Group 13 Modelling, Simulation, and Control Report

Anton Stigemyr Hill David Espedalen Siyu Yi Yinsong Wang

***Abstract*—This project is all about math and controlling a quadrotor drone called the Crazyflie 2.0. We're diving deep into it, looking at everything from how it moves to how we can make it do what we want. We're using lots of different things, like math to describe how it works, fancy techniques to figure out which way it's facing, and special filters to make sure we get accurate measurements of its angles. We're also simplifying the tricky math to make it easier to work with, and using a special type of control called LQR to keep the drone steady in the air.**

At the start, we're using tools like Matlab and Simulink to play around with the settings and see how everything behaves in a virtual world. It's like a video game where we can tweak things until they work just right. These tools help us fine-tune our control methods before we take them out into the real world.

***Index Terms*—Crazyflie 2.0, model-based design, quadrotor simulation, LQR control**

I. Introduction

In a more and more automated everyday life, model-based design can be seen in all kinds of different systems from heat regulation in homes to control of autonomous drive systems. Even though these systems may seem far from each other at first glance, the control theory behind them works in the same way. This is the reason why control theory is an important issue for many people in today's world. Model-based design is a tool to speed up development processes and save time and resources by simulating, analyzing, and optimizing these systems before implementation.

In this Project, the development and implementation of a control system is done for the Quadcopter Crazyflie from Bitcraze. This is done to get hands-on experience with the theories learned in the MPSYS program at Chalmers. This is an important set of knowledge since to implement control systems in real life some additional steps that lay outside the range of control theory are needed. These include setting up a test environment, converting the control system into code the drone can understand, and uploading the code to the drone. This was done in a Matlab environment in Simulink and Simscape created by the course which was then compiled to C-code and later uploaded to the quadcopter via Bluetooth using a VirtualBox. [1]

To be able to understand this project, some prior knowledge of basic control systems design is needed however, the concepts will be explained rather thoroughly.

II. Orientation Estimation

The position of the quadrotor is defined by a state vector with six states, $x, y, z, \phi, \theta, \psi$. Where ϕ, θ, ψ represents the rotation in the three different directions. The rotation is denoted by applying the right-hand rule which easily gives the positive or the negative rotation direction. To get the quadrotor's orientation from each motor, i.e. body, to the world frame a rotation matrix is used. The rotation matrix is given by the product of the three angles rotation matrices. To be more clear, the angles roll (ϕ), pitch (θ), yaw (ψ). Which got the rotation matrices R_ϕ, R_θ and R_ψ is multiplied together to get the rotation and transformation from the body to the world frame. $R_{xyz} = R_x(\phi) * R_y(\theta) * R_z(\psi)$.

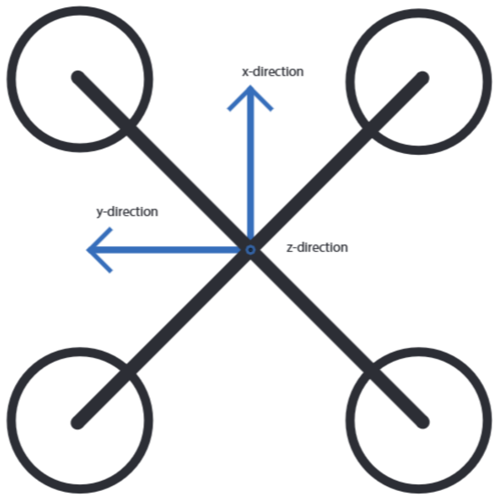

Fig. 1. Drone orientation

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\varphi) & -sin(\varphi) \\ 0 & sin(\varphi) & cos(\varphi) \end{bmatrix} \tag{1}$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{2}$$

$$R_z(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

A. Estimate angles from accelerometers

The roll, pitch, and yaw angles are derived using the gyroscope and the accelerometer in the quadrotor. The gyroscope measures the angular velocity, i.e. the rate of change of the quadrotor's orientation. The accelerometer defines the external G-force on the quadrotor sensors and the quadrotor's gravity itself.

$$\begin{bmatrix} {}^{B}f_x \\ {}^{B}f_y \\ {}^{B}f_z \end{bmatrix} = m *{}^{B} R_{\omega} * \left(\begin{bmatrix} {}^{\omega}a_x \\ {}^{\omega}a_y \\ {}^{\omega}a_z \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \right) \tag{4}$$

When body is at rest, ${}^{\omega}a_x = 0$, ${}^{\omega}a_y = 0$ and ${}^{\omega}a_z = 0$, the accelerometer only measures the acceleration as gravity. The accelerometer was tested which revealed it was calibrated to give -1 when aligned with the earth gravity field. We then have:

$$\begin{bmatrix} {}^{B}f_x \\ {}^{B}f_y \\ {}^{B}f_z \end{bmatrix} = m *{}^{B} R_{\omega} * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{5}$$

From this we know that ${}^{B}R_{\omega} = ({}^{\omega}R_B)^T$, which gives the following expression for ${}^{B}f_x$, ${}^{B}f_y$ and ${}^{B}f_z$.

$$\begin{aligned} \begin{bmatrix} {}^{B}f_x \\ {}^{B}f_y \\ {}^{B}f_z \end{bmatrix} &= m * R_x^T(\varphi) * R_y^T(\theta) * R_z^T(\psi) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ &= m * \begin{bmatrix} -sin(\theta) \\ cos(\theta)sin(\varphi) \\ cos(\theta)cos(\varphi) \end{bmatrix} \end{aligned} \tag{6}$$

With this we can solve the angles φ and θ, which gives:

$$\varphi = arctan(\frac{{}^{B}f_y}{{}^{B}f_z}) \tag{7}$$

and

$$\theta = arctan(\frac{-{}^{B}f_x}{\sqrt{({}^{B}f_y)^2 + ({}^{B}f_z)^2}}) \tag{8}$$

B. Estimate angles from gyroscope

The gyroscope gives the angular velocity of the angles, in the discrete time system we just get the angle's value from:

$$\theta_{g,k} = \theta_{g,k-1} + \dot{\theta}_{g,k-1} \cdot T_s \tag{9}$$

C. Complementary filter

The estimations from the accelerometer are noisy for short time intervals but are accurate for longer periods. The gyroscope is accurate over short time periods but very sensitive to overshoot, therefore a complementary filter is applied to the system where a low-pass filter is used through the accelerometer and a high-pass filter through the gyroscope. The complementary filter is applied at discrete time which means that the continuous system needs to be converted to discrete time before using the complementary filter. A chart of the complementary filter is shown in Figure 2.

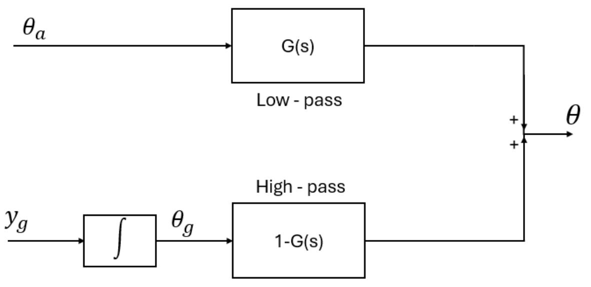

Fig. 2. Complementary filter

Here we have the two filters, i.e. the low-pass and the high-pass filter defined as:

$$G(s) = \frac{1}{\alpha s + 1} \tag{10}$$

and

$$1 - G(s) = 1 - \frac{1}{\alpha s + 1} = \frac{\alpha s}{\alpha s + 1} \tag{11}$$

Now let θ_a be the estimation from the accelerometer and θ_g be the estimation from the gyroscope after one integration, $\theta_g = \int_0^t y_g dt$. From figure 2 the estimated θ after the complementary filter is calculated as:

$$\theta(S) = G(S) \cdot \theta_a(S) + (1 - G(S)) \cdot \theta_g(S) \tag{12}$$

III. Plant Modeling

When modeling the plant, Matlab's Simscape was used. This is a tool to model systems using only equations. The difference between Simulink and Simscape is that in Simscape, block diagrams are not used. Instead, the relevant relations between variables are stated and then all possible equations are extracted automatically. Below is a table to introduce the different variables used.

Parameter	Explanation
v	speed
T	total thrust
F	air friction coefficient
τ	Torque
L	lift
D	drag
d	distance between motor and origin
K	LQR feedback gain
Q	states weighting matrix
R	inputs weighting matrix
${}^{B}R_{\omega}$	rotation matrix from body to world
${}^{\omega}R_{B}$	rotation matrix from world to body
θ_a	Estimation from accelerometer
θ_g	Estimation from gyroscope
y_g	Estimation from gyroscope (before integration)
φ	Roll angle
θ	Pitch angle
ψ	Yaw angle
T_s	Sampling time
An	Continious time A matrix
Bn	Continious time B matrix
A_d	Discrete time A matrix
B_d	Discrete time B matrix
r	reference

When modeling the plant, all equations should be stated in the world frame and therefore we will have to use the rotation matrices developed in the previous chapter to switch between frames. First of we begin with the law of motion. To do this in the world frame the following equation is used.

$$ma =^{B} R_{\omega} * \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ m * g \end{bmatrix} - F * v \tag{13}$$

Next, we need to define the angular equations of motion. Once again we begin in the body frame and later move it to the world. The torque generated in the body frame can be expressed in x, y, and z directions.

$$\tau_{tot} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} d \cdot cos(\pi/4) \cdot (T_3 + T_4 - T_1 - T_2) \\ d \cdot cos(\pi/4) \cdot (T_2 + T_3 - T_1 - T_4) \\ D/L \cdot (T_2 + T_4 - T_1 - T_3) \end{bmatrix} \tag{14}$$

Now Newton's second law of rotation in vector space can be used

$$J \cdot \dot{\omega} = -\omega \times (J\omega) + \tau_{tot} \tag{15}$$

The last step is then to multiply this with the Euler angle transform matrix to get it to the world frame.

$$\omega_b = \begin{bmatrix} 1 & 0 & sin(\theta) \\ 0 & cos(\varphi) & -sin(\varphi)cos(\theta) \\ 0 & sin(\varphi) & cos(\varphi)cos(\theta) \end{bmatrix} \omega \tag{16}$$

To check whether the Simscape model worked as intended, two different test cases were used in an open-loop system. For the first one motor 1 and motor 2 were set to 10 at time 4s which results in the drone rotating with a negative roll. This can be seen in the plots for case 1 in figure 3 and 4. For the other case, motor 1 was set to 10000 at time 4s, and motor 3 was set to 2000 at time 6s. Now all angles are affected at 4s since the drone starts spinning in all directions and the accelerations jump from -1 to 1 because of that as well which can be seen in figure 5 and 6.

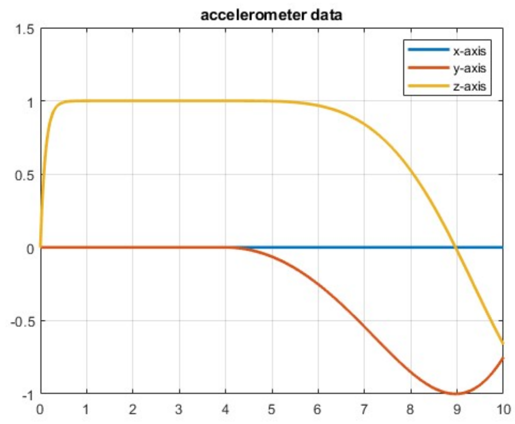

Fig. 3. Accelerometer data for case 1

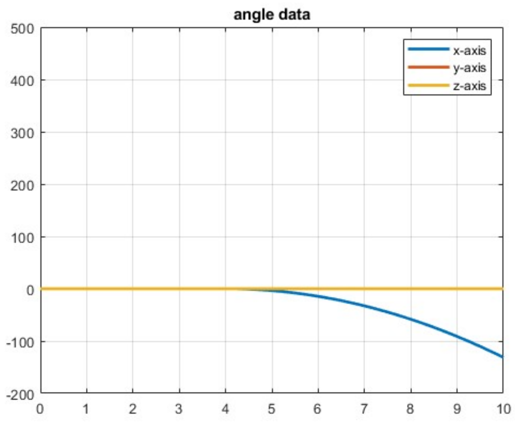

Fig. 4. Angle data for case 1

IV. Plant Linearization

We have chosen the state representation for this system as

$$x = \begin{bmatrix} \varphi & \theta & \dot{\varphi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{17}$$

where the model of the plant is described by the equation $\dot{x} = f(x, u)$. The dynamics of the system are captured by the function

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{\varphi} & \dot{\theta} & \ddot{\varphi} & \ddot{\theta} & \ddot{\psi} \end{bmatrix}^T \tag{18}$$

The input to the system consists of the thrust generated by four motors. To linearize the model, the operating point is set as

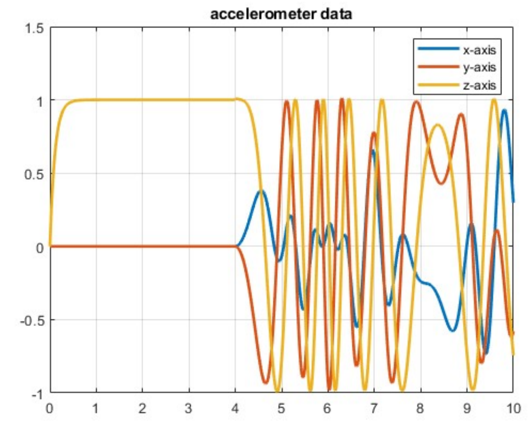

Fig. 5. Accelerometer data for case 2

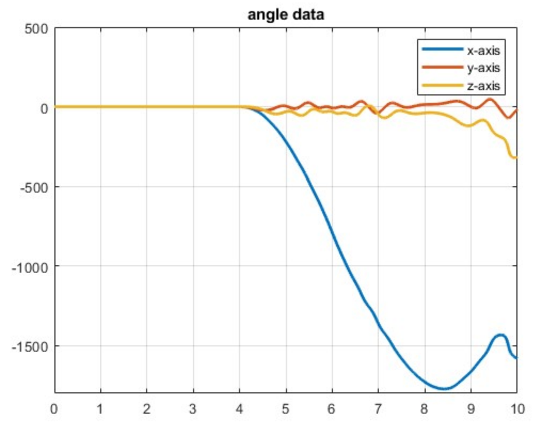

Fig. 6. Angle data for case 2

$u_{\text{lin}} = [0.0662\ 0.0662\ 0.0662\ 0.0662]^T$ and the initial state is $x_0 = [0\ 0\ 0\ 0\ 0]^T$. Subsequently, we can derive the state-space model equations as follows:

$$\dot{x} = \underbrace{\left.\frac{\partial f}{\partial x}\right|_{x_o, u_{\text{lin}}}}_{A} \Delta x + \underbrace{\left.\frac{\partial f}{\partial u}\right|_{x_o, u_{\text{lin}}}}_{B} \Delta u \tag{19}$$

Using the Jacobian method, we calculated the matrices A_n, B_n as follows:

$$A_n = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B_n = 10^3 \times \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -2.8376 & -2.8376 & 2.8376 & 2.8376 \\ -1.9141 & 1.9141 & 1.9141 & -1.9141 \\ -0.4282 & 0.4282 & -0.4282 & 0.4282 \end{bmatrix}$$

Now, using the linearized matrices A_n and B_n, we can calculate the discretized model. Given a sampling time of $T_s = 0.01$ seconds, the discrete-time system matrices, A_d and B_d, can be computed as follows: [2]

$$A_d = e^{A_n T_s} \tag{20}$$

$$B_d = \left(\int_{\tau=0}^{T_s} e^{A_n \tau} d\tau \right) B_n \tag{21}$$

the matrices A_d, B_d as follows:

$$A_d = \begin{bmatrix} 1 & 0 & 0.01 & 0 & 0 \\ 0 & 1 & 0 & 0.01 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B_d = \begin{bmatrix} -0.1419 & -0.1419 & 0.1419 & 0.1419 \\ -0.0957 & 0.0957 & 0.0957 & -0.0957 \\ -28.3756 & -28.3756 & 28.3756 & 28.3756 \\ -19.1414 & 19.1414 & 19.1414 & -19.1414 \\ -4.2821 & 4.2821 & -4.2821 & 4.2821 \end{bmatrix}$$

V. Design of the Linear Quadratic Controller

The Linear Quadratic Controller(LQR) could be developed following the block diagram in Figure 7.

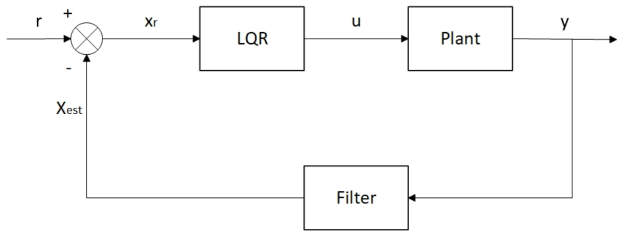

Fig. 7. LQR block diagram

Reference tracking is applied on the angular pitch and roll as well as yaw rate as angular velocity.

$$r = \begin{bmatrix} \varphi \\ \theta \\ \dot{\psi} \end{bmatrix} \tag{22}$$

VI. Design of the Linear Quadratic Regulator (LQR) Controller

Using the discrete-time state-space model, we design our LQR controller by focusing on minimizing the infinite time quadratic regulation cost function. This function is critical for evaluating the performance of the controller across an unlimited time horizon. The cost function J is defined as:

$$J = \min \sum_{k=0}^{\infty} x_k^\top Q x_k + u_k^\top R u_k \tag{23}$$

where x_k represents the state vector at step k, and u_k denotes the control input at step k. The matrices Q and R are weighting factors that balance the state error and control effort, respectively.

Next, we define the feedback gain, which is crucial for determining the effectiveness of the control response:

$$u(k) = -K\hat{x}(k) \tag{24}$$

Here, $\hat{x}(k)$ represents the estimate of the state at time k, and K is the feedback gain matrix calculated using the discrete-time Riccati equations:

$$\begin{aligned} K &= (B_d^\top S B_d + Q)^{-1} B_d^\top S A_d \\ S &= A_d^\top S A_d + Q - A_d^\top S B_d (B_d^\top S B_d + R)^{-1} B_d^\top S A_d \end{aligned} \tag{25}$$

These equations ensure that the feedback gain K optimizes the control actions based on the dynamics of the system encapsulated by matrices A and B.

The choice of Q and R matrices profoundly influences the controller's behavior. By adjusting Q, we prioritize the reduction of state deviations, leading to a more aggressive control action. Conversely, increasing R makes the system more conservative, reducing the magnitude of the control input to minimize effort and potential overshoot. These adjustments allow for fine-tuning the controller to meet specific performance criteria and system demands.

Then we can implement our theory into Simulink to establish our own block which we can see in Figure 8

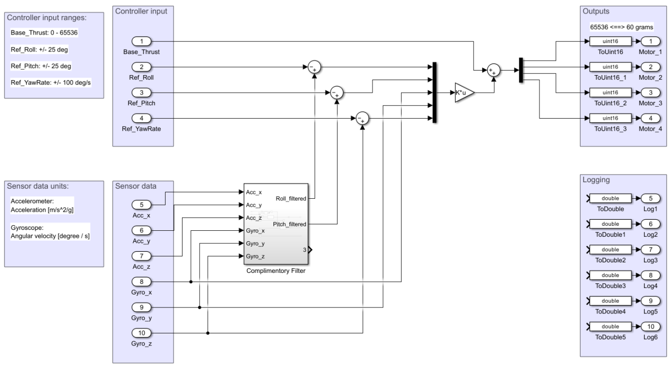

Fig. 8. Simulink block diagram

VII. Evaluation of Control Design

The controller was subsequently tested using the model. Figure 9 illustrates the system's ability to resist interference. A disturbance was introduced, and it was evident that the system promptly adjusted to mitigate its effects.

In the final part of the experiment, we fine-tuned the controller by changing values in the Q and R matrices. Specifically, we focused on adjusting the roll and pitch parts in the Q matrix. When the R matrix is small, the system responds slowly. Conversely, if the Q matrix is too big, the system overshoots. To get the best performance, we found that using the matrix described in Equation 26 resulted in Figure 10, showing the most favorable tuning.

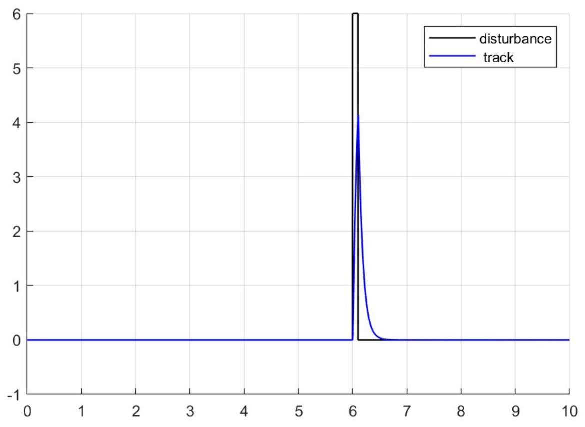

Fig. 9. Disturbance test of the controller

$$\begin{aligned} Q &= \begin{bmatrix} 1000000 & 0 & 0 & 0 & 0 \\ 0 & 1000000 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix} \\ R &= \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0.1 \end{bmatrix} \end{aligned} \tag{26}$$

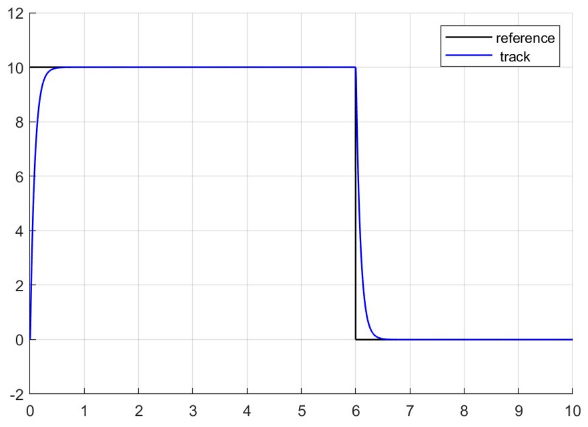

Fig. 10. Track following test of the controller

VIII. Conclusions

In this project, we have successfully developed and implemented a control system for the Quadcopter Crazyflie. Through our modeling, simulation, and control implementation, several challenges arose, particularly with the alignment between theoretical models and real-world behavior. These challenges included the uneven weight distribution of the

Quadcopter Crazyflie and the way the force from the fishing string was applied. Additionally, disturbances such as the drone hitting the table during takeoff in physical flight tests added further complexity to our control challenges.

We also encountered numerous issues during the control simulation phase. When attempting to use data from actual flights, we found that the control components seemed ineffective. We traced this ineffectiveness to the version of our modeling tool, MATLAB Simulink. This experience underscores the importance of maintaining a unified environment for consistent and reliable modeling.

However, by meticulously modeling and simulating each component of the Quadcopter Crazyflie, we gained a deep understanding of its operation, which makes it easier to modify or enhance the system. This detailed approach has proven immensely beneficial for our ongoing development efforts.

References

[1] SSY191 Course Team, "SSY191 Model-based development of cyber-physical systems," From the SSY191_Gettingstarted, Chalmers University of Technology, 2023.

[2] T. Glad and L. Ljung, *Control Theory - Multivariable and Nonlinear Methods*, Taylor & Francis, ISBN: 978-0-748-40878-8.