Controlling a quadcopter SSY191 - Group 9

Emil Erikmats
Systems, Control and Mechatronics (MPSYS)
Chalmers University of Technology
Gothenburg, Sweden
emieri@student.chalmers.se

Yuhui Bi
Systems, Control and Mechatronics (MPSYS)
Chalmers University of Technology
Gothenburg, Sweden
yuhuib@student.chalmers.se

Alan Ali Doosti
Systems, Control and Mechatronics (MPSYS)
Chalmers University of Technology
Gothenburg, Sweden
payama@student.chalmers.se

V D V Vamsi Krishna N
Systems, Control and Mechatronics (MPSYS)
Chalmers University of Technology
Gothenburg, Sweden
dwaraka@student.chalmers.se

***Abstract*—The aim of this report is to describe the processes on how to develop control algorithms and implementation by C for a quadcopter. An amalgamation of five sub projects within the SSY191 course culminated into necessary steps on how to derive equations needed for designing an LQR controller for a drone and relative methods to implement control over the drone in the C programming language. Acausal modelling displays a strength when mechatronic systems are being designed which is proven with a drone in this case. With the reliance of two sensors, the importance of a complementary filter is emphasised. This report sheds light on the way a quad-copter has been modelled, simulated and controlled in a software environment such as MATLAB®, and later implemented the model in a real-time quad-copter using techniques from real time operating system such as multi-thread programming.**

***Index Terms*—LQR controller, quadcopter, complementary filter, acausal modelling, roll, pitch, yaw, semaphores, RTOS**

I. Introduction

The development of drones has skyrocketed the last couple of years, leading to new areas of use and improved user experience of already existing ones. The control of a drone is arguably the most important part of the user experience. This report thus aims to clarify how the control laws of a drone, specifically the Crazyflie 2.1 can be developed and what steps have to be taken to get there.

II. The work flow basics and Model based design approach

Before decisions are made on how to control the drone, a few steps have to be taken. The first part of the project involves an estimation of the orientation of the drone. Since the drone is equipped with a gyroscope (from here on called *gyro*) and an accelerometer (from here on called *accel*), the data from both sensors can be combined to minimize the error. In other words a sensor fusion technique is applied in the form of a complementary filter.

The rest of the development of the controller was made according to a model-based design. The first step in a model-based design is the plant modelling which in this case is based on first-principles modelling, meaning the behaviour of the drone is modelled by deriving differential equations from mechanical relationships. The rest of the steps in a model-based design seen in Fig. 1 are not discussed under this section.

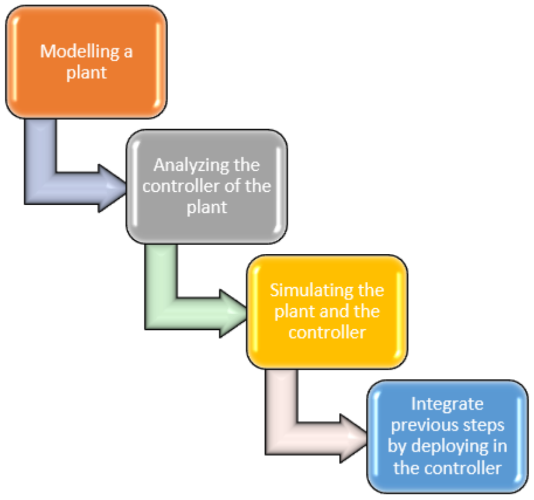
Modelling a plant
Analyzing the controller of the plant
Simulating the plant and the controller
Integrate previous steps by deploying in the controller

Fig. 1. The workflow of a model based design approach

Since the system and equations are non-linear, a linearization had to be made to be able to control the drone with a Linear Quadratic Regulator (LQR).

After that, the LQR was implemented and evaluated. The pole placement of a controller for a system as fast as this one is extra fussy since a small displacement can make a noticeable difference on the robustness. The LQR offers a relatively fast way to derive relatively accurate pole placements to control a system where multiple states are being controlled compared to other methods.

After this, the outcome of the project will be discussed and conclusions will be drawn on the performance and why it behaves as it does.

This is followed by a brief introduction to Real Time Operating Systems (RTOS) and how the concepts of this were used to implement the LQR in the C programming language.

III. Estimation of orientation

The first part is the estimation of the orientation of the drone. As mentioned before, the sensor data from the two sensors do contain errors, partly because of noise (for both sensors) and partly because of a numerical integration error. Since the *gyro* gives the angular rate, it has to be integrated to get the angle. The integration is however only approximate since the measurements are taken at discrete time instances. This means information is lost, leading to an accumulated error over time. The *gyro* is therefore not as trustworthy in the long run. Nonetheless, the integration also attenuates the disturbances making it more reliable in the short term.

The *accel* on the other hand doesn't have to be integrated. The angle is instead estimated by the relative difference between the measured accelerations. Since only roll and pitch angles are affected by gravity according to the configuration as the one seen in Fig. 5, only these can be estimated using the *accel* readings. The estimations were derived by first finding the intrinsic rotation matrix from body to world frame ${}^{w}R_B$:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\alpha & -s_\alpha \\ 0 & s_\alpha & c_\alpha \end{bmatrix}$$

$$R_y(\alpha) = \begin{bmatrix} c_\alpha & 0 & s_\alpha \\ 0 & 1 & 0 \\ -s_\alpha & 0 & c_\alpha \end{bmatrix}$$

$$R_z(\alpha) = \begin{bmatrix} c_\alpha - & s_\alpha & 0 \\ s_\alpha & c_\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{w}R_B = R_{zyx}(\psi,\theta,\phi) = R_z R_y R_x = \\ = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\phi s_\theta - c_\phi s_\psi & s_\phi s_\psi + c_\phi c_\psi s_\theta \\ s_\psi c_\theta & c_\phi c_\psi + s_\phi s_\psi s_\theta & c_\phi s_\psi s_\theta - c_\psi s_\theta \\ -s_\theta & c_\theta s_\phi & c_\phi c_\theta \end{bmatrix}$$

where *c* stands for cosine, *s* for sine, "sub-φ" means "of φ" , "sub-θ" means "of θ" and "sub-ψ" means "of ψ". Some elements were identified to hold enough information to estimate the roll (φ) and pitch (θ) and were picked out as follows:

$$f = {}^{w}R_B^T \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -s_2 \\ c_2 s_1 \\ c_2 c_1 \end{bmatrix} \tag{1}$$

The expressions in f in (1), which are measured by the *accel*, can then be used to derive expressions for the angles. Estimation of the pitch can be derived from the following expressions:

$$f_1 = -\sin(\theta) \tag{2}$$
$$\theta = \arcsin(-f_1) = -\arcsin(f_1) \tag{3}$$

However the scaling factor from the *accel* readings have to be considered, therefore another method where the constants cancel out was used:

$$\theta = \arctan(-f_1, \sqrt{f_2^2 + f_3^2}) \tag{4}$$

Then the roll was estimated as:

$$\frac{f_2}{f_3} = \frac{c_2 s_1}{c_2 c_1} \tag{5}$$
$$= \frac{s_1}{c_1} \tag{6}$$
$$= \tan(\varphi) \tag{7}$$
$$\Rightarrow \varphi = \arctan(\frac{f_2}{f_3}) \tag{8}$$

By conversion the noise doesn't get attenuated as in the case for the integration of the *gyro* readings. Therefore the angles derived from the *accel* readings are less trustworthy in the short term but since there's no numerical drift occurring, they are more trustworthy in the long run.

As mentioned previously, the angles were then combined through a complementary filter, which was designed around the discretized Euler backward method:

$$\hat{\theta}_k = (1-\gamma)\theta_{a,k} + \gamma(\hat{\theta}_{k-1} + h \cdot y_{g,k}) \tag{9}$$

where, h is the sampling time, γ = α/(α+h), α is the cut-off frequency, $\theta_{a,k}$ is the accelerometer readings at time instance k and $y_{g,k}$ is angle estimation using gyroscope at the same time instance. The output from the filter is a percentage of one of the calculated angles and the complementary percentage (percentage left to reach 100 %) for the other one.

The complementary filter setup can be seen in Fig. 2-3. As seen in the figures and the equations above, only roll and pitch are filtered. Since yaw cannot be measured by the *accel*, due to the fact that the gravitational pull points in the negative z-direction, i.e. it is parallel to the yaw-axis when the drone is at its equilibrium point. This means the measured acceleration doesn't change noticeably with different yaw angles. Ultimately, the state vector was chosen to be $[\phi\ \ \omega_\phi\ \ \theta\ \ \omega_\theta\ \ \omega_\psi]^T$.

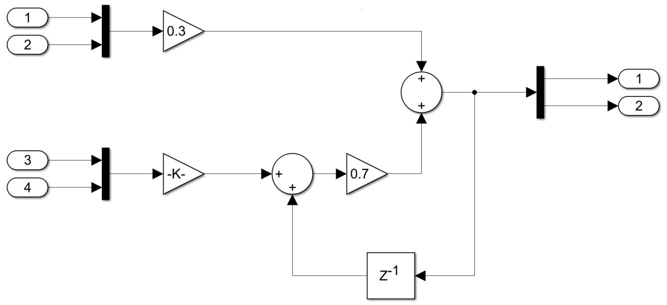

Fig. 2. Complementary filter combining the angles calculated from the *gyro*'s and *accel*'s readings. The K gain is the sample interval ($K = 0.01$) making a zero order hold integration. Input 1 and 2 are roll ϕ and pitch θ respectively from the *accel* and input 3 and 4 are the equivalent angles from the *gyro*.

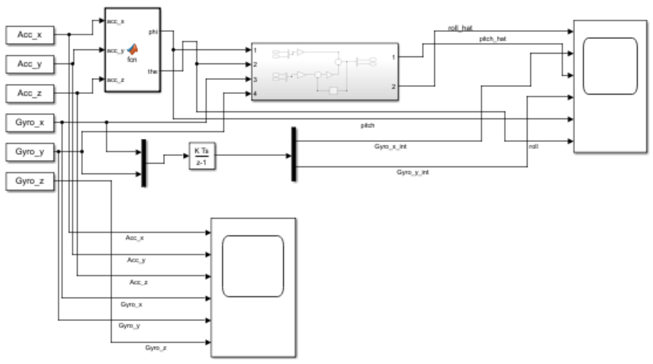

Fig. 3. Complementary filter together with accel, acceleration to degrees conversion and gyro readings.

Worth pointing out is that the complementary filter was tested in a simulation with noisy roll and pitch readings. The output can be seen in Fig. 4 and was much less noisy, and follows the estimated values fairly well which confirms the advantage of implementing such a filter.

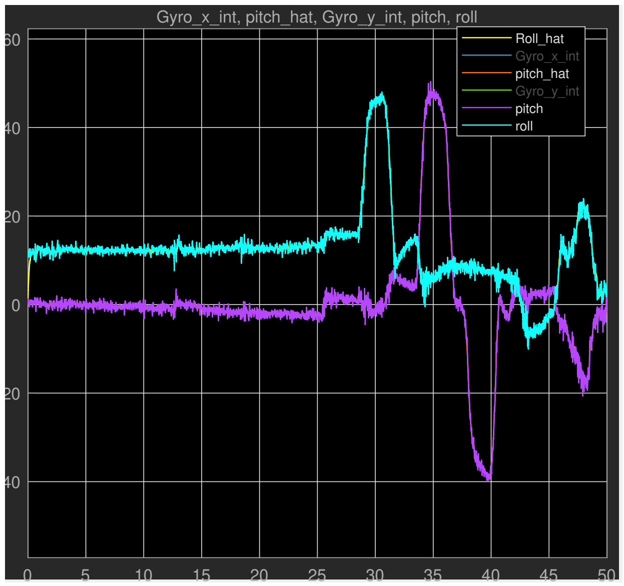

Fig. 4. Roll and pitch angles estimated by the complementary filter applied on flight data.

IV. Plant modelling

The end goal is to be able to control the quadcopter's orientation. Therefore, equations describing the change in position and angle with respect to the world frame were derived. The equations expressing the drone's position / velocity and angle / rotational speed can be seen in equation (10)-(16). They were declared in Simscape®, which is a programming language for equation based modelling. One special feature with this programming language is that it allows for acausal modelling, meaning that the compiler can do the necessary algebraic manipulations to the equations in order to get the desired output. This decreases the number of lines of code needed for the expressions which in turn decreases the time needed for modelling and the complexity of the model.

$$R = \begin{bmatrix} 1 & 0 & s_2 \\ 0 & c_3 & -s_3 c_2 \\ 0 & s_3 & c_3 c_2 \end{bmatrix} \tag{10}$$

$$^{w}R_B = \begin{bmatrix} c_3 c_2 & c_3 s_2 s_1 - c_1 s_3 & s_3 s_1 + c_3 c_1 s_2 \\ c_2 s_3 & c_3 c_1 + s_3 s_2 s_1 & c_1 s_3 s_2 - c_3 s_1 \\ -s_2 & c_2 s_1 & c_2 c_1 \end{bmatrix} \tag{11}$$

$$\omega = R \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{12}$$

$$\Gamma = \begin{bmatrix} \gamma_x \\ \gamma_y \\ \gamma_z \end{bmatrix} = \begin{bmatrix} \frac{d}{\sqrt{2}}(T_3 + T_4 - T_1 - T_2) \\ \frac{d}{\sqrt{2}}(T_2 + T_3 - T_1 - T_4) \\ \frac{k}{b}(T_2 + T_4 - T_1 - T_3) \end{bmatrix} \tag{13}$$

$$J = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{14}$$

$$J\dot{\omega} = -\omega \times J\omega + \Gamma \tag{15}$$

$$ma = -\begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} +^{w} R_B \begin{bmatrix} 0 \\ 0 \\ T_1 + T_2 + T_3 + T_4 \end{bmatrix} - \mu_{\text{airfriction}} v \tag{16}$$

J is the moment of inertia matrix. R is the transformation matrix converting $\dot{\phi}, \dot{\theta}, \dot{\psi}$ to ω (angular velocity).$^{w}R_B$ is the rotation matrix converting from the body frame to the world frame. The vector Γ (torque) was derived assuming that the the rotors were spinning in the directions according to Fig. 5. This means that a positive roll is obtained by increasing the thrust from M3 and M4 while decreasing on M1 and M2. An increase in pitch means M2 and M3 have to increase while M1 and M4 need to decrease.

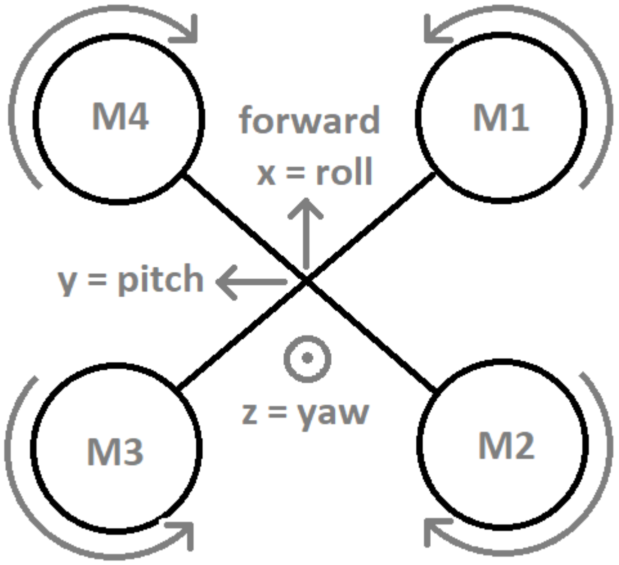

Fig. 5. Visualization of the quadcopter seen from above. x or roll axis pointing forward, y or pitch axis pointing to the left and z or yaw axis pointing out of the screen i.e. upwards when the drone is stabilized and parallel to the ground.

To change the yaw angle one must first understand how the side forces from the propellers generate torque around the center of the drone: The explanation lies in the difference in distance between two blades on a rotor and the z-axis. If a rotor blade is closer to the center, the lever gets smaller while the horizontal force component (*hfc*) remains the same. This means the *hfc* from the other blade on the same rotor, being an equal but opposite directed has a bigger lever and therefore generates more torque around the z-axis. The *hfc* is a friction component meaning its direction is the opposite from the speed of the rotor blade. So to increase the yaw, the motors rotating in the negative direction should therefore be increased and the other two decreased. Once again following the schematic diagram from Fig. 5, an increase in yaw is obtained by increasing the thrust from M2 and M4 while decreasing on M1 and M3. This is all reflected in the equation for Γ above.

The equations were then used to derive expressions for a state space model. The output of the model is what is measured. Since the measurement model was already known, only a motion model was derived. The results can be seen in equation (17)

$$\begin{cases} \ddot{\phi} &= \frac{\gamma_x + I_y \dot{\psi}\dot{\theta} - I_z \dot{\psi}\dot{\theta}}{I_x} \\ \ddot{\theta} &= \frac{\gamma_y + I_z \dot{\phi}\dot{\psi} - I_x \dot{\phi}\dot{\psi}}{I_y} \\ \ddot{\psi} &= \frac{\gamma_z + I_x \dot{\phi}\dot{\theta} - I_y \dot{\phi}\dot{\theta}}{I_z} \end{cases} \tag{17}$$

V. Linearization of plant

For this project, the chosen controller is a Linear Quadratic Controller (LQR, which will be explained in next chapter). However, to be able to implement an LQR, a linear system is needed. Upon looking at the expressions for the state space model in (17), several non-linearities are observed. The model was therefore linearized around a working point, in this case an equilibrium point when the body is at rest was selected $[0\ 0\ 0\ 0\ 0]^T$. The principle of linearisation can be found in (18)

$$\dot{x} = \frac{\text{df}}{\text{dx}}|_{x0,u0}\ \Delta x + \frac{\text{df}}{du}|_{x0,u0}\ \Delta u = A\ x + B\ u \tag{18}$$

Where x and u can be found below in (19):

$$x = \begin{bmatrix} \phi \\ \omega_x \\ \theta \\ \omega_y \\ \omega_z \end{bmatrix},\ u = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{19}$$

The resulting A and B matrices were the following:

$$A = \begin{bmatrix} \frac{\partial\dot{\phi}}{\partial\phi} & \frac{\partial\dot{\phi}}{\partial\dot{\phi}} & \frac{\partial\dot{\phi}}{\partial\theta} & \frac{\partial\dot{\phi}}{\partial\dot{\theta}} & \frac{\partial\dot{\phi}}{\partial\dot{\psi}} \\ \frac{\partial\ddot{\phi}}{\partial\phi} & \frac{\partial\ddot{\phi}}{\partial\dot{\phi}} & \frac{\partial\ddot{\phi}}{\partial\theta} & \frac{\partial\ddot{\phi}}{\partial\dot{\theta}} & \frac{\partial\ddot{\phi}}{\partial\dot{\psi}} \\ \frac{\partial\dot{\theta}}{\partial\phi} & \frac{\partial\dot{\theta}}{\partial\dot{\phi}} & \frac{\partial\dot{\theta}}{\partial\theta} & \frac{\partial\dot{\theta}}{\partial\dot{\theta}} & \frac{\partial\dot{\theta}}{\partial\dot{\psi}} \\ \frac{\partial\ddot{\theta}}{\partial\phi} & \frac{\partial\ddot{\theta}}{\partial\dot{\phi}} & \frac{\partial\ddot{\theta}}{\partial\theta} & \frac{\partial\ddot{\theta}}{\partial\dot{\theta}} & \frac{\partial\ddot{\psi}}{\partial\dot{\psi}} \\ \frac{\partial\ddot{\psi}}{\partial\phi} & \frac{\partial\ddot{\psi}}{\partial\dot{\phi}} & \frac{\partial\ddot{\psi}}{\partial\theta} & \frac{\partial\ddot{\psi}}{\partial\dot{\theta}} & \frac{\partial\ddot{\psi}}{\partial\dot{\psi}} \end{bmatrix} \tag{20}$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{21}$$

$$B = \begin{bmatrix} \frac{\partial\dot{\phi}}{\partial T_1} & \frac{\partial\dot{\phi}}{\partial T_2} & \frac{\partial\dot{\phi}}{\partial T_3} & \frac{\partial\dot{\phi}}{\partial T_4} \\ \frac{\partial\ddot{\phi}}{\partial T_1} & \frac{\partial\ddot{\phi}}{\partial T_2} & \frac{\partial\ddot{\phi}}{\partial T_3} & \frac{\partial\ddot{\phi}}{\partial T_4} \\ \frac{\partial\dot{\theta}}{\partial T_1} & \frac{\partial\dot{\theta}}{\partial T_2} & \frac{\partial\dot{\theta}}{\partial T_3} & \frac{\partial\dot{\theta}}{\partial T_4} \\ \frac{\partial\ddot{\theta}}{\partial T_1} & \frac{\partial\ddot{\theta}}{\partial T_2} & \frac{\partial\ddot{\theta}}{\partial T_3} & \frac{\partial\ddot{\theta}}{\partial T_4} \\ \frac{\partial\ddot{\psi}}{\partial T_1} & \frac{\partial\ddot{\psi}}{\partial T_2} & \frac{\partial\ddot{\psi}}{\partial T_3} & \frac{\partial\ddot{\psi}}{\partial T_4} \end{bmatrix} \tag{22}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{-d}{\sqrt{2}I_x} & \frac{-d}{\sqrt{2}I_x} & \frac{d}{\sqrt{2}I_x} & \frac{d}{\sqrt{2}I_x} \\ 0 & 0 & 0 & 0 \\ \frac{-d}{\sqrt{2}I_y} & \frac{d}{\sqrt{2}I_y} & \frac{d}{\sqrt{2}I_y} & \frac{-d}{\sqrt{2}I_y} \\ \frac{-k}{bI_z} & \frac{k}{bI_z} & \frac{-k}{bI_z} & \frac{k}{bI_z} \end{bmatrix} \tag{23}$$

VI. Design of Linear Quadratic Controller

The Linear Quadratic Regulator (LQR) is a well-known method that provides optimally controlled feedback gains to enable the closed-loop stable and high performance design of systems [1].

First an understanding of the analytical approach for using LQR is provided. The LQR is a control technique which uses state space equations, to operate a dynamic system at minimum cost [2] [3]. The cost function for the LQR is given by

$$J = \int_0^\infty x^T Q x + u^T R u \, dt \tag{24}$$

where x and u are the previously mentioned input and state vectors, Q contains the costs of control errors (e = $x_{i,reference}$ - $x_{i,measured}$) and R signifies the cost matrix, penalising input parameters based on the importance of each state and inputs respectively. By tuning the Q and R matrices, a desired behaviour of the controller can be achieved. The LQR control law is defined as:

$$u = -kx \tag{25}$$

where *k* is the gain matrix which can be calculated by solving the Riccati equation by using defined Q and R matrices before), the equation above is the starting point to model the LQR, on how to go forward and implementing it in application are the important part. The LQR has been modelled as follows:

$$u = -k(r - x) + \text{BaseThrust} \tag{26}$$

The input for the controller has to equal the gain times reference tracking, the base thrust plays a vital role only when the drone is in motion.

The next step was to use the linearized model to find the LQR gain matrix. This was done with MATLAB's lqrd(A,B,Q,R,Ts) function where *A* and *B* are the linearization of the non-linear state space (see in previous section). *Q* and *R* are the cost matrices and *Ts* is the system sampling interval. *Q* is set to:

$$Q = 1000 \cdot \begin{bmatrix} 100 & 0 & 0 & 0 & 0 \\ 0 & 8000 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 9000 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The angular velocity of pitch and roll were penalized more since the angular momentum and thereby angular speed has to be counteracted fast. Otherwise the errors in angels will rapidly grow even bigger. In other words, penalising the angular speed more than the angels themselves can decrease the error in angle more than penalising the angle directly, following the concepts of derivatives in PID-controllers.

R is simply expressed as an unit matrix (created with the MATLAB command eye(4) where the 4 indicates a 4×4 matrix). The values of the *R* matrix are set to very low relative to the *Q* matrix since the power consumption is of less importance (as long as the motors aren't saturated) than the speed of the system. One could also say that it is more important to quickly stabilize the drone so it does not fall too far down or fly too far to the side before it stabilizes since it might crash. The order of magnitude on the weightings within the *Q* matrix were guessed and then fine tuned by repeatedly simulating and tuning the parameters to find a good compromise between a quick approach to the equilibrium point and oscillations. Further details about the simulations are treated in the section VII below.

When all parts for the controller were finished, the control scheme of the system was constructed and can be seen in Fig. 6. The scheme shows how the inputs from the *accel* are sent to a MATLAB function which calculates the angles as described in section III. The result is then added with the *gyro* readings in the complementary filter. The angles are in turn combined with the reference roll, pitch and disturbances signals before being combined with the angular velocities into the state vector. The state vector is then multiplied with the LQR gain matrix. Afterward the outcome is added to the base thrust to form the output to the different motors.

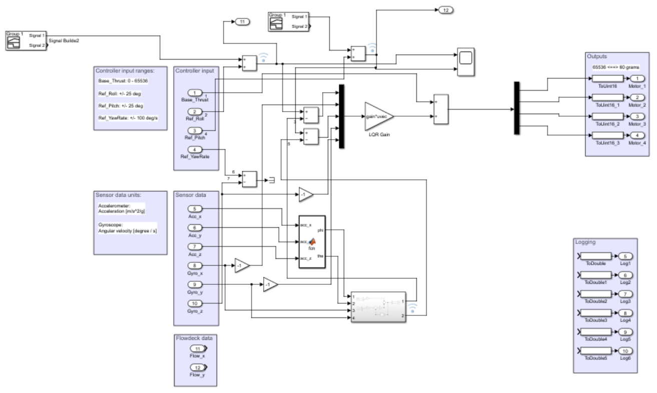

Fig. 6. linear quadratic regulator implementation in the controller.

VII. Evaluation of the control design

A. Simulink and the generated code

The result of the final Q-matrix can be seen in Fig. 7-9 below. The figures show the LQR controller in action which makes a relatively fast convergence and with low overshoot. However, the roll has less overshoot than the pitch, possibly because the drone is slightly longer than it is wide, giving it more moment of inertia around the pitch axis than the roll axis. This will in turn make $\dot{\theta}$ harder to control than $\dot{\varphi}$, meaning that it either takes more time to converge or gives a bigger overshoot, which is seen in these test cases.

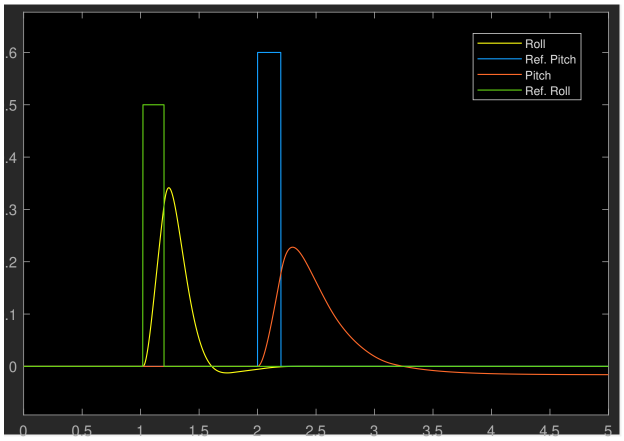

Fig. 7. Roll and pitch angles and how they change when time limited disturbances are applied.

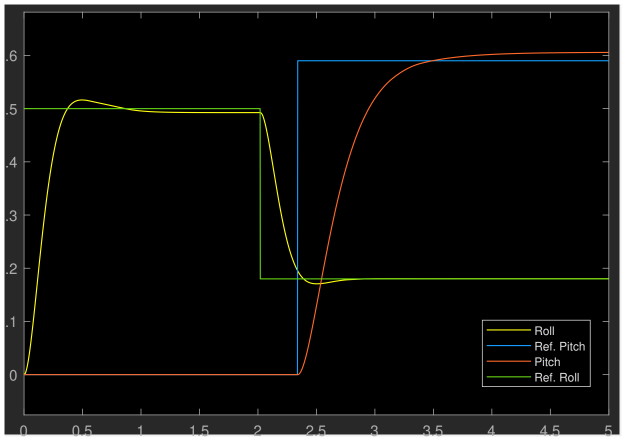

Fig. 8. Roll and pitch angles and how they change when step function disturbances are applied.

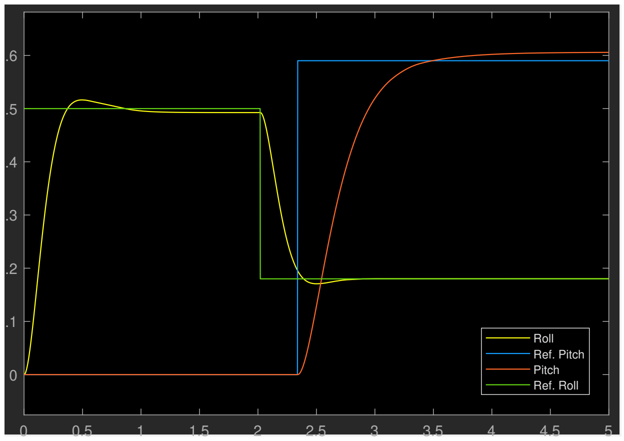

Fig. 9. Roll and pitch angles and how they change when step function disturbances are applied but this time without base thrust

Another drawback with the controller as a whole is that it doesn't contain integral action. This absence results in residual errors as can be seen on the pitch in Fig. 7 and both pitch and roll in Fig. 8-9. If studies on the drone had been performed for a longer time, it would have been beneficial to implement it with an integral action to minimise error as much as possible.

When comparing Fig. 8 and 9 they seem to be identical. The only difference is that base thrust is added in Fig. 8 and not in Fig. 9. As discussed earlier, the base thrust comes in effect only when set in motion. Intuitively, the base thrust can be understood as a throttle and LQR acts as a balance to the drone. So when when the drone is set in motion without the base thrust it will crash down.

C code for the LQR was generated and ran on the Crazyflie. When it started it had to be held in place by two fishing lines. The drone was subject to a tilt by pushing it upwards on the side, the rotors on the opposing side from the push suddenly increased their speed as a means of counterbalancing the created disturbance. When it got close to the equilibrium point it started to wobble. It wobbled too much to let it fly. The system was too unstable.

B. *“Hand written” version of the LQR*

The LQR was also implemented in C. Because of struggles with the “hand written” controller we weren’t able to obtain reasonable simulation results (see results in Fig. 10). At least the true roll angle keeps the correct second derivative, i.e. turning the right way almost all the time. There also seems to be some resemblance between the measured angles and the reference.

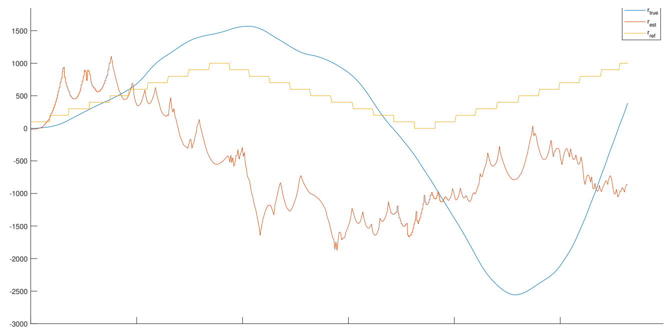

Fig. 10. Simulation of “hand written” LQR. The blue line is the actual roll, the red line is the estimated roll and the yellow line is the reference roll. Note that the functions are scaled, i.e. the unit is not degrees or radians.

The scaling in Fig. 10 was a way to make the functions be of the same order of magnitude to be able to draw any conclusions on the relations between the three lines.

A possible explanation behind the poor performance could be that the LQR-gain wasn’t good enough. Another possible explanation is that the delays of the different tasks were too small and that they had too long execution times, keeping the control task from running at the preferred 100 Hz or every 10 ms. This would make the controller output a certain output for too long, which in turn leads to an over all slower system, which is what is seen in figure 10

VIII. Real Time Operating Systems

The program ran on a Real Time Operating System (RTOS) which allows for multi-threading, that is, to queue several functions or parts of functions (from here on called tasks) to run interchangeably on the processor but also sharing memory. The idea with this is to utilize the processor as much as possible by:

1) “blocking” or “unblocking” tasks, meaning that they are removed or placed in the queue of tasks to be run on the processor.
2) Implementing a scheduling policy. For example Rate-Monotonic Scheduling (RMS) which sorts the unblocked tasks ascending in the queue or Earliest Deadline First (EDF) which, as the name suggests, sorts the unblocked tasks according to how close the different tasks’ deadlines were in time. For this project, the freeRTOS built in scheduling policy was used.
3) If two or more tasks have equal priority according to the scheduling policy, those tasks are sorted according to the individual tasks’ priorities.

A. *Delays*

Tasks can however delay themselves which is one way of a task being blocked. This can be helpful to give more processing time for more time critical tasks. For example, out of estimation, control and reference generation, the estimation is the most time critical since the system quickly can grow unbounded if the controller tries to compensate for faulty measurements. The reference generation is not of any significant importance for the robustness of the system and can therefore be delayed way more than the other two tasks. The estimation task therefore delays itself with 5 ms, the control task, 10 ms and the reference generator with 100 ms.

B. *Semaphores*

If the delays get too small, the different tasks might run interchangeably on the processor, which can be a good thing since each task by average gets to run more often, allowing for the control algorithm to become more robust. However, it can also make different tasks running on different threads reading or writing to memory being used by some other task before that other task has finished. A so called race condition arises. This usually results in algorithms not working properly. To solve this, semaphores are used. A semaphore is an integer where a task which wants to run “takes”, or decreases if it’s bigger than 0, then continues running. If it’s 0, the task is blocked until the semaphore’s value is increased, i.e. given. In this project semaphore gives and takes were utilised during, for instance, sensor readings, estimation readings and writings and reference readings and writings.

IX. Conclusions

A. *Acausal modelling*

The project has initially led to a working simulation of an LQR controlled quadcopter within relatively few modelling equations. This would not have been the case if a causal programming language would have been used instead of Simscape even for a model such as a quadrotor. This demonstrates the strengths of acausal modelling, modeling the system in a convenient and rapid way.

B. *Complementary filters and multiple sensors*

As mentioned before, sensor fusion (in this case the complementary filter) was proven to give less noisy data (from both *gyro and accel*), not only implying that complementary filters are useful in an orientation estimation contexts but also that multiple sensors can be combined to enhance the accuracy of the estimations from sensors’ data.

C. *Control design*

When it comes to the control of the drone, the noise in real time has a great effect on the LQR gain. Since the sensors are placed on the drone, the vibrations from the drone can influence the measurements of the sensor, and there might be other external factors such as wind etc. The values that worked during simulation were not the optimal ones while when running it on the actual drone. The angular velocity of pitch and roll have been penalized more during simulation since they can help in converging towards the changed reference faster.

References

[1] K. Alexis. *LQR Control*. [Online]. Available:http://www.kostasalexis.com/lqr-control.html#: :text=The%20Linear%20Quadratic%20Regulator%20(LQR,high%20performance%20design%20of%20systems.

[2] E. Okyere, A. Bousbaine, G.T. Poyi, A.K. Joseph and J.M. Andrade.'LQR controller design for quad-rotor helicopters'. *The Journal of Engineering*.pp.1-7. June 2018.

[3] Wikipedia. (2022, Mar. 2). *Linear–quadratic regulator*. [Online].Available:https://en.wikipedia.org/wiki/Linear%E2%80%93quadratic_regulator