AE 483 Lab 4: Modeling and Simulation of the Crazyflie 2.1

Linyi Hou[1]

University of Illinois at Urbana-Champaign, Urbana, IL, 61820, United States

The Crazyflie 2.1 quadcopter's dynamics are modeled in a MATLAB simulation, and the cascaded proportional, integral, derivative controller used by the Crazyflie was emulated in MATLAB along with hardware models and perturbations. Simulation flights were successfully conducted to reproduce the physical flights done in a previous lab, showing that the simulation is a good representation of the quadcopter's actual performance. Future improvements to the simulation are identified by comparing the simulation and actual flights.

I. Nomenclature

$\boldsymbol{r}$	=	position in inertial frame	$\mathcal{I}$	=	moment of inertia matrix	U	=	PID output
$\boldsymbol{v}$	=	velocity in inertial frame	R_x	=	rotation matrix about x-axis	D	=	desired state
$\boldsymbol{\theta}$	=	attitude in inertial frame	R_y	=	rotation matrix about y-axis	E	=	state error
$\boldsymbol{\omega}$	=	attitude rate in body frame	R_z	=	rotation matrix about z-axis	M	=	measured state
$\boldsymbol{\tau}$	=	torque in body frame	R_{zyx}	=	Tait-Bryan rotation matrix	K_p	=	PID proportional gain
F	=	thrust in body frame	m	=	mass	K_i	=	PID integral gain
$\boldsymbol{x}$	=	state vector	L	=	length	K_d	=	PID derivative gain
$\dot{\boldsymbol{x}}$	=	time derivative of state vector	g	=	gravity, $9.81 m/s^2$	RPM	=	rotations per minute
$\boldsymbol{u}$	=	control input						

II. Introduction

The Crazyflie 2.1 is a small and versatile quadcopter with open-source firmware produced by bitcraze. In previous lab reports, features of the Crazyflie's attitude and positioning system have been examined, and the effects of tuning the proportional-integral-derivative (PID) controller were explored. In addition, physical flights were conducted to evaluate the performance of the quadcopter, and the position PID controller was optimized to minimize flight time.

However, PID tuning with the physical quadcopter may be time-consuming. The testing duration is limited by the battery capacity of the quadcopter, and recharge times are significantly longer than flight times. Crashes can damage motor and cause inconsistencies in flight results, while replacement parts such as propellers, though inexpensive, might not be readily available. In contrast, PID tuning via software simulations is highly desirable as it can mitigate all of the above constraints.

In this report, a MATLAB simulation of the Crazyflie will be developed. First, a physics simulation will be created by solving the equations of motion (EOMs). Next, the Crazyflie firmware will be emulated to recreate the PID controller. Then, perturbations and noise models will be implemented to reflect physical flight conditions. Finally, the trajectory from a physical flight test will be compared against the fully developed simulation to validate its accuracy.

[1] Undergraduate Student, Department of Aerospace Engineering, AIAA Student Member.

III. Simulation Development

A. Equations of Motion

The quadcopter has six degrees of freedom, meaning it can translate and rotate along any axis in 3D space. As a result, the states of the quadcopter include position, velocity, attitude, and attitude rate. The state vector is given as:

$$\boldsymbol{x} = [\boldsymbol{r}, \boldsymbol{\theta}, \boldsymbol{v}, \boldsymbol{\omega}]^T = \left[r_x, r_y, r_z, \theta_z, \theta_y, \theta_x, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z\right]^T \tag{1}$$

The control input to the quadcopter, u, comprises of the net thrust produced by all four rotors, as well as the body torques imparted by the thrust difference between motors and the motor torques. The control input vector is then

$$\boldsymbol{u} = \left[\tau_x, \tau_y, \tau_z, F\right]^T \tag{2}$$

The EOMs of the quadcopter are found by taking the time derivative of the state and applying the control inputs:

$$\dot{\boldsymbol{x}} = \left[\dot{\boldsymbol{r}}, \dot{\boldsymbol{\theta}}, \dot{\boldsymbol{v}}, \dot{\boldsymbol{\omega}}\right]^{\boldsymbol{T}} \tag{3}$$

To begin, it is trivial to show the time derivatives of position and velocity:

$$\dot{\boldsymbol{r}} = \boldsymbol{v} \quad ; \quad \dot{\boldsymbol{v}} = [0, 0, mg]^T - R_{zyx}(\boldsymbol{\theta})[0, 0, F]^T \tag{4}$$

where the rotation matrix $R_{zyx}(\boldsymbol{\theta})$ is the z-y-x Tait-Bryan rotation sequence defined in Equations (5) and (6).

$$R_{zyx}(\boldsymbol{\theta}) = R_z R_y R_x \tag{5}$$

$$R_z = \begin{bmatrix} cos\,\theta_z & -sin\,\theta_z & 0 \\ sin\,\theta_z & cos\,\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; R_y = \begin{bmatrix} cos\,\theta_y & 0 & sin\,\theta_y \\ 0 & 1 & 0 \\ -sin\,\theta_y & 0 & cos\,\theta_y \end{bmatrix}, \; R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\theta_x & -sin\,\theta_x \\ 0 & sin\,\theta_x & cos\,\theta_x \end{bmatrix} \tag{6}$$

To obtain the attitude and attitude rate derivatives, some additional algebra is required. The attitude is given as z-y-x Tait-Bryan angles, so $\dot{\boldsymbol{\theta}}$ can be obtained by converting $\boldsymbol{\omega}$, which is in the body frame, to the inertial frame. The equation for the inverse operation, converting attitude rate from the inertial frame to body frame, is as follows [1]:

$$\boldsymbol{\omega} = \boldsymbol{I_3} \begin{bmatrix} \dot{\theta}_x \\ 0 \\ 0 \end{bmatrix} + R_x^T \begin{bmatrix} 0 \\ \dot{\theta}_y \\ 0 \end{bmatrix} + \left(R_x R_y\right)^T \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_z \end{bmatrix} \tag{7}$$

The body to inertial frame transformation is then easily found:

$$\dot{\boldsymbol{\theta}} = \left[\mathrm{col}_1(\boldsymbol{I_3}), \mathrm{col}_2(R_x^T), \mathrm{col}_3\left(R_y^T R_x^T\right)\right]^{-1} \boldsymbol{\omega} \tag{8}$$

The time derivative of attitude rate in the body frame is found by taking the derivative of the moment equation, accounting for the Coriolis effect in the body frame. This yields the torque applied:

$$\frac{d}{dt}(\mathcal{J}\boldsymbol{\omega}) = \mathcal{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathcal{J}\boldsymbol{\omega} = \boldsymbol{\tau} \tag{9}$$

The above can then be rearranged to find $\dot{\boldsymbol{\omega}}$ in Equation (10), thus completing the derivation of the EOMs.

$$\dot{\boldsymbol{\omega}} = \mathcal{J}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathcal{J}\boldsymbol{\omega}) \tag{10}$$

The EOMs are numerically propagated for the simulation, yielding the final equation to implement the simulation:

$$\boldsymbol{x_{t+1}} = \boldsymbol{x_t} + \dot{\boldsymbol{x}}_{\boldsymbol{t}} \cdot dt \tag{11}$$

B. Cascaded PID Controller

The Crazyflie 2.1 uses a cascaded PID controller to convert vehicle and target state knowledge to motor commands. The cascaded PID controller is comprised of four individual PID controllers for position, velocity, attitude, and attitude rate. Given the desired position and current position, the position PID generates the desired velocity. The velocity PID then uses that information to generate the desired attitude, and so forth, cascading until the attitude rate PID outputs the motor power distribution commands. A detailed schematic describing the Crazyflie's firmware implementation of the cascaded PID controller is shown in Figure 1.

The simulation developed in this report will model all four PID controllers as well as the motor power distribution as shown in Figure 1, whereas the state estimator and the sensor measurements will be represented by random perturbations applied to ground truth states. Implementation details are discussed in the next section.

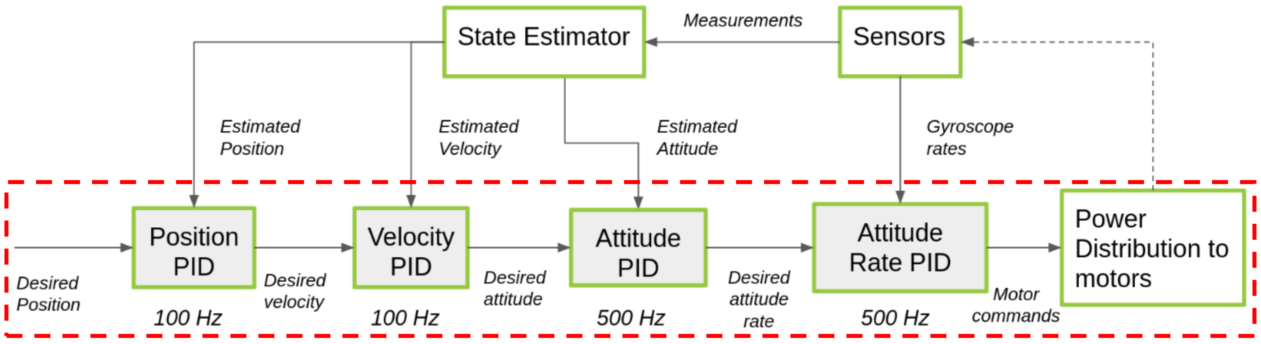

Figure 1: The cascaded PID controller implemented in Crazyflie 2.1 [2].

C. Crazyflie Firmware Emulation

The Crazyflie's cascaded PID controller firmware is extensively documented on GitHub [3]. The firmware, written in C, was translated into MATLAB for this report. Its detailed implementation is described in this section.

1. PID Controller

Each segment of the cascaded PID controller comprises of an individual PID controller, governed by the following:

$$E(t) = D(t) - M(t) \tag{12}$$

$$U = K_p E(t) + K_i \int_0^t E(t')dt' + K_p \frac{dE(t)}{dt} \tag{13}$$

Furthermore, the control law can be discretized for implementation, assuming a constant timestep dt:

$$U(t_{i+1}) = K_p E(t_i) + K_i dt \sum_{j=0}^{i} E(t_j) + \frac{K_p}{dt}\big(E(t_i) - E(t_{i-1})\big) \tag{14}$$

Each of the four PID controllers are operated at their respective frequencies indicated in Figure 1.

2. Low Pass Filter

Low pass filters are implemented for the position and velocity PID controllers, with a cutoff frequency of 20Hz. Specifically, the Crazyflie firmware uses a moving average filter across the three most recent timesteps [4]:

$$S_{i+1} = b_0 S_i + b_1 S_{i-1} + b_2 S_{i-2} \tag{15}$$

with the following definitions:

$$S_i = I_i - a_1 S_{i-1} - a_2 S_{i-2} \tag{16}$$

$$a_1 = \frac{2(\Omega^2 - 1)}{c} \quad , \quad a_2 = \frac{1 - \sqrt{2}\Omega + \Omega^2}{c} \tag{17}$$

$$b_0 = \frac{\Omega^2}{c} \quad , \quad b_1 = \frac{2\Omega^2}{c} \quad , \quad b_2 = \frac{\Omega^2}{c} \tag{18}$$

$$\mathcal{F} = \frac{f_{sample}}{f_{cutoff}} \quad , \quad \Omega = tan\left(\frac{\pi}{\mathcal{F}}\right) \quad , \quad c = 1 + \sqrt{2}\Omega + \Omega^2 \tag{19}$$

The sample frequency, f_{sample}, is the rate at which the position/velocity PID is run, which is 100Hz.

3. *Post-Processing for the Velocity PID*

The velocity PID controller outputs the desired attitude. However, in accordance with Crazyflie firmware, additional processing is needed to convert the raw output from the velocity PID to the desired attitude [5]. Denote the outputs from the x-, y-, and z-axis velocity PIDs as U_{vx}, U_{vy}, U_{vz}; the pre-processed thrust value is simply the raw output of the z-axis velocity PID, as shown in Equation (20).

$$F^* = U_{vz} \tag{20}$$

The desired attitude, $\tilde{\theta}$, is computed according to lines 210-211 of [5]:

$$\widetilde{\theta_x} = -U_{vy}\, cos\, \theta_z + U_{vx}\, sin\, \theta_z \tag{21}$$

$$\widetilde{\theta_y} = U_{vx}\, cos\, \theta_z + U_{vy}\, sin\, \theta_z \tag{22}$$

The desired yaw angle, $\widetilde{\theta_z}$, is implemented differently than the other angles. From Lab 1, it was found that the yaw angle gradually decayed asymptotically toward 0°, at a rate of approximately 7.5°/sec. This was found to be a result of the Kalman filter attempting to mitigate yaw drift due to gyro measurements [6]. For the sake of simplicity, the gyro drift and Kalman filter were not implemented, and as a result the only relevant firmware implementation sets the desired yaw angle equal to the yaw angle of the previous timestep, as seen in line 70 of [7]:

$$\widetilde{\theta_z}(t_{i+1}) = \theta_z(t_i) \tag{23}$$

4. *Post-Processing for the Attitude Rate PID*

The output of the attitude rate PID, $U_{\omega_x}, U_{\omega_y}, U_{\omega_z}$, are converted to raw motor outputs in the form of unsigned 16-bit integers according to the following formulas found between lines 87-92 of [8]:

$$F_1^{\#} = F^* - \frac{1}{2}U_{\omega x} + \frac{1}{2}U_{\omega y} - \widetilde{\theta_z} \tag{24}$$

$$F_2^{\#} = F^* - \frac{1}{2}U_{\omega x} - \frac{1}{2}U_{\omega y} + \widetilde{\theta_z} \tag{25}$$

$$F_3^{\#} = F^* + \frac{1}{2}U_{\omega x} - \frac{1}{2}U_{\omega y} - \widetilde{\theta_z} \tag{26}$$

$$F_4^{\#} = F^* + \frac{1}{2}U_{\omega x} + \frac{1}{2}U_{\omega y} + \widetilde{\theta_z} \tag{27}$$

5. *Output Limits*

The raw output of the cascaded PID controller, $F_1^{\#} - F_4^{\#}$, are constrained between 0 and 65535, which is the maximum value of an unsigned 16-bit integer [8]. Integration limits are applied to the integral errors of each PID controller as shown in Table 1. Finally, control limits are applied to the desired states as shown in Table 2.

Table 1: Integration limit for the integral errors of the cascaded PID controller [5] [9].

Axis	Position (m)	Velocity (m/s)	Attitude (°)	Attitude Rate (°/s)
X	±5000	±5000	±20	±33.3
Y	±5000	±5000	±20	±33.3
Z	±5000	±5000	±360	±166.7

Table 2: Output limit for elements of the cascaded PID controller [5].

Axis	Position (m)	Velocity (m/s)	Attitude (°)	Attitude Rate (°/s)
X	-	±1.1	±22	-
Y	-	±1.1	±22	-
Z	-	±1.1	-	-

6. Converting Motor Commands to Torque and Thrust

The EOMs derived in subsection A take torque and thrust in the body frame as the input $\boldsymbol{u}$. To convert from raw motor input to torque and thrust, the physical performance of the Crazyflie motors must be modeled. The Bitcraze Wiki contains testing data relating the raw motor signal, propeller RPM, and thrust [10], from which the following 3rd order polynomial fit (R=0.995) is derived, shown in Equation (28) and Figure 2:

$$RPM = 1.280 \times 10^{-10}(F^{\#})^3 - 1.513 \times 10^{-5}(F^{\#})^2 + 0.8267(F^{\#}) + 874 \quad (28)$$

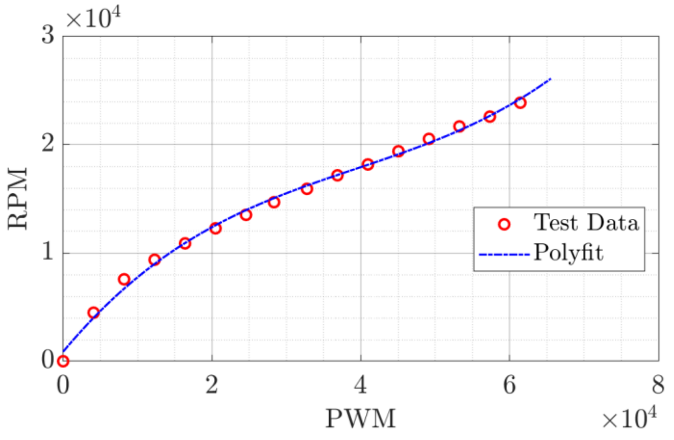

Figure 2: Transfer function from motor raw input (PWM) to propeller RPM.

The rotations per second, n, of the motor is then trivially:

$$n = RPM/60 \quad (29)$$

The thrust and torque are then modeled by the following equations [1]:

$$F = C_T \rho n^2 D^4 \quad (30)$$

$$\tau = \frac{1}{2\pi} C_P \rho n^2 D^5 \quad (31)$$

where C_T and C_P are experimentally determined coefficients, ρ is the air density ($1.225 kg/m^3$), and D is the diameter of the propeller ($45mm$). Past studies have characterized the coefficients specifically for the Crazyflie's motor and propeller, indicating that C_T and C_P are actually functions of n [1]. However, since the coefficients do not vary significantly across the range of possible values of n, C_T and C_P were set as constants in the simulation ($C_T = 0.15$, $C_P = 0.10$).

The total torque along the z-axis is found by summing the torques from each motor. Note that motors 2 and 4 rotate clockwise, so the torques from motors 2 and 4 need to be subtracted instead of added:

$$\tau_z = \tau_1 - \tau_2 + \tau_3 - \tau_4 \tag{32}$$

For the x- and y-axes, the torques can be found by the following:

$$\tau_x = L/2 \cdot (F_3 + F_4 - F_1 - F_2) \tag{33}$$

$$\tau_y = L/2 \cdot (F_2 + F_3 - F_1 - F_4) \tag{34}$$

Where L is the distance between two adjacent motors, which is $92mm$. Finally, the thrust is simply:

$$F_{tot} = F_1 + F_2 + F_3 + F_4 \tag{35}$$

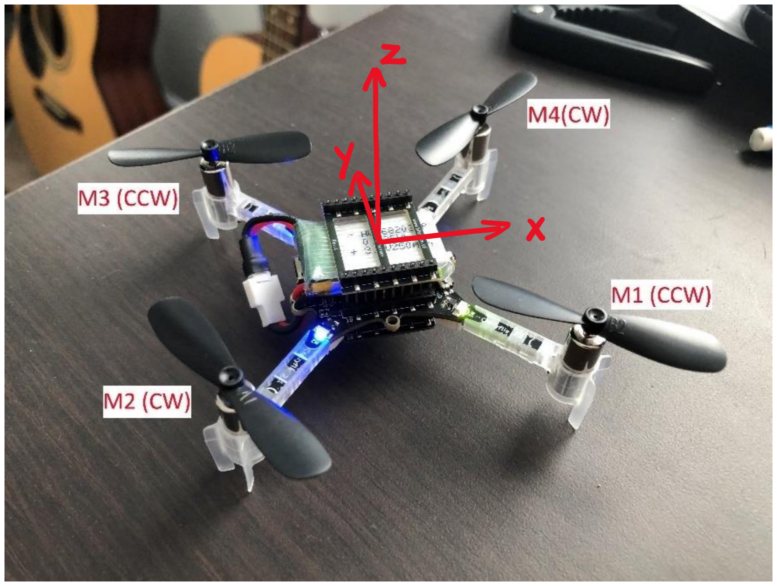

Figure 3: The Crazyflie 2.1 quadcopter's body frame orientation and motor arrangement.

D. Hardware and Perturbation Modeling

In addition to emulating Crazyflie firmware, the physical parameters of the Crazyflie are modeled including the mass and moment of inertia tensor. Furthermore, practical limitations on the sensing and output of the quadcopter are also simulated to achieve a more accurate model, including the motor ramp-up time and sensor noise.

1. Inertial Properties

The mass of the Crazyflie is 27 grams, while the Flow Deck v2 expansion board adds another 1.6 grams [11] [12] for a total of 28.6 grams. The moment of inertia of the quadcopter is calculated by modeling the Flow Deck, battery, and main board as three separate cuboids and modeling motors as point masses. The resulting moment of inertia tensor is shown in Equation (36).

$$I_{xx} = 2.965 \times 10^{-5} kg \cdot m^2 \quad , \quad I_{yy} = 2.990 \times 10^{-5} kg \cdot m^2 \quad , \quad I_{zz} = 5.948 \times 10^{-5} kg \cdot m^2 \tag{36}$$

2. Motor Delay

The motor used by the Crazyflie 2.1, when fitted with the propeller, takes approximately $180ms$ to reach full speed from rest [13]. This motor delay effectively serves as a low pass filter for signals sent to the motor. This delay is approximately modeled by limiting the change in motor RPM to $RPM_{max}/0.18 \cdot dt$ at each timestep, where dt is the timestep of the attitude rate PID ($2ms$) since it commands the motor.

3. *Sensor Noise*

Sensor noise is modeled by adding a zero-mean normal distribution random perturbation to each of the 12 states. There is limited information from literature regarding the sensor noise variance, so the values in Table 3 were used with limited justification. Future work in characterizing sensor noise variance would be greatly beneficial to improving the sensor model and thus simulation accuracy.

Table 3: 1-σ sensor noise standard deviations.

Axis	Position (m)	Velocity (m/s)	Attitude (°)	Attitude Rate (°/s)
X	0.005	0.02	0.2	3
Y	0.005	0.02	0.2	3
Z	0.005	0.02	0.2	3

IV. Flight Data Comparison

The simulation developed in Section III was used to predict flight performance when following flight plans A and B from Lab 3. For each flight, five waypoints are set up within a 1-meter cube. Flight plan A requires the quadcopter to hover within 5cm of each waypoint for 1 second before proceeding to the next waypoint. For flight plan B, only the first and last waypoints have this requirement, and the quadcopter needs only pass by the other waypoints within 5cm. A summary of the two flight plans is shown in Table 4. In Lab 3, only the position PID controller gains were tuned. The gain values used for each flight plan is shown in Table 5.

Table 4: Summary of position and hover requirements for flight plans A and B.

Waypoint #	Waypoint Position [x, y, z] (m)	Hover Time (s)	
		Flight Plan A	Flight Plan B
1	[0.0, 0.0, 0.5]	1	1
2	[1.0, 0.0, 1.0]	1	0
3	[1.0, 1.0, 0.5]	1	0
4	[0.0, 1.0, 1.0]	1	0
5	[0.0, 0.0, 0.5]	1	1

Table 5: PID controller gains for flight plans A and B and corresponding flight times.

Flight Plan	$[K_p, K_i, K_d]_x$	$[K_p, K_i, K_d]_y$	$[K_p, K_i, K_d]_z$
A	[2.2, 0.0, -0.05]	[2.2, 0.0, -0.05]	[2.6, 0.4, 0.05]
B	[3.3, 0.1, 0.1]	[3.6, 0.1, 0.1]	[1.9, 0.2, 0.1]

A. Flight Plan A Comparison

The trajectory from simulation and experimental data are coplotted in Figure 4. Qualitatively, the simulation flight performed very similarly to the actual flight. The shape of the trajectory between waypoints shows strong resemblance, though the simulation appears to overshoot each waypoint by a few centimeters more than its physical counterpart.

Upon closer examination of the profile along each axis in Figure 5, the simulation successfully produced the expected rise time and settling time for all three axes, though along the y-axis the rise time predicted by the simulation was shorter than the experimental results. It appears that the actual system was more damped than predicted by the simulation, which may be caused by factors such as drag, which are not accounted for by the simulation.

Finally, the time spent at each waypoint is shown in Figure 6. For each waypoint, the time spent was comparable between the simulation and the actual flight, and the overall flight time difference was approximately 1 second. This indicates that the simulation is a decent predicter of the physical flight performance of the Crazyflie for flight plan A.

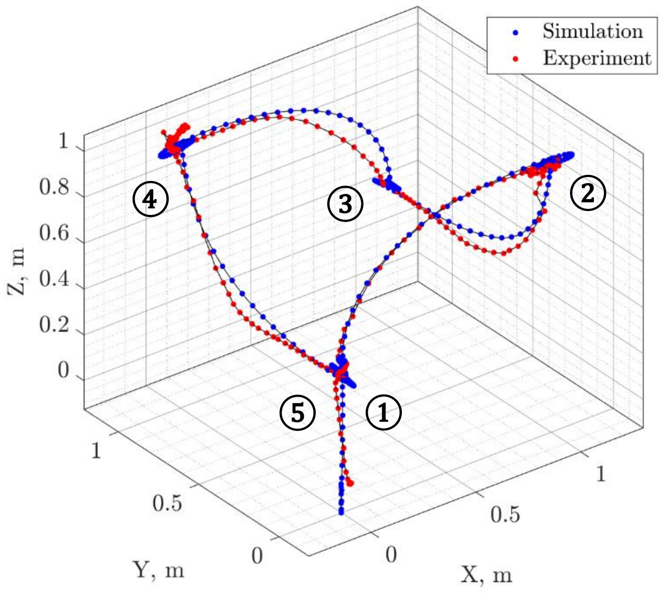

Figure 4: Flight trajectories for plan A generated from simulation and experimental data.

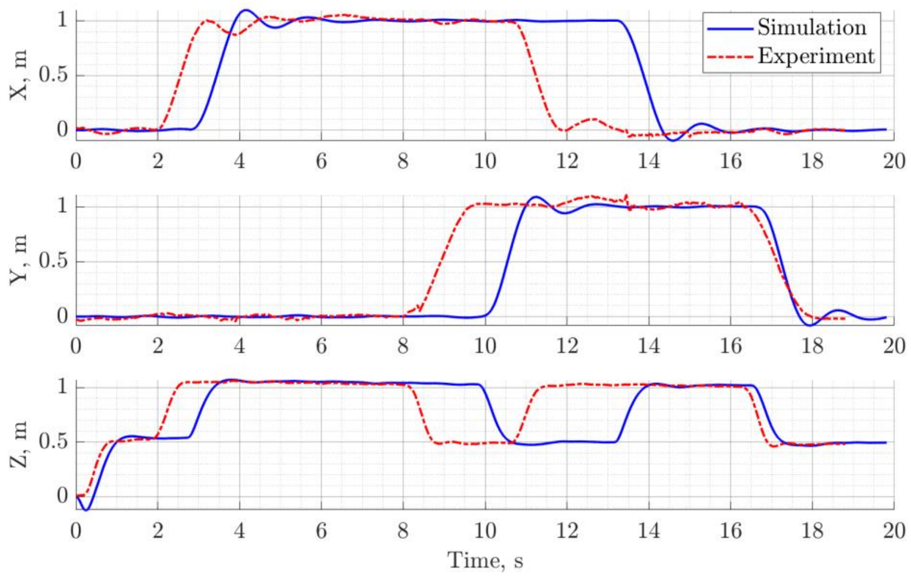

Figure 5: Flight plan A profile comparison between simulation and experimental data.

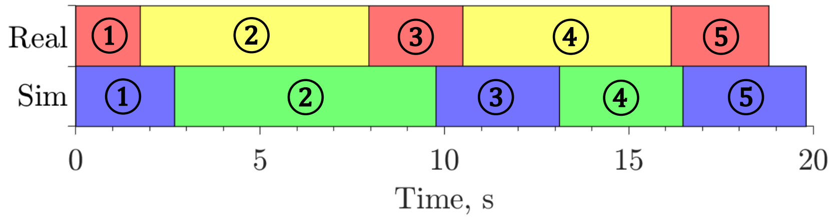

Figure 6: Time spent reaching each waypoint for the real and simulated flights in plan A.

B. Flight Plan B Comparison

Flight plan B removes the intermediate waypoint hover time requirements, meaning the quadcopter could proceed to the next waypoint immediately after flying within $5cm$ of the current waypoint. This caused the overshoot observed in the simulation for flight plan A to propagate into the subsequent waypoints, since the simulated quadcopter did not have time to correct its position error. As a result, the simulation flight matched poorly with the actual flight data.

Once again, while the simulation and experimental data shared the same PID controller parameters, the physical flight appears to be more damped than the simulation, as seen in Figure 8. Surprisingly, the simulation flight still matched the flight time of the experimental data (most likely due to the lack of hover time requirements), until the last waypoint where hover was required. At this point, all the previously accumulated errors caused the simulation to spiral around the waypoint, leading to a significantly longer flight time near the end as seen in Figure 9.

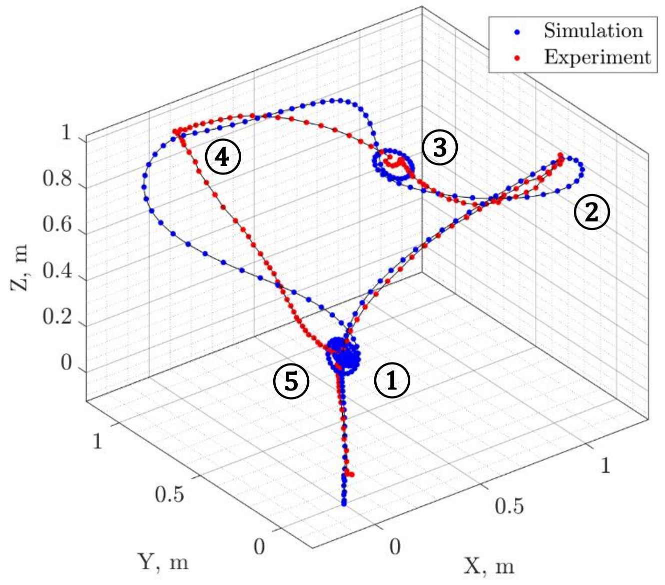

Figure 7: Flight trajectory for plan B generated from simulation and experimental data.

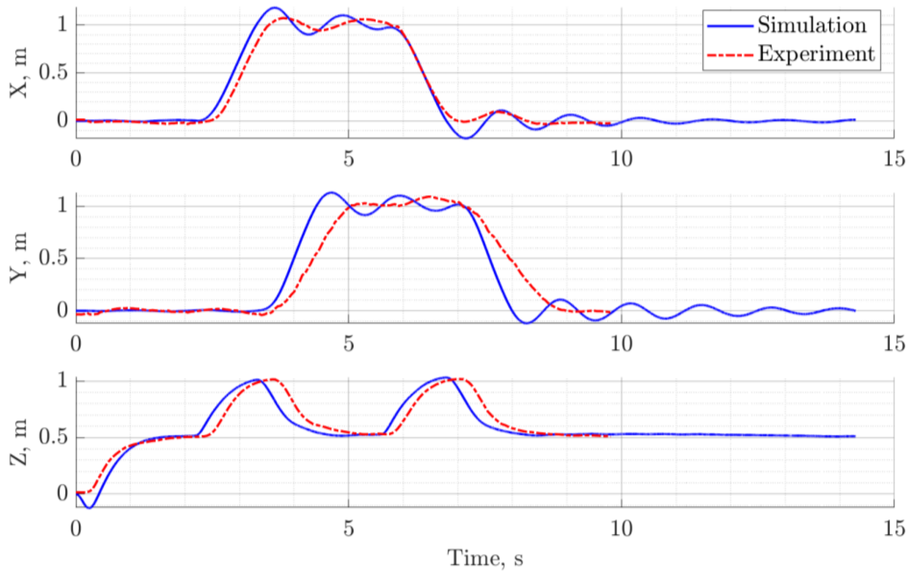

Figure 8: Flight plan B profile comparison between simulation and experimental data.

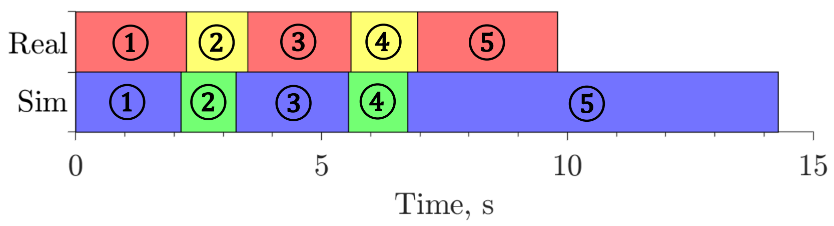

Figure 9: Time spent reaching each waypoint for the real and simulated flights in plan B.

C. Error Sources and Model Improvements

Inconsistencies between the simulation and experimental data can be largely attributed to the difference in damping and overshoot. Specifically, the simulated system was less damped than the real system, causing the overshoot and settling time to be longer. Potential causes for this difference are discussed in this subsection.

While state data in collected via sensors in the actual quadcopter, they are instead directly collected from the ground truth states in the simulation. Consequently, the sensor reading errors have to be artificially generated, for which the variance and bias was largely unknown. The actual sensor noise can be drastically different from the idealized zero-mean normal distribution model used in the simulation.

Furthermore, aerodynamic effects beyond propeller thrust were not included, such as drag and the ground effect. Since the quadcopter was hovering between 0.5 and 1.0 m above the ground, it is possible the turbulence caused by the propellers generated airflow that contributed to the dampening of the system.

In addition, it is known that the motors and propellers have power/thrust variance due to imperfections in manufacturing as well as deformations caused by crashes. This would cause the thrust and torque input to be slightly perturbed, but the variance was omitted in the simulation.

Other limitations include sources such as the accuracy/resolution of numerical integration, or the approximation of the quadcopter component geometry for moment of inertia calculations.

V. Conclusion

The dynamics of the Crazyflie 2.1 quadcopter was successfully modeled, and its firmware was implemented along with the quadcopter dynamics in a detailed MATLAB simulation. Comparisons between the actual flight data and simulation data indicate that the simulation decently reflects actual flight performance, although discrepancies in the damping of the system were identified. A broad range of improvements that can be made to the simulation to improve its fidelity were discussed.

References

[1] G. P. Subramanian, "Nonlinear Control Strategies for Quadrotors and Cubesats," University of Illinois at Urbana-Champaign, Department of Aerospace Engineering, Urbana, IL, 2015.

[2] bitcraze, "Controllers in the Crazyflie," 2020. [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/#cascaded-pid-controller. [Accessed 6 December 2020].

[3] bitcraze, "Crazyflie Firmware GitHub Repository," bitcraze, 2020. [Online]. Available: https://github.com/bitcraze/crazyflie-firmware/tree/master/src. [Accessed 11 December 2020].

[4] bitcraze, "filter.c," 21 October 2020. [Online]. Available: https://github.com/bitcraze/crazyflie-firmware/blob/dceb05160abc1c2250b97fe2ed4275dc447aaba8/src/utils/src/filter.c. [Accessed 13 December 2020].

[5] bitcraze, "position_controller_pid.c," 2 August 2018. [Online]. Available: https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/position_controller_pid.c. [Accessed 13 December 2020].

[6] bitcraze staff (arnaud), "Bitcraze Forums," 6 February 2007. [Online]. Available: https://forum.bitcraze.io/viewtopic.php?t=3436. [Accessed 13 December 2020].

[7] bitcraze, "controller_pid.c," 15 January 2020. [Online]. Available: https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/controller_pid.c. [Accessed 13 December 2020].

[8] bitcraze, "power_distribution_stock.c," 8 October 2020. [Online]. Available: https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/power_distribution_stock.c. [Accessed 13 December 2020].

[9] bitcraze, "attitude_controller_pid.c," 13 August 2019. [Online]. Available: https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/src/attitude_pid_controller.c. [Accessed 13 December 2020].

[10] bitcraze, "Finding a PWM to Thrust Transfer Function," 15 July 2015. [Online]. Available: https://wiki.bitcraze.io/misc:investigations:thrust. [Accessed 14 December 2020].

[11] bitcraze, "Crazyflie 2.1," 2020. [Online]. Available: https://store.bitcraze.io/products/crazyflie-2-1. [Accessed 14 December 2020].

[12] bitcraze, "Flow v2 Deck," 10 March 2020. [Online]. Available: https://wiki.bitcraze.io/projects:crazyflie2:expansionboards:flow-v2. [Accessed 14 December 2020].

[13] bitcraze, "Measuring Propeller RPM: Part 3," 9 February 2015. [Online]. Available: https://www.bitcraze.io/2015/02/measuring-propeller-rpm-part-3/. [Accessed 14 December 2020].

[14] bitcraze, "Crazyflie 2.1," 3 October 2020. [Online]. Available: https://www.bitcraze.io/products/crazyflie-2-1/.

[15] K. Ang, G. Chong and Y. Li, "PID Control System Analysis, Design, and Technology," *IEEE Transactions on Control Systems Technology*, vol. 13, pp. 559-576, 2007.

[16] AE 483 Course Staff, "Compass2g - Lab 2 Week 1," October 2020. [Online]. [Accessed 8 November 2020].

[17] AE 483 Course Staff, "Compass2g - Lab 2," 29 October 2020. [Online]. [Accessed 2 December 2020].