Final Modelling, Simulation and Control Project

SSY191, Group 24

1st Carl Emvin
System, Control and Mechatronics
Chalmers University Of Technology
Gothenburgh, Sweden
emvin@student.chalmers.se

2nd Fikri Farhan Witjaksono
Mechanics and Maritime Science
Chalmers University of Technology
Gothenburg, Sweden
fikrif@student.chalmers.se

3rd Katell Hauss
Erasmus Student
Chalmers University of Technology
Gothenburg, Sweden
katellh@student.chalmers.se

4th Madhu Suresh
System Control and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden
madhus@student.chalmers.se

***Abstract*—In the project, we implemented a mathematical model and developed a LQ controller to the Quadrotor, i.e. Crazyflie. The flow of this project was to first make an estimation of orientation using complementary filter for the angular measurements, built a dynamic model using Simscape, obtain the linearized state space model and developed a LQR controller by minimizing the cost function and finally implemented everything in C using FreeRTOS for the scheduling.**

I. Introduction to control of Quadrocopters

The model-based approach is a system engineering approach that utilizes plant models to simulate the behaviour of the designed control system using softwares such as Matlab and Simulink before implementing it into the embedded system as well as validating and verifying the system behavior in real-time. This engineering approach is known to be more efficient in terms of the steps required to complete as well as faster in terms of development time. In order to implement a model-based design, we will need to have the dynamic model of the behavior of the system, a controller and scheduling procedure in real time [1].

The quadrotor that we want to model has a X-configuration, and the x-axis represent the roll, y-axis the pitch and z-axis the yaw. In the quadrotor motion modelling, to control individually the thrust, roll, pitch and yaw, we need to have the motors M1 and M3 spin counter-clockwise direction but clockwise for the motors M2 and M4, as in Figure 1.

In order to control the quadrotor, the rotational velocity in each motor will define the thrust that each motor is experiencing which will finally influence the position and attitude of the quadrotor. The thrust applied on each rotor could be defined mathematically as below, with ω for the rotational velocity $[\frac{rad}{s}]$ and b as the lift constant [kgm] :

$$T_i = b \cdot \omega_i^2 \ [N], \quad i \in 1, 2, 3, 4 \tag{1}$$

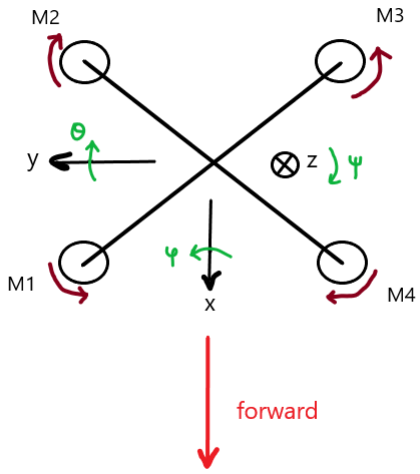

Fig. 1: Quadrotor Modelling

The control methods which are used for the regulation as well as trajectory tracking of the quadrotor are widely available in the literature. They ranged from the linear to non-linear methods (e.g. backstepping, sliding mode and feedback linearization) [2]. In our project, we will implement the control based on linearization of the non-linear dynamics of the quadrotor around a certain operating point.

Furthermore, to design the control of the system, we will need the accelerometer as well as the gyroscope as a measurement tool of the current position and attitude of the quadrotor. The gyroscope measures attitude in terms of rotational velocity and the accelerometer measures the external force acting on the quadrotor (acceleration and earth's gravity), so the inertial sensor will combine the measurements of these two tools.

However, the accelerometer is noisy for short periods of

time but accurate over long periods, and the gyroscope is accurate over short term, therefore we will need to have a filter (e.g. complementary filter) which could reliably estimate the attitude of the quadrotor for our control system.

II. Estimation of Orientation

First, we will start by introducing the I/O states. The inputs to the state estimator consists of acceleration in the directions of roll, pitch and yaw as well as the velocity. Where the gyroscope is responsible for the velocities and the accelerometer for the accelerations.

While it is of interest to estimate the roll pitch angle, it is sufficient to use a complementary filter in order to account for both fast and slow changes in angle. The complementary filter is composed of a low pass filter and a high pass filter in order to minimize the noise from tools, as well as an integral to obtain the orientation angle from the gyroscope velocity. Since our simulation model operates in the discrete time domain, it means that we have to construct our filter accordingly like in the Figure 2.

In the continuous domain, the Complementary Filter could mathematically be expressed as:

$$\theta(s) = G(s) \cdot \theta_a(s) + (1 - G(s)) \cdot \underbrace{\frac{1}{s} Y_g(s)}_{\theta_g} \tag{2}$$

Moreover, to discretize eq.(2), we will be using the Euler-backward method, which mathematically expressed as :

$$\dot{x}(t) \approx \frac{x(kh) - x((k-1)h)}{h}, \quad h \rightarrow \text{Sampling period} \tag{3}$$

The equation (2) will be transformed into the equation (4) as a result of the discretization.

$$\hat{\theta}_k = (1 - \gamma) \cdot \theta_{a,k}(s) + \gamma(\hat{\theta}_{k-1} + h \cdot y_{g,k}) \tag{4}$$

where,

TABLE I: The table represents the description of variables used in the eqn (4)

Variable	Description
γ	$\frac{\alpha}{\alpha+h} (with\ 0 < \gamma < 1)$
α	tuning parameter
h	sampling interval

According to the final discretized equation (4), the parameter γ acts as parameter that we had to choose as a tuning parameter. The range of γ is between 0 and 1. Therefore, in our tuning, we would have the following consequences :

- Small α (γ is close to 0) results in high cut-off frequency in the filter, hence imply higher significance of signal output from the accelerometer.
- Large α (γ is close to 1) results in low cut-off frequency in the filter, hence imply higher significance of signal output from the gyroscope.

As the gyroscope is integrated, its noise will be smaller than the noise from the accelerometer. Thus, we will choose to use a large α, so γ will be close to 1. In the figure 2, K represents this γ and it is equal to 0.98 [3].

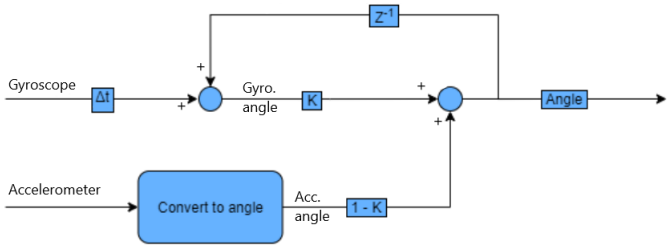

Fig. 2: Flowchart representing the quadrotor complementary filter

For estimating the angles from the accelerometer, we need to use all the forces on the readings.

$$^{B}f = {}^{B}R_w \left(\begin{bmatrix} {}^{w}a_x \\ {}^{w}a_y \\ {}^{w}a_z \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \right) \tag{5}$$

where ${}^{w}a_x$, ${}^{w}a_y$,${}^{w}a_z$ refers to the acceleration in the x,y,z direction in the world coordinates respectively, g refers to gravitational acceleration [$\approx 9.81 \frac{m}{s^2}$] and ${}^{B}R_w$ refers to coordinate transformation between world coordinates and body coordinates. The coordinate transformation matrix is found through the equation below, assuming the rotation order used is XYZ (roll-pitch-yaw).

$$^{B}R_w = R_{xyz} = R_x(\phi) \cdot R_y(\theta) \cdot R_z(\psi) \tag{6}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_1 & -s_1 \\ 0 & s_1 & c_1 \end{bmatrix} \cdot \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{bmatrix} \cdot \begin{bmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + s_1s_2c_3 & c_1c_3 - s_1s_2s_3 & -s_1c_2 \\ s_1s_3 - c_1s_2c_3 & s_1c_3 + c_1s_2s_3 & c_1c_2 \end{bmatrix}$$

where,

TABLE II: The table represents description of variables used in the above matrix

Variable	Description
c	Cosine function
s	Sine function
1	Roll angle
2	Pitch angle
3	Yaw angle

Assuming the body is at rest and the accelerometer is calibrated to give -1 when aligned with earth's gravitational field, the equation (5) becomes :

$$\begin{bmatrix} {}^{B}f_x \\ {}^{B}f_y \\ {}^{B}f_z \end{bmatrix} = R_{xyz} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \sin\theta \\ -\cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} \tag{7}$$

By solving the system of equations for roll and pitch we have :

$$\phi = \arctan\frac{A_y}{A_z} \quad \theta = \arctan\frac{-A_x}{||A_y + A_z||} \tag{8}$$

The result from Figure 4 indicates that the integrated gyro estimation result will drift away from the supposed orientation of the system as predicated from Figure 3. This is due to the fact that integration of the gyro will result in errors which are accumulated over time which is commonly called as integration drift. On the other hand, the complementary filter result seems to be more in line with our intuitive prediction on how the orientation should behave. This is evident since we choose to trust the accelerometer more than the gyro in the complementary filter hence putting more weight to the accelerometer input.

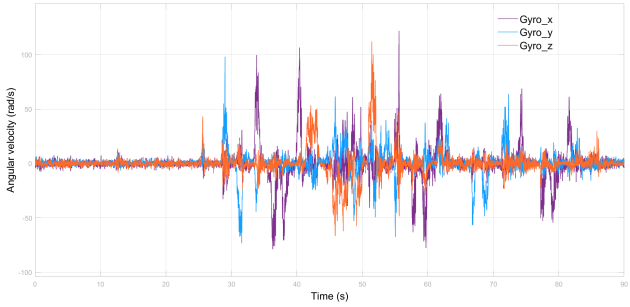

Fig. 3: Measured Gyro Rotational Velocity

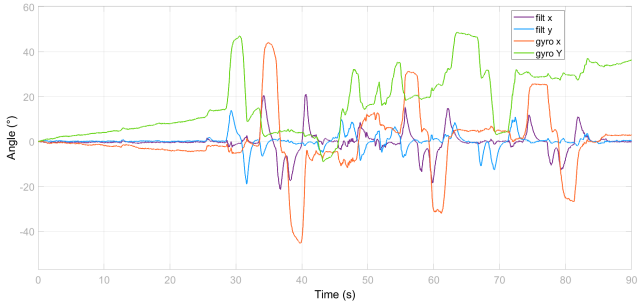

Fig. 4: Gyro Integration vs Complementary Filter

III. Plant Modelling

Plant modeling of the Quadrotor is done using Simscape toolbox where we need to list the equations required in modeling of the system. These equations will have thrust of 4 motors as an input and provide position, velocity, acceleration, angle and angular velocity as output which is further used to control the quadrotor. These equations are differentiated into linear and angular equations of motion. The linear equations of motion are defined to obtain position, velocity and acceleration using Newton's law under world co-ordinate frame, given by

$$m \cdot \dot{v} = -\begin{bmatrix} 0 \\ 0 \\ m \cdot g \end{bmatrix} + {}^{B}R_w \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} - B \cdot v \tag{9}$$

$$\dot{v} = a$$

$$\dot{s} = v$$

where T is total thrust generated by the propellers, B is the aerodynamic friction experienced by the quadrotor which is neglected in this case, R_b^w is the coordinate transformation matrix from the body coordinates to world coordinates, m is mass of quadrotor and a is linear acceleration. The angular equation of motion is defined to obtain angles and angular velocity by using the equation given below,

$$J \cdot \dot{\omega} = -\omega \times J \cdot \omega + \tau \tag{10}$$

$$\tau = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$$

where

TABLE III: The table represents the description of variables used in the eqn. (10)

Variable	Description
ω	Angular velocity of the quadrotor
τ	Torque applied to Quadrotor in 3D-space
τ_x	Torque applied to airframes in x-direction
τ_y	Torque applied to airframes in y-direction
τ_z	Torque applied to airframes in z-direction

Thrust applied to rotor in x, y and z directions are given by the following equations,

$$T_x^b = d * cos(\frac{\pi}{4}) \times (\tau_3 + \tau_4 - \tau_1 - \tau_2)$$

$$T_y^b = d * cos(\frac{\pi}{4}) \times (\tau_2 + \tau_3 - \tau_1 - \tau_4)$$

The aerodynamic drag generated by propeller movement applies a reaction torque on the frame about the propeller shaft in the opposite direction of rotation as,

$$T_z^b = b * \times(-\tau_1 + \tau_2 - \tau_3 + \tau_4)$$

where b is aerodynamic drag constant and $\tau_1, \tau_2, \tau_3, \tau_4$ is the torque applied on rotor 1,2,3,4 respectively. Thus the only part to define is euler angles which can be obtained by coupling angular velocity in co-ordinates to the euler angle derivatives shown in the eqn. (11) as done in the complimentary filter.

However, theoretically, this is not fully accurate due to the fact that order of rotation will influence the derivation.

$$\Theta = \int \dot{\Theta} * dt \tag{11}$$

Where Θ is Euler's angle, the following are the representation of equations implemented in the Simscape tool box.

```
%Rotation matrix,Body to World co-ordinate frame,
Rz = [cos(angle(3)), -sin(angle(3)), 0; sin(angle(3)), cos(angle(3)), 0; 0, 0, 1];
Ry = [cos(angle(2)), 0, sin(angle(2)); 0, 1, 0; -sin(angle(2)), 0, cos(angle(2))];
Rx = [1, 0, 0; 0, cos(angle(1)), -sin(angle(1)); 0, sin(angle(1)), cos(angle(1))];
wRb = Rx*Ry*Rz;

in
%Dynamics in world co-ordinate system, To find the outputs,
[0;0;-m*g] + wRb * [0;0;sum(ctrl)];

%Velocity,
v.der==a;
pos.der==v;

%Angular velocity,
J * w.der == cross(-w,J*w) + T ;
angle.der==w;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

Fig. 5: Implemented Simscape Plant Dynamics Equation

IV. Linearization of Plant

As we have a non-linear model, in order to build a good controller strategy, the system has to be linearized first and then a linear control technique has to be applied to it. Generally, The LTI state model to be linearized is expressed as,

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

where,

$$A = \frac{\partial f}{\partial x}|_{x_0,u_0} \quad B = \frac{\partial f}{\partial u}|_{x_0,u_0} \tag{12}$$

Here, we choose the following as our states to represent a system,

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

where

TABLE IV: The table represents the description of variables used in the above matrix

Variable	Description
ϕ	Roll angle
θ	Pitch angle
$\dot{\phi}$	Roll angle rate
$\dot{\theta}$	Pitch angle rate
$\dot{\psi}$	Yaw angle rate

The system is linearized to the point $x_0 = 0$, by assuming that the quadrotor will be having small oscillation in its initial states, thus we could assume that the sine and cosine function approximation for the linearization is approximately unity [2]. Thus by using Linearization method, as in eqn. (12), the dynamic matrix A and input matrix B of an continous plan can be expresses as,

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

$$B = \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ -\frac{d}{\sqrt{2}\mathcal{J}_x} & -\frac{d}{\sqrt{2}\mathcal{J}_x} & \frac{d}{\sqrt{2}\mathcal{J}_x} & \frac{d}{\sqrt{2}\mathcal{J}_x}\\ -\frac{d}{\sqrt{2}\mathcal{J}_y} & \frac{d}{\sqrt{2}\mathcal{J}_y} & \frac{d}{\sqrt{2}\mathcal{J}_y} & -\frac{d}{\sqrt{2}\mathcal{J}_y}\\ -\frac{k}{\mathcal{J}_z C_L} & \frac{k}{\mathcal{J}_z C_L} & -\frac{k}{\mathcal{J}_z C_L} & \frac{k}{\mathcal{J}_z C_L}\end{bmatrix}$$

where d is the arm length of the quadcopter, J_x, J_y, J_z is the moment of inertia in the x,y,z direction respectively and C_L is the lift constant of the quadcopter. Next, the above system is converted to discrete time with sampling time T_s = 0.01 sec in order to implement it in the controller. The number T_s comes from the fact that in our Simulink implementation, the model only works well with that specific sampling time.

$$x(k+1) = A_d x(k) + B_d u(k)$$

where,

$$A_d = e^{AT_s}$$

$$B_d = \int_0^{T_s} (e^{At} * dt) \times B$$

Thus by using the above equations, following are the A and B matrices obtained for Discrete time system.

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

$$B_d = \begin{bmatrix}-0 & -0 & 0 & 0\\ -0 & 0 & 0 & -0\\ -\frac{\sqrt{2}*d}{2e4*J_x} & -\frac{\sqrt{2}*d}{2e4*J_x} & \frac{\sqrt{2}*d}{2e4*J_x} & \frac{\sqrt{2}*d}{2e4*J_x}\\ -\frac{\sqrt{2}*d}{2e4*J_y} & \frac{\sqrt{2}*d}{2e4*J_y} & \frac{\sqrt{2}*d}{2e4*J_y} & -\frac{\sqrt{2}*d}{2e4*J_y}\\ -\frac{-k}{1e4*J_z b} & \frac{-k}{1e4*J_z b} & -\frac{-k}{1e4*J_z b} & \frac{-k}{1e4*J_z b}\end{bmatrix}$$

Thus from matrix A, We can observe that the system is affected mainly by angular velocity of Roll and Pitch. By using the above matrices in the control strategy, we can able to converge the deviations to the origin. However, as we have taken linearization point as 0, if the system deviates too far from the point, then the control system may become unstable.

V. Design of Linear Quadratic Controller

The controller is to be designed by the optimal feedback law $u = -k\hat{x}$ where the state estimate $\hat{x}$ is composed by the complementary filter as well as the gyroscope readings together with the reference states. Where one has the ability

to control the quadcopter with regards to angular velocity and tilt angle. If one were to control the quadcopter with regards to its path, one would have to estimate the orientation angles required to arrive with a certain pose. It was sufficient enough to just follow a reference tilt angle for this project. In order to grasp the concept of lqr feedback, one can describe our model using following flow chart:

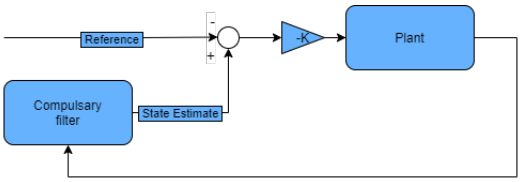

Fig. 6: LQR feedback controller concept design

Where the Linear Quadratic Regulator (LQR) gain K is the solution to eq. (13). With A_d and B_d matrices defined by the discrete LTI model previously determined. The disturbance intensity matrix R is said to be zero since we work in a simulation environment without added noise.

$$K = (B_d^T P B_d + R)^{-1}(B_d^T P A_d) \tag{13}$$

With P being the solution to the discrete time Riccati equation [4].

$$P = A_d^T P A_d - (A_d^T P B_d)(R + B_d^T P B_d)^{-1}(B^T P A_d) + Q$$

and the feedback control input is defined as

$$u(k) = -K\hat{x}(k) \tag{14}$$

in which the controller minimizes the cost function defined as:

$$J = \sum_{k=0}^{\infty}(x(k)^T Q x(k) + u(k)^T R u(k) + 2x(k)^T N u(k))$$

Where Q and R is square matrices. Given Q and R as diagonal matrices, we can utilize the Bryson's rule. which is defined as:

$$\bar{Q}_{ii} = \frac{1}{\text{maximum acceptable value of } x_i^2}$$
$$\bar{R}_{jj} = \frac{1}{\text{maximum acceptable value of } u_j^2}$$

Assuming zero state cross dependence, we have a diagonal Q-matrix. With respect to the Bryson's rule, since we have the max angle as $25\,\deg$ and the maximum angular velocity as $100\,\deg/s$ as our reference input value range that we want, which consequently gives us the following Q matrix:

$$Q = \begin{bmatrix} 1/25^2 & 0 & 0 & 0 & 0 \\ 0 & 1/25^2 & 0 & 0 & 0 \\ 0 & 0 & 1/100^2 & 0 & 0 \\ 0 & 0 & 0 & 1/100^2 & 0 \\ 0 & 0 & 0 & 0 & 1/100^2 \end{bmatrix}$$

For the weighting matrix R we can use the same analogy. It is reasonable to assume that each motor is independent with regards to torque. With a max torque defined by the maximum motor signal 65536 and the *motor signal to torque* ratio $\frac{0.06g}{65536*4}$ resulting in a R matrix as:

$$R = 65536\frac{0.06g}{65536 * 4}I_4 \tag{15}$$

Where I_4 is the 4 by 4 identity matrix. The cross dependence for the state-input is said to be zero resulting in:

$$N = 0$$

With Q, R and N we end up with K as:

$$K = 10^{-3}\begin{bmatrix} -2.94 & -2.94 & -1.03 & -1.14 & -0.74 \\ -2.94 & 2.94 & -1.03 & 1.14 & .74 \\ 2.94 & 2.94 & 1.03 & 1.14 & -0.74 \\ 2.94 & -2.94 & 1.03 & -1.14 & 0.74 \end{bmatrix}$$

For convenience reasons we choose to use the matlab command, *lqrd()* which calculates the discrete lqr gain for our LTI system using the same procedure as been illustrated above.

VI. Evaluation of the control design

To evaluate the performance of our lqr controller we made a few tests. First we start by checking the roll control using a impulse response test. The pulse start at 3 seconds and dies at 5 with the amplitude of 1 degree. By the looks of figure (7) we can conclude that the quadcopter is certainly converging to the correct value even though it is a bit slow, implying that we have an over-damped system.

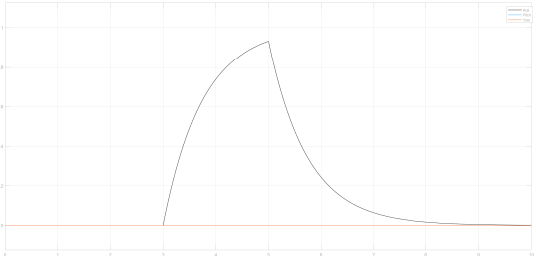

Fig. 7: Impulse response test for roll with amplitude of 1 degree

To validate that the controller works in the pitch direction as well, we choose to do a step analysis. With a step of 1 degree starting at 3 seconds one can do the same analysis for the pitch control as one did for the roll control. The controller is a bit slow implying under-damped system.

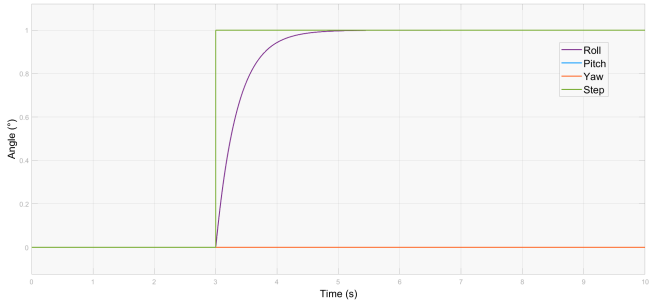

Fig. 8: Step response analysis for Pitch with amplitude of 1 degree

To speed up the controller we can make a dead-beat reconstruction. The dead-beat controller has all its poles in the origin, forcing the system to converge within n samples, if n is the system degree. Using the matlab command *place()* we were not able to place all the poles in the origin because of rank related problems. Instead we choose to place a double pole in 0 and a triple pole in 0.1. The controller should have almost dead-beat behaviour and be good enough for measure. The dead beat gain matrix we end up with is presented below.

$$K_{db} = 10^4 \begin{bmatrix} -0.79 & -1.18 & -0.00 & -0.00 & -0.00 \\ -0.79 & 1.18 & -0.00 & 0.00 & 0.00 \\ 0.79 & 1.18 & 0.00 & 0.00 & -0.00 \\ 0.79 & -1.18 & 0.00 & -0.00 & 0.00 \end{bmatrix}$$

The dead-beat controller is obviously responding much faster than the lqr controller but with the cost of a bumpy ride. The quadcopter overshoots with less than 1 degree, and bounces back and forth for a few samples before converging.

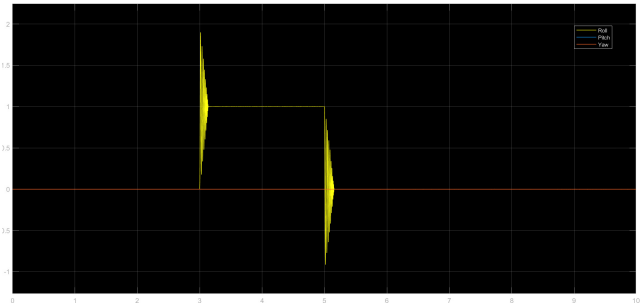

Fig. 9: Deadbeat-controller Roll impulse response test.

Depending on the application, one might prefer one over the other. One need to take in consideration as well that the dead-beat controller only performs well under very controlled circumstances, where noise are close to absent. But the point of the dead-beat controller still remains regarding the LQR controller being to slow. One could definitely tune Q and R in order to archive a faster behaviour if that is desired.

VII. Implementation in C

The implementation of a Quadrotor control as done before with Matlab and Simulink is carried out in C programming. This is specifically divided into 2 tasks,

- Implementation of complementary filter in C.
- Implementation of controller in C.

For the implementation, we utilized the operating system freeRTOS which are used to manage the threads and processes that we have in the system with respect to the processor that we have in the memory (multitasking through scheduling). It is also used for communication between hardwares through the TCP/IP communication protocol. We also used the Simulink coder to convert the Simulink blocks which represented our filter and controller, into general C programs. The general nature of the output of Simulink coder which does not fit perfectly well with the provided skeleton code in C, has required us to create our own C code implementation while using the coder output as our general framework reference.

Since the scheduling of tasks in real-time control environment is a hard requirement due to limited processing power, we could done this through the utilization of semaphores which are already created by the given skeleton code. Our work in the project is specifically to design an algorithm where these semaphores are used to synchronize our concurrent tasks. Generally, it is recommended to use the controller sampling frequency of 100 Hz while using higher and lower sampling frequency for the reference generator and filter respectively.

A. Complementary Filter

Initially, our implementation starts with our complementary filter which is done in `filter.c`. The complementary filter semaphore is added to the each threads which is integral to all the sub-tasks well being, by using a signaling mechanism (e.g. binary semaphores) whose main function is to prevent race condition. We have also implemented the period of each task through the `vTaskDelayUntil()` by defining the task period as an input to the function. The algorithm pseudo-code could be expressed as below in Algorithm 1

Algorithm 1: Complementary filter pseudocode

Result: estimate
Initialize:Sensor Semaphore Handle, Estimation Semaphore Handle, **estimate** array definition ;
while `true` **do**
  Get tick count for **vTaskDelayUntil()**
  Take Semaphore for sensor readings
  Read the sensor data
  Give Semaphore for sensor readings
  Take Semaphore for angle estimation
  Estimate the angle using complementary filter
  Write the result to the memory using the pointer to array **estimate**
  Give Semaphore for angle estimation
  Sleep 8 ms to make this task run at 125Hz using **vTaskDelayUntil()**
end

Observing the algorithm 1, our complementary filter will essentially function by taking the value from the sensor data reading (i.e. accelerometer and gyro) and estimate the orientation angle in real-time. The real-time estimation result will then be written to the **estimation** global state variable. This imply that whenever we got a condition in our sensor reading thread where it is deprived from the memory resource due to some bug or errors in our code, the estimation of the angle from the complementary filter will be impossible to calculate. That is why we only put semaphores on the sub-tasks which are of highest criticalities.

B. LQR Controller

In the LQR controller, we first try to take the readings from the sensor, the reference generator input and the estimate from `filter.c`. Then using eqn. (14) as defined in Chapter V to define the feedback control motor input which utilized the LQR gain obtained in Simulink model K, the states which were calculated in real-time, the thrust converter constant and the base thrust. This operation could mathematically be expressed as below

$$u(k) = (-Kx(k) \cdot R) + BT \tag{16}$$

where BT is the Base Thrust which is equal to 30000 ($\approx 45.78\%$ of maximum torque) and R is the motors signal to torque ratio defined as in eqn. (15) in Chapter V. Then, the motor input calculation result were passed into the C program translation of the saturate block in Simulink model which is basically giving us maximum torque value (e.g. Motor Torque $=$ 65536) and minimum torque value (e.g. Motor Torque $=$ 0), if the torque is more than the maximum and less than its minimum value respectively. Finally, the output of this saturate function will be written in the `motor` global state variable.

Algorithm 2: LQR Controller pseudocode

```
Result: motors
Initialize:Reference Generator Semaphore Handle,
  r_rpdy, Estimation Semaphore Handle, estimate,
  Sensor Semaphore Handle, acc_data,
  gyro_data, Controller Gain (K), Motors
  Semaphore Handle, motors;
while true do
    Get tick count for vTaskDelayUntil()
    Take Semaphore for sensor readings
    Read the sensor data
    Give Semaphore for sensor readings
    Take Semaphore for reference generator
    Read the reference value array from memory
    Give Semaphore for reference generator
    Take Semaphore for angle estimation
    Read the estimated angle from memory in estimate
    Give Semaphore for angle estimation
    Take Semaphore for input motors
    Calculate motor input using LQR controller
    Write the result to the memory using the pointer to
      array motors
    Give Semaphore for input motors
    Sleep 10 ms to make this task run at 100Hz
end
```

In the **Algorithm 2**, we are using the semaphore as a memory resource only for the sub-tasks that are of critical to be processed in time. For example, the reading and writing of the values from/to the memory. The rest of the sub-tasks that are of less memory consuming (e.g. the simple substraction or addition mathematical operation, definitions) were not assigned semaphores, whereas the sub-tasks that are of high criticality (e.g. calculate motor input using LQR controller using for loop) and requires higher computational power, were assigned semaphores. This is done to ensure that the controller has the lowest latency in its operation as well as to avoid racing conditions since we are having limited memory resources.

The implemented C code gave a satisfactory results for roll angle as can be shown in figure 10 below. It could be seen that the estimated angle could track the given reference value to certain degree. There seems to be latency in the controller that caused some kind of bias in the estimated angle responses. It could possibly caused by some bias in our scheduler. This problem could possibly be solved by using the Response Time Analysis (RTA) which will make sure the sub-task is controlled perfectly on time.

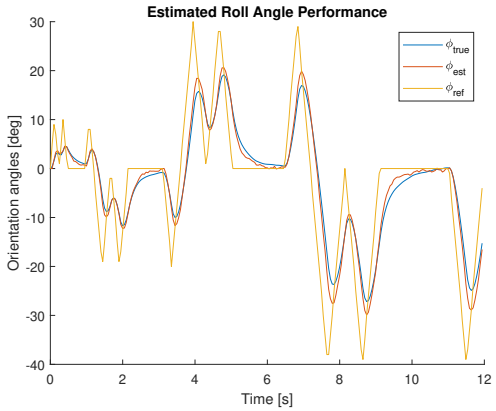

Fig. 10: Roll Angle Estimation with LQR Controller Performance

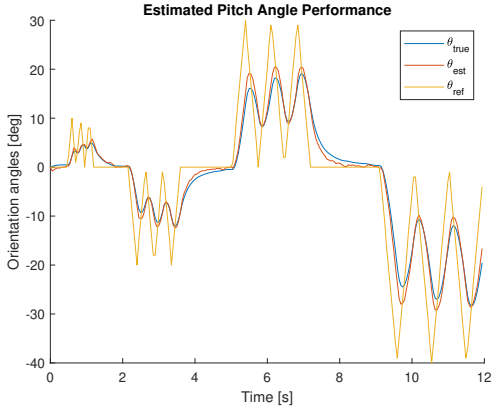

Fig. 11: Pitch Angle Estimation with LQR Controller Performance

The estimated pitch is shown in Figure 11 where it behaves similarly with the roll angle estimation. The motor input signal performance seems to hover around the initially set base thrust ($30000 \approx 45\%$) as shown in Figure 12 above.

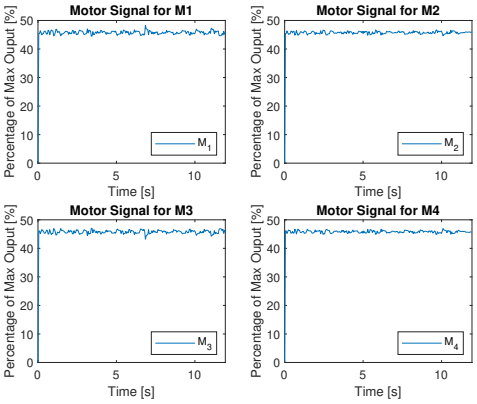

Fig. 12: Motor Input Signal Performance

VIII. Conclusions

In this project work, firstly, we are modelling the non-linear dynamics of the plant according to Chapter III. These equations are then be linearized around zero and converted to discrete time equation as explained in Chapter IV. In order to create the controller, the estimation of current state of orientation of the quadrotor needs to be obtained through a filter. The method used was complementary filter as explained in Chapter II. This filter will then be the basis of one of our inputs, apart from the Thrust applied to each rotor and reference input, for our LQR controller design as explained in Chapter III. In the designed LQR controller, we chose R-Covariance matrix to be larger than the Q-Covariance matrix while the correlation matrix N is chosen to be zero since the Torque dynamics is independent from the state dynamics. Then, the result was that the quadcopter roll angle output was converging to the desired value of reference input slowly due to the fact the system we used were an over-damped system. Similarly, the performance of the pitch angle output was good, converging to our desired input reference. In our C-implementation as explained in Chapter VII, we got a stable output response result.

References

[1] K. Åkesson, Model-Based Development of Cyber-Physical System (MB-DCPS) Lecture Notes, Chalmers Univ., 2021.

[2] F. Sabatino, "Quadrotor control: modeling, nonlinear control design, and simulation", Master Thesis, Kungliga Tekniska Högskolan, 2015.

[3] B. Douglas, "Drone Control and the Complementary Filter", [Video]. *Youtube*, Nov. 5, 2018. Available : https://www.youtube.com/watch?v=whSw42XddsU

[4] G. Torkel and L. Lennart, *Control theory: Multivariable and Nonlinear Methods*, 2010. (London : CRC Press, 2000.)