Model-based development of a quadrocopter control system

Ludvig Hazard
Chalmers University of Technology
Gothenburg, Sweden
hazardl@student.chalmers.se

Georg Hess
Chalmers University of Technology
Gothenburg, Sweden
georghe@student.chalmers.se

Tilani Gallege
Chalmers University of Technology
Gothenburg, Sweden
tilani@student.chalmers.se

William Ljungbergh
Chalmers University of Technology
Gothenburg, Sweden
willju@student.chalmers.se

***Abstract*—Model-based development consists of deriving a mathematical model, simulating the model, designing a control system, and testing and verification. In this project, such an approach was used to model, simulate, and control the Crazyflie 2.1 quadrocopter. The controller was implemented with custom C-code on FreeRTOS and controller evaluation was done by simulation.**

***Index Terms*—Quadrocopter, Linear Quadratic Regulator, Model-based development, Bitcraze, Crazyflie 2.1, Embedded System, Simulink, Simscape, FreeRTOS**

NOMENCLATURE

Name	Value	Unit	Description
m	0.027	kg	Mass of quadrocopter
g	9.81	$\mathrm{m/s^2}$	Gravitational acceleration
d	0.046	m	Arm length
k_d	$1.9796 \cdot 10^{-9}$		Drag constant
b	$2.5383 \cdot 10^{-11}$		Lift constant
J_x	$1.1463 \cdot 10^{-5}$	$\mathrm{kgm^2}$	Moment of inertia
J_y	$1.6993 \cdot 10^{-5}$	$\mathrm{kgm^2}$	Moment of inertia
J_z	$2.9944 \cdot 10^{-5}$	$\mathrm{kgm^2}$	Moment of inertia

I. INTRODUCTION TO CONTROL OF QUADROCOPTERS

Model based approach is a mathematical model that is used in engineering to solve complex problems where it reduces costs and increases efficiency by leveraging simulations and thus avoiding expensive hardware. The methodology consists of thorough equation-based modeling that forms the foundation for a robust simulation environment where most of the development takes place. When satisfactory properties are achieved, the simulated control behavior can be tested, fine-tuned, and verified in a real-world setting. Here, the model-based approach was applied to the development of a quadrocopter with the purpose of designing a control system able to keep the quadrocopter hovering and allowing for reference-tracking.

A quadrocopter is an aerial carrier that changes its orientation and position using four propellers and in this paper, the *Crazyflie 2.1* from *Bitcraze* was used, which can be seen in Figure 1. Two sensors hosted on the Crazyflie were used in this project, a gyroscope, and an accelerometer. These sensors were used to estimate the orientation of the quadrocopter. Since the measurements taken from these sensors were not accurate enough on their own, a complementary filter was designed to improve the accuracy of the estimates.

Further, a plant model was developed to enable simulations of the quadrocopter and ease the controller synthesis. The controller, a linear-quadratic regulator, was designed based on a linearized version of the plant model and utilizes the orientation estimates from the complementary filter for computing appropriate control signals. Due to limitations, no physical Crazyflie was available during this project and hence the plant model also was used for controller tuning and evaluation.

In order to implement the software MATLAB, Simulink, and Simscape were used in this project. After model verification and controller tuning were finished in these tools, the controller was implemented in C-code on the real-time operating system FreeRTOS, to allow for future integration in the physical Crazyflie.

II. ESTIMATION OF ORIENTATION

Effective control requires a good estimate of how the Crazyflie is oriented in space. This section will explain how such an estimate can be generated by leveraging a complementary filter and fusing different sources of information.

A. Quadrocopter coordinate system and rotation matrices

Two different coordinate systems were used throughout this project for estimating the orientation of the Crazyflie, the inertial frame, and the body frame. The inertial frame is the world coordinate system that is considered fixed in the world. In comparison, the body frame is fixed to the body of the Crazyflie and can move and rotate in relation to the inertial frame and is the one used by the internal sensors. Figure 1 shows the body frame and its orientation and position on the quadrocopter.

In the body frame, the x-axis is pointing to the front of the Crazyflie between motors 1 and 4, the y-axis is pointing to the left between motors 4 and 3 and the z-axis is pointing up.

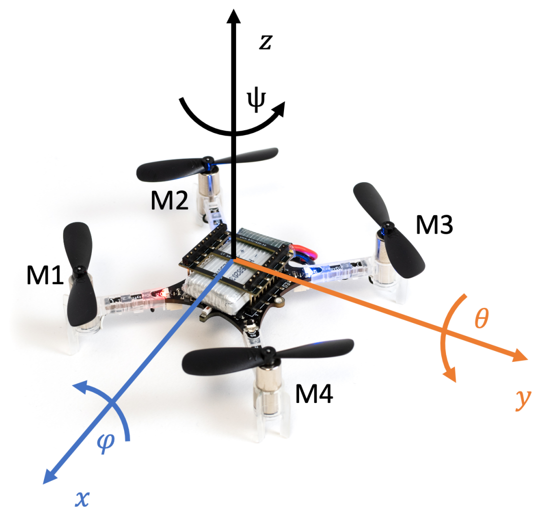


Fig. 1: Coordinate system used by the internal sensors of the Crazyflie, also known as body frame. M1 to M4 indicate numbering of motors. Image of Crazyflie 2.1 adapted from [1] under CC BY-SA 3.0.

The angles roll φ, pitch θ, and yaw ψ are positive around axes x, y, z respectively according to the right-hand rule.

Coordinates and measurements expressed in one frame can easily be translated to the other frame by using rotation matrices. The key idea is to do consecutive rotations around three axes where the amount of rotation is captured by the so called Euler angles (φ, θ, ψ). The rotation matrices from inertial to body frame for individual rotations from [2] are

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

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

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

where for example, $R^{bi}(\varphi)$ would rotate a vector by an angle φ around the x-axis. The full rotation can be found combining the elementary rotations. Multiple combinations are possible, depending on in which order the individual rotations are applied, but throughout this project the Tait-Bryan Euler angles XYZ (roll-pitch-yaw) rotation order was applied

$$R^{bi}_{XYZ}(\varphi, \theta, \psi) = R^{bi}(\varphi)R^{bi}(\theta)R^{bi}(\psi) \tag{4}$$

To clarify, a vector x^i in the inertial coordinate system can be expressed in the body frame as

$$x^b = R^{bi}_{XYZ}x^i \tag{5}$$

B. Accelerometer and gyroscope

Two of the sensors hosted on the Crazyflie are an accelerometer and a gyroscope, which measure acceleration experienced by the quadrocopter and angular velocities respectively. When standing still on a flat surface, the accelerometer will read $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ where the unit is force divided by the gravitational acceleration. Similarly, if rotated +90 degrees around the x-axis the readings would be $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$. In practice, the accelerometer measurement vector might not always be of unit length due to noise and biases, hence the measurements are normalized before further use. The gyroscope measures angular velocities around each axis and the unit for the sensor output is deg/s.

An estimate of the orientation from the accelerometer can be found by using

$$\begin{bmatrix} f^b_x \\ f^b_y \\ f^b_z \end{bmatrix} = R^{bi}_{XYZ} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{6}$$

where f^b_x, f^b_y and f^b_z are the normalized accelerometer readings, R^{bi}_{XYZ} according to (4) and the right hand side vector is the gravitational force in the inertial frame. This assumes that the quadrocopter is stationary and that the accelerometer only measures the gravitational force, e.g. the quadrocopter is either hovering at a constant position or sitting still on a surface. The assumption is based on that the accelerometer measurements mostly are dominated by the gravitational force. In order to find estimates for the Euler angles (6) can be rewritten as

$$\hat{\varphi}_{acc} = \arctan\frac{f^b_y}{f^b_z} \tag{7}$$

$$\hat{\theta}_{acc} = \arctan\frac{-f^b_x}{\sqrt{(f^b_y)^2 + (f^b_z)^2}} \tag{8}$$

where arctan is implemented using `atan2()`. Also, note that the yaw ψ cannot be estimated with this approach and will not be controlled.

To generate estimates using the gyroscope one can simply integrate the readings, for example for roll,

$$\hat{\varphi}_{gyro} = \int \omega_\varphi dt \tag{9}$$

where ω_φ is the reading for angular velocity around the x-axis.

C. Complementary filter

In general, the orientation estimation from an accelerometer is noisy but precise over a long time, while the estimation from a gyroscope is accurate over a short time window but tends to drift with time [2]. This can be exploited in the frequency domain, i.e. using the gyroscope for fast dynamics while trusting the accelerometer over time, and this is known as a complementary filtering method. Concretely, the accelerometer readings are fed through a low-pass filter $G(s)$ while the gyroscope readings are fed through a high-pass filter $1-G(s)$. These use the same cut-off frequency such that the sum of the low-pass and high-pass filters is one in the frequency domain, which also is a reason for the name complementary filter. In other words, the complementary filter calculates orientation estimates $\hat{\Theta}(s)$ from accelerometer estimates $\Theta_{acc}(s)$ and gyroscope estimates $\Theta_{gyro}(s)$ as

$$\hat{\Theta}(s) = G(s)\Theta_{acc}(s) + (1 - G(s))\Theta_{gyro}(s) \tag{10}$$

In [2] the equation for the complementary filter is expressed in a discrete-time setting by applying Euler backward discretization,

$$\hat{\theta}_k = (1 - \gamma)\theta_{a,k} + \gamma(\theta_{k-1} + Ty_{\omega,k}) \tag{11}$$

where $\hat{\theta}_k$ is the estimated angle at time k, $\theta_{a,k}$ is the angle estimate from the accelerometer at time k, T is the sampling time and $y_{\omega,k}$ the readings from the gyroscope at time k. γ is a tuning parameter between 0 and 1, used to select which sensor to trust. For a value close to 0, the complementary filter trusts only the accelerometer, while a value close to 1 emphasizes the gyroscope. For this project $\gamma = 0.98$ was found to give adequate behavior, i.e. suppressing the noise from the accelerometer while avoiding the drifting from gyroscope estimates.

III. Plant modeling

To be able to simulate the quadrocopter per the model-based approach, the equations defining the behavior of the plant were modeled in the software Simscape, a tool created by MATLAB to facilitate equation-based modeling.

$$\mathbf{x} = \begin{cases} x_1 : \text{positions} & = \begin{bmatrix} x & y & z \end{bmatrix}^T \\ x_2 : \text{velocities} & = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \\ x_3 : \text{angles} & = \begin{bmatrix} \varphi & \theta & \psi \end{bmatrix}^T \\ x_4 : \text{angular velocities} & = \begin{bmatrix} \dot{\varphi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \end{cases} \tag{12}$$

Provided the notation set in (12) the differential equations from Corke [3] can be adapted to describe the state of the quadrocopter,

$$\dot{x}_1 = x_2 \tag{13}$$

$$\dot{x}_2 = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T + R^{ib}_{XYZ} \begin{bmatrix} 0 & 0 & \sum_i \frac{T_i}{m} \end{bmatrix}^T \tag{14}$$

$$\dot{x}_3 = x_4 \tag{15}$$

$$\dot{x}_4 = J^{-1}\big(- x_4 \times Jx_4 + \Gamma\big) \tag{16}$$

where

$$\Gamma = \begin{bmatrix} \frac{d}{\sqrt{2}}\left((T_4 + T_3) - (T_1 + T_2)\right) \\ \frac{d}{\sqrt{2}}\left((T_2 + T_3) - (T_1 + T_4)\right) \\ \frac{k}{b}\left((T_1 + T_3) - (T_2 + T_4)\right) \end{bmatrix}$$

describes how different motors affect roll, pitch and yaw rate and T_i is the thrust at the motor i. Furthermore, m is the mass of the Crazyflie, k_d is a drag constant depending on air density and rotor blade parameters, b is the lift constant depending on the same factors as k_d, g is the acceleration of gravity, J is the moment of inertia matrix containing J_x, J_y and J_y on the diagonal, and R^{ib}_{XYZ} is the rotation matrix from the body to the inertial frame which can be found as

$$R^{ib}_{XYZ} = (R^{bi}(\varphi))^T (R^{bi}(\theta))^T (R^{bi}(\psi))^T \tag{17}$$

IV. Linearization of plant

To design a Linear Quadratic controller for the model of the Crazyflie, a linear model was needed. The operating state, x^0, around which the linearization was done, was selected to be the equilibrium for all states, in other words,

$$x_1 = x_2 = x_3 = x_4 = \mathbf{0} \tag{18}$$

To maintain states at zero, a total thrust, $T^0 = mg$, is needed to counteract the gravitational force acting upon the Crazyflie. In order to linearize the plant, the two nonlinear state equations (14) and (16) are differentiated with respect to the states x_1 to x_4 and evaluated at the selected operating point. Note that the right hand side of (14) and (16) is hereafter referred to as f_2 to f_4 respectively.

$$\nabla_{x_1} f_2 = \nabla_{x_2} f_2 = \nabla_{x_4} f_2 = \mathbf{0} \tag{19}$$

$$\begin{aligned} \nabla_{x_3} f_2\Big|_{x^0, T^0} &= \begin{bmatrix} 0 & cos(\theta) & 0 \\ -cos(\varphi)cos(\theta) & sin(\varphi)sin(\theta) & 0 \\ -sin(\varphi)cos(\theta) & -cos(\varphi)sin(\theta) & 0 \end{bmatrix} \frac{T^0}{m} \\ &= \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \frac{T^0}{m} \end{aligned} \tag{20}$$

Furthermore, f_2 is differentiated with respect to the input $T = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 \end{bmatrix}^T$, yielding

$$\nabla_T f_2\Big|_{x^0, T^0} = \frac{1}{m} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{21}$$

Now, differentiating f_4 w.r.t the states x_1 to x_3 gives the following result

$$\nabla_{x_1} f_4|_{x^0,T^0} = \nabla_{x_2} f_4|_{x^0,T^0} = \nabla_{x_3} f_4|_{x^0,T^0} = \mathbf{0} \tag{22}$$

Since Γ does not depend on x_4, differentiating w.r.t. x_4 simply becomes:

$$\nabla_{x_4} f_4 = \begin{bmatrix} 0 & \frac{\dot{\psi}(J_y - J_z)}{J_x} & \frac{\dot{\theta}(J_y - J_z)}{J_x} \\ \frac{\dot{\psi}(J_z - J_x)}{J_y} & 0 & \frac{\dot{\varphi}(J_z - J_x)}{J_y} \\ \frac{\dot{\theta}(J_x - J_y)}{J_z} & \frac{\dot{\varphi}(J_x - J_y)}{J_z} & 0 \end{bmatrix} \tag{23}$$

Evaluating (23) at the operating point yields $\nabla_{x_4} f_4|_{x^0,T^0} = \mathbf{0}$, i.e. the linearized model disregards the influence of the angular velocities when solving for the angular acceleration. Now, f_4 is differentiated w.r.t. the input T,

$$\nabla_T f_4|_{x^0,T^0} = J^{-1} \begin{bmatrix} -\alpha & -\alpha & \alpha & \alpha \\ -\alpha & \alpha & \alpha & -\alpha \\ \beta & -\beta & \beta & -\beta \end{bmatrix} \tag{24}$$

where $\alpha = \frac{d}{\sqrt{2}}$ and $\beta = \frac{k_d}{b}$

For control purposes only a subset of the states (12) need to be considered. For controlling roll φ, pitch θ and yaw rate

$\dot{\psi}$, the linearized model can be reduced to include only five states since no other states affect roll and pitch,

$$\mathbf{x}_l = \begin{bmatrix}\varphi & \theta & \dot{\varphi} & \dot{\theta} & \dot{\psi}\end{bmatrix}^T \tag{25}$$

where subscript l indicates the states in the linearized model. The model used during controller design hence is

$$\dot{\mathbf{x}}_l = A\mathbf{x}_l + BT \tag{26}$$

where

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{27}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{\alpha}{J_x} & -\frac{\alpha}{J_x} & \frac{\alpha}{J_x} & \frac{\alpha}{J_x} \\ -\frac{\alpha}{J_y} & \frac{\alpha}{J_y} & \frac{\alpha}{J_y} & -\frac{\alpha}{J_y} \\ \frac{\beta}{J_z} & -\frac{\beta}{J_z} & \frac{\beta}{J_z} & -\frac{\beta}{J_z} \end{bmatrix} \tag{28}$$

V. Design of Linear Quadratic Controller

For the Crazyflie to follow any desired movement, it needs to have a controller. It was decided to use a Linear Quadratic (LQ) controller. Such a controller is suitable to use for Multiple-Input-Multiple-Output (MIMO) systems [4], which a quadrocopter is.

A. Design

A LQ controller makes use of the control law

$$\mathbf{u} = -K\tilde{\mathbf{x}},\ \tilde{\mathbf{x}} = (\hat{\mathbf{x}}_l - \mathbf{x}_r) \tag{29}$$

where K is the feedback gain matrix, $\hat{\mathbf{x}}_l$ the state estimates and $\mathbf{x}_r = \begin{bmatrix}\varphi_r & \theta_r & 0 & 0 & \dot{\psi}_r\end{bmatrix}$ a vector with reference states. Estimates for roll and pitch are generated using the complementary filter, while the angle rates are estimated using gyroscope measurements without any further modifications. The LQ controller strives to navigate states to their origin and in order to specify a desired movement, the controller is fed the error between estimated and reference states.

In practice, the feedback gain matrix K was constructed using MATLAB's `lqrd()`. The function uses a state space system in continuous time, e.g. matrices A and B in (27) and (28) respectively, and cost matrices Q and R and outputs the feedback gain K for a discrete time controller. The function minimizes the cost

$$J = \int \tilde{\mathbf{x}}(t)Q\tilde{\mathbf{x}}^T(t) + \mathbf{u}(t)R\mathbf{u}^T(t)dt \tag{30}$$

where Q and R represent the cost matrices for states and inputs respectively.

B. Tuning

Q and R can be tuned to achieve different behaviors of the controller. For convenience and clarity, the cost matrices were chosen as diagonal matrices such that each nonzero element directly affects the cost associated with a certain state or input. As it is the relationship between Q and R that has any impact on the feedback gain matrix, R was chosen and kept as the identity matrix times 100 whilst Q was what was changed.

The reasoning behind the tuning is attributed to several aspects.

- The Crazyflie inputs are mostly comprised of an electrical system and are hence relatively fast compared to the states which are a mechanical system. Therefore input cost should be relatively large compared to state cost.
- The desired behavior is that the Crazyflie steadily follows reference states whilst remaining responsive to any changes in reference. As the roll and pitch rates are not given any reference and are closely linked to their respective angle, adding cost to the pitch and roll rates is not necessary. However, by adding weights to the roll and pitch rate one could achieve a smoother but less responsive control behavior.
- As the yaw rate $\dot{\psi}$ is considered less critical for stability, deviations from the reference are penalized less compared to the more stability critical states, pitch, and roll.

Through simulation, and trial and error, a suitable mixture of stability and responsiveness was achieved with the cost matrices.

$$Q = \text{diag}(\begin{bmatrix}1 & 1 & 0 & 0 & 0.01\end{bmatrix}),\ R = 100I_4 \tag{31}$$

VI. Controller implementation

Following the simulation and tuning of the controller in Simulink, the controller and the complementary filter were implemented in the real-time operating system FreeRTOS using C code. The reason for this is that the Crazyflie firmware is based on FreeRTOS, hence these systems must be implemented in this framework to be able to control the physical quadrocopter. However, as mentioned before, no physical Crazyflie was available in this project and thus the purpose of the C implementation is mainly for future usage and the correctness of the C implementation was verified through simulation.

The controller and complementary filter were implemented as two separate tasks to be handled by the scheduler in FreeRTOS. Further, semaphores were used to avoid race conditions where different tasks access shared memory and to ensure data integrity. For example, the sensors write to a set memory location which is also accessed by the complementary filter for calculating its estimate. Similarly, the complementary filter writes the estimates to a certain memory location which is accessed by the controller for calculating motor output.

The task for the complementary filter consists of reading the sensor data, normalizing the accelerometer readings, applying (11), and storing the estimates for roll and pitch. Here, the reading of sensor data and the storing of estimates involve accessing shared memory. To avoid race conditions and ensure data integrity, each shared memory resource is associated with a semaphore that must be taken before accessing the corresponding memory location and given back as soon as the task is done reading or writing. For the filter task to run periodically, after it has written its estimates, the task is delayed until 5 ms from the last time the task was run. This effectively runs the task at 200 Hz.

The control task, in turn, consists of reading the gyroscope data, the filter estimates, and the control reference. Following this, (29) is applied to calculate required motor thrusts. Further, this is converted to appropriate motor signals and written to the motors. Finally, the control task is delayed 10 ms from the last time the task was run, resulting in a 100 Hz task period. Similar to the complementary filter task, the control task is enforced to take relevant semaphores before reading/modifying shared memory and giving them back once the specific action is finished. Thus, this is used for reading the gyroscope data, the filter estimates, and the control reference, and for writing the motor output.

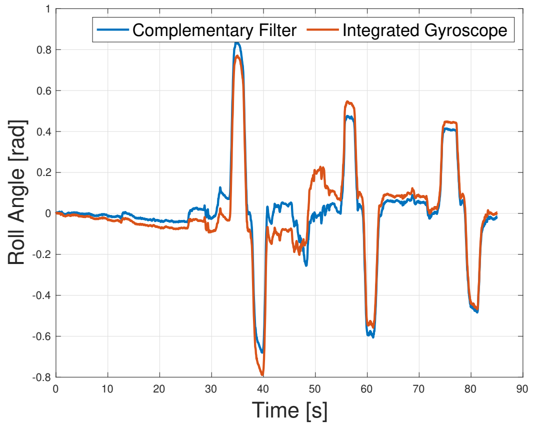
Other tasks used for simulation included a reference generator, a plant model of the quadrocopter, and a model of the sensors. The code for these tasks was provided by teachers for the course Model-based development of cyber-physical Systems at the Chalmers University of Technology.

VII. Evaluation of the control design

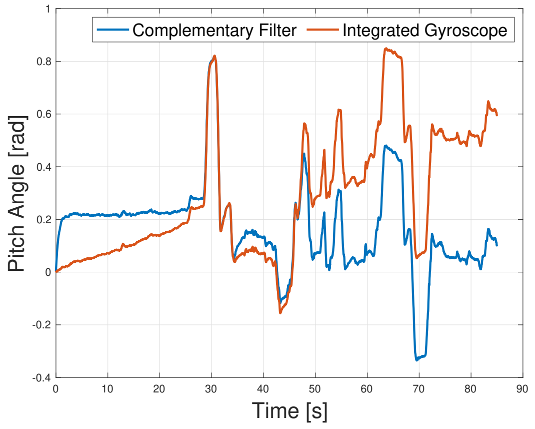
As a part of the model-based approach, it is important to determine whether or not the methods applied in this paper yield satisfactory control behavior. For this, the complementary filter was evaluated against a more naive and simple discrete integration method, while the controller was evaluated based on its ability to track external reference values in the roll θ, pitch φ and yaw-rate $\dot{\psi}$ during different scenarios.

A. Complementary Filter

In Figure 2 the roll and pitch angles are estimated using two different methods, namely, a complementary filter and discrete integration of the gyroscope measurements. The sensor data used was collected using the physical system. In the estimation of the pitch angle, shown in Figure 2b, the gyroscope measurements exhibit a clear bias. This causes the estimates to drift for the discrete integration method as the bias accumulates over time. However, the complementary filter can suppress the effects of the bias giving a more correct estimate. Further, when the sensor is not exhibiting any substantial bias the two methods perform similarly as seen in Figure 2a.



(a) Roll angle



(b) Pitch angle

Fig. 2: Complementary filter estimation (blue) compared to integrated gyroscope measurements (red) for roll and pitch angles, using $\gamma = 0.98$.

B. LQ reference tracking

To evaluate the performance of the designed LQ controller in Simulink different scenarios were simulated and one of these is shown in Figure 3. Here, the reference roll and pitch angles are intermittently raised to different values while keeping the yaw rate reference at zero. As the figure clearly shows the roll and pitch follow their references closely with a slight overshoot.

Another scenario that was simulated had the reference yaw-rate displaying a pulse behavior. As seen in Figure 4 the controller drives the yaw-rate to its reference without overshoot. The corresponding yaw angle estimation, produced with discrete integration, exhibits the cyclic linear behavior that is expected.

C. C-code implementation of controller

For verification purposes, the performance of the C implementation of the controller was compared to its Simulink counterpart. This was done by simulating the same scenario as in Figure 3 and the result of the FreeRTOS simulation is

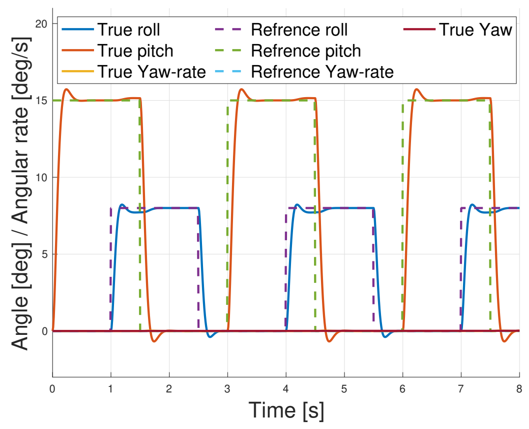


Fig. 3: Reference tracking for pitch and roll angle. Simulated using Simulink.

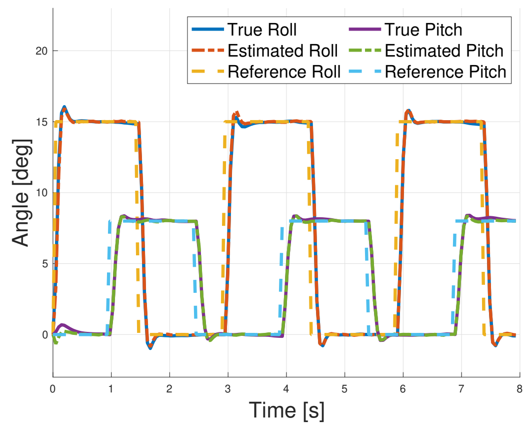


Fig. 5: Reference tracking for pitch and roll angle. Simulated using FreeRTOS.

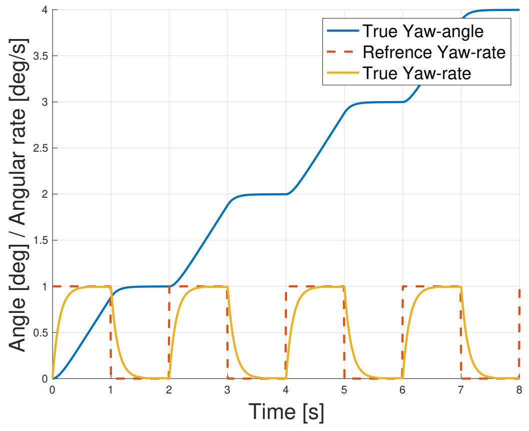


Fig. 4: Reference tracking for yaw rate. Simulated using Simulink.

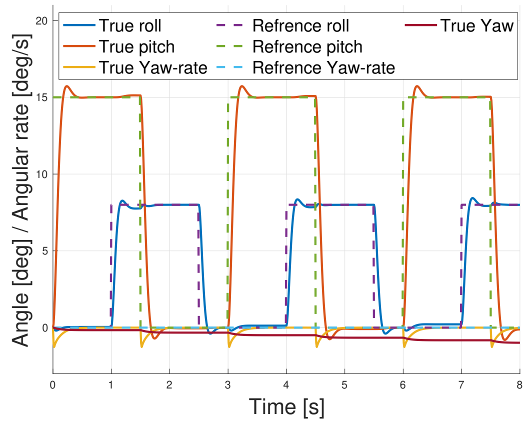


Fig. 6: Reference tracking with degraded motor performance of M1 and M3. Simulated using Simulink.

shown in Figure 5. The behavior is very similar to the one seen in Figure 3, indicating the correctness of the implementation. Furthermore, the controller manages to track the given reference states and the complementary filter gives estimates close to the true states although accelerometer readings are slightly noisy and the gyroscope measurements have a small bias.

D. Robustness

Robustness is an important measure of controller performance since robustness allows the controller to keep the plant in a good state in unforeseen scenarios. Such could be degrading of actuator or sensor performance, deviations between model and real-world settings, and other disturbances. To evaluate some robustness properties of the LQ controller, the scenario shown in Figure 3 was simulated with the addition of unevenly degraded motor performance and is shown in Figure 6. This was implemented through saturating the motor signals to motors M1 and M3 at 50% of their maximum capacity.

As can be observed, with degraded motor performance, the quadrocopter exhibits a drift in its yaw angle since the yaw rate cannot be kept at zero during rapid changes in roll or pitch. This is an effect of the selected cost matrices, where deviations in pitch and roll are penalized more than deviations in yaw rate. However, the controller is still able to follow its reference values in pitch and roll similar to that in Figure 3, indicating a certain level of robustness in the controller.

Further, the robustness of the controller was tested by increasing the noise of the accelerometer measurements when using the scenario in Figure 3. The increase in noise was done by a factor 10 compared to Figure 5. In a physical system, this could be due to external electromagnetic disturbances or other non-modeled disturbances. The result of the simulation can be seen in Figure 7. As a result, the orientation estimates are also noisier when compared to Figure 5, leading to the controller believing it is deviating from the reference state, while it might be close to in reality. However, the controller manages to keep the true state close to the reference state in general.

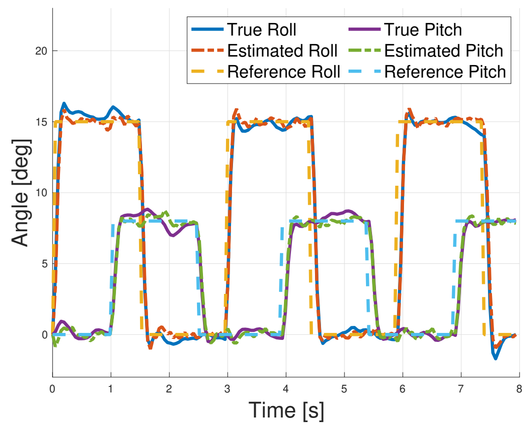


Fig. 7: Simulation with increased noise in accelerometer measurements. Simulated using FreeRTOS.

VIII. Discussion

The following section will discuss the relevance of the results presented in Section VII as well as potential areas for further development and the suitability of the model-based approach.

The complementary filter exhibited good behavior and was able to suppress the effect of the sensor bias in the gyroscope. Since it was evaluated on data recorded in a real-world setting, the filter could be expected to perform equivalently when implemented in the Crazyflie. The LQ controller displayed the ability to track its reference states in a fast but controlled manner. When the controller was subjected to disturbances, such as degraded motor performance and increased sensor noise, it displayed a robust behavior as it was able to track its reference states in roll and pitch even in the presence of the aforementioned disturbances.

While the controller seemingly has outstanding performance it should be considered that the results are completely based on simulation. By creating mathematical models of real-world systems the behavior of the controller and the quadrocopter can be accurately simulated, which allows for more efficient development as well as limiting the requirement of use of hardware. Specifically, Simscape proved to be an easy-to-use tool to perform acausal modeling of real-world systems in an effective manner. Simulink allowed doing fast simulations for trying different approaches without having to worry about negative consequences for the system. However, even though models used can be considered comprehensive, they can never capture every aspect of the real world. Furthermore, several assumptions are made, such as ignoring disturbances, to ease the modeling of the above-mentioned systems. These are assumptions are needed since trying to model in all factors, such as creating advanced airflow simulations and complex mathematical models for each component can make the modeling process slow and costly. It can even destroy the purpose of using simulation if available processing power results in simulations that are far slower than physical testing or if the time spent modeling outweigh the hardware costs.

In addition, it is important to note that the behavior exhibited by the controller is largely influenced by its tuning. This project's purpose, to achieve hovering and reference tracking, does not put many desires in controller performance other than to achieve stable flight. The tuning used certainly achieves such during simulation. However, should the need for a specific behavior arise, the cost matrices can easily be adjusted for the controller to yield performance appropriate to the need. For example, should less overshoot at the expense of a slower rise time be desired, the cost associated with roll and pitch rate could be increased in relation to cost associated with motor signals or input. If more responsive behavior is desired at the expense of a larger overshoot, the cost associated with input could be lowered in relation to cost associated with deviations from the reference states. This back and forth tuning would be especially prevalent and useful in a real-world implementation and testing where the true performance of the controller is shown. As this ultimately did not become part of the project, the true impact, and importance of the tuning process are not reflected in this report.

Although the project achieved its purpose, to produce a functional controller that allows reference tracking for roll and pitch angles, as well as yaw rate, multiple points allow for further development.

Firstly, the complementary filter uses orientation estimations from the accelerometer that are based on the assumption that the acceleration of the Crazyflie is zero. This is normally not the case for a quadrocopter maneuvering in the physical world. Further, the developed models use other approximations such as no air resistance, no delays in the electrical systems, and simplified equations for motor thrust. These simplifications create mismatches between model and real-world plant, making it harder to apply the model-based controller on the physical Crazyflie and getting stable performance. A more accurate model will most certainly increase the controller performance when applied in a real-world setting. However, with no physical Crazyflie available, this could not be evaluated during this project.

Secondly, the orientation estimates can potentially be improved in multiple ways. One way is to utilize the known behavior of a quadrocopter and using this model when doing state estimations, e.g. using a Kalman filter instead of a complementary filter if there is enough computational power. This way, other sensor measurements can be used, such as a magnetometer, which is already mounted on the Crazyflie. The Kalman filter can also approximate the position of the Crazyflie and enable more complex control schemes.

Lastly, the performance of the controller should be thoroughly evaluated in a real-world setting. As mentioned before, approximations introduce discrepancies between the model and the physical world. Since the controller is based on these approximated models, its performance will naturally not be optimal when applied on a different model. This enhancement would also be in line with the model-based methodology.

The model-based approach has proven itself very useful for a project of this type. It has given a meaningful structure to the workflow and allowed the development of a controller without any access to the physical plant. Further, the modeling of quadrocopter has been explored in the literature, [3] for example, easing the modeling work and speeding up the development. The model-based framework provides tools for iterative improvements, where implementing a more accurate model, quickly would affect the controller parameters and allow for re-tuning. Further, the approach has reduced development cost since different controller parameters easily can be simulated in a safe environment, without the risk of harming expensive hardware.

IX. Conclusions

Following the model-based development approach yielded a satisfactory controller, exhibiting responsive yet smooth behavior. The designed LQ controller displayed a robust performance when exposed to different levels of sensor noise and unevenly degraded motor performance. However, it should be noted that throughout this project the controller and its performance have been purely simulated using models of the physical world. Even though these models are considered accurate, they can never capture the entirety of the real world, thus creating a deviation in the simulated versus the actual behavior of the controller. While it serves as a good first guess, it would be necessary to fine-tune the controller's cost matrices to achieve excellent performance in a real-world setting.

References

[1] Bitcraze, "Crazyflie 2.1," 2020. [Online]. Available: https://www.bitcraze.io/products/crazyflie-2-1/ Accessed on: 2020-05-25.

[2] M. Kok, J. D. Hol, and T. B. Schön, "Using inertial sensors for position and orientation estimation," *Foundations and Trends® in Signal Processing*, vol. 11, no. 1-2, p. 1–153, 2017. [Online]. Available: http://dx.doi.org/10.1561/2000000094

[3] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*. Cham, Switzerland: Springer International Publishing, 2017, pp. 114–119. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-20144-8 Accessed on: 2020-05-25.

[4] T. Glad and L. Ljung, *Control Theory - Multivariate and Nonlinear Methods*. Taylor & Francis Group, 2000, vol. 1.