Final report
SSY191
Group 22

Abdulrahman Hameshli
Department of electrical engineering
Chalmers technical university
Gothenburg, Sweden
hameshli@chalmers.se

Philip Fredriksson
Department of electrical engineering
Chalmers technical university
Gothenburg, Sweden
Philiper@chalmers.se

Mohammad Kanjo
Department of electrical engineering
Chalmers technical university
Gothenburg, Sweden
kanjom@chalmers.se

Sanjeet Malawade
Department of electrical engineering
Chalmers technical university
Gothenburg, Sweden
sanjeet@chalmers.se

***Abstract*—This report aims to explain how a crazyflie 2.0 can be modeled and controlled by using matlab, C, simscape, simulink and so on. Orientation estimation is a crucial part when controlling the crazyflie, therefore a complementary filter was used to estimate the roll (φ) and pitch (θ) angles. Furthermore a model for the drone was used and implement in simscape. This was done mainly because of the controller and simulation purposes. Since the model is non-linear and the LQR is developed for linear models, a linearization based on taylor expansion was made and thereafter discretized. The model was later used to calculate the LQR gain for the controller and tuned accordingly.**

***Index Terms*—Complementary filter, model, controller, simscape, linearization, LQR**

I. Introduction

Setting the context for quadrocopter control encompasses various critical aspects, including orientation estimation, plant modeling, plant linearization, design of Linear Quadratic Regulator (LQR), and evaluation of control design.

In the realm of orientation estimation, understanding the quadrocopter's spatial orientation in real-time is fundamental for precise control. The complementary filter provides a way to combine data from sensors such as accelerometers, and gyroscopes, to accurately determine the quadrocopter's position and orientation relative to its surroundings.

Plant modeling plays a pivotal role in developing effective control strategies. Firstly, quadrocopters, with their complex dynamics and high degrees of freedom, require a deep understanding of their behavior in various conditions. This understanding forms the foundation of model-based design, where mathematical models of the quadrocopter's dynamics are utilized to predict its behavior and design control algorithms accordingly.

Plant linearization further refines these models by simplifying complex dynamics into linear approximations around specific operating points. This enables the application of linear control techniques, such as LQR, which are invaluable for designing stable and optimal control laws.

The design of LQR controllers leverages the linearized plant models to optimize control performance by minimizing a cost function that penalizes deviations from desired trajectories while considering control effort.

This holistic approach is indispensable for developing precise, reliable, and efficient control strategies that enable quadrocopters to excel in real-world applications while ensuring safety and performance.

II. Orientation Estimation

The quadcopter is free to rotate in three dimensions, i.e., along the roll (x), pitch (y), and yaw (z) axes, and these rotations are represented by ϕ, θ, and ψ respectively. These three angles are used to represent the orientation of the quadcopter.

To estimate the pose, i.e., the position and orientation of the quadcopter, we use the inertial measurement unit (IMU), which consists of a 3-axis accelerometer and a 3-axis gyroscope.

The accelerometer measures the external specific force

acting on the sensor. It simply measures the acceleration due to gravity when it as at rest or moving with constant velocity. When we have the acceleration due to gravity as a reference, and the quadcopter rotates, we can estimate the orientation using this gravity vector as a reference.

The 3-axis accelerometer will output the forces acting on the quadcopter, which is provided by Equation 1 given below:

$$^{B}f = m\,^{B}R_{w}\left(\begin{bmatrix}^{w}a_{x}\\ ^{w}a_{y}\\ ^{w}a_{z}\end{bmatrix} - \begin{bmatrix}0\\0\\g\end{bmatrix}\right) \tag{1}$$

where ^{B}f represents the forces acting on the body (drone), $^{B}R_{w}$ is the rotation matrix from body to world (i.e., body with respect to world), m is the mass of the quadcopter, and $^{w}a_{x}$, $^{w}a_{y}$, and $^{w}a_{z}$ represent the acceleration along each axis with respect to the world coordinate frame.

Assuming the quadcopter is flying with constant velocity or at rest and the forces are normalized to gravity, the acceleration in the world coordinate frame would be equal to 0, which would modify the equation above to:

$$^{B}f = m\,^{B}R_{w}\begin{bmatrix}0\\0\\1\end{bmatrix} \tag{2}$$

The rotation matrix in Equation 2is derived by multiplying the rotation matrices in the extrinsic rotation order of roll-pitch-yaw (XYZ). Considering the above equation, only the third column is relevant for calculating the forces from the accelerometer. The final equations for the forces using the above equation are:

$$\begin{bmatrix}^{B}f_{x}\\ ^{B}f_{y}\\ ^{B}f_{z}\end{bmatrix} = m\underbrace{\begin{bmatrix}1 & 0 & -\sin(\theta)\\ 0 & \cos(\varphi) & \sin(\varphi)\cos(\theta)\\ 0 & \sin(\varphi) & \cos(\varphi)\cos(\theta)\end{bmatrix}}_{^{W}R_{B}}\begin{bmatrix}0\\0\\1\end{bmatrix} \tag{3}$$

Given the above Equation 3 consisting of forces along each axis, the roll and pitch orientation angles can be found by considering two forces and passing them through atan2 functions to avoid the scaling factors.

$$\frac{^{B}f_{y}}{^{B}f_{z}} = \frac{m\cos(\theta)\sin(\varphi)}{m\cos(\theta)\cos(\varphi)} = \tan(\varphi) \tag{4}$$

Therefore,

$$\varphi = \text{atan2}(^{B}f_{y}, ^{B}f_{z}) \tag{5}$$

Similarly, to avoid scaling factors, we can use all three forces to estimate the roll angle.

$$^{B}f_{y}{}^{2} + ^{B}f_{z}{}^{2} = m^{2}\cos^{2}(\theta)(\sin^{2}(\varphi) + \cos^{2}(\varphi)) \tag{6}$$

$$= m^{2}\cos^{2}(\theta) \tag{7}$$

$$\cos(\theta) = \frac{1}{m}\sqrt{^{B}f_{y}{}^{2} + ^{B}f_{z}{}^{2}} \tag{8}$$

Using $^{B}f_{x}$,

$$\tan(\theta) = \frac{\sin(\theta)}{\cos(\theta)} = \frac{\frac{-\,^{B}f_{x}}{m}}{\frac{\sqrt{^{B}f_{y}{}^{2}+^{B}f_{z}{}^{2}}}{m}} \tag{9}$$

Therefore,

$$\theta = \text{atan2}(-\,^{B}f_{x}, \sqrt{^{B}f_{y}{}^{2} + ^{B}f_{z}{}^{2}}) \tag{10}$$

The gyroscope measures the angular velocity, i.e., the rate of change of the sensor's orientation. We can use the measured angular velocity to calculate how the angle has changed, by updating the current estimates based on the previous estimates in a given time frame or time step. By doing this, we seek to integrate the angular velocity to get the estimated orientation angle. This approach is called dead reckoning, and it is good for short periods of time.

Therefore, using this IMU sensor, we can estimate the position and orientation of the quadcopter. However, these sensors tend to be noisy and have bias, which could lead to integration drift from the true position and orientation values.

The accelerometer readings are noisy as they are affected by the body's acceleration but accurate in estimating the orientation of the body over long periods or when the body is not accelerating. On the other hand, the gyroscope provides accurate orientation readings but is sensitive to drift over time.

To overcome this, we use a complementary filter which utilizes the advantages of both these sensors to get a good estimate of the orientation by relying on the gyroscope in the short term and the accelerometer on a long term. Gyroscope readings are integrated and passed through a first order high-pass filter with transfer function $1 - G(s)$, meanwhile accelerometer readings are passed through a complementary first order low-pass filter with transfer function $G(s)$, which are combined to get an accurate estimation of the orientation.

The complementary filter is given by:

$$\theta = G(s)\theta_{a}(s) + (1 - G(s))\theta_{g}(s) \tag{11}$$

where:

$$\theta_{g}(s) = \frac{1}{sY_{g}(s)}$$

$\theta_{a}(t)$ is the estimated angle from the accelerometer, and $y_{g}(t)$ is the angular velocity from the gyroscope.

By using Euler backward, Equation 11 can be discretized and will yield:

$$\theta_k = (1-\gamma)\theta_{a,k} + \gamma(\theta_{k-1} + h \cdot y_{g,k}) \tag{12}$$

where:

$$\gamma = \frac{\alpha}{h+\alpha}$$

After implementing the complementary filter in Simulink and tuning it, we found that a value of $\alpha = 0.05$ worked best, providing us with good and stable estimates of the orientation of the quadcopter. The final Simulink model for the complementary filter as well as the results are illustrated in Figure 1.

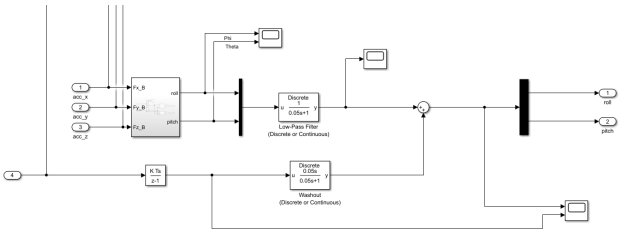

Fig. 1. Complementary filter implementation in Simulink

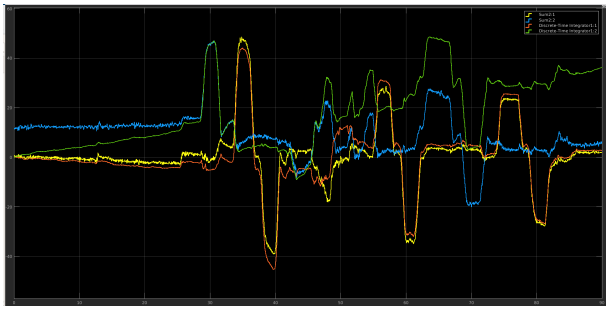

Fig. 2. Output of the filtered and Integrated Roll and Pitch Data

Using the above two figures, one can easily see the advantages of using a complementary filter rather than simply integrating the gyroscope readings to estimate the roll and pitch angles with the given data. When integrating gyroscope readings, although the noise level is reduced, it leads to integral drift, which diverges from the original data. On the other hand, when using a complementary filter, the noise levels are reduced, and the integral drift is prevented, providing accurate estimates of the angles.

III. Plant Modeling

When the motors starts to spin, they generate a force named thrust. The thrust is always directed in the negative z-direction and can be calculated by:

$$T_i = b\omega_i^2, \qquad i = 1,2,3,4 \tag{13}$$

Where i represent each motor and $b > 0$ is the lift constant.

By applying Newton's second law and assuming that the drone is at rest or moving with constant velocity, the following result yields:

$$m\dot{\boldsymbol{v}} = -\begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + {}^{w}R_B \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \tag{14}$$

Where $T = \sum_{i=1}^{4} T_i$.

Since thrust is a force, a torque will be generated on the quadrotor. This torque can be categorized in 3 different torques namely: rolling torque, pitching torque and yawing torque.

Rolling torque

Torque is defined as force cross distance. Assume that the distance is d. Then the rolling torque is defined as:

$$\tau_x = d \cdot cos\left(\tfrac{\pi}{4}\right) \cdot \left(T_4 + T_3 - T_2 - T_1\right) \tag{15}$$

Pitching torque

In the same way as in rolling torque, the pitching torque is defined as:

$$\tau_y = d \cdot sin\left(\tfrac{\pi}{4}\right) \cdot \left(T_3 + T_2 - T_4 - T_1\right) \tag{16}$$

Yawing torque

Because of the conservation of angular momentum, each motor will experience aerodynamic drag that wants to rotate the drone around its z-axis. This torque can be described as:

$$Q_i = k\omega_i^2, \qquad i = 1,2,3,4 \tag{17}$$

Therefore the torque acting on the z-axis, i.e. yawing torque can be described as:

$$\tau_z = Q_4 + Q_2 - Q_3 - Q_1 \tag{18}$$

The rotational acceleration of the airframe is given by Euler's equation of motion

In order to calculate the drone's acceleration, one can utilize Euler's equation of motion which says:

$$\boldsymbol{J}\dot{\omega} = -\omega \times \boldsymbol{J}\omega + \boldsymbol{\Gamma} \tag{19}$$

By using the rotation matrix from base to world. The rotational velocities from the world coordinate can be expressed as:

$$\omega_W = {}^W R_B \cdot \omega_B \tag{20}$$

Where ${}^W R_B$ is derived in equation 3.

From equation 19 and 20 the states used for the LQR controller can be derived as following:

The states of interests and inputs are:

$$X = \begin{bmatrix} \varphi \\ \theta \\ \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad U = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} \tag{21}$$

The states $\dot{\varphi}$, $\dot{\theta}$ and $\dot{\psi}$ can be derived from equation 19 as follows:

$$\boldsymbol{J}\dot{\omega} = -\omega \times \boldsymbol{J}\omega + \boldsymbol{\Gamma} \tag{22}$$

$$\tag{23}$$

$$\boldsymbol{J} \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = - \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \boldsymbol{J} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{24}$$

$$\tag{25}$$

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \boldsymbol{J}^{-1} \left(\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \boldsymbol{J} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \right) \tag{26}$$

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} (2 \cdot J_y \cdot \omega_y \cdot \omega_z - 2 \cdot J_z \cdot \omega_y \cdot \omega_z - 2^{1/2} \cdot d \cdot t_1 - 2^{1/2} \cdot d \cdot t_2 + 2^{1/2} \cdot d \cdot t_3 + 2^{1/2} \cdot d \cdot t_4)/(2 \cdot J_x) \\ -(2 \cdot J_x \cdot \omega_x \cdot \omega_z - 2 \cdot J_z \cdot \omega_x \cdot \omega_z - 2^{1/2} \cdot d \cdot t1 + 2^{1/2} \cdot d \cdot t2 + 2^{1/2} \cdot d \cdot t3 - 2^{1/2} \cdot d \cdot t4)/(2 \cdot J_y) \\ (drag \cdot |t1| - drag \cdot |t2| + drag \cdot |t3| - drag \cdot |t4| + J_x \cdot lift \cdot \omega_x \cdot \omega_y - J_y \cdot lift \cdot \omega_x \cdot \omega_y)/(J_z \cdot lift) \end{bmatrix}$$

To model the drone, a simscape model was created. This is because simscape offers a convenient way to write equations such that no consideration for derivative states is needed. This means that one does not have to derive all equations for the derivative and integral states, for example, if one has an equation for the acceleration, there is no need to calculate the velocity. By using simscape the number of equations and complexity for equations drops significantly. Note that there is some assumption for example we assume that the air resistance is equal for all directions.

IV. Plant Linearization

With reference to *Control Theory - Multivariable and Nonlinear Methods* by Torkel Glad and Lennart Ljung [1], we know that this system is not linear because of the squared functions, for example, in equation 15. To be able to control this system with an LQ controller, the system has to be linearized. By using the following formula one can linearize the system to get A and B matrix:

The nonlinear system dynamics are described by:

$$\dot{x} = f(x, u)$$

where:

x : the $n \times 1$ state vector,

u : the $m \times 1$ input vector,

$\dot{x}$: the derivative of x with respect to time,

$f(x, u)$: the nonlinear function describing the system dynamics.

To linearize around an equilibrium point $\bar{x}, \bar{u}$, we define small deviations from this point as $x = \bar{x} + \Delta x$ and $u = \bar{u} + \Delta u$. Then, we approximate the nonlinear function $f(x, u)$ using a first-order Taylor series expansion:

$$f(x, u) \approx f(\bar{x}, \bar{u}) + \left.\frac{\partial f}{\partial x}\right|_{\substack{x=\bar{x} \\ u=\bar{u}}} \Delta x + \left.\frac{\partial f}{\partial u}\right|_{\substack{x=\bar{x} \\ u=\bar{u}}} \Delta u$$

where:

$$\left.\frac{\partial f}{\partial x}\right|_{\substack{x=\bar{x} \\ u=\bar{u}}} : \text{the } n \times n$$

Jacobian matrix of f with respect to x evaluated at the equilibrium point,

$$\left.\frac{\partial f}{\partial u}\right|_{\substack{x=\bar{x} \\ u=\bar{u}}} : \text{the } n \times m$$

Jacobian matrix of f with respect to u evaluated at the equilibrium point.

This approximation allows us to represent the system dynamics in a linear form:

$$\Delta\dot{x} = A\Delta x + B\Delta u$$

where:

$$A = \left.\frac{\partial f}{\partial x}\right|_{\substack{x=\bar{x}\\u=\bar{u}}} \text{ is the } n \times n \text{ state matrix,}$$

$$B = \left.\frac{\partial f}{\partial u}\right|_{\substack{x=\bar{x}\\u=\bar{u}}} \text{ is the } n \times m \text{ input matrix.}$$

These matrices A and B are crucial for linear control design and analysis.

In our system, the number of states is 5 and corresponds to

$$X = \begin{bmatrix} \phi \\ \theta \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

And we get

$$\dot{X} = \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix}$$

and the number of inputs is 4:

$$U = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix}$$

These variables represent:

$$X : \text{the } 5 \times 1 \text{ state vector,}$$
$$U : \text{the } 4 \times 1 \text{ input vector.}$$

By calculating the jacobian of the equation 26 and add the two first rows in A matrisx with respect to our states we get the A matrix:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{2J_y\omega_z - 2J_z\omega_z}{2J_x} & \frac{2J_y\omega_y - 2J_z\omega_y}{2J_x} \\ 0 & 0 & -\frac{2J_x\omega_z - 2J_z\omega_z}{2J_y} & 0 & -\frac{2J_x\omega_x - 2J_z\omega_x}{2J_y} \\ 0 & 0 & \frac{J_x\text{lift}\omega_y - J_y\text{lift}\omega_y}{J_z\text{lift}} & \frac{J_x\text{lift}\omega_x - J_y\text{lift}\omega_x}{J_z\text{lift}} & 0 \end{bmatrix} \tag{27}$$

Now, by using

$$J = \begin{bmatrix} 1.146 \times 10^{-5} & 0 & 0 \\ 0 & 1.699 \times 10^{-5} & 0 \\ 0 & 0 & 2.994 \times 10^{-5} \end{bmatrix}$$

,

and equilibrium points $X = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ which is chosen because we want the drone to have zero angles initially, ensuring no tilting and input $U = \begin{bmatrix} 30000 \\ 30000 \\ 30000 \\ 30000 \end{bmatrix}$ is set to maintain a hover state without any lifting. With these parameters, the A matrix finally becomes:

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

This matrix represents the state transition matrix for the given system at the specified equilibrium point and input.

We calculate the same equation with respect to the inputs to calculate the B matrix and by adding then inputed equilibrium point , we get:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.0325 & 0.0325 & -0.0325 & -0.0325 \\ -0.0325 & 0.0325 & 0.0325 & -0.0325 \\ -0.0275 & 0.0275 & -0.0275 & 0.0275 \end{bmatrix}$$

All the steps is made in matlab to simplify the workflow.

The system is now linearized. But we still need to discretize the system now since the programs is always discretized.

Now given a continuous-time linearized state-space model:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^m$ is the input vector, and $y(t) \in \mathbb{R}^p$ is the output vector. A, B, C, and D are constant matrices of appropriate dimensions.

To discretize the system using the zero-order hold (ZOH) method with a sampling time $T_s = 0.01$, we can use the following formula:

$$x(k+1) = (I + AT_s)x(k) + \int_0^{0.01} (I + A(0.01 - \tau))Bu(\tau)d\tau$$
$$y(k) = Cx(k) + Du(k)$$

For the integral term, when $T_s = 0.01$, we have:

$$\int_0^{0.01} (I + A(0.01 - \tau))Bu(\tau)d\tau = \int_0^{0.01} (I + 0.01A)Bu(\tau)d\tau$$

Since I and A are constant matrices, we can pull them out of the integral:

$$(I + 0.01A)\int_0^{0.01} Bu(\tau)d\tau = (I + 0.01A)B \cdot \int_0^{0.01} u(\tau)d\tau$$

We know that the integral of $u(\tau)$ over the interval $[0, T_s]$ is simply $T_s \cdot u(k)$. Therefore, we have:

$$(I + 0.01A)B \cdot T_s = B \cdot T_s + 0.01AB$$

So, the simplified expression for B is:

$$B = (I - 0.01A)^{-1}(I + 0.01A)T_s$$

The discretization was performed in MATLAB to simplify the workflow, and the function `c2d(system,0.01,"zoh")` was used to obtain discretized matrices A and B.

$$A_{\text{discretized}} = \begin{bmatrix} 1.0000 & 0 & 0.0100 & 0 & 0 \\ 0 & 1.0000 & 0 & 0.0100 & 0 \\ 0 & 0 & 1.000 & 0 & 0 \\ 0 & 0 & 0 & 1.000 & 0 \\ 0 & 0 & 0 & 0 & 1.000 \end{bmatrix}$$

$$B_{\text{discretized}} = \begin{bmatrix} -0.1419 & -0.1419 & 0.1419 & 0.1419 \\ -0.0957 & 0.0957 & 0.0957 & -0.0957 \\ -28.3756 & -28.3756 & 28.3756 & 28.3756 \\ -19.1414 & 19.1414 & 19.1414 & -19.1414 \\ -9.1838 & 9.1838 & -9.1838 & 9.1838 \end{bmatrix}$$

V. Design of the Linear Quadratic Controller

The Linear Quadratic Regulator (LQR) is a potent control design technique employed to optimize the performance of linear dynamic systems. LQR seeks to identify the optimal control that minimizes a performance criterion, represented by a quadratic cost function. Specifically tailored for linear time-invariant systems, LQR necessitates linearization of nonlinear systems around an operating point to facilitate its implementation. The cost function is structured as a quadratic equation. In our application, full state feedback was favored over integral state feedback due to its ease of implementation and tuning.

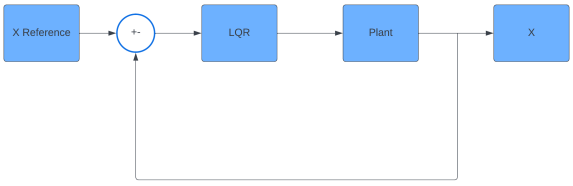

Fig. 3. Feedback system

The cost function is mathematically defined as:

$$J = \boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{u}^T R \boldsymbol{u} \tag{28}$$

where:

- J represents the cost function.
- $\boldsymbol{x}$ denotes the state vector.
- $\boldsymbol{u}$ signifies the control input.
- Q is the state weighting matrix.
- R is the control weighting matrix.

The Discrete-Time Riccati Equation is expressed as:

$$P_k = A^T P_{k+1} A - A^T P_{k+1} B (R + B^T P_{k+1} B)^{-1} B^T P_{k+1} A + Q$$

where:

- A represents the system matrix.
- B is the control input matrix.
- P signifies the solution to the discrete Riccati equation.

To derive the general LQR optimal gain for discrete A and B matrices, the following steps are undertaken:

1) **Definition of System Matrices:**

$$A \text{ (discrete matrix)}, \quad B \text{ (discrete matrix)}$$

2) **Definition of Weighting Matrices:**

$$Q = \text{weighting matrix for states}, \quad R = \text{weighting matrix for inputs}$$

3) **Solving the Discrete-Time Riccati Equation:**

$$P_k = A^T P_{k+1} A - A^T P_{k+1} B (R + B^T P_{k+1} B)^{-1} B^T P_{k+1} A + Q$$

4) **Computing the Control Gain:**

$$K_k = (R + B^T P_{k+1} B)^{-1} B^T P_{k+1} A$$

$$u_k = -K_k x_k$$

It is noteworthy that in MATLAB, the computed K matrix already incorporates the negative sign.

Adjusting the values in the Q matrix enables the modulation of punishment on different states. Thus, increasing or decreasing the value in the Q matrix allows for the adjustment of controller aggressiveness for specific states. Similarly, adjustments in the R matrix impact controller behavior concerning inputs.

By fine-tuning the Q and R matrices, diverse controller behaviors can be achieved. Therefore, meticulous tuning of these matrices is imperative to develop a robust controller capable of effectively tracking reference signals. The process of tuning lacks a universally prescribed methodology, thus requiring a considerable amount of time to achieve optimal values. However, this duration is contingent upon the developer's desired behavior for the controller, whether it be aggressive or more subdued.

Tuning Q and R matrices:
Starting from the following matrices:

$$Q = \text{diag}([1, 1, 1, 1, 1]), \quad R = \text{diag}([1, 1, 1, 1])$$

Then roll vs roll reference yields as the following:

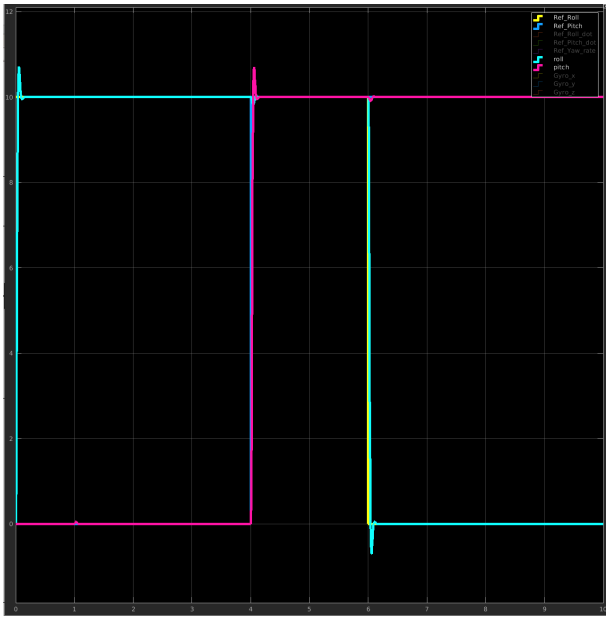

Fig. 4. Simulation of roll vs roll reference

With some different values for Q and R we obtain:

Fig. 5. Simulation of roll vs roll reference

The result is still good, which is logical due to not punishing the inputs. The chosen Q and R matrices are the following:

$$Q = \text{diag}([0.15, 0.15, 1, 1, 1]), \quad R = \text{diag}([1, 1, 1, 1])$$

By using the discrete A and B matrices from the Plant Linearization section in the report and the chosen Q and R matrices, we could use the MATLAB function *dlqr* to get the optimal gain. The choice behind the function *dlqr* is because our plant is discretized.

$$K_d = \begin{bmatrix} -0.0034 & -0.0050 & -0.0088 & -0.0130 & -0.0265 \\ -0.0034 & 0.0050 & -0.0088 & 0.0130 & 0.0265 \\ 0.0034 & 0.0050 & 0.0088 & 0.0130 & -0.0265 \\ 0.0034 & -0.0050 & 0.0088 & -0.0130 & 0.0265 \end{bmatrix}$$

VI. Evaluation of control design

Figure 5 depicts how well the system will follow a step function. As mentioned before this gave a good result. However there is a ”bump” at around 4 seconds, which could be caused due to coupling effects. The roll and pitch movements are not entirely independent in a quadrocopter. Adjusting the thrust on one motor to control roll can inadvertently affect the pitch, and vice versa. Figure 6 illustrates this more clearly, where both the roll and pitch angle and their respective references are plotted.

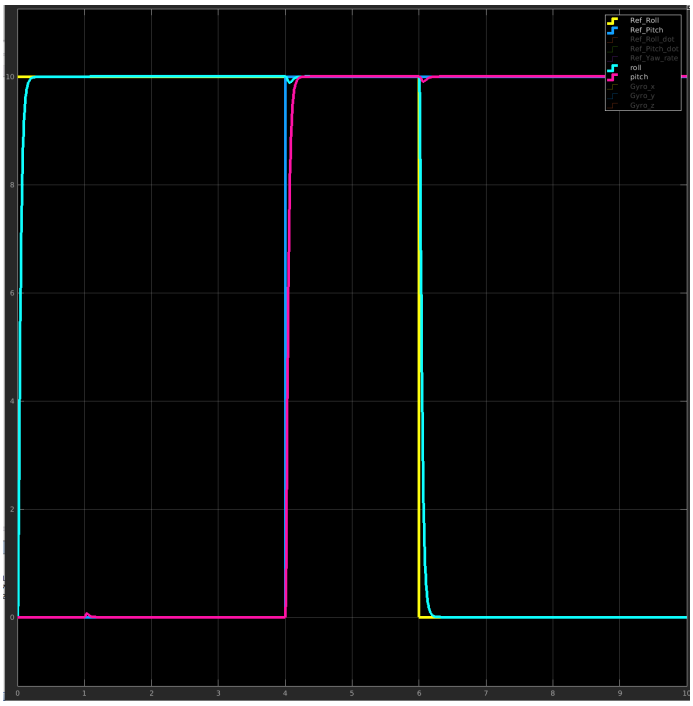

Fig. 6. Caption

After the evaluation in matlab, a separate evaluation in the c-code was made. This is presented in figure 7.

It is clear that the system is able to follow the reference signal well. However, the system operates somewhat slowly (i.e. there is a small delay), this depends on the complementary filter. How fast an estimation can be made is crucial, on the other side it can not be very noisy. This is a trade off and can be tuned depending on the operator’s priorities. In figure 7, $\alpha = 0.05$ and $\gamma = 0.83$.

As mention above, it is also clear that the estimation is

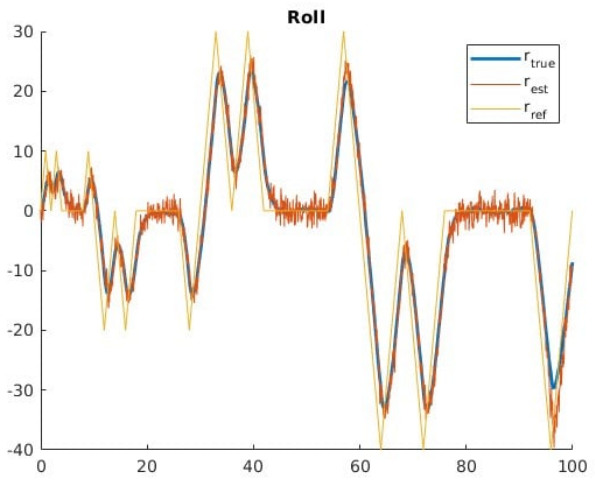

Fig. 7. Evaluation of controller in C

noisy. The filter could do a better noise attenuation at the cost of a larger time delay. This can be proved by the fact that when α is increasing, the time constant for the filters are also increasing.

When testing the controller in reality, the system started good, although after about 2 second it became unstable.

VII. Conclusions

Figure 8 illustrates how the complete system works.

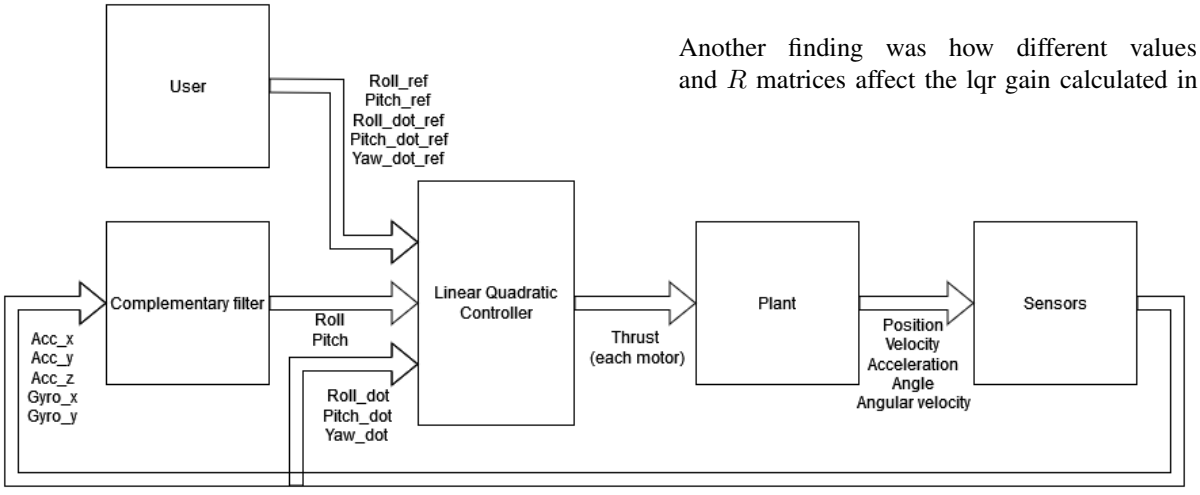

Fig. 8. Drone flowchart

Complementary filter

The complementary filter is a crucial subsystem for the controller since it generates the roll and pitch angles. These are compared to the reference signal for roll and pitch which generates the error signal for the controller.

As mentioned before, the complementary filter consists of 1 high-pass filter and 1 low-pass filter. Both complementary filter depends on a common variable named α. Equation 12 shows the relation between α and γ. It is clear that in some sense γ represents the cut-off frequency for the low-pass and high-pass filter. One of the findings here is how different values of α affects the attenuation of noise and delay. This is a well known trade-off and basically says that good attenuation means a long delay and vice versa.

Plant modeling

The plant modeling part can be developed further as all models can. Some examples of what could be improved are:

- Each motor is different and all of them should maybe not be linearized around 30 000 as input.
- The air resistance is not constant in every direction.
- Dynamic Load Variation - Improving models to dynamically adjust parameters based on load conditions can lead to more robust performance and predictive capabilities.

Linear Quadratic Controller

For the design of the Linear Quadratic Controller there was a lot of conclusions. One of them was to use full state feedback as the control strategy. This was the case since it made it simple to set the reference value for a specific angle or angular velocity.

Another finding was how different values for the Q and R matrices affect the lqr gain calculated in matlab.

References

[1] T. Glad and L. Ljung, *Control Theory - Multivariable and Nonlinear Methods*. Taylor and Francis, 2002.