Final report: Modelling, Simulation and Control Group 8

1st Gustav Isaksson
Systems, Control and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden
igustav

2nd Gonzalo Urbanos
Systems, Control and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden
urbanos

3rd Mostafa Husseini
Systems, Control and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden
moshus

4th Yaochen Song
Systems, Control and Mechatronics
Chalmers Technical University
Gothenburg, Sweden
yaochen

Abstract— This report outlines a design approach for a quadrotor that is based on a model. The design process begins with the development of a complementary filter that is employed to estimate the orientation of the quadrotor. Following this, an acausal modeling technique is utilized to describe the quadrotor's dynamics using differential equations. As the resulting plant model exhibits nonlinear dynamics, a linearization of the plant model is undertaken to facilitate the use of well-established control design methods for linear systems. Subsequently, a linear quadratic controller is designed to regulate the quadrotor around 0° in roll and pitch angles.

I. Introduction to control of Quadrocopters

Nowadays, quadrocopters have become increasingly prevalent in various industries and fields, including both military and civilian applications. Two common quadrotor configurations are the '+' and '×' models [1]. The quadrocopters utilized in the project, feature an X-configuration as illustrated in figure 1. The figure highlights the directions considered positive for x, y, and z positions, as well as the positive directions for roll, pitch, and yaw angles. With 6 degrees of freedom (x, y, z, φ, θ, and ψ) and only four motors, the drone is under-actuated. This means that the quadrocopters must rotate toward a particular direction before proceeding in that trajectory. Moreover, the figure also shows the direction of motor rotation and the upward direction of the thrusts generated by the motors.

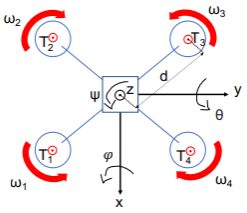

Figure 1. X configuration quadrocopters

II. Estimation of orientation

In order to achieve the main goal, control of the orientation of the quadrocopter, an effective method must be found in order to estimate the angles of the quadrocopter. In this section, there will be an analysis of how accurate estimations of the roll (φ) and pitch (θ) angle can be made. Estimating the yaw (ψ) angle is something that will not be possible and the reason why this is the case, will be brought up later in this section.

The quadrocopter is equipped with an Inertial Measurement Unit (IMU) which contains a 3-axis accelerometer and gyroscope. The accelerometer measures the external specific force acting on the sensor in each axis of the quadrocopter while the gyroscope measures the angular velocity in each axis. The main question in this section is how we can obtain accurate estimations of roll and pitch based on measurements from the accelerometer and gyroscope.

A. Estimation of Roll (φ) and Pitch (θ) via Accelerometer

The first thing which will be analyzed is how estimations of roll and pitch angle can be derived given measurements from the accelerometer. The derivations will be based on equation 1-2 below where $\boldsymbol{f}^B$ is the accelerometer readings, k is a proportionality constant, $\mathbf{R}_W^B$ is the rotation matrix in the WCF (world coordinate frame) with respect to the body frame with XYZ extrinsic rotation order, $\mathbf{a}^W$ is the linear acceleration vector with respect to the WCF and $\mathbf{g}$ is the gravitational force vector. Note that $\mathbf{R}_i(\alpha)$ is an elementary rotation matrix in a certain coordinate frame that rotates with the angle α around the axis $i \in \{x, y, z\}$.

An important note in equation 1 is that the linear accelerations of the quadrocopter were assumed to be zero. This is a strict demand since the estimations of roll and pitch rely on that changes in accelerometer readings are because of changes in orientation and not linear acceleration. Another note is that

the accelerometer is calibrated to give -1 when aligned with earths gravitational field.

$$\underbrace{\begin{bmatrix} f_x^B \\ f_y^B \\ f_z^B \end{bmatrix}}_{\boldsymbol{f}^B} = k\,\mathbf{R}_W^B \left(\underbrace{\begin{bmatrix} a_x^W \\ a_y^W \\ a_z^W \end{bmatrix}}_{\boldsymbol{a}^W} - \underbrace{\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}}_{\boldsymbol{g}} \right) = k\,\mathbf{R}_W^B \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

$$\mathbf{R}_B^W = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\varphi) \tag{2}$$

By applying equation 1-2, the final expression of our accelerometer readings in terms of the proportionality constant k, roll angle φ and pitch angle θ can be derived accordingly. Continuing, the derivation of the estimation of roll and pitch can then be expressed as follows below. As mentioned earlier, the yaw angle ψ can't be estimated and the reason for that is because it does not appear in the last column of $(\mathbf{R}_B^W)^\top$.

$$\begin{bmatrix} f_x^B \\ f_y^B \\ f_z^B \end{bmatrix} = k\,\mathbf{R}_W^B \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = k\,(\mathbf{R}_B^W)^\top \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} =$$

$$k\,\mathbf{R}_x(\varphi)^\top \mathbf{R}_y(\theta)^\top \mathbf{R}_z(\psi)^\top \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = k \begin{bmatrix} -\sin(\theta) \\ \cos(\theta)\sin(\varphi) \\ \cos(\theta)\cos(\varphi) \end{bmatrix} \tag{3}$$

Estimation of Roll angle φ

$$\frac{f_y^B}{f_z^B} = \frac{k\cos(\theta)\sin(\varphi)}{k\cos(\theta)\cos(\varphi)} = \tan(\varphi) \implies \varphi = \text{atan2}(f_y^B, f_z^B)$$

Estimation of Pitch angle θ

$$(f_y^B)^2 + (f_z^B)^2 = k^2\cos^2(\theta)\underbrace{(\sin^2(\varphi) + \cos^2(\varphi))}_{=1} = k^2\cos^2(\theta)$$

$$\tan(\theta) = \frac{\sin(\theta)}{\cos(\theta)} = \frac{-k\,f_x^B}{k\sqrt{(f_y^B)^2 + (f_z^B)^2}}$$

$$\implies \theta = \text{atan2}(-f_x^B, \sqrt{(f_y^B)^2 + (f_z^B)^2})$$

B. Complementary Filter

Now there is a straight forward method for estimating roll φ and pitch θ via the accelerometer readings but there is one catch. The estimation hardly relies on the assumption that the linear accelerations of the quadrocopter are zero but this is something that can't be guaranteed during the whole time. Another possible method is to estimate the roll φ and pitch θ via the gyro sensor by integrating the measurements. The problem with this though is that the angle estimations will start to deviate after a while because of the integration and the drift in the gyroscope. Therefore, a complementary filter will be utilized which takes both the measurements from the accelerometer and gyroscope into account in order to estimate the roll φ and pitch θ.

The complementary filter sends the angle estimations from the accelerometer into a low-pass filter (LP) since the measurements from the accelerometer are quite noisy. On the other side, the filter sends the angle estimations from the gyro into a high-pass filter (HP), since the gyroscope tends to drift over time, and then adds these two different results together in order to get one single estimation of roll φ and pitch θ. This can be studied in figure 2 below and from that figure, the angle estimations in continuous Laplace domain can be gathered according to equation 4 and 5.

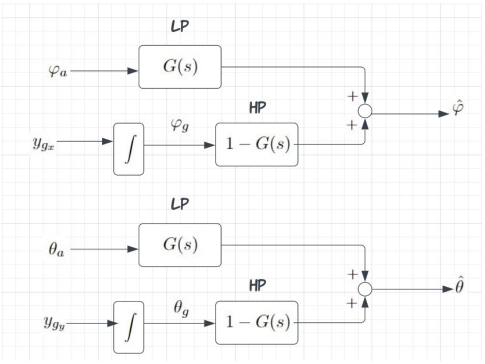

Figure 2. The complementary filter in continuous Laplace domain

$$\varphi(s) = G(s)\varphi_a(s) + (1 - G(s))\,\frac{1}{s}Y_{g_x}(s) \tag{4}$$

$$\theta(s) = G(s)\theta_a(s) + (1 - G(s))\,\frac{1}{s}Y_{g_y}(s) \tag{5}$$

where φ_a, θ_a is the angle estimations from the accelerometer, Y_{g_x}, Y_{g_y} is the angular velocity measurements from the gyro and $G(s) = \frac{1}{\alpha s + 1}$

In order to simulate this complementary filter in a computer, it has to be discretized first. This can easily be done by applying the Euler-backward discretization method in equation 6 below. After discretization, the roll φ and pitch θ estimations are obtained according to equation 7 and 8 below. Note that the value of γ determines how much trust each sensor should be given. This means when $\gamma \approx 0$, then only the accelerometer is being trusted and when $\gamma \approx 1$, then only the gyro is being trusted.

$$\dot{x}(t) \approx \frac{x(kh) - x((k-1)h)}{h} = \frac{x_k - x_{k-1}}{h},\ t = kh \tag{6}$$

$$\hat{\varphi}_k = (1-\gamma)\varphi_{a,k} + \gamma\,(\hat{\varphi}_{k-1} + h\,y_{gx,k}),\ \ \gamma = \frac{\alpha}{h+\alpha} \tag{7}$$

$$\hat{\theta}_k = (1-\gamma)\theta_{a,k} + \gamma\,(\hat{\theta}_{k-1} + h\,y_{gy,k}),\ \ \gamma = \frac{\alpha}{h+\alpha} \tag{8}$$

C. Evaluation of the Complementary Filter

In figure 3 it can be seen how the discretized complementary filter is implemented and the tuning factor γ was chosed to be 0.98. In figure 4 and 5 the result of the angle estimations using the complementary filter can be studied. Illustrated below, the estimated roll φ and pitch θ (red line) is following the estimation of the accelerometer (blue line) but with significantly less noise. Continuing, notice that the angle estimations from the gyro are deviating from the estimations from the accelerometer which is because of the drift in the gyro sensor and that it's being integrated over time.

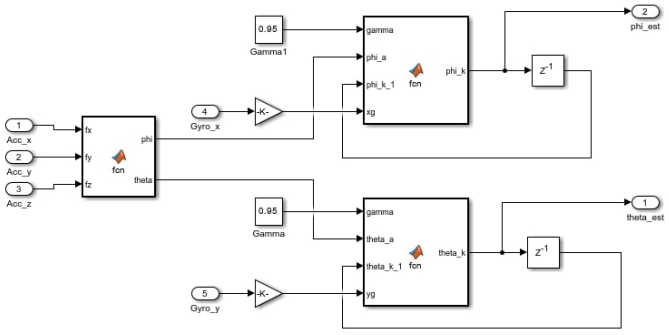

Figure 3. The implementation of the complementary filter

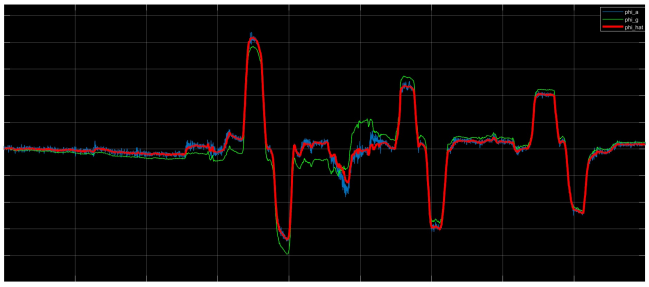

Figure 4. Simulation of the estimated roll φ angle using the complementary filter

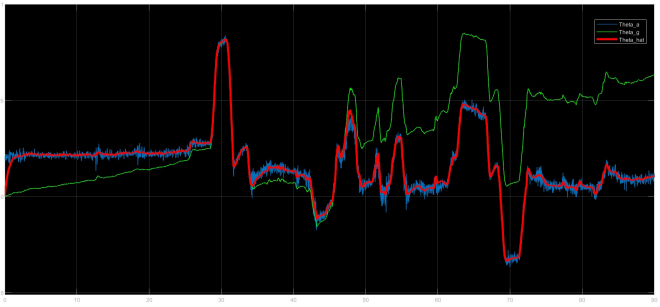

Figure 5. Simulation of the estimated pitch θ angle using the complementary filter

III. Plant Modelling in Simscape

The thrust of each rotor is modeled by the following equation. When the rotor speed $\omega_i > 0$ the thrust will be an upward vector, where b is the **lift** constant.

$$T_i = b\omega_i^2, \quad \text{for } i = \{1, 2, 3, 4\} \tag{9}$$

Since there is an upward vector in each rotor (thrust) we will have a force opposing to the upward motion called aerodynamic drag. This torque is modeled by the following equation where k is the **drag**.

$$Q_i = k\omega_i^2, \quad \text{for i = 1, 2, 3, 4} \tag{10}$$

The traditional dynamics of the drone in the world coordinate frame are given by:

$$m\dot{\boldsymbol{v}} = -\begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \mathbf{R}_B^W \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} - B\boldsymbol{v} \tag{11}$$

where B stands for the **aerodynamic friction** coefficient and T stands for the sum of the thrusts generated by each rotor, $T = T_1 + T_2 + T_3 + T_4$.

The rotational accelerations ($\dot{\boldsymbol{\omega}}$) is governed by Euler's equation:

$$\boldsymbol{J}\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} + \boldsymbol{\Gamma} \tag{12}$$

where $\boldsymbol{\Gamma} = [\tau_x\ \tau_y\ \tau_z]^\top$ is the torque applied to the drone (13)

where,

$$\tau_x = \frac{d}{\sqrt{2}}(T_3 + T_4 - T_1 - T_2) \tag{14}$$

$$\tau_y = \frac{d}{\sqrt{2}}(T_3 + T_2 - T_1 - T_4) \tag{15}$$

$$\tau_z = Q_2 + Q_4 - Q_1 - Q_3 = \frac{k}{b}(T_2 + T_4 - T_1 - T_3) \tag{16}$$

There is a difference from the '+' to '×' configuration in the thrust equations for x and y components. In the ×-configuration the distance from the motors to the center is now $d\cos 45$ and that's where the $\frac{d}{\sqrt{2}}$ factor comes from. It is important to note that the gyro measures the angular velocity in the body frame $[\dot{\varphi}\ \dot{\theta}\ \dot{\psi}]^\top$ while in the model we need the orientation of the WCF, which means that we need corresponding angular velocities in WCF $[\omega_r\ \omega_p\ \omega_y]^\top$. Therefore, we have to take into consideration a rotation matrix that transforms the body coordinate frame into the WCF in order to find the orientation in the WCF. Then, the following transformation is made:

$$\begin{bmatrix} \omega_r \\ \omega_p \\ \omega_y \end{bmatrix} = \begin{bmatrix} \dot{\varphi} \\ 0 \\ 0 \end{bmatrix} + \mathbf{R}_X \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathbf{R}_X\mathbf{R}_Y \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{17}$$

$$\begin{bmatrix} \omega_r \\ \omega_p \\ \omega_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & s(\theta) \\ 0 & c(\varphi) & -s(\varphi)c(\theta) \\ 0 & s(\varphi) & c(\varphi)c(\theta) \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{18}$$

IV. Open-loop simulations

The following scenarios represent the ones given in the assignment description. Figures 6 and 7 serve as evidence to show that the open-loop model of the plant is correctly designed. Scenario 1 is the result of a thrust of value 10 for M_1 and M_2 at a step time 4s, whereas M_3 and M_4 will have a 0 constant value. Scenario 2, represents a motor configuration of 0 constant value in M_2 and M_4 whereas M_1 takes a step of 1000 at time 4s and in M_3 a step of 2000 in time 6s.

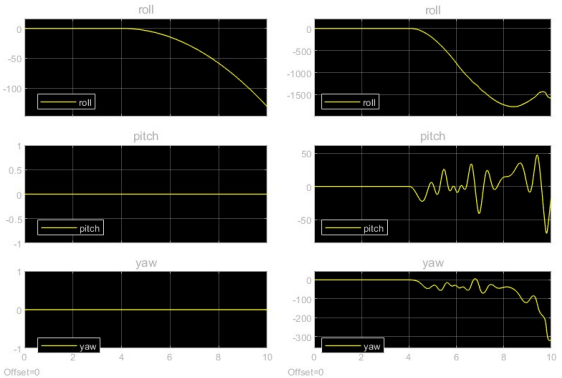

Figure 6. Scenario 1

Figure 7. Scenario 2

In figure 8 and 9 the simulation of the open-loop plant can be studied with our own scenarios in order to confirm that it works as expected. In figure 8, a constant Pulse Width Modulation (PWM) thrust of 1000 was applied to motor M_2 and M_3 while the thrust to motor M_1 and M_4 was initially set to zero. After 5 seconds, the thrust to motor M_2 and M_3 was set to zero while the PWM thrust to motor M_1 and M_4 was set to 1000 instead. The expected behavior is that the pitch angle first increases and at 5 seconds one can see that the angle still increases but at a slower rate since motor M_1 and M_4 are trying to rotate the other direction around the pitch axis. After a while, one can see that the angle is back to zero and then it will be negative and continue to decrease. For scenario 4, similar test was initiated as in the previous scenario but around the yaw axis instead. This means that motor M_2 and M_4 was initialized with the PWM thrust 1000 and M_1 and M_3 to zero. It can be confirmed that this also works as intended.

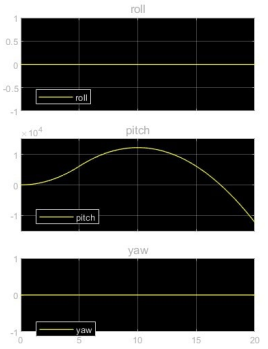

Figure 8. Scenario 3

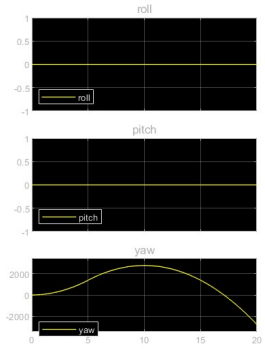

Figure 9. Sceanrio 4

V. Linearization of the plant

In the derivations of a linearized state space model, only the angles and angular velocities was considered as states since the main goal is to control the orientation of the drone. Therefore, only the differential equation 12 will be used and not 11 from the modelling part. The states and inputs (thrust) of the system for linearization are:

$$\boldsymbol{x} = \begin{bmatrix} \varphi \\ \theta \\ \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \ , \ \boldsymbol{u} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}$$

Since $\boldsymbol{\omega}$ is the angular velocity of the drone with respect to the world coordinate frame, the orientation of the body frame can be updated using the relationship between the derivative of the Euler angles and angular velocity according to 19. By utilizing equation 19, the non-linear state space model can be derived according to equation 20.

$$\underbrace{\begin{bmatrix} \omega_r \\ \omega_p \\ \omega_y \end{bmatrix}}_{\boldsymbol{\omega}} = \begin{bmatrix} 1 & 0 & s(\theta) \\ 0 & c(\varphi) & -s(\varphi)c(\theta) \\ 0 & s(\phi) & c(\varphi)c(\theta) \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{19}$$

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \ddot{\varphi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \boldsymbol{J}^{-1}\Big(-\boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} + \boldsymbol{\Gamma} \Big) \end{bmatrix} \tag{20}$$

The linearized model will be:

$$\Delta\dot{\boldsymbol{x}} = \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}} \, |_{\boldsymbol{x_0}, \boldsymbol{u_0}} \, \Delta\boldsymbol{x} + \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}} \, |_{\boldsymbol{x_0}, \boldsymbol{u_0}} \, \Delta\boldsymbol{u} \tag{21}$$

$$\Delta\dot{\boldsymbol{x}} = \boldsymbol{A}\Delta\boldsymbol{x} + \boldsymbol{B}\Delta\boldsymbol{u} \ , \tag{22}$$

$$\boldsymbol{A} = \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}} \, |_{\boldsymbol{x_0}, \boldsymbol{u_0}} \ , \ \boldsymbol{B} = \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}} \, |_{\boldsymbol{x_0}, \boldsymbol{u_0}} \tag{23}$$

The stability of the linearized system is depending on the choice of operating point. For some operating points, the system is unstable and for others, the system is marginally stable. However, this doesn't really matter because the main interest is the stability of the closed-loop system. The only important condition which is required from the linearized model is that it's controllable. If it is controllable, then a controller, which places the poles of the closed-loop system in the left half plane, can be designed and thus guarantee a stable closed-loop system. Since the linearized model contains five states, the rank of the controllability matrix has to be five in order to obtain full rank and thus imply controllability. Based on this reasoning, a desirable operating point was chosen 24 such that the linearized model is controllable.

$$\boldsymbol{x_0} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top \tag{24}$$

Then the following linearized state space model is obtained linearized around $\boldsymbol{x_0}$:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\,\boldsymbol{x}(t) + \boldsymbol{B}\,\boldsymbol{u}(t) \tag{25}$$

where,

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

and

$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{\sqrt{2}\,d}{2\,\mathrm{Jx}} & -\frac{\sqrt{2}\,d}{2\,\mathrm{Jx}} & \frac{\sqrt{2}\,d}{2\,\mathrm{Jx}} & \frac{\sqrt{2}\,d}{2\,\mathrm{Jx}} \\ -\frac{\sqrt{2}\,d}{2\,\mathrm{Jy}} & \frac{\sqrt{2}\,d}{2\,\mathrm{Jy}} & \frac{\sqrt{2}\,d}{2\,\mathrm{Jy}} & -\frac{\sqrt{2}\,d}{2\,\mathrm{Jy}} \\ -\frac{k}{\mathrm{Jz\,b}} & \frac{k}{\mathrm{Jz\,b}} & -\frac{k}{\mathrm{Jz\,b}} & \frac{k}{\mathrm{Jz\,b}} \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -2837.557 & -2837.557 & 2837.557 & 2837.557 \\ -1914.136 & 1914.136 & 1914.136 & -1914.136 \\ -428.209 & 428.209 & -428.209 & 428.209 \end{bmatrix}$$

In order to simulate in the Crazyflie $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices are discretized. For doing the discretization, the `c2d` command was used such that:

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

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

The thing is we had to convert $\boldsymbol{B_d}$ matrix to degrees and convert the output thrust from the controller to Pulse width modulation (PWM), therefore, we had to multiply it by $1.2865 \cdot 10^{-4}$ and the following matrix was obtained:

$$\boldsymbol{B_d} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.0037 & -0.0037 & 0.0037 & 0.0037 \\ -0.0025 & 0.0025 & 0.0025 & -0.0025 \\ -0.0006 & 0.0006 & -0.0006 & 0.0006 \end{bmatrix}$$

VI. Design of Linear Quadratic Regulator

When the non-linear model has been linearized, the Linear Quadratic Regulator (LQR) can finally be derived. The idea behind LQR is to minimize the cost function V in equation 26. The cost matrices $\boldsymbol{Q}, \boldsymbol{R}$ are the tuning parameters for the LQR, and depending on the desired behavior of the controller, these cost matrices should be tuned accordingly. These cost matrices are then used in order to compute the state feedback gain (LQR gain) for the closed-loop system. The LQR gain is computed by first solving the ***Discrete Time Algebraic Riccati Equation*** in 27. When $\boldsymbol{P}$ has been solved, the LQR gain $\boldsymbol{K}$ can be derived according to equation 28. From this LQR gain, we can compute the optimal control action as $\boldsymbol{u} = \boldsymbol{K}\boldsymbol{x}$ and thus describe the closed-loop system according to the equation 30.

$$V = \sum_{k=0}^{\infty} \boldsymbol{x}(k)^\top \boldsymbol{Q}\,\boldsymbol{x}(k) + \boldsymbol{u}(k)^\top \boldsymbol{R}\,\boldsymbol{u}(k), \quad \boldsymbol{Q}, \boldsymbol{R} \succ 0 \tag{26}$$

$$\boldsymbol{P} = \boldsymbol{Q} + \boldsymbol{A_d}^\top \boldsymbol{P} \boldsymbol{A_d} - \boldsymbol{A_d}^\top \boldsymbol{P} \boldsymbol{B_d} (\boldsymbol{B_d}^\top \boldsymbol{P} \boldsymbol{B})^{-1} \boldsymbol{B_d}^\top \boldsymbol{P} \boldsymbol{A_d} \tag{27}$$

$$\boldsymbol{K} = -(\boldsymbol{B_d}^\top \boldsymbol{P} \boldsymbol{B_d} + \boldsymbol{R})^{-1} \boldsymbol{B_d}^\top \boldsymbol{P} \boldsymbol{A_d} \tag{28}$$

$$\boldsymbol{x}(k+1) = \boldsymbol{A_d}\boldsymbol{x}(k) + \boldsymbol{B_d}\boldsymbol{u}(k) \tag{29}$$

$$\boldsymbol{x}(k+1) = \boldsymbol{A_d}\boldsymbol{x}(k) + \boldsymbol{B_d}\boldsymbol{K}\boldsymbol{x}(k) \tag{30}$$

A. Implementation of LQR in Matlab/Simulink

For building the LQR, the `dlqr` command in MATLAB was used. The main difference between `dlqr` and `lqrd` is that in `dlqr` the input parameters used are the already discretized matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ in comparison to `lqrd` which uses continuous-time matrices and it carries out the discretization automatically. Since the same gain $\boldsymbol{K}$ matrix was obtained using both approaches, the `dlqr` command was selected because of the desire to compute the discretized matrices separately.

After tuning the state cost matrix and the control input cost matrix with the following weightings, the following LQR gain $\boldsymbol{K}$ was obtained:

$$\boldsymbol{Q} = \begin{bmatrix} 1000 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} 0.0001 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0.0001 & 0 \\ 0 & 0 & 0 & 0.0001 \end{bmatrix}$$

$$\boldsymbol{K} = \begin{bmatrix} -1504.2 & -1523.4 & -65.7 & -73.6 & -155.4 \\ -1504.2 & 1523.4 & -65.7 & 73.6 & 155.4 \\ 1504.2 & 1523.4 & 65.7 & 73.6 & -155.4 \\ 1504.2 & -1523.4 & 65.7 & -73.6 & 155.4 \end{bmatrix}$$

In order to assemble the reference, controller, and plant model simulink blocks we had to perform several steps. We decided to work with degrees instead of radians in the controller. Since our φ and θ estimates from the complementary filter were given in radians and the reference values for the angles were given in degrees, we had to convert the roll and pitch from complementary filter to degrees so that we could compute the difference to later multiply our state vector with the LQR gain. The output from the gyrosensor is always in radians per second, so the conversion takes place before entering the complementary filter. The references for the angular velocities are already in degrees per second. Since we only wanted to check roll and pitch angles we set a 0 reference for the angular velocities in the state vector.

After multiplying the LQR gain with the state vector we added the base thrust as a PWM signal to the one calculated from the LQR and sent it out to the plant. After that, new angle estimations are made and sent into the LQR. Observe that the calculated thrust after the LQR gain is already as a PWM signal since we modified the $\boldsymbol{B_d}$ matrix before we calculated our LQR gain. In Figure 10 we can see how the structure of the LQR looks like in Simulink.

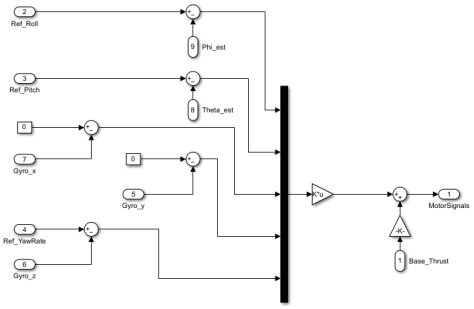

Figure 10. Linear Quadratic Regulator

B. Implementation of LQR in C language

Once the LQR has been designed via Matlab and Simulink, it's time to implement the corresponding controller in C code. The difference between the control design in Matlab/Simulink and C code is that in C we have to define the different components (complementary filter, LQR, reference generator) as separate tasks. Each task also contains a set of properties that should be considered such as priority, period, and deadline. The priority determines which task that should be prioritized more than the other tasks. This means that the task with the highest priority should start to execute immediately once it's released. The period determines when a task can start to execute again and sometimes the period is equal to the deadline of a task, which was the case in our implementation.

Another very important difference is how the reading and writing operations are constructed in C since there is a risk for race conditions to occur if we don't handle them. Race conditions are something that can occur if we perform multiple read/writing operations at the same time. The consequence of this is that variables might not have the predicted assigned value which will further affect the rest of the tasks. This is something that needs to be avoided and we can do so by utilizing a data structure called semaphore. A semaphore that is shared between several tasks has the ability to temporarily block other tasks. There are several different semaphores but in this implementation, semaphore mutex was used and its initial value is 1. The semaphore will temporarily block a task once its value reaches -1 and will unblock at values 0 and 1. By using the semaphore operations Take (decrease the value of semaphore by 1) and Give (increase the value of semaphore by 1) we can obtain the desired behavior of when certain tasks should be temporarily blocked or not. This enables us to protect the read and writing operations that occur in the three tasks, filter, controller, and reference generator.

Despite these differences, we used the same equations for the complementary filter and the same LQR gain as the one obtained via Matlab/Simulink. The priorities for the different tasks were set to 1 for all of them and period for the controller was set to 0.01 seconds and the period for the filter was set to 0.004 seconds. We will later see in the evaluation of the control design via C how these settings affect the performance of the controller.

VII. Evaluation of the control design

In this section we will present our results obtained from the control design using an LQ controller and we will carry out a discussion regarding how the different cost-weighting matrices affect the control of the system. Several simulation results, with and without base thrust, will be carried out in order to view the differences.

A. Evaluation of LQR in simulation via Matlab/Simulink

For the given weighting matrices and the given LQR gain in the previous section, two different scenarios was carried out in order to test the performance of the LQ controller.

Scenario 1: In this scenario, the behaviour of the control system will be studied in the vicinity of a disturbance. The system will be given one impulse disturbance of 0.1s for each angle at different times:

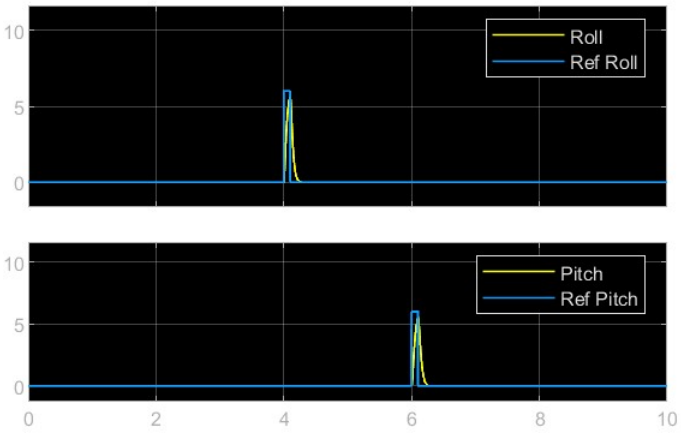

Figure 11. Scenario 1. System behavior to a state disturbance with base thrust.

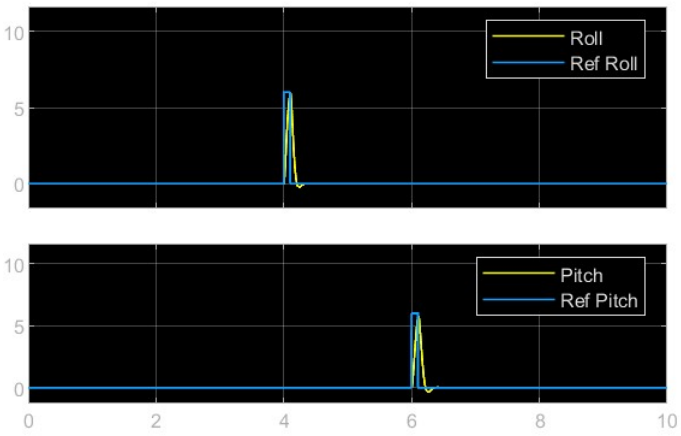

Figure 12. Scenario 1. System behavior to a state disturbance without base thrust.

From Figure 11 one can see that both, roll and pitch respond to the disturbance really fast. As well, the expected behavior is obtained because the LQ-controller stabilizes the quadrotor pose in roll and pitch angles around zero after the disturbance. From Figure 12 one can see that without base thrust, it takes a little bit longer time to stabilize the angles around zero since there is a very small overshoot. Basically, it behaves as expected, without base thrust there is a small delay when stabilizing. The base thrust is trying to compensate for the air friction force. It means when the base thrust is zero the drone will fall down and the air friction will disturb the controller, and this will lead to a harder stabilization.

Scenario 2: In this scenario, we are testing how does the roll and pitch angles behave when we want to follow a step reference. One step reference of 6s will be given for each angle at different times:

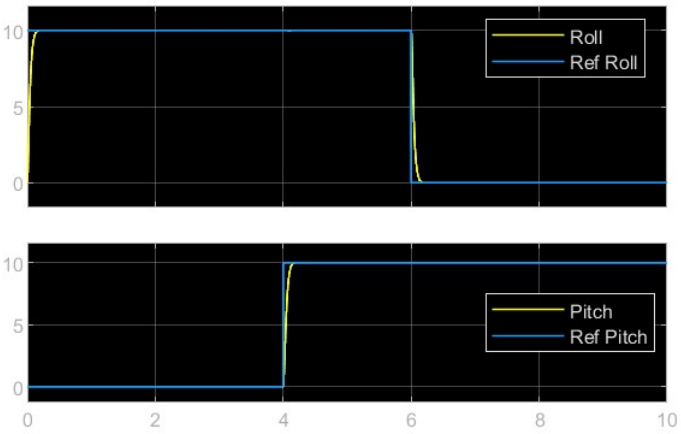

Figure 13. Scenario 2. System behavior to a step reference with base thrust.

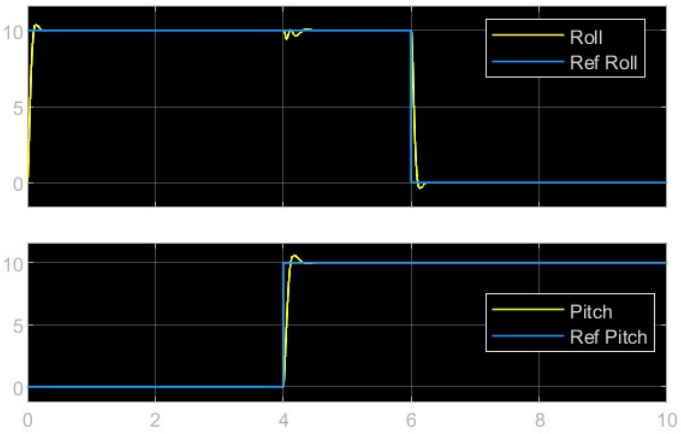

Figure 14. Scenario 2. System behavior to a step reference without base thrust.

From Figure 13 we can see that both, roll and pitch track their corresponding reference signals properly and without

overshoot. As well, when the reference signals go back to zero, so do the roll and pitch angles. From Figure 14 we can see the difference when we remove the base thrust. We get a small overshoot since it is harder to stabilize without base thrust. Moreover, in this case, we can see how the step in the pitch angle affect the roll angle as we can see a small oscillation right above.

Scenario 3: Apart from the impulse and step response scenarios we decided to try out our own case. It consists of a step response at the same time but with different final angle values for roll (10º) and pitch (5º).

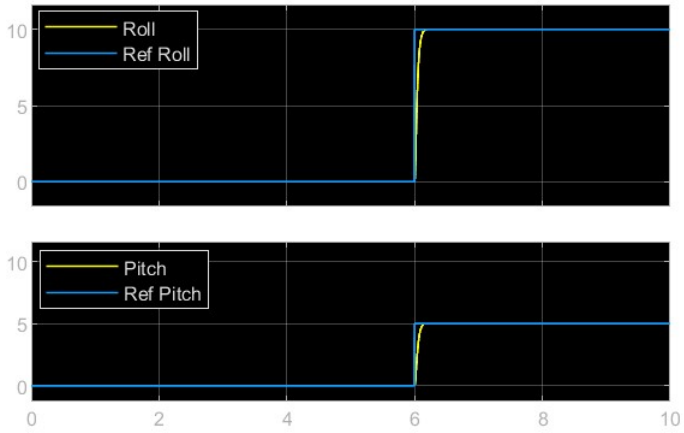

Figure 15. Scenario 3. System behavior to a step reference with base thrust.

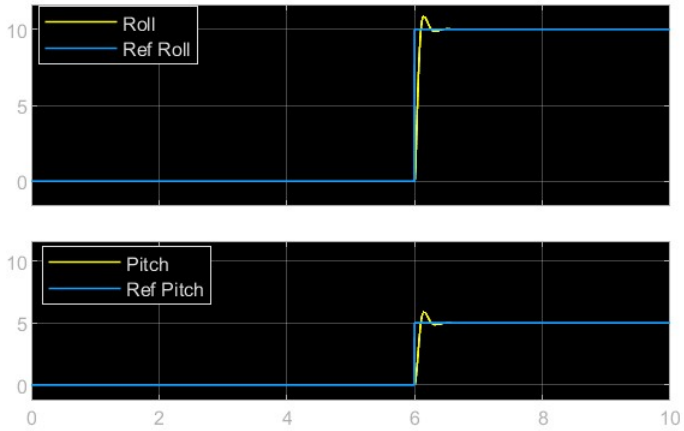

Figure 16. Scenario 3. System behavior to a step reference without base thrust.

From Figure 15, both the roll and pitch angles track their corresponding reference signals appropriately with no overshoot. When the reference signals return to the target angle, so do the roll and pitch angles. This demonstrates that under the condition of base thrust, our controller can properly control the quadrotor's attitude and allow it to reach the designated angles.

However, as can be seen in Figure 16, the system exhibits slightly different behavior when we remove the base thrust. There is a minor overshoot in the roll and pitch angles when tracking the respective reference signals, as it is more challenging to stabilize the system without base thrust. Furthermore, the angles also take some time for the target angles. This outcome is within our expectations, as without base thrust, the system would have some delay in stabilization.

From the different tunings which were carried out, we came across some observations on how the values of the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ changed the control of the LQ-controller. We know that the $\boldsymbol{Q}$ matrix is the state weighting matrix and therefore the first two diagonal elements correspond to roll φ and pitch θ respectively and the last three diagonal elements correspond to the angular velocities $\dot{\varphi}$, $\dot{\theta}$, $\dot{\psi}$ respectively. For example, by increasing roll and pitch weightings, a higher overshoot but a faster response will be obtained, and alternatively, when decreasing these weightings, a slower response and low overshoot will be obtained. The results can be observed in figures 17 and 18. It behaves in the opposite way for the angular velocities weightings, by increasing the weightings, the output overshoots less and by decreasing the weightings it overshoots more.

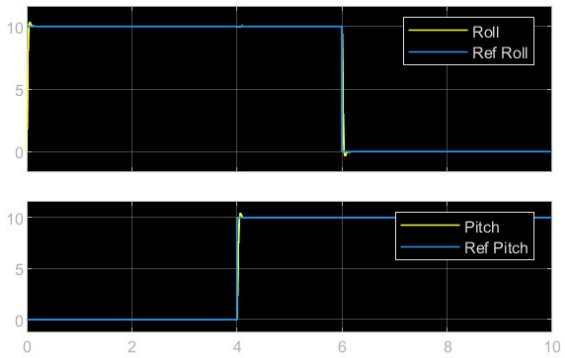

Figure 17. $\boldsymbol{Q}$ = diag([100000 100000 0.01 0.01 1]).

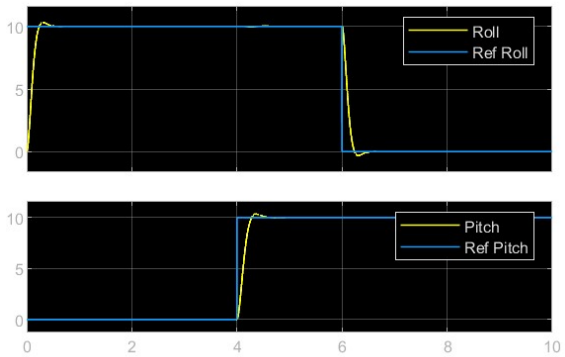

Figure 18. $\boldsymbol{Q}$ = diag([10 10 0.01 0.01 1]).

Now, having said how does the $\boldsymbol{Q}$ matrix affect the control of the system we are going to comment on how does $\boldsymbol{R}$ influence the control of the system. By increasing the value of the diagonal elements of the $\boldsymbol{R}$ matrix, a worse control will be obtained since the controller is slower as the roll and pitch takes time before they follow their corresponding reference signal. On the other hand, by decreasing the main diagonal elements of the $\boldsymbol{R}$ matrix, the response will be really fast and won't have an overshoot compared to higher values in the $\boldsymbol{R}$ matrix where the control is slower and with a slight overshoot. The results for these tryouts can be observed in Figures 19 and 20 below.

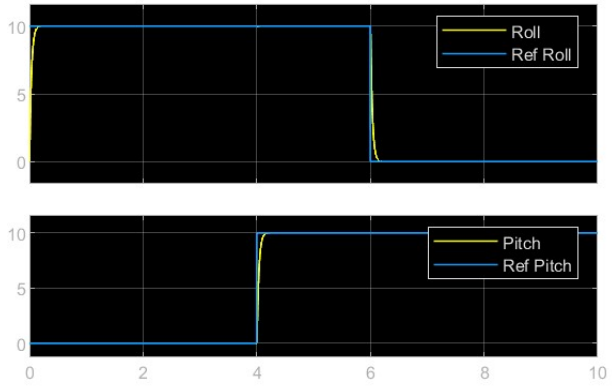

Figure 19. $\boldsymbol{R}$ = diag([$10^{-7}10^{-7}$ 10^{-7} 10^{-7} 10^{-7}]).

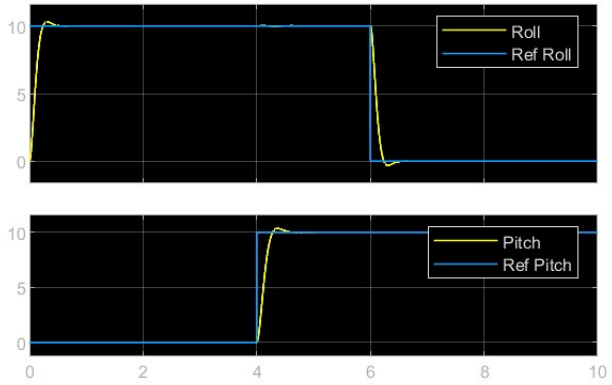

Figure 20. $\boldsymbol{R}$ = diag([0.01 0.01 0.01 0.01]).

B. Evaluation of LQR in simulation via C

Once the implementation of the LQR was made in C, we could start to create simulations of the controller via C. In figure 21 we can study how the true, estimated, and reference roll and pitch angle trajectories evolved over time. As we can see by the plots, it seems that our estimated roll and pitch angles manage to track their corresponding reference in a good way. Also, note that these simulations are based on the same LQR gain as the one obtained via Matlab/Simulink. We will see in the next sections how this current LQR gain affects the real physical drone.

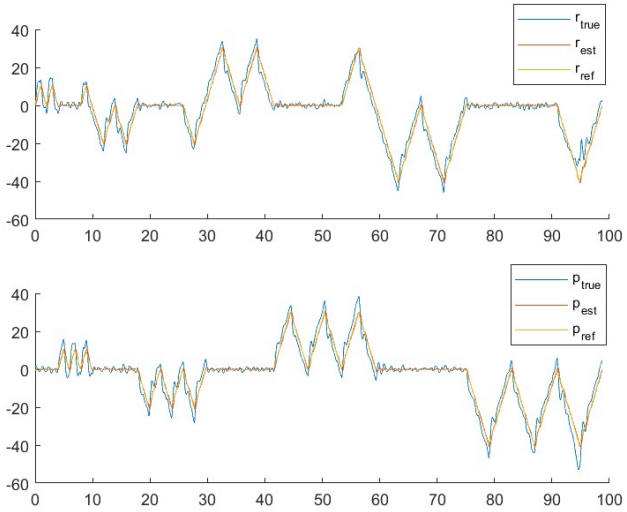

Figure 21. Simulation of the LQR in C

C. Evaluation of LQR in Physical Crazyflie via Matlab/Simulink

By utilizing the LQR gain, which has exhibited successful performance in the C simulation, and by generating C code from the Crazyflie model in Simulink and subsequently loading the code onto the physical drone, it was observed that the Crazyflie was not able of maintaining its balance at all. This implies that this gain is not an appropriate choice, despite its satisfactory performance in the C simulation. Hence, it is important to modify the components within the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, aiming to enable the drone to maintain its balance. After applying multiple tests (the drone could only apply pitch), and loading the generated C code from the Simulink model to the physical drone, the drone could eventually keep its balance. The appropriate state cost matrix, control input cost matrix, and LQR gain have been shown below:

$$\boldsymbol{Q} = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot 10^{-7}$$

$$\boldsymbol{K} = \begin{bmatrix} -5866.7 & -7688.9 & -593.5 & -779.0 & -1329.5 \\ -5866.7 & 7688.9 & -593.5 & 779.0 & 1329.5 \\ 5866.7 & 7688.9 & 593.5 & 779.0 & -1329.5 \\ 5866.7 & -7688.9 & 593.5 & -779.0 & 1329.5 \end{bmatrix}$$

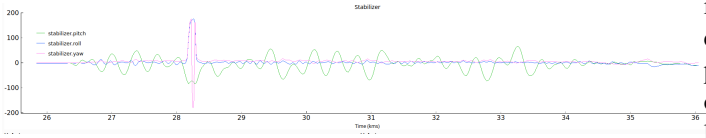

Figure 22. Estimated angles captured by Crazyflie client

According to figure 22, since the drone can only rotate around the y-axis (pitch) in our tests, it tries to keep its balance around zero. Despite oscillating around zero pitch, the drone can successfully keep its balance around the reference point and that's why these $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices are appropriate.

D. Evaluation of LQR in Physical Crazyflie via C

After getting appropriate results from generated C code from Simulink, and determining the optimal LQR gain, we can utilize this gain in our C code and check if the result is the same as the previous part. Based on figure 23, the result is more or less the same as figure 22. The drone is oscillating around the reference pitch angle and it keeps its balance successfully.

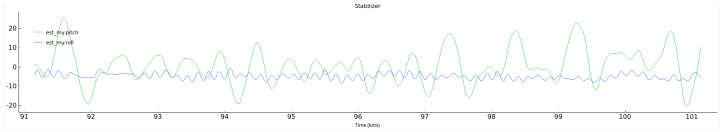

Figure 23. Estimated angles captured by Crazyflie client

VIII. Conclusions

Throughout this project, we have successfully completed all the different independent tasks required for achieving the desired control of the orientation of the drone. All the covered tasks include:

1) Calculating an estimate of the orientation using the complementary filter.
2) Modelling of the Crazyflie by deriving differential equations in order to describe the system's dynamics.
3) We have linearized the non-linear model in order to later apply a LQR.
4) We have designed an LQR to stabilize the drone's orientation around 0º in roll and pitch angles.
5) Finally, we have implemented our controller in simulation and on the real physical Crazyflie.

While working on the project, we encountered several ideas and challenges that are worth noticing. One of the most common issues that appeared continuously during the whole project is the units. The units affected the implementations of the complementary filter, open loop plant, and closed loop plant. It is crucial that all units are handled correctly (e.g. radians or degrees, thrust or PWM, etc), and this was something that was easy to miss.

Having said this, we managed to get matching all the units and right after that, we obtained great results from the implementation of the LQR. We tuned the weight matrices until we obtained a balance between a fast response and no overshoot. Three different scenarios were tested where impulse and step response analysis was carried out. From these simulations, we found some interesting observations which we think are worth discussing. If we increased the weightings of the states in the $\boldsymbol{Q}$ matrix we obtained a much faster response and lots of overshoot compared to low values of $\boldsymbol{Q}$. The opposite goes for the control cost matrix since decreasing $\boldsymbol{R}$ value resulted in a faster control without overshoot and increasing $\boldsymbol{R}$ resulted in a slower control with a slight overshoot.

Something else which is worth discussing is that the simulations doesn't represent the true world. Only because our closed loop simulations were promising doesn't mean that we will obtain the same results while carrying out the tests in the real Crazyflie. If we compare the evaluation of the LQR controller via simulation (both Matlab/Simulink and C code) and simulation on the real crazyflie, we can see that the LQR in the real scenario couldn't track the reference as good as it did in the simulations. This is, however, reasonable since our models and simulations doesn't represent the true physical crazyflie. With that said, the simulations of the real crazyflie which we managed to obtain in figure 22 and 23, are the ones which gave the best results among the different tunings which was explored.

IX. References

[1]Bitcraze, "Crazyflie 2.1," Bitcraze. [Online]. Available: https://www.bitcraze.io/products/crazyflie-2-1/. [Accessed: 22-May-2023]