBullet-drones model for the CF 2.1 [16] against real flight data from a CF 2.1 (Fig. 4). The results show that the PyBullet-drones model significantly deviates from the real behavior, demonstrating that our CFB model achieves substantially higher accuracy than this established baseline.

While the presented model captures the fundamental behavior of the CFB, some discrepancies remain. To address this, we investigate the effect of domain randomization on RL performance in the following. The required degree of domain randomization for successful sim-to-real transfer serves as an additional metric for evaluating the accuracy of the dynamics model and the uncertainties in its key parameters.

VI. EVALUATION: SIMULATION-BASED REINFORCEMENT LEARNING

The previous section assessed the model's accuracy. In this section, we demonstrate the model's suitability for learning end-to-end NN controllers by designing and evaluating a simulation-based RL pipeline. Additionally, we use this pipeline to examine the impact of domain randomization, uncertainties applied to model parameters during training, which provides insights into the sim-to-real gap. A video accompanying this section is available at tiny.cc/CFBVideo.

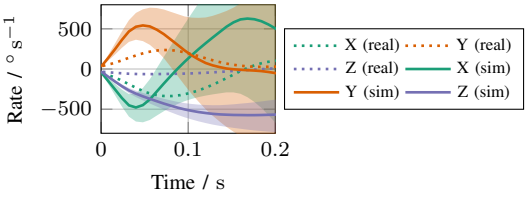


Fig. 4: Comparison of hardware measurements (dotted lines) versus simulation (solid lines) of angular rates for the CF 2.1. *We use the model parameters used in PyBullet-drones [16] for baselines comparisons.*

A. Method

We train quadcopters on two distinct tasks. First, we focus on training a quadcopter to fly and maintain a specified target position. Second, we design a controller specifically for executing backflips.

Baselines. We compare the learned target controller with the firmware's PID and Mellinger controller [30], [39], using parameters provided by the firmware.

Learning Algorithm. We employ proximal policy optimization (PPO) [40] for learning, utilizing BRAX's PPO implementation [41] and its interface to MJX allowing the entire training to run on the GPU. To assess the sim-to-real gap, no dedicated fine-tuning is performed on hardware.

Controller. The controller is a fully connected NN with four hidden layers, each containing 32 neurons. Although the CFB provides estimates of position, attitude, velocity and angular rates, it cannot measure motor rotation speeds Ω. Therefore, past estimates and actions are also included in the controller's input as a state representation.

Reward. We use sums of triangular shaped rewards [42]

$$\Lambda(e, e_{\max}) = \text{relu}(1 - |e/e_{\max}|), \tag{11}$$

where e is a deviation from a target value (e.g. $p - p_{\text{target}}$) and $e_{\max}$ is the maximum interval on which we want to give a reward. This function helps to normalize the reward making the RL training better conditioned. We also add a large negative reward in case of a failure (e.g., crash to the ground or too high angular velocities). The next paragraphs give the exact formulation for the specific controller.

Target Control. Goal of target control is to steer the quadcopter to the origin. We change the target position the quadcopter should fly to by subtracting it from the position estimate. We use the following reward function to encourage proximity to the target position while penalizing high rotation speeds, angular velocities and generated torques (to ensure smooth maneuvers)

$$\begin{aligned} r(x) = {} & \Lambda(||p||_2, 10\,\text{m}) + 0.05\Lambda(||v||_2, 10\,\text{m s}^{-1}) \\ & + 0.15\Lambda(||q - [1,0,0,0]||_2, 2) \qquad (12) \\ & + 0.15\Lambda(||\omega||_2, 10\,\text{rad s}^{-1}) \\ & + 0.02\Lambda(||Lu||_2, 0.01\,\text{N m}) - 300\text{fail}(x), \qquad (13) \end{aligned}$$

where fail is one when the quadcopter's velocity exceeds $10\,\text{m s}^{-1}$ or its angular velocity ω is above $30\,\text{rad s}^{-1}$. In

this case, we terminate the episode.

Backflips. For backflips, we use a multi-phase approach similar to [43]. First, we use the controller from the last section to accelerate the quadcopter upwards and then switch to a second NN controller (called flip-controller in the following) that controls the quadcopter to have a constant angular rate around its x-axis and a given velocity along its local frame's z-axis. After some time, we switch back to the stabilizing controller that catches the quadcopter and steers its back towards its starting position. The possibility to vary angular rate and velocity setpoints $v_{\text{target}}, \dot{\theta}_{\text{target}}$ and the time points on which we switch the controllers allows us to change the behavior of the drone during the backflip, generating looping of different sizes and speeds.

The NN controller for the second phase is trained via the following reward

$$\begin{aligned} r_{\text{flip}}(x) = 0.6\Lambda(||R(q)^T v - \begin{bmatrix}0 & 0 & v_{\text{target}}\end{bmatrix}^T ||_2, 20\,\text{m}\,\text{s}^{-1}) \\ + 0.15\Lambda(||R(q)[:,0] - \begin{bmatrix}0 & 1 & 0\end{bmatrix}^T ||_2, 3.0) \\ + 1.0\Lambda(||\omega - \begin{bmatrix}\dot{\theta}_{\text{target}} & 0 & 0\end{bmatrix} ||_2, 0.01\,\text{N}\,\text{m}) \\ - 300\text{fail}(x), \end{aligned} \tag{14}$$

where the first part rewards keeping the velocity, the second keeping the quadcopter's attitude close to its rotation axis and the third rewarding controlling it to the pitch rate.

Domain Randomization. We randomized key model parameters: mass ($\pm 10\,\%$), inertia and motor dynamics ($\pm 20\,\%$), motor thrusts and torques scalings ($\pm 20\,\%$) and center of gravity position ($\pm 1\,\text{cm}$), sampling from uniform distributions across these intervals. Additionally, we introduced random offsets in attitude measurements ($\pm 15°$) to simulate real-world misalignment between the drone's local coordinate frame, the global motion capture system frame and the direction of gravity.

We quantify the effect of domain randomization in Sec. VI-D by training NN controllers with varying magnitudes of randomization, defined as a factor for the interval size. A magnitude of 1.0 corresponds to the full intervals specified above, while 0.5 and 0.0 correspond to half-intervals and no randomization, respectively.

B. Evaluation: Target Control

Fig. 5 illustrates a maneuver executed by the CFB, flying between two positions $6\,\text{m}$ away. The quadcopter successfully reaches its target within $2\,\text{s}$, achieving a maximum velocity of around $5\,\text{m}\,\text{s}^{-1}$.

In Table III, we compare the learned controller for different domain randomization magnitudes with the PID and Mellinger controllers from the firmware. We conducted a hover test for $6\,\text{s}$, assessing mean distance to target position and motor command standard deviation to evaluate motor command smoothness.

The PID has the best positional accuracy, followed by the NN controller with domain randomization magnitude 1.0 and then the Mellinger controller. The NN controller performs slightly better in motor command smoothness.

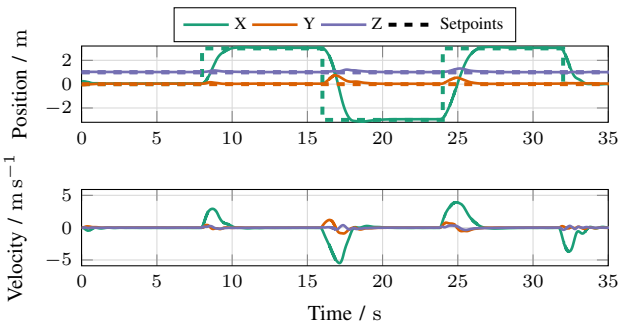


Fig. 5: Target controller flying back and forth between two points $6\,\text{m}$ apart. *The NN controller guides the CFB to its target with an accuracy of a few* cm *(cf. Table III), achieving velocities of approximately* $5\,\text{m}\,\text{s}^{-1}$ *during the maneuver.*

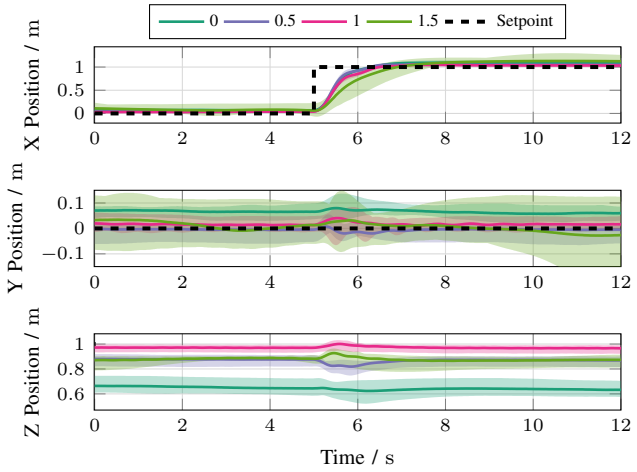


Fig. 6: Performance of the position controller with respect to domain randomization. *Each configuration represents a different magnitude of domain randomization trained over five random seeds to test the robustness of the trained policy.*

These results demonstrate the RL pipeline's ability to learn a controller using our model, whose stationary behavior is comparable to the firmware's controllers. We want to note that we used the PID and Mellinger controller's standard parameters and tuning of these controllers could potentially improve their performance.

C. Evaluation: Backflips

Single Backflips. Fig. 7 illustrates various backflips executed by the NN controllers. The shape of each backflip varies according to the setpoint velocity and roll rate. As the CFB cannot control its velocity when inverted, the velocity is not kept exactly. However, the adjustable setpoints still allow the users to customize the backflips to their preferences.

Double Backflips. Fig. 8 depicts the trajectory of a CFB executing a backflip with two rotations. This maneuver is performed requiring only $1.8\,\text{m}$ movement in z-direction. The quadcopter reduces its rotational velocity from $1000\,\text{deg/s}$ in $0.35\,\text{s}$ and its vertical velocity from $3.5\,\text{m}\,\text{s}^{-1}$

TABLE III: Comparison of the learned NN controllers for different domain randomization magnitudes (denoted in parentheses, cf. Sec. VI-A) against the PID and Mellinger controllers from the firmware. *The first column is the mean differences from setpoint position measured for* 5 s *and the second mean of motor command standard deviations (std) over the four motors.*

Controller	Mean Distance (mm)	Motor command std
NN (0)	284.76	0.0203
NN (0.5)	100.07	0.0245
NN (1.0)	43.16	0.0197
NN (1.5)	71.39	0.0548
PID	27.81	0.0226
Mellinger	54.68	0.0394

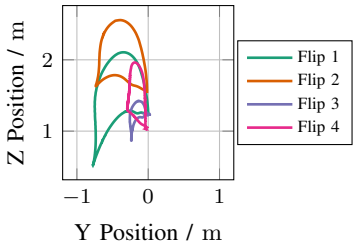


Fig. 7: Trajectory in YZ plane for four flip maneuvers. *Depending on the hyperparameters of the maneuver (more specifically duration of the liftoff and roll rate), the shape changes.*

to zero in 0.6 s, demonstrating the CFB's capacity for agile, high-thrust maneuvers.

In summary, RL using the presented model enables the CFB to execute high-performance aerial acrobatics.

D. Effect of Domain Randomization

To assess the impact of domain randomization, we trained the target controller using various randomization magnitudes. The results of these experiments are presented in Table III and Fig. 6.

Low magnitudes of domain randomization reduce positional accuracy, primarily in the z-direction. Nevertheless, even with low or no randomization, the drone maintains a level and stable flight. Conversely, excessively high domain randomization significantly degrades controller performance, as the increased demand for robustness comes at the expense of optimal performance. These findings suggest that the rotational dynamics are less sensitive to low domain randomization than the vertical dynamics. We hypothesize that this is due to slight discrepancies in mass and thrust between the model and the real quadcopter, combined with reduced controller robustness to these parameters when trained with limited randomization.

We note that the reward functions were not explicitly tuned for different randomization magnitudes. Such tuning could potentially improve performance. Alternatively, the loss of accuracy in the z-direction might be mitigated by

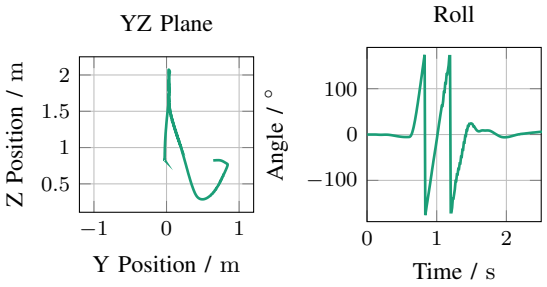


Fig. 8: Double backflip maneuver showing trajectory in YZ plane and roll angle over time.

removing mass compensation from the NN controller and instead compensating for it with a constant offset to the motor commands.

In summary, our experiments demonstrate that the presented model is accurate enough to successfully train NN controllers in simulation that can be successfully deployed on a real CFB. To enhance robustness against model-to-real-world discrepancies, we recommend domain randomization of ±10 % to ±20 % for mass, inertia, motor model and thrust/torque scaling. For controllers designed primarily for attitude control, where translational dynamics are less critical, a reduced amount or even the omission of domain randomization may be sufficient.

VII. CONCLUSION AND OUTLOOK

This work derived a dynamics model of the recently released CFB, to support future research with this new version of the popular Crazyflie platform. Specifically, we identified the parameters of the dynamics model and provided procedures to quickly re-identify the inertias when payloads are added or removed. Evaluations of the model's prediction capabilities demonstrated that the model is accurate with minor discrepancies due to unmodeled effects such as aerodynamics and observer dynamics. Additionally, we demonstrated the model's effectiveness by training end-to-end NN controllers through RL that transfer from simulation to real world. Our experiments revealed that domain randomization between 10 %–20 % effectively bridges the sim-to-real gap for RL, allowing end-to-end NN controllers that generalize from simulation to real-world deployment on the CFB.

To our knowledge, this is the first openly available model of the new CFB, effective for common tasks like RL. Future work could extend this model by identifying the CFB's aerodynamics, for instance using methods like those in [37], which is particularly relevant for high-speed applications like quadcopter racing. Another interesting direction is modeling the aerodynamic interactions between multiple CFB, as explored in [44], to facilitate the development of high-performance swarms.

ACKNOWLEDGMENT

We thank Emma Cramer, Bernd Frauenknecht, Henrik Hose, David Stenger, Devdutt Subhasish, Khaled Wahba

and Lukas Wildberger for helpful discussions and valuable feedback.

REFERENCES

[1] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[2] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2017.

[3] C. Budaciu, N. Botezatu, M. Kloetzer, and A. Burlacu, "On the evaluation of the crazyflie modular quadcopter system," in *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[4] R. Socas, R. Dormido, M. Guinaldo, and S. Dormido, "A control engineering framework for quadrotors: An application for the crazyflie 2.x:," in *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics*, 2021.

[5] K. Wahba, J. Ortiz-Haro, M. Toussaint, and W. Hönig, "Kinodynamic motion planning for a team of multirotors transporting a cable-suspended payload in cluttered environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[6] A. Gräfe, J. Eickhoff, M. Zimmerling, and S. Trimpe, "DMPC-Swarm: Distributed model predictive control on nano uav swarms," *Autonomous Robots*, 2025.

[7] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/jax-ml/jax

[8] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[9] M. W. Mueller, "Quadcopter dynamics," *Dynamics and Control of Autonomous Flight*, 2025.

[10] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "Neurobem: Hybrid aerodynamic quadrotor model," *arXiv preprint arXiv:2106.08015*, 2021.

[11] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, 2023.

[12] C. Paz, E. Suárez, C. Gil, and J. Vence, "Assessment of the methodology for the cfd simulation of the flight of a quadcopter uav," *Journal of Wind Engineering and Industrial Aerodynamics*, 2021.

[13] S. Yoon, P. V. Diaz, D. D. Boyd Jr, W. M. Chan, and C. R. Theodore, "Computational aerodynamic modeling of small quadcopter vehicles," in *American Helicopter Society (AHS) 73rd Annual Forum Fort Worth, Texas*, 2017.

[14] J. Förster, "System identification of the crazyflie 2.0 nano quadrocopter," 2015, Bachelor's thesis. [Online]. Available: https://doi.org/10.3929/ethz-b-000214143

[15] B. Landry *et al.*, "Planning and control for quadrotor flight through cluttered environments," Ph.D. dissertation, Massachusetts Institute of Technology, 2015.

[16] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

[17] J. Eschmann, D. Albani, and G. Loianno, "Learning to fly in seconds," *IEEE Robotics and Automation Letters*, 2024.

[18] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[19] C. Llanes, Z. Kakish, K. Williams, and S. Coogan, "Crazysim: A software-in-the-loop simulator for the crazyflie nano quadrotor," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[20] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[21] M. Schuck, M. Rath, Y. Hua, A. Goudar, S. Zhou, and A. P. Schoellig, "Crazyflow," https://github.com/utiasDSL/crazyflow, 2025.

[22] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester, "TinyMPC: Model-predictive control on resource-constrained microcontrollers," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[23] A. Gräfe, J. Eickhoff, and S. Trimpe, "Event-triggered and distributed model predictive control for guaranteed collision avoidance in UAV swarms," in *9th IFAC Conference on Networked Systems NECSYS*, 2022.

[24] K. Wahba and W. Hönig, "Efficient optimization-based cable force allocation for geometric control of a multirotor team transporting a payload," in *IEEE Robotics and Automation Letters*, 2024.

[25] A. Javeed and V. L. Jiménez, "Reinforcement learning-based control of crazyflie 2.x quadrotor," *arXiv preprint arXiv:2306.03951*, 2023.

[26] F. Felten, "Multi-Objective Reinforcement Learning," PhD Thesis, Unilu - Université du Luxembourg [FSTM], Luxembourg, June 2024. [Online]. Available: https://hdl.handle.net/10993/61488

[27] T.-D. Do, N. Xuan-Mung, Y.-S. Lee, and S.-K. Hong, "End-to-end deep reinforcement learning-based nano quadcopter low-level controller," in *24th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2024.

[28] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[29] R. Ferede, C. De Wagter, D. Izzo, and G. C. De Croon, "End-to-end reinforcement learning for time-optimal quadcopter flight," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[30] Bitcraze, "Crazyflie firmware," https://github.com/bitcraze/crazyflie-firmware, 2025, online repository; accessed 2025-08-26.

[31] "Crazyflie 2.1 brushless schematics rev.g," https://www.bitcraze.io/documentation/hardware/crazyflie_2_1_brushless/cf2.1_bl_schematics_Rev.G.pdf, 2025, accessed: 2025-08-26.

[32] "Crazyflie 2.1 schematics rev.b," https://www.bitcraze.io/documentation/hardware/crazyflie_2_1/crazyflie_2.1_schematics_rev.b.pdf, 2025, accessed: 2025-08-26.

[33] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *Signal processing, sensor fusion, and target recognition VI*. Spie, 1997.

[34] M. W. Mueller, M. Hamer, and R. D'Andrea, "Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[35] M. W. Mueller, M. Hehn, and R. D'Andrea, "Covariance correction step for kalman filtering with an attitude," *Journal of Guidance, Control, and Dynamics*, 2017.

[36] R. Ferede, G. de Croon, C. De Wagter, and D. Izzo, "End-to-end neural network based optimal quadcopter control," *Robotics and Autonomous Systems*, 2024.

[37] D. Hanover, A. Loquercio, L. Bauersfeld, A. Romero, R. Penicka, Y. Song, G. Cioffi, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing: A survey," *IEEE Transactions on Robotics*, 2024.

[38] (2025) Pwm to thrust conversion. Accessed: 2025-08-26. [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/pwm-to-thrust/

[39] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," 2011.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[41] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax - a differentiable physics engine for large scale rigid body simulation," 2021. [Online]. Available: http://github.com/google/brax

[42] E. Cramer, L. Jäschke, and S. Trimpe, "CHEQ-ing the box: Safe variable impedance learning for robotic polishing," in *14th IFAC Symposium on Robotics*. Elsevier, 2025.

[43] P. Antal, T. Péni, and R. Tóth, "Backflipping with miniature quadcopters by gaussian-process-based control and planning," *IEEE Transactions on Control Systems Technology*, 2023.

[44] G. Shi, W. Hönig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions," *IEEE Transactions on Robotics*, 2021.