Development of a high-fidelity simulation environment for a Crazyflie2.1 quadcopter

Luke Richardson[1*] and *Arnold* Pretorius[1]

[1]MechatronicsSystems.Group, Department of Mechanical Engineering, University of Cape Town, South Africa

Abstract. This paper presents an open-source, Simulink-based, high-fidelity model of a Crazyflie2.1 quadcopter, addressing the lack of an accurate, verified, and readily adaptable simulation in the MATLAB ecosystem. The 6-DOF dynamic model of the quadcopter includes non-linearities, battery drain, and off-diagonal inertia effects that are present on the physical platform. A cascaded PID control scheme, directly ported from firmware, including sensor filtering and data-type parity, is implemented in the simulation environment to align with the physical counterpart. The closed-loop simulation model is validated against real-world flight tests, with position tracking parity shown to be within 2.5 cm. This open-source framework allows for future work in model-based, low-level control modification, as well as modular use in advanced vision and swarming applications.

1 Introduction

Due to their open-source and low-cost ecosystem, the Bitcraze Crazyflie2.1 is a popular choice for undergraduate and postgraduate research on drone control systems, pose estimation, and swarm behaviour [1-3]. While the Crazyflie2.1's firmware is fully customisable, modification requires manual firmware configuration using C, Python or the Crazyflie client, with each of these options having increasingly limited capabilities. In contrast, Simulink's visual block-programming language allows for rapid control system development and simulation, as well as trajectory generation, path planning and obstacle avoidance routines [4-6].

Research on the Crazyflie has previously focused on system identification [7] and controller replacement [8] on the platform, with a lesser focus on developing a high-fidelity digital twin of the platform for simulation and testing in Simulink. Work in [1] analysed the changes in flight performance for different hardware configurations of the Crazyflie, due to their noted lack of documentation on the matter. The authors in [9] and [4] designed a controller for the Crazyflie in Simulink, and simulated the platform in Gazebo using ROS, with no hardware testing. Work in [10] and [2] made use of the stock attitude controller on the Crazyflie with a position controller run externally on a PC, simulated using Gazebo and ROS, and then deployed to the Crazyflie, sending the position controller commands over the Crazyflie's radio. None of the above-mentioned research has analysed the controller code in the stock Crazyflie firmware, nor have they simulated the platform in Simulink.

In [5], a cascaded PID controller and dynamics simulation for an S500 quadcopter platform was developed. However, the simulation was done using Simscape Multibody, not low-level Simulink blocks, and the controller was implemented and simulated with continuous-time controllers, meaning that their performance would differ if it were to be deployed to hardware, where the controllers would be run in discrete time.

In this paper, a high-fidelity model of the quadcopter dynamics, along with a digital twin of the Crazyflie's Proportional-Integral-Derivative (PID) controllers, is developed and implemented in Simulink, and quantitatively compared to flight test data. The closed-loop model is shown to have strong parity with the physical counterpart when in generalised flight, with position tracking within 2.5 cm in all three axes. The simulation environment, created exclusively in MATLAB and Simulink, addresses the lack of an accurate, verified and lucid model of the Crazyflie2.1, which can be used for rapid controller tuning, and an education tool for quadcopters and their control systems. The MATLAB/Simulink project is openly available for use, modification and/or replication, and can be found on GitHub.

2 Quadcopter model

Fig. 1 provides a schematic of the Crazyflie, along with the axis conventions and notation. The Crazyflie uses the X-frame configuration, and the dynamics are formulated using the North-West-Up (NWU) axis orientation convention [11]. The body frame, indicated by $\{B\} = \{\hat{\boldsymbol{x}}_B, \hat{\boldsymbol{y}}_B, \hat{\boldsymbol{z}}_B\}$, is located at the Crazyflie's assumed centre of gravity (CG), and the world frame, indicated by $\{W\} = \{\hat{\boldsymbol{x}}_W, \hat{\boldsymbol{y}}_W, \hat{\boldsymbol{z}}_W\}$, is located at a fixed reference point.

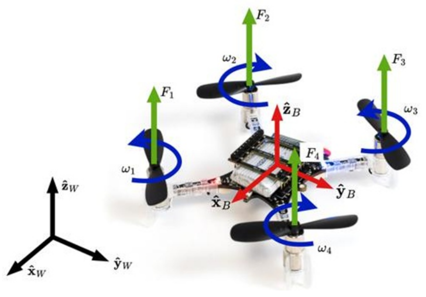

Fig. 1. Crazyflie notation schematic. The body frame $\{B\}$, is shown in red in the North-West-Up convention and X-frame configuration. The world frame $\{W\}$, is shown in black, thrust forces are indicated in green, and drag torques in blue, with their indicated number sequence.

2.1 Moment equations and rotational dynamics

Euler's dynamic equation for general three-dimensional (3D) rigid-body rotational motion, expressed in the body frame coordinates, is defined as [12, 13]

$$^{\mathrm{B}}\mathbf{M} = \boldsymbol{J}\,^{B}\dot{\boldsymbol{\omega}} + {}^{\mathrm{B}}\boldsymbol{\omega} \times (\boldsymbol{J}\,^{B}\boldsymbol{\omega}). \tag{1}$$

Vector $^{B}\mathbf{M}$ denotes the sum of the external moments applied to the body, about each of the three body-frame axes, $\boldsymbol{J}$ is the 3×3 mass moment of inertia matrix of the quadcopter body, and $^{B}\boldsymbol{\omega} = [\omega_x\ \omega_y\ \omega_z]^T$ and $^{B}\dot{\boldsymbol{\omega}} = [\dot{\omega}_x\ \dot{\omega}_y\ \dot{\omega}_z]^T$ are the angular velocity and acceleration experienced in $\{B\}$, respectively. Equation (1) can be rearranged to solve for $^{B}\dot{\boldsymbol{\omega}}$, which can be numerically integrated to find $^{B}\boldsymbol{\omega}$. Finally, to avoid issues related to the use of Euler angles to represent orientation, $^{B}\boldsymbol{\omega}$ can be integrated using the quaternion formulation of

$$^{W}\boldsymbol{q}_{(t+\Delta t)} = {}^{W}\boldsymbol{q}_{(t)} \otimes \Delta\boldsymbol{q}, \tag{2}$$

where

$$\Delta\mathbf{q} = \left[cos\left(\frac{|^{\mathrm{B}}\boldsymbol{\omega}|\Delta t}{2}\right) \quad sin\left(\frac{|^{\mathrm{B}}\boldsymbol{\omega}|\Delta t}{2}\right)\frac{^{B}\omega_x}{|^{\mathrm{B}}\boldsymbol{\omega}|} \quad sin\left(\frac{|^{\mathrm{B}}\boldsymbol{\omega}|\Delta t}{2}\right)\frac{^{B}\omega_{\mathrm{y}}}{|^{\mathrm{B}}\boldsymbol{\omega}|} \quad sin\left(\frac{|^{\mathrm{B}}\boldsymbol{\omega}|\Delta t}{2}\right)\frac{^{B}\omega_{\mathrm{z}}}{|^{\mathrm{B}}\boldsymbol{\omega}|}\right]^{T},$$

and Δt is the sampling period [14].

The dominating forces and moments that act on the quadcopter frame are the thrust forces and drag torques due to the aerodynamic interaction between each of the rotating propeller blades and the surrounding air [12, 13]. These interactions are highly complex; however, their dynamics are often neglected due to the assumed fast response of the motors with respect to the overall body motion [15]. The commonly used equations for the thrust force, F_i, and drag torque, τ_{i}, experienced by motor i, are given respectively by

$$F_i = k_{\mathrm{t}}\bar{\omega}_i{}^2, \tag{3}$$

and

$$\tau_i = k_q\bar{\omega}_i{}^2, \tag{4}$$

where k_t is the (approximately) constant thrust coefficient, k_q is the (approximately) constant drag coefficient, and $\bar{\omega}_{\mathrm{i}}$ is the angular velocity of propeller i [12]. According to [8] and [16], the propeller angular velocity, in steady-state, is related to the Pulse Width Modulation (PWM) command, which is a 16-bit unsigned integer, sent to the motors through the static, linear mapping of

$$\bar{\omega}_{\mathrm{i}} = \frac{2\pi}{60}(0.2685P_i + 4070.3), \tag{5}$$

where P_i is the PWM command of motor i.

The thrust forces and drag torques can be used to calculate the overall moments acting on the quadcopter body assuming no external disturbances are acting on the platform. With reference to Fig. 1, differential thrust between motors m_1 and m_2, and motors m_3 and m_4 generates a moment about the body frame $\hat{\boldsymbol{x}}_B$ axis. A moment about the $\hat{\boldsymbol{y}}_B$ axis can be generated in a similar manner. These moments are also referred to as roll and pitch moments respectively, and are collectively referred to as the moments due to thrust, given by [12, 13]

$$^{B}\boldsymbol{M}_T = \begin{bmatrix} \frac{d}{2}(-F_1 - F_2 + F_3 + F_4) \\ \frac{d}{2}(-F_1 + F_2 + F_3 - F_4) \\ 0 \end{bmatrix}, \tag{6}$$

where d is the distance between adjacent motors on the Crazyflie.

The final dominant moment acting on the body is the overall moment due to the drag torque of the propellers, which is commonly referred to as the yaw moment, $^{B}\boldsymbol{M}_D$. This yaw moment arises when pairs of opposite motors rotate at different speeds to their adjacent motors and is described by [12, 13]

$$^{B}\boldsymbol{M}_D = [0 \quad 0 \quad -\tau_1 + \tau_2 - \tau_3 + \tau_4]^T. \tag{7}$$

Equations (1)-(7) can be implemented in Simulink to simulate the attitude dynamics of the quadcopter.

2.2 Force equations and translational dynamics

Newton's second law of motion can be used to describe the translational dynamics of the quadcopter in the world frame as [12, 13]

$$\Sigma^{W}\boldsymbol{F} = m^{W}\ddot{\boldsymbol{p}}_{B}, \tag{8}$$

where $\Sigma^{W}\boldsymbol{F}$ is the sum of the external forces acting on the body, expressed in world-frame coordinates, m is the mass of the quadcopter, and ${}^{W}\ddot{\boldsymbol{p}}_{B}$ is the translational acceleration of the quadcopter, expressed in world frame coordinates. Equation (8) is expressed in world frame coordinates so that the calculation of Coriolis forces can be avoided. The only implication of this is that the body frame forces need to be mapped into the world frame counterparts using the current orientation of the body frame, as calculated in equation (2). The translational acceleration, ${}^{W}\ddot{\boldsymbol{p}}_{B}$, can then be numerically integrated twice to determine the position of the quadcopter in the world frame.

The dominant forces acting on the quadcopter body are the thrust forces due to the rotating propellers and gravity [12]. The resultant thrust force in $\{W\}$ is calculated by first determining the thrust force in $\{B\}$, given by ${}^{B}\boldsymbol{F}_{T}$, followed by rotating this vector into world-frame coordinates. This is accomplished using quaternion vector conjugation, as given by

$$\begin{bmatrix} 0 \\ {}^{W}\boldsymbol{F}_{T} \end{bmatrix} = {}^{W}\boldsymbol{q}_{\boldsymbol{B}} \otimes \begin{bmatrix} 0 \\ {}^{B}\boldsymbol{F}_{T} \end{bmatrix} \otimes {}^{W}\boldsymbol{q}_{\boldsymbol{B}}{}^{-\boldsymbol{1}} \;, \tag{9}$$

where

$${}^{B}\boldsymbol{F}_{T} = [0 \quad 0 \quad \textstyle\sum_{i=1}^{4} F_{i}]^{T}. \tag{10}$$

The sum of external forces follows as

$$\Sigma^{W}\boldsymbol{F} = {}^{W}\boldsymbol{F}_{T} + {}^{W}\boldsymbol{F}_{g} = {}^{W}\boldsymbol{F}_{T} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}, \tag{11}$$

where m is the mass of the quadcopter and $g \approx 9.8\ \mathrm{ms}^{-2}$ is the gravitational acceleration in Cape Town, South Africa. Equations (8)-(11) can be implemented in Simulink to simulate the translational dynamics of the quadcopter.

2.3 Battery drain modelling

The dynamic modelling presented thus far is commonplace in quadcopter research and considers the idealised platform behaviour [12, 13]. However, it can also be helpful to understand how battery drain effects the quadcopter's performance, and this is yet to be analysed for the Crazyflie. Flight stability can degrade as the battery drains, since the control authority is closely tied to the available battery voltage. As the voltage drops, higher PWM duty cycles are required to maintain the same motor thrust. However, this compensation has limits — eventually, the duty cycle approaches its maximum, and the motors can no longer produce the necessary thrust, reducing the system's ability to respond effectively to control inputs [10].

To model the battery drain during flight, the battery voltage was directly logged from the Crazyflie during a 300 second hover flight test. An empirical model of the voltage decrease over time was found to be:

$$V(t) = -2.555 \cdot 10^{-12} t^{5} + 2.028 \cdot 10^{-9} t^{4} - 6.315 \cdot 10^{-7} t^{3} + 9.654 \cdot 10^{-5} t^{2}$$

$$-0.008303t + 3.706, \tag{12}$$

where t is the time variable in seconds. A comparison of the actual battery voltage over time plotted against the above equation is shown in Fig. 2.

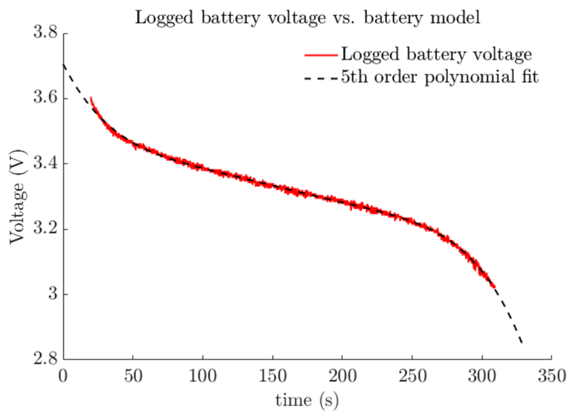

Fig. 2. Battery voltage logged during a hover flight test plotted with the 5th order polynomial used as the battery model.

The battery model can be implemented by scaling the angular velocity of the propellers before the thrust and torque calculations are made. The scale factor is the current battery voltage divided by the fully charged battery voltage at hover, to account for the battery sag when loaded to approximately hover conditions. The calculation of the corrected angular velocity, ω_i^*, is given by

$$\overline{\omega}_i^* = \frac{V(t)}{3.7}\overline{\omega}_i, \tag{13}$$

where the nominal battery voltage at hover load is measured to be 3.7 Volts. The corrected angular velocity is then used in the dynamics and is also fed back to the appropriate controllers to be used in a battery compensation function, which will be presented in Section 3.1.5. Notably, this battery model is an approximation for the battery voltage drop under the near-hover-flight assumption. The voltage drop rate would increase relative to hover when more power is demanded from the batteries, such as during transient motion [17].

2.4 Model parameters

The model parameters that were obtained from other sources are shown in Table 1 for sake of completeness.

Table 1. Model parameters used for simulation.

Parameter description	Parameter symbol	Value	Unit	Ref.
Mass	m	0.032	kg	[8]
Inertia matrix	$\boldsymbol{J}$	$\begin{bmatrix} 16.57171 & 0.830806 & 0.718277 \\ 0.830806 & 16.55602 & 1.800197 \\ 0.718277 & 1.800197 & 29.26165 \end{bmatrix} \cdot 10^{-6}$	$kg.m^2$	[7]
Thrust coefficient	k_t	$2.879933 \cdot 10^{-8}$	$\frac{N \cdot s^2}{rad^2}$	[8]
Drag coefficient	k_q	$7.23850 \cdot 10^{-10}$	$\frac{Nm \cdot s^2}{rad^2}$	[8]
Adjacent motor distance	d	0.065	m	[8]

3 Control system digital twinning in Simulink

This section will discuss the process taken to create the digital twin of the Crazyflie's Proportional-Integral-Derivative (PID) controllers, which are implemented on the Crazyflie in C. The intention is to obtain a more readable version of the controllers on the Crazyflie, which can be used for controller demonstration purposes, and to also allow for tuning of the controllers in an accurate software-in-the-loop environment.

The default control scheme onboard the Crazyflie has a cascaded proportional-integral-derivative (PID) structure, as depicted in Fig. 3 and in [5]. PID controller structures are present for each of the roll, pitch, and yaw angles, as well as their respective rates. Similarly, PID controller structures are implemented for each of the x, y and z position loops, and their respective rates, for a total of twelve PID controllers. While all controllers have the archetypal PID structure, some of the gains are set to zero by default and therefore operate as proportional (P) or proportional-plus-integral (PI) controllers, as shown in Table 2. The position and velocity controllers operate at 100 Hz, and the attitude and attitude rate counterparts operate at 500 Hz. The ported Simulink controllers are implemented with the above-mentioned rates, along with discrete derivatives and integrals to emulate the digital implementation of the controllers. Furthermore, all units, datatypes, saturations and PID gains are consistent with the C-code implementation onboard the Crazyflie. The PID gains obtained from the Crazyflie code are shown in Table 2.

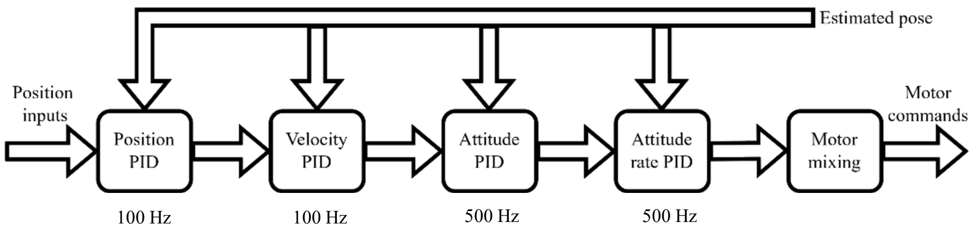

Fig. 3. Crazyflie cascaded PID controller structure. Feedback loop update rates are also indicated.

Table 2. Default Crazyflie PID gains. K_p refers to the proportional gain, K_I is the integral gain, and K_D is the derivative gain.

Controller	Gain	Value	Controller	Gain	Value
x and y position	K_p	2	x and y velocity	K_p	25
	K_I	0		K_I	1
	K_D	0		K_D	0
z position	K_p	2	z velocity	K_p	25
	K_I	0.5		K_I	15
	K_D	0		K_D	0
Roll and pitch	K_p	6	Roll and pitch rates	K_p	250
	K_I	3		K_I	500
	K_D	0		K_D	2.5
Yaw	K_p	6	Yaw rate	K_p	120
	K_I	1		K_I	16.67
	K_D	0.35		K_D	0

The following subsections will discuss some of the additional intricacies of the Crazyflie controller implementation that may be difficult to decipher from the C-code, as well as the method used for simulation visualisation, and its future potential.

3.1 Attitude and attitude rate controllers

3.1.1 Attitude and attitude rate units and sense

An important thing to note in the Crazyflie's attitude control is that the pitch angle is inverted from the conventional right-hand rule, and therefore the state values for pitch and pitch rate are inverted before being injected into their respective controllers. Furthermore, the state values for attitude and attitude rate have units of radians and radians per second, respectively. However, these signals are converted to degrees and degrees per second before being used in the controller.

3.1.2 Derivative action and filtering

The only derivative gains that are non-zero in the Crazyflie controller are the ones in the attitude rate controllers. There are two common methods for implementing derivative action; one is to differentiate the feedback error, and the other is to differentiate only the measurement signal, which closely resembles a state variable feedback control approach. The Crazyflie controllers use the latter method throughout.

Another common addition to derivative action in PID controllers is the use of a low-pass filter on the measurement or error signal before taking its derivative, as this assists with filtering out high frequency control action that can result from differentiation. The Crazyflie firmware has the attitude rate derivative filter disabled, however, elsewhere in the firmware, the filter onboard the gyroscope is enabled to have a bandwidth of 116Hz.

3.1.3 Integral anti-windup

The Crazyflie firmware uses integrator clamping for the PID controllers when a saturation is detected.

3.1.4 Motor mixing

The motor mixing equations govern how the roll, pitch, yaw, and thrust commands are mapped to the requisite command signals for each of the motors to generate the desired forces and torques. A notable feature of the Crazyflie implementation is that the roll and pitch commands are halved before being injected into the motor mixing algorithm. It is common to have all four control signals to be scaled by half or a quarter based on the axis configuration [18] or to not scale them at all [19]. However, there is no reasoning provided for only the roll and pitch being halved.

3.1.5 Battery compensation function

Once the command signals have been passed through the motor mixing algorithm, the individual motor commands can be sent to the motors. However, the Crazyflie attempts to adjust this command to account for the voltage sag of the battery under load, using an empirical model identified by the Crazyflie developers. This model can be found in the *Crazyflie-firmware/src/drivers/src/motors.c* file in the firmware [16].

3.2 Position and velocity controllers

3.2.1 Yaw alignment

Before the x, y, and z inputs and states are injected into the position and velocity controllers, their values are rotated based on the current yaw angle. This is done because the attitude control loops make use of Euler angle information, with the translational velocity controller outputs directly mapping to the Euler roll and pitch angles required to move the quadcopter to the target location.

3.2.2 Velocity and attitude command saturations

Given the use of Euler angles in the attitude feedback loops, along with the fact that the controllers are designed around near-hover assumptions, the Crazyflie firmware implements physical constraints on the accessible flight envelope. This is done by limiting the velocity commands in the x, y and z channels to ±1 m/s, as well as limiting the roll and pitch commands to $\pm20^\circ$. The velocity command saturations are also multiplied by 1.1 as an 'overhead allowance', and a reason for this allowance is not provided in the Crazyflie documentation.

3.2.3 Total thrust control

The z-position of the Crazyflie is controlled by cascaded PID controllers for position and then velocity, which outputs a value analogous to a commanded acceleration in the z-direction. This value is then scaled by 1000 and added to a base thrust command of 36000. The scaling is done to adjust the order of magnitude of the thrust command to the same as the 16-bit unsigned integer that is sent to the motors. The base thrust is a pre-calculated thrust command value that would be required to maintain hover, because otherwise the integral windup in the *z* position controller would be relied on to increase the thrust command to maintain hover. The resulting output is a 16-bit integer, which is injected into the motor mixing algorithm.

3.3 Visualisation

The Crazyflie's six-degree-of-freedom motion is visualised using Simulink's Unreal Engine integration, with the built-in quadcopter UAV 3D visualisation in the default scene configuration. An example of the visualisation is shown in Fig. 4.

Fig. 4. Visualisation example of a quadcopter in the Unreal Engine Simulink visualiser.

4 Simulink implementation

This section aims to present the graphical Simulink environment in which the dynamics and digital twin of the controller are implemented. This section will demonstrate the improved readability that the Simulink implementation provides. Fig. 5 shows the high-level view of the system in a familiar form, with the inputs being injected into the controller from the left, the controllers' outputs being injected into the plant block, and the plant output being fed back to the controller through the sensor block, as well as being sent to the visualisation subsystem.

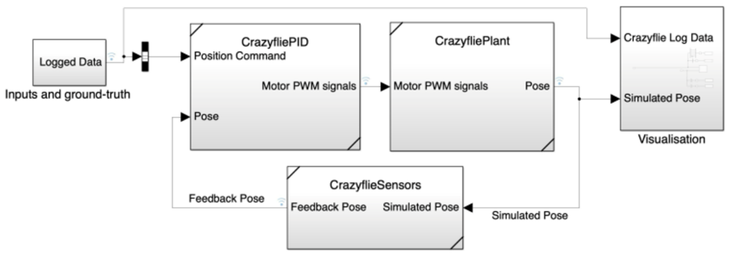

Fig. 5. High-level overview of the Simulink environment.

Throughout the Simulink model, bus signals are used extensively for two reasons. Firstly, they reduce the number of signal lines significantly, and secondly, they maintain the individual signal names, improving the logging and debugging process significantly.

4.1 Dynamics

The dynamic equations of motion and force and moment calculations presented in Section 2 are implemented in the *CrazyfliePlant* subsystem. The parameters used in this subsystem, which are those that were presented in Table 1, are loaded into the MATLAB workspace using a startup script, which is executed automatically when the MATLAB project is opened.

4.1.1 Equations of motion

The equations of motion that were presented in Section 2 are implemented as any other ordinary differential equation (ODE) would be implemented in Simulink. The benefit of using Simulink is that ODE’s are solved numerically with reasonably high accuracy, without the need for the user to be familiar with numeric ODE solvers. In this case, Simulink is set to automatically choose a solver, and a variable time step is used.

4.1.2 Force and moment calculations

The implementation of the force and moment calculations begins with the approximation of the motor angular velocity from the commanded PWM value. This value is then scaled by the previously mentioned battery drain scale factor from (12). From this, the thrust force and drag torque produced by each propeller can be calculated according to (3) and (4). Fig. 6 illustrates these calculations as implemented in Simulink.

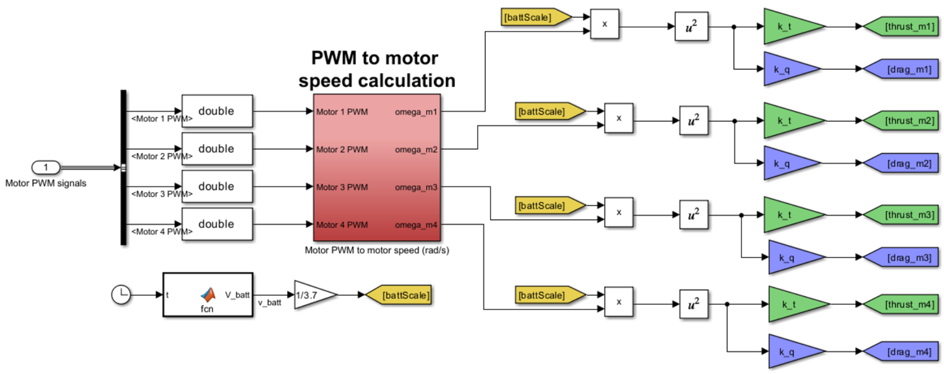

Fig. 6. Simulink block diagram for individual propeller thrust calculations

The thrust and drag values are then used to calculate the overall moments on the body according to (6) and (7). The attitude equation of motion can then be solved. The Simulink block diagram is shown in Fig. 7 and illustrates the ease of traceability of each of the signals.

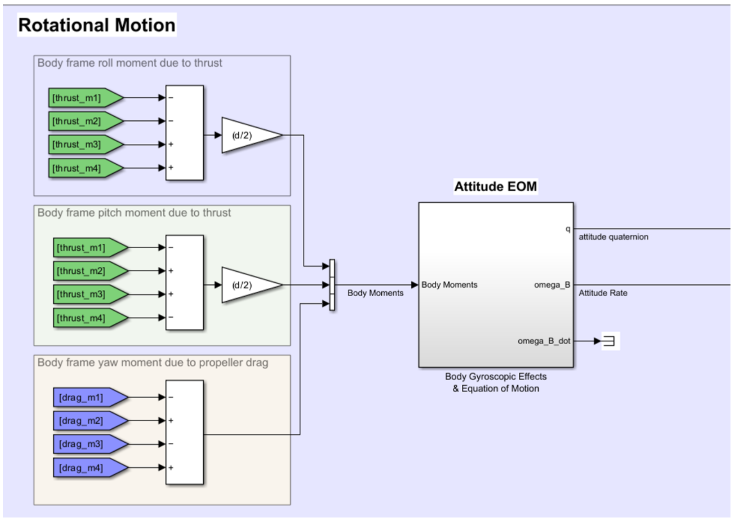

Fig. 7. Simulink block diagram for body-frame moment calculations.

Fig. 8 shows the calculation of the body frame forces due to thrust based on (10), and how this signal, along with the body frame orientation, is directed into the EOM for translation.

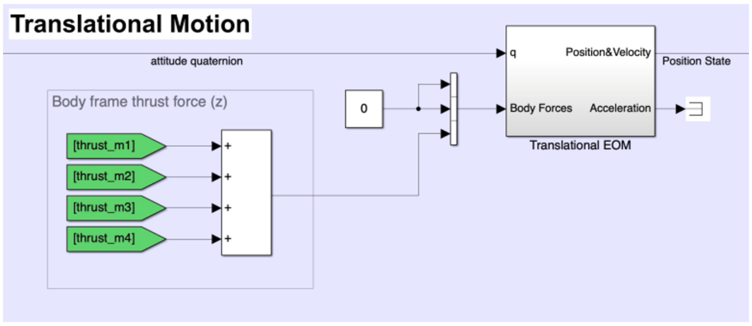

Fig. 8. Simulink block diagram for body-frame force calculations.

4.2 Controllers

Simulink is very popular for control systems design due to the graphical block diagram programming language. This will be emphasised in this subsection by presenting portions of the Simulink implementation of the Crazyflie controller.

4.2.1 Attitude controllers

The initial work done on reproducing the attitude controllers was completed as part of an undergraduate project [20]. The decision was made to use basic Simulink blocks to model the controllers, instead of a dedicated PID block, to maintain full transparency of the controller. This approach also improves readability, as one can easily see internal features of the controller at a glance. Examples of the roll and roll rate controllers are provided in Fig. 9.

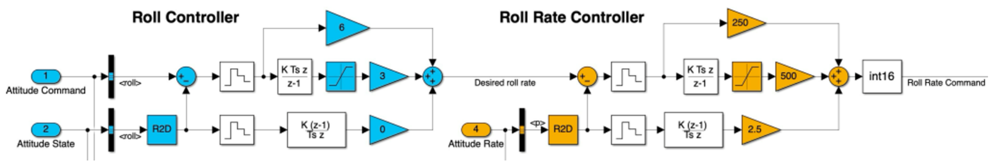

Fig. 9. Attitude PID example for the roll and roll rate controllers.

The above figures show the benefit of using Simulink bus signals, as the required signal can be extracted from the bus using its name, such as where the <*roll*> command signal is extracted from the *Attitude Command* bus.

4.2.2 Position controllers

The format of the position and velocity controllers in Simulink are very similar to the attitude and attitude rate controllers, as shown in Fig. 10.

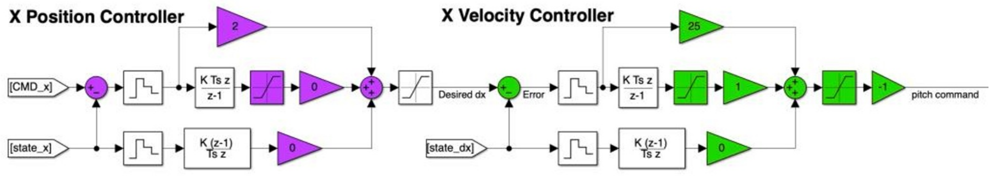

Fig. 10. Position and velocity PID example for the x-channel.

The figures above show the benefit of using basic Simulink blocks to build the PID controllers. One can easily decipher the features of the controller, such as the gain values, the zero-order-holds used to limit the controller execution rate, and the types of discrete derivatives and integrators being used.

5 Experimental setup, results and discussion

To validate the Simulink model, a set of experimental tests were conducted on the Crazyflie and it's Simulink digital twin, with the full pose of the quadcopter being logged throughout the tests. The logged responses of the real-world flights, logged at 100 Hz, were then compared to the simulated responses for comparison.

5.1.1 Lighthouse positioning system

To obtain a ground truth measurement for the Crazyflie position, the lighthouse positioning system was used. This system uses two or more SteamVR Base Stations that emit a laser array into the flight area, where a set of photodiodes onboard the quadcopter are used to calculate its position relative to the Base Stations [21]. The Crazyflie can use this position estimate in feedback to maintain position stability. The work presented in [22] shows the lighthouse positioning system maintaining a mean and median Euclidian error of 2-4 cm.

5.1.2 Flight test regime

The flight tests consisted of a set of reference commands at the position level, firstly to the individual x, y and z positions, followed by a set of simultaneous steps combining motion in multiple axes to activate the coupled dynamics of the system. This set of inputs was chosen to excite the non-linear dynamics present in the system. The sequence of inputs is shown in Fig. 11 below.

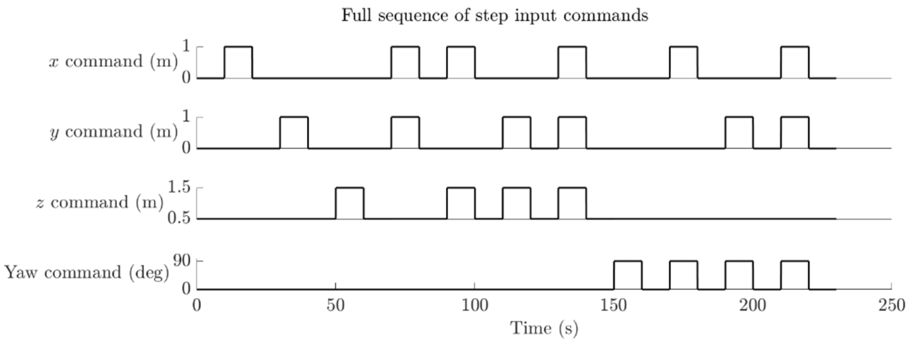

Fig. 11. Sequence of step input commands sent to the Crazyflie over the 230 second flight, starting with single channel position setpoint commands, followed by step inputs in multiple directions simultaneously.

5.1.3 Results and discussion

Fig. 12 below shows the isolated step response for x, y, z and yaw step inputs. The figure shows the input commands, simulated responses and real-world responses. Fig. 13 shows the simulated and experimental roll and pitch responses for the respective y- and x-position step commands. Fig. 14 shows the simulated and experimental PWM output commands from the controller for motor 1, as an exemplar, over the flight test. Finally, Table 3 presents the root-mean-square error (RMSE) for each element in the pose of the quadcopter, measured over the entire flight, based on the high-level commands issued in Fig. 11.

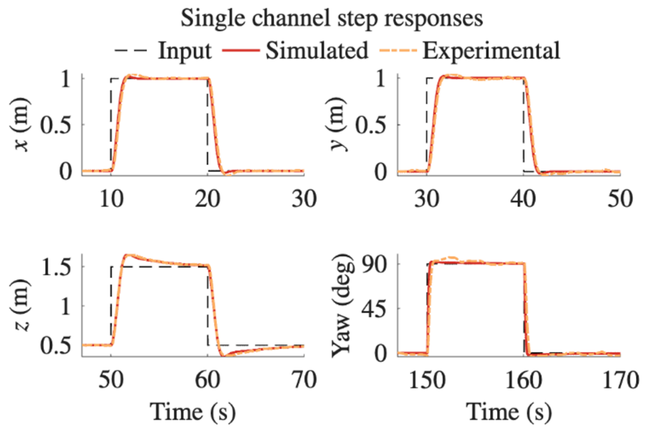

Fig. 12. Step response for isolated x, y, z and yaw step inputs, plotted with the inputs, simulated responses and the experimental data.

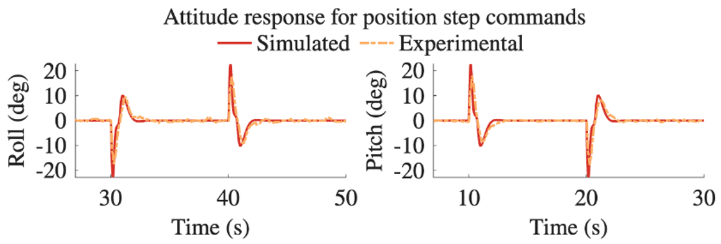

Fig. 13. Roll and pitch responses for step inputs in the y- and x- directions respectively, plotted with the simulated and experimental data.

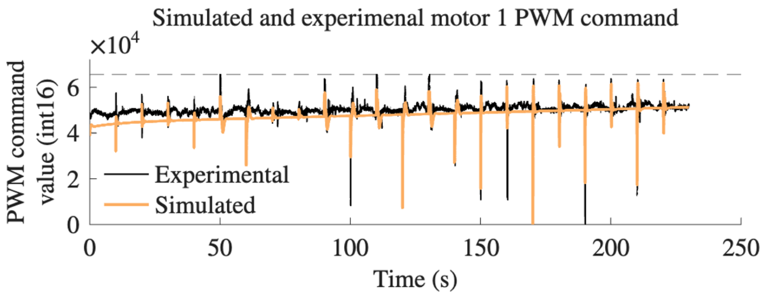

Fig. 14. PWM output commands from the controller for motor 1 over the flight test, plotted with the simulated and experimental data.

Table 3. RMSE values for each element of the pose of the quadcopter over the entire 230 second flight test.

Channel	RMSE (cm)	Channel	RMSE (deg)
x	2.32	Roll	1.82
y	2.17	Pitch	1.83
z	1.73	Yaw	3.55

The step responses Fig. 12 and RMSE values Table 3 indicate that there is good parity between the simulation and the real-world flight test measurements, including during coupled motion that requires simultaneous rolling and pitching. This parity is further substantiated by the roll and pitch responses in Fig. 13. The motor commands in Fig. 14 shows the steady increase in command values to compensate for battery drain, and that the simulated battery drain model is resulting in a similar steady increase in commands.

The RMSE values for position can likely be attributed to lighthouse system inaccuracies shown in [22], combined with the lack of a sensor noise model. Furthermore, the slight differences in overshoot and settling time, seen evidently in the x and y position step responses, can likely be attributed to parameter inaccuracies and uncertainties, as most of the parameters were obtained from other sources, which may have been using a slightly different configuration of the Crazyflie. An additional explanation for this minor disparity could be the lack of motor dynamics included in the simulation.

6 Conclusion

In this paper, an open-source, high-fidelity Simulink model of the quadcopter dynamics, along with a digital twin of the Crazyflie's PID controllers, was developed. This addressed the lack of an accurate and verified simulation entirely in the MATLAB/Simulink ecosystem, providing an easily readable simulation environment for the platform over the stock C-code implementation. The 6-DOF, non-linear dynamic model, including a battery drain model and digital controller twin, was shown to have strong parity with the physical counterpart when in generalised flight, with position tracking parity within 2.5 cm and low-level motor commands closely matching in both scale and bandwidth. The Simulink-based simulation environment allows for easy tuning and refinement of the PID controller gains while in simulation and can also be used in an education setting without the risk of damaging hardware.

This research was partially funded by the Council for Scientific and Industrial Research (CSIR). All files and data used in this research can be found on the author's GitHub repository: github.com/lukejrichardson/CrazyflieSimulation.

References

1. C. Budaciu, N. Botezatu, A. Burlacu, On the Evaluation of the Crazyflie Modular Quadcopter System. 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 1189-1195 (2019). https://doi.org/10.1109/ETFA.2019.8869202
2. C. Llanes, Z. Kakish, K. Williams, S. Coogan, CrazySim: A Software-in-the-Loop Simulator for the Crazyflie Nano Quadrotor. IEEE International Conference on Robotics and Automation (ICRA), 12248-12254 (2024). https://doi.org/10.1109/ICRA57147.2024.10610906

3. W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, P. Kozierski, Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. 22nd International Conference on Methods and Models in Automation and Robotics (MMAR), 37-42 (2017). https://doi.org/10.1109/MMAR.2017.8046794
4. M. Nithya, M. Rashmi, Gazebo – Simulink Framework for Trajectory Tracking in a Multi-Quadcopter Environment. International Review of Automatic Control (IREACO), **15**, 164-175 (2022). https://doi.org/10.15866/ireaco.v15i4.21432
5. V, Budnyaev, I. Filippov, V. Vertegel, S. Dudnikov, Simulink-based Quadcopter Control System Model. 1st International Conference Problems of Informatics, Electronics, and Radio Engineering (PIERE), 246-250 (2020). https://doi.org/10.1109/PIERE51041.2020.9314676
6. M. Tahir, I. Mir, T. Islam, Control Algorithms, Kalman Estimation and Near Actual Simulation for UAVs: State of Art Perspective. Drones, **7**, 339 (2023). https://doi.org/10.3390/drones7060339
7. J. Förster, M. Hamer, R. D'Andrea, System Identification of the Crazyflie 2.0 Nano Quadrocopter. Master Thesis, ETH Zurich, 2015. https://doi.org/10.3929/ETHZ-B-000214143
8. C. Luis, J. Ny, Design of a Trajectory Tracking Controller for a Nanoquadcopter. Master Thesis, Polytechnique Montreal, 2016. https://doi.org/10.48550/arXiv.1608.05786
9. M. Nithya, M. Rashmi, Gazebo - ROS - Simulink Framework for Hover Control and Trajectory Tracking of Crazyflie 2.0. IEEE Region 10 Conference (TENCON), 649-653 (2019). https://doi.org/10.1109/TENCON.2019.8929730
10. G. Garcia, A. Kim, E. Jackson, S. Keshmiri, D. Shukla, Modeling and flight control of a commercial nano quadrotor. International Conference on Unmanned Aircraft Systems (ICUAS), 524-532 (2017). https://doi.org/10.1109/ICUAS.2017.7991439
11. Bitcraze, The Coordinate System of the Crazyflie 2.x | Bitcraze (2025). https://www.bitcraze.io/documentation/system/platform/cf2-coordinate-system/
12. S. Bouabdallah, R. Siegwart, Full control of a quadrotor. IEEE/RSJ International Conference on Intelligent Robots and Systems, 153-158 (2007). https://doi.org/10.1109/IROS.2007.4399042
13. C. Masse, O. Gougeon, D. Nguyen, D. Saussie, Modeling and Control of a Quadcopter Flying in a Wind Field: A Comparison Between LQR and Structured H-infinity Control Techniques. International Conference on Unmanned Aircraft Systems (ICUAS), 1408-1417 (2018). https://doi.org/10.1109/ICUAS.2018.8453402
14. K. Lynch, F. Park, Modern Robotics: Mechanics, Planning, and Control, (Cambridge University Press, 2017). https://www.cambridge.org/core/product/identifier/9781316661239/type/book
15. T. Luukkonen, Modelling and control of quadcopter. Independent research project in applied mathematics, Espoo, Aalto University (2011). https://sal.aalto.fi/publications/pdf-files/eluu11_public.pdf
16. Bitcraze, GitHub - bitcraze/crazyflie-firmware: The main firmware for the Crazyflie Nano Quadcopter, Crazyflie Bolt Quadcopter and Roadrunner Positioning Tag (2025). https://github.com/bitcraze/crazyflie-firmware/tree/master
17. Y. Chen, D. Baek, A. Bocca, A. Macii, E. Macii, M.Poncino, A Case for a Battery-Aware Model of Drone Energy Consumption. International Telecommunications Energy Conference (IN^{TEL}EC), 1-8 (2018). https://doi.org/10.1109/INTLEC.2018.8612333

18. B. Milhim, Y. Zhang, C. Rabbath, Gain Scheduling Based PID Controller for Fault Tolerant Control of Quad-Rotor UAV. AIAA Infotech@Aerospace (2010). https://doi.org/10.2514/6.2010-3530
19. H. Huang, G. Hoffmann, S. Waslander, C. Tomlin, Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. IEEE International Conference on Robotics and Automation, 3277-3282 (2009). https://doi.org/10.1109/ROBOT.2009.5152561
20. G. Hasewinkel, Analysis of the orientation control schemes present on the Crazyflie quadrotor. Undergraduate Thesis, University of Cape Town, 2022.
21. Bitcraze, Lighthouse Positioning System | Bitcraze (2025). https://www.bitcraze.io/documentation/system/positioning/ligthouse-positioning-system/
22. A. Taffanel, B. Rousselot, J. Danielsson, K. McGuire, K. Richardsson, M. Eliasson, T. Antonsson, W. Hönig, Lighthouse Positioning System: Dataset, Accuracy, and Precision for UAV Research. ArXiv preprint, arXiv:2104.11523 (2021). https://doi.org/10.48550/arXiv.2104.11523