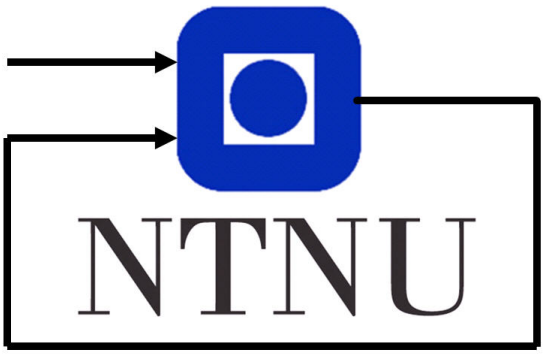
TTK4115 Lab Report

Group 27
Kristoffer Steinsland, Student 507739
Theodor Johannes Line Forgaard, Student 507032

November 16, 2020



Department of Engineering Cybernetics

Contents

1 Part I - Monovariable Control

1.1 Equations of motion

In this part we shall derive the equations of motion for the helicopter around the axis p, e and λ. We will find these derivations to be useful in our discussion later. To compute these equations of motion we can use Newtons second law for rotation, which state that the sum of torques acting on a rigid body about a fixed axis, equals the moment of inertia J times the angular acceleration α about the same axis

$$\sum \tau = J\alpha. \tag{1}$$

For the pitch angle p, we have $\alpha = \ddot{p}$ and let J_p be the moment of inertia about the axis. Using equation 1 and referencing figure 10 we get the following equation

$$\begin{aligned} J_p\ddot{p} &= l_p(F_b - F_{g,b}) - l_p(F_f - F_{g,f}) \\ &= l_p(F_b - m_p g - F_f + m_p g) \\ &= l_p(F_b - F_f) \\ &= l_p K_f(V_b - V_f) \\ &= -l_p K_f V_d, \end{aligned}$$

where F_b and F_f are the force generated from the back and front propeller respectively, and $F_{g,c}$, $F_{g,b}$ and $F_{g,f}$ are the gravitational forces acting on the counterweight, front motor and back motor. It is also worth noting that we assume a linear relation between the voltages supplied the motors and the propeller forces

$$\begin{aligned} F_f &= K_f V_f \\ F_b &= K_f V_b, \end{aligned}$$

and we write the sum and difference of the motor voltages as

$$\begin{aligned} V_s &= V_f + V_b \\ V_d &= V_f - V_b. \end{aligned}$$

Elevation e and travel λ can be modelled in a similar manner. Again J_e and J_λ denotes the moments of inertia about the respective axis,

$$\begin{aligned} J_e\ddot{e} &= m_c g l_c \cos e - 2m_p g l_h \cos e + l_h(F_f - F_b)\cos p \\ &= m_c g l_c \cos e - 2m_p g l_h \cos e + l_h K_f(V_f - V_b)\cos p \\ &= (m_c l_c - 2m_p l_h) g \cos e + l_h K_f V_s \cos p \end{aligned}$$

$$\begin{aligned} J_\lambda\ddot{\lambda} &= l_h(F_f + F_b)\cos e \sin p \\ &= l_h K_f V_s \cos e \sin p. \end{aligned}$$

We can now compress these equations by setting

$$\begin{aligned} L_1 &= -l_p K_f \\ L_2 &= (m_c l_c - 2m_p l_h)g \\ L_3 &= l_h K_f \\ L_4 &= l_h K_f, \end{aligned}$$

such that our model becomes

$$J_p \ddot{p} = L_1 V_d \tag{2a}$$
$$J_e \ddot{e} = L_2 \cos(e) + L_3 V_s \cos(p) \tag{2b}$$
$$J_\lambda \ddot{\lambda} = L_4 V_s \cos(e) \sin(p). \tag{2c}$$

1.2 Linearization

In order to use the methods examined in this course, it is necessary to have our system on linear form. We define $e = p = 0$ to be the angle when the arm extended by the joint angle e and p are horizontal, and linearize around $\lambda = e = p = 0$. Furthermore, to make calculations simpler, a variable transformation is introduced such that $e = 0$ is an equilibrium point. This is achieved by setting $\tilde{V}_s = V_s - V_{s,0}$, where $V_{s,0}$ is the voltage needed to keep the system at rest at our linearization point. After calculating all the partial differentials and inserting values for our linearization point, we end up with these equations describing our linearized model

$$\ddot{p} = K_1 V_d \tag{3a}$$
$$\ddot{e} = K_2 \tilde{V}_s \tag{3b}$$
$$\ddot{\lambda} = K_3 p, \tag{3c}$$

where the constants K_i are as follows

$$\begin{aligned} K_1 &= \frac{L_1}{J_p} \\ K_2 &= \frac{L_3}{J_e} \\ K_3 &= \frac{L_4 V_{s,0}}{J_\lambda}. \end{aligned}$$

1.3 Manual control

In this task we connected the outputs from the joystick directly to the inputs V_s and V_d. No feedback was used, and we quickly noticed that the helicopter was difficult to control. Taking the Laplace transform of our linearized model and looking at the transfer functions for $(p/V_d)(s)$ and $(e/\tilde{V}_s)(s)$, we get an idea of why the system is not suited for manual control. We see that we have double poles at origin and the linearized system

is unstable, which might explain why controlling the helicopter is quite challenging. We also guess that small differences in the motors thrust and/or weight differences makes manual control difficult. Because of this we can conclude that feedback control is needed to maintain stable operation of the helicopter.

1.4 Parameter identification

Because the encoder values are set to zero every time simulink connects to the helicopter, we added an offset of -0.53 radians, to make $e = 0$ correspond to the arm being horizontal. As discussed earlier, it is necessary to determine a constant motor voltage $V_{s,0}$ such that a $e = 0$ becomes an equilibrium point for the helicopter. After some experimentation we found $V_{s,0} = 7.5$ V to work well. Now we used equation 2b and the fact that in equilibrium $\ddot{e} = e = p = 0$, to determine K_f.

$$0 = L_2 + L_3 V_{s,0} \implies K_f = \frac{-(m_c l_c - 2m_p l_h)g}{l_h V_{s,0}}$$

This way we found $K_f = 0.133$ NV^{-1}.

1.5 PD control

To regulate the pitch angle, p, we used a PD controller given as

$$V_d = K_{pp}(p_c - p) - K_{pd}\dot{p}, \tag{4}$$

where p_c is the reference for the pitch angle. We then found the transfer function from p_c to p by combining equation (4) and (3a), and taking the Laplace transform

$$\mathcal{L}\{\ddot{p}\} = \mathcal{L}\{K_1(K_{pp}(p_c - p) - K_{pd}\dot{p})\} \implies \frac{p}{p_c}(s) = \frac{K_1 K_{pp}}{s^2 + K_1 K_{pd} s + K_1 K_{pp}}.$$

We can then express K_{pp} and K_{pd} as a function of the poles we want the second order system to have, giving us the equations

$$K_{pp} = \frac{\lambda_1 \lambda_2}{K_1}$$
$$K_{pd} = -\frac{(\lambda_1 + \lambda_2)}{K_1}.$$

The controller in equation 4 was implemented in Simulink, and is shown in figure 1.5.

We began experimenting with different complex conjugated pole placement and tested multiple different values for the angle and magnitude of the polar coordinates for the poles. To be able to compare the pole placements we made p_c follow a square wave and captured the models states and inputs, to let us plot the different responses together. During the testing we found that a greater angle corresponded to a more oscillatory response, and a greater magnitude corresponded to a faster response, at least within a

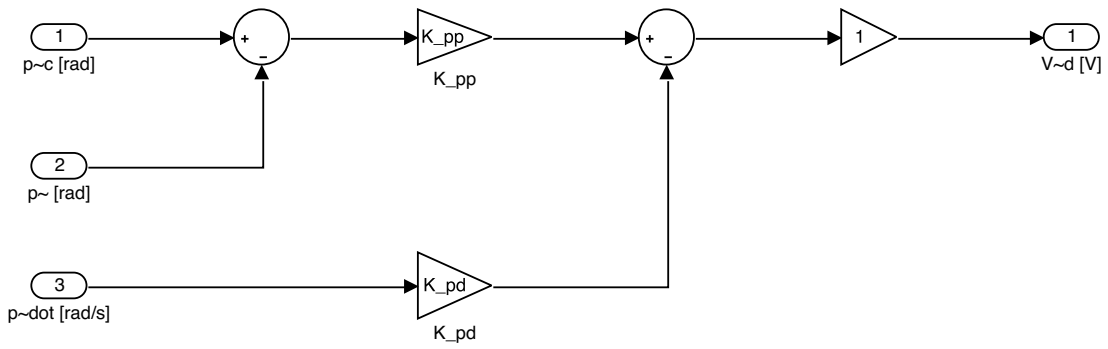


Figure 1: PD-controller in Simulink.

certain range. This can be seen in figure 2, where we see the oscillatory response from an angle close to 90° and the slow response of a low magnitude. A critically damped response with both poles on top of each other on the real line was also tried and can be seen in the same figure. We learned that the placing the poles with a high magnitude, e.g. 5 in this case, resulted in saturation of the motor voltages V_f and V_b as there is a saturation block with ± 5 V limit. This, and also the fact that we are using a linearized model which is only a good approximation around the linearization point, resulted in a oscillatory response when increasing the magnitude of the poles. Other nonlinear effects like transport delay and saturation on the rate of change of the force from the propellers, are also not taken into effect, and we cannot exclude that they would have a negative effect on our system. We also tried to validate our calculations and pole placements by estimating our transfer function model from our observed data, but the poles from the estimated transfer function did not match our placed poles well. Again this shows the limitations of our model.

After much trial and error, we concluded that poles with a magnitude of 2.5 and an angle of $\pm 45°$ resulted in a satisfactory response. We opted not to go for a critically dampened pole placement to achieve a faster response, with a trade off of only minor oscillatory characteristics. Looking back, it might have been beneficial to try some more conservative values for the angle and magnitude of the poles.

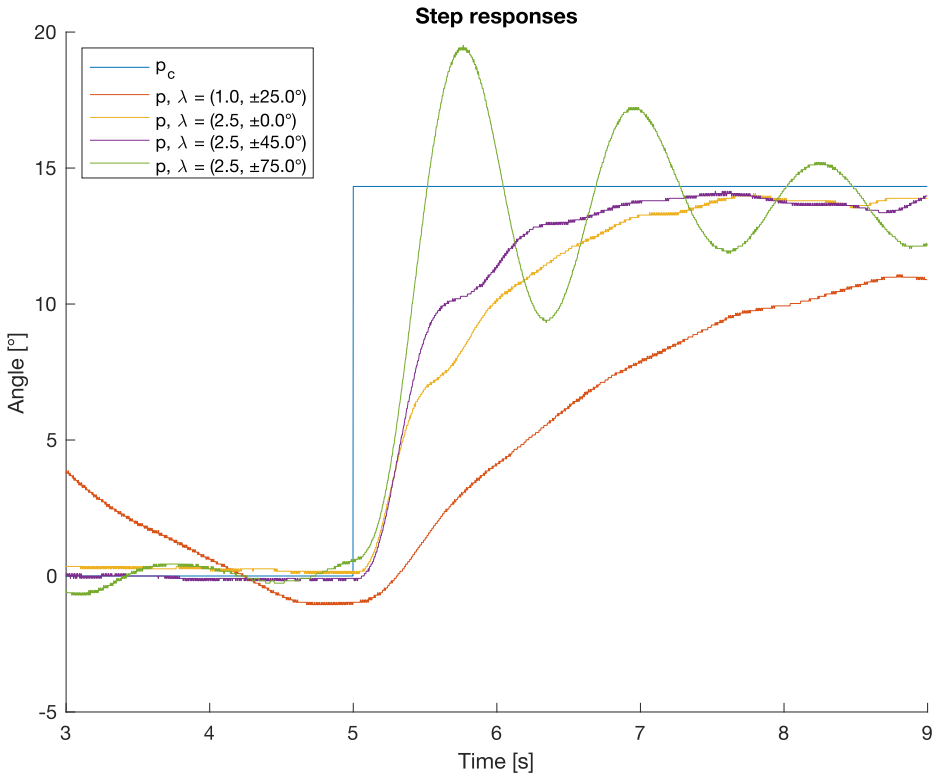


Figure 2: Pitch step responses for different pole placements from part 1, task 3.

2 Part II - Multivariable control

2.1 State space formulation

Because we want to control a system with multiple inputs and output it is beneficial to use state space formulation. To do this we will equations 3a and 3b on the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{5}$$

where $\mathbf{x} = [p,\, \dot{p},\, \dot{e}]^T$ and $\mathbf{u} = [V_s\, V_d]^T$. This gave us the matrices

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & K_1 \\ K_2 & 0 \end{bmatrix}.$$

We used state-feedback along with reference-feed-forward to generate an input signal to control the helicopter. The input u is given as

$$\mathbf{u} = \mathbf{F}\mathbf{r} - \mathbf{K}\mathbf{x}, \tag{6}$$

where, $\mathbf{r} = [p_c,\, \dot{e_c}]^T$, is the reference for the pitch angle p and elevation rate $\dot{e}$, respectively. Our Simulink implementation of this controller can be seen in figure 3. The state feedback allows us to alter the dynamics of our system, and ensure that the states converge towards zero, while the reference-feed-forward allow us to alter which setpoint we want the states to stay at rest at. Substituting equation 6 into 5 and rewriting we get

$$\dot{\mathbf{x}} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}\mathbf{F}\mathbf{r}. \tag{7}$$

We wanted $\mathbf{F}$ to make the measured states converge on the reference $\mathbf{r}$ as time tends towards infinity. At steady state we have that $\dot{\mathbf{x}} = \mathbf{0}$, then equation 7 can be rewritten as

$$\mathbf{x} = (\mathbf{B}\mathbf{K} - \mathbf{A})^{-1}\mathbf{B}\mathbf{F}\mathbf{r}$$

$$\mathbf{y} = \mathbf{C}(\mathbf{B}\mathbf{K} - \mathbf{A})^{-1}\mathbf{B}\mathbf{F}\mathbf{r}. \tag{8}$$

If our $\mathbf{C}$ matrix transposed has the same dimensions as $\mathbf{B}$, we can extract the optimal $\mathbf{F}$ by inversion,

$$\mathbf{F} = (\mathbf{C}(\mathbf{B}\mathbf{K} - \mathbf{A})^{-1}\mathbf{B})^{-1} \tag{9}$$

Substituting this into equation 8 we see that $\mathbf{y} = \mathbf{r}$ as $t \to \infty$. One could also solve the equations yielding $\mathbf{y} = \mathbf{r}$ by hand if the matrix cannot be inverted. As we wish to control the pitch angle and elevation rate, we set the $\mathbf{C}$ matrix as

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Solving equation 9, we get the following reference gain matrix

$$\mathbf{F} = \begin{bmatrix} k_{11} & k_{13} \\ k_{21} & k_{23} \end{bmatrix}, \tag{10}$$

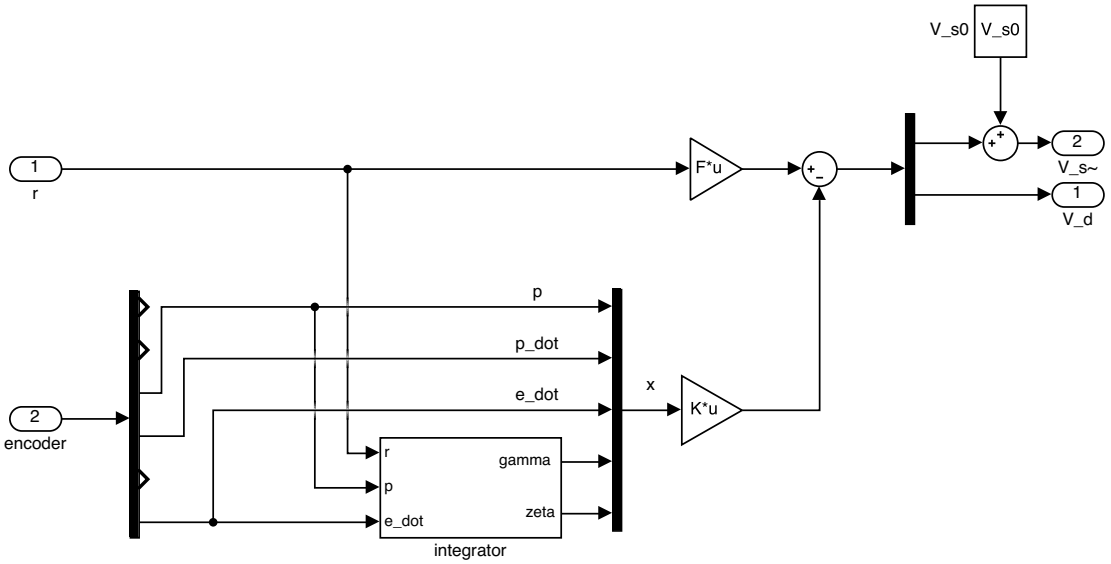


Figure 3: Simulink implementation of controller used. In task 1 and 2 the integrator block was not used, but the rest remained the same.

where k_{ij} are the terms in the $\mathbf{K}$ matrix.

We can examine the controllability of the system in equation 5 by calculating the controllability matrix given by $\mathcal{C} = \begin{bmatrix}\mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix}$, and checking the rank.

$$\mathcal{C} = \begin{bmatrix} 0 & 0 & 0 & K_1 & 0 & 0 \\ 0 & K_1 & 0 & 0 & 0 & 0 \\ K_2 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We see that as long as $K_1 \neq 0$ and $K_2 \neq 0$ the three rows are linearly independent so $rank(\mathcal{C}) = n = 3$, and the system is controllable. From theorem 8.3 in [1] the eigenvalues of $\mathbf{A} - \mathbf{BK}$ can now be arbitrarily assigned provided that complex conjugate eigenvalues are assigned in pairs. By choosing a $\mathbf{K}$ we now have the ability to alter the dynamics of the system given by equation 7. Different techniques to choose this $\mathbf{K}$ was examined and tested, and will be described in the following.

2.2 Pole placement

In this task we will manually place the poles of system given by equation 7. Our pole placement options are generally limited by the available actuator effort (up to 5v applied to the motor in this case) and stability criterion. According to [1], p.290, it is beneficial to place all eigenvalues evenly around an arc symmetric over the negative real axis. Furthermore, a large radius for the arc is supposed to give a faster response by using more actuator force, and a larger angle should according to theory result in a larger overshoot. We did experiments to test out these statements. To place the poles at

our desired coordinates we used the MATLAB function $K = place(A, B, p)$ where p is a vector containing our desired poles. Firstly we did some unsystematic testing to try to understand what it meant to now have three poles and how they corresponded to each state. What we found was that the two complex conjugated poles magnitude corresponded with the rate of which pitch reached its reference. A lower angle between the poles also seemed to decrease this rate, while a higher one made it faster but more oscillatory. The location of the third pole correlated with the $\dot{e}$ state, and again a higher magnitude made this state reach its reference faster. Interestingly, moving this pole did not seem to have any effect on pitch, and moving the conjugated poles seemed to have no effect on elevation rate. This might be because there is no connection between pitch and elevation in the **A** and **B** matrices.

After this, the poles were placed along an arc, with varying magnitude and angle. A step input was applied to the pitch reference and the response was captured. Some of the tests we did can be seen in figure 4. First notice the difference in the purple and red response. Both pole configuration have the same magnitude but in the purple one the poles are placed at a higher angle. We see that a higher angle results in a faster and more oscillatory response. Furthermore, when we keep the angle low but set a higher magnitude as the yellow response shows, we also get a faster and more oscillatory response. Notice how there seems to be more noise in the yellow response as well. Additionally, when increasing the radius even more, the motors eventually saturated which led to the system becoming unstable. Overall we found the theory to work well in practice. A final tuning was found by doing multiple tests and choosing the one we found to be most appropriate in terms of a compromise between speed and stability. Additional manual tests where done to assure that controlling the elevation rate also worked as intended. In the end, we ended up choosing the tuning shown by the red line in figure 4, where the three poles are 45° apart with a magnitude of 5. This resulted in the following **K** matrix.

$$\mathbf{K} = \begin{bmatrix} 0 & 0 & 29.40 \\ 11.83 & 6.69 & 0 \end{bmatrix}$$

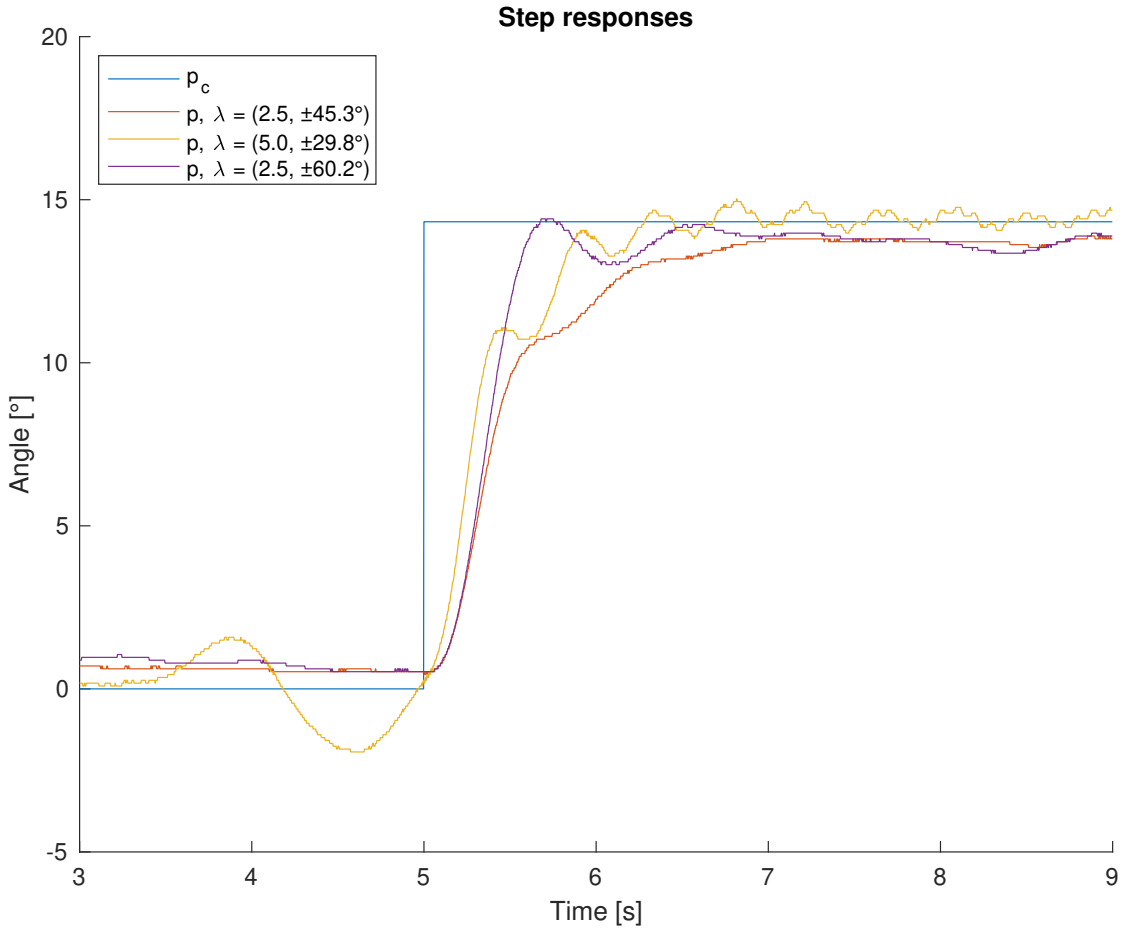


Figure 4: Step responses for different pole placements, λ shows the polar coordinate of the complex conjugated pair of poles, where the angle is measured relative to the negative real axis. The third pole is placed at the same magnitude, but at 0°.

2.3 LQR

Pole placement works as intended, but it is often difficult for us to know where to place the poles to get a desired response. To overcome this we can use LQR optimization. In this technique we still use the same input $\mathbf{u} = \mathbf{Fr} - \mathbf{Kx}$, but now we let the coefficients in the $\mathbf{K}$ matrix be chosen such that they minimize the cost function in equation 11 with regard to the input. Using this method we are able to weight the relative importance of the performance of each state and the cost of using the actuators by choosing the coefficients in $\mathbf{Q}$ and $\mathbf{R}$ in such a way that meets our design criteria.

$$J = \int_0^\infty \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\, dt \tag{11}$$

We hypothesized by looking at equation 11, that by increasing a term corresponding to a state in the **Q** weight matrix relative to other states, we effectively put a higher "penalty" on that state being non zero, and the lqr algorithm should adjusts the **K** matrix accordingly to use more actuating effort to pull this state to zero.

To test out this hypothesis we did some experiments which can be seen in figure 5 and 6. For simplicity the matrices **Q** and **R** were chosen to be diagonal. Firstly we set all the diagonal terms to 1, thus giving all states equal weighting. From figure 5 we see that this was not effective. We believe this response can be explained by the fact that since the terms in **R** have the same weighting as those in **Q**, **R** is for this system set too high compared to **Q**. This results in that the cost of using the actuators is higher than the cost of deviations in the states. So in this case the the optimal strategy is to use less actuator effort than what is needed for **x** to go to zero, and therefore we get a terrible response as shown.

When we increased the term in **Q** that corresponds to pitch as shown in the upper rightmost plot i figure 5, we see how the pitch responds fairly fast to the reference. This is because we placed a cost in **Q** that is much higher than the cost in **R** for using actuator effort. In the leftmost plot at the bottom we see that when the term in **Q** that corresponds to pitch rate is set high, the helicopter cant reach its pitch reference because a change in pitch rate is penalised highly. This term seems to have a dampening effect as shown in the last plot in figure 5, and we should be able to use it to improve our stability margins should the need arise.

The same type of experiments but for elevation rate can be seen in figure 6. As we discovered with pitch, elevation rate too seamed to reach its reference faster when we increased the corresponding term in in **Q**. However, we found elevation rate to be a bit more difficult to control, firstly because the encoders are noisy. And secondly because it is impossible for the helicopter to keep a constant elevation rate for longer periods of time. In hindsight we realise that the tests performed for elevation rate should have had a much smaller step input as reference, thus giving the helicopter the opportunity to sustain the rate longer. Also, it might have been illuminating to perform both the elevation rate and pitch test simultaneously as to discover how this would affect the overall response.

The final values in **Q** that we arrived at was a result of a the systematic testing explained above, and some manual testing to see how the helicopter handled. When it comes to the **R** matrix we found that increasing the diagonal terms made the helicopter use less actuator effort. We could not find a compelling argument as to why penalize the use of V_d or V_s over the other. Furthermore we see from equations 3 that the states p and $\dot{p}$ is dependent on V_d but not on V_s. The opposite is true for $\dot{e}$. This has the consequence that increasing the actuator cost in **R** for the input V_s would be the same as decreasing the cost for an error in the $\dot{e}$ state in **Q** by the same factor. The same goes for V_d versus p and $\dot{p}$. For this reason, and simplicity, the **R** matrix is set to just have ones on its

diagonal. Tests of the final tuning can be seen in figure 7. The tuning we arrived was

$$\mathbf{Q} = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 500 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Which gave us

$$\mathbf{K} = \begin{bmatrix} 0 & 0 & 22.37 \\ 10.00 & 6.92 & 0 \end{bmatrix},$$

This tuning works fairly well, but notice how we have a steady state error present in all our different tests. This problem will now be addressed.

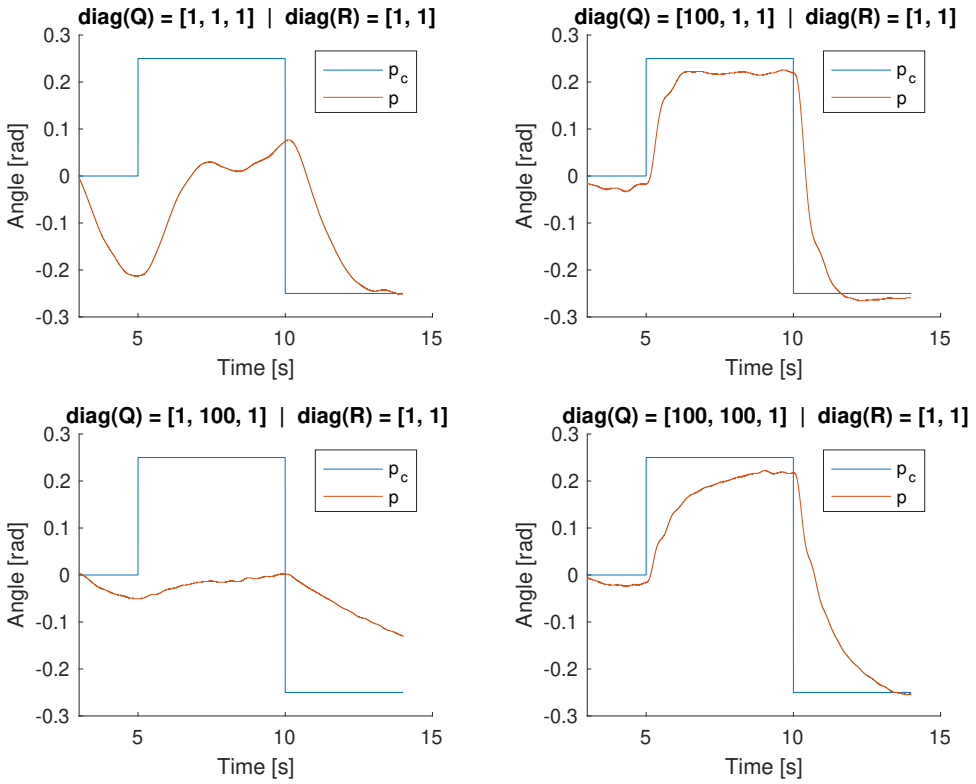


Figure 5: Pich step responses for different LQR opimizations

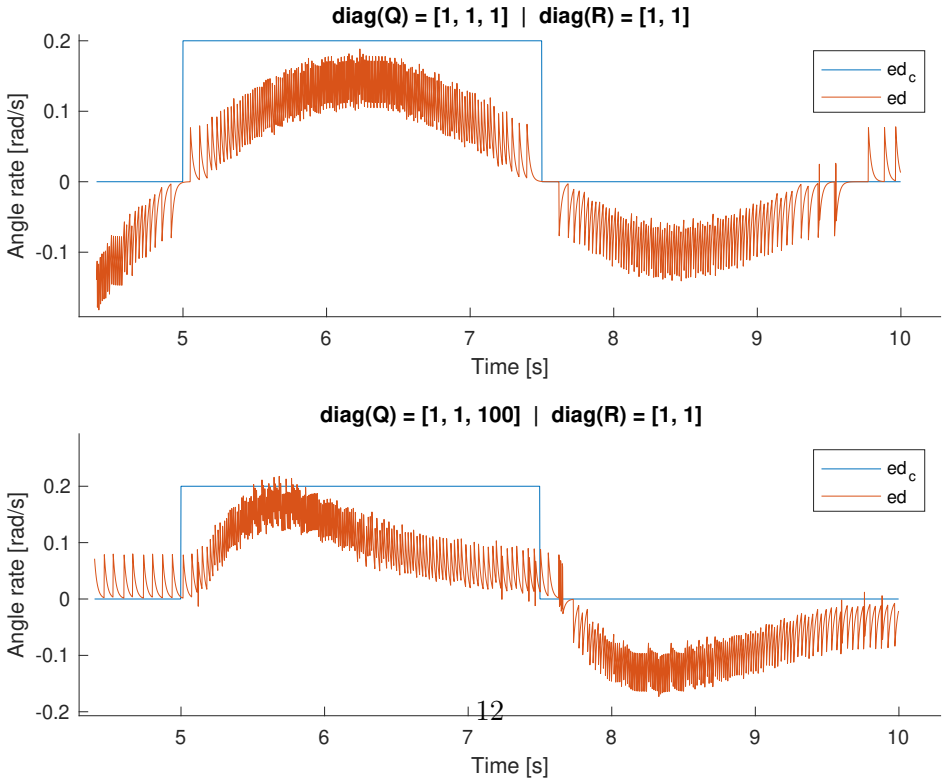


Figure 6: Elevation rate step responses for different LQR opimizations

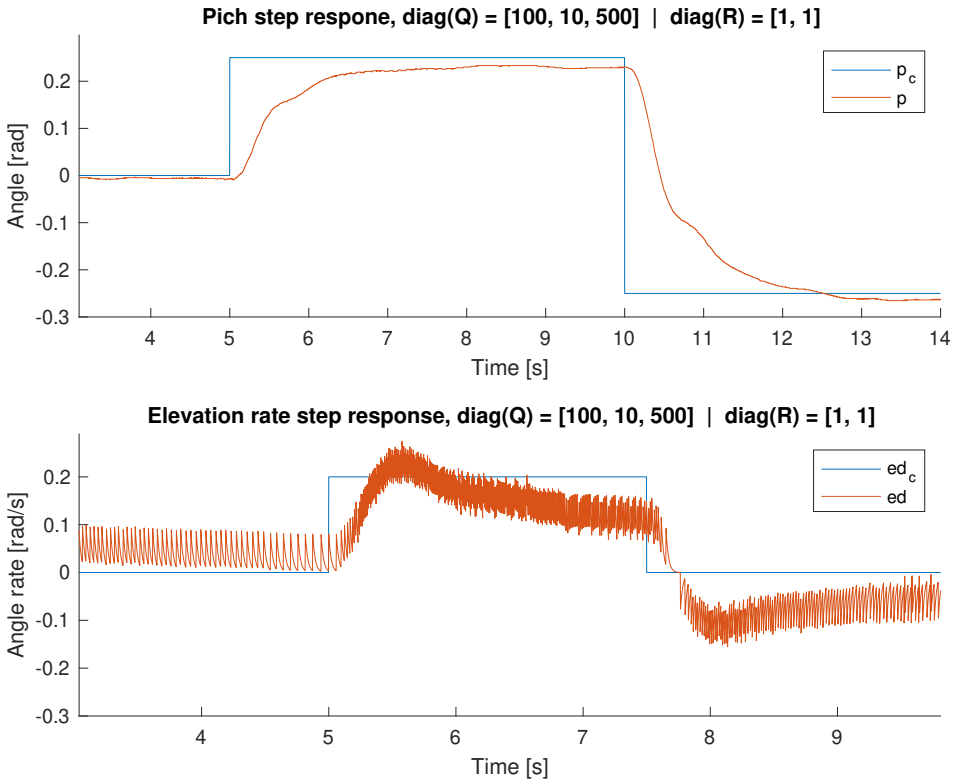


Figure 7: Elevation rate and pitch step responses with final tuning

2.4 LQR with integral action

After the development of equation 9, we made the claim that by choosing an appropriate reference gain matrix the output would reach the reference as time tends towards infinity. However, this claim assumes a perfect model and that no external disturbances are acting on the system which is not realistic. This becomes clear by looking at the step response in figure 7 from the previous task. To deal with this problem we implemented integral action in our controller. To do this we added two new states to our system γ and ζ, defined as

$$\dot{\gamma} = p - p_c$$
$$\dot{\zeta} = \dot{e} - \dot{e_c}.$$

If we make a vector containing our new states as $\mathbf{x}_a = [\gamma, \zeta]^T$ and keep $\mathbf{x}$ as before, we can write the new augmented system as

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{x}}_a \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ -\mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_a \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \mathbf{u} + \begin{bmatrix} \mathbf{0} \\ \mathbb{I} \end{bmatrix} \mathbf{r} \tag{12}$$

where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are the same matrices as before, only changing the input $\mathbf{u}$ to accommodate the new states.

$$\mathbf{u} = \mathbf{Fr} - \begin{bmatrix} \mathbf{K} & \mathbf{K_a} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_a \end{bmatrix} \tag{13}$$

The $\mathbf{F}$ matrix must also be updated Solving the equations yielding $\mathbf{y} = \mathbf{r}$ from 2.1 by hand gives us the following $\mathbf{F}$ matrix

$$\mathbf{F} = \begin{bmatrix} k_{11} & k_{14} \\ k_{21} & k_{24} \end{bmatrix}. \tag{14}$$

One could also argue that we see from our last $\mathbf{F}$ matrix 10 that we want the positions in the $\mathbf{K}$ matrix corresponding to p and $\dot{e}$, and $\dot{e}$ has been moved one index to the right.

We used the tuning from the last past as a starting point, and experimented with different values for $\mathbf{Q}$. Firstly we noticed that now it was necessary to decrease the term in $\mathbf{Q}$ corresponding to the $\dot{e}$ state to make the helicopter less jittery. Then we started altering terms in the $\mathbf{Q}$ corresponding to the γ and ζ states. We found that increasing the term for γ decreased the steady state error for the pitch angle p, but increasing the term too much resulted in quite an overshoot as shows in figure 8. A good compromise can be seen in the last plot in figure 8. We had a bit more trouble with setting the term for the ζ state. Notice in figure 9 that we had to increase the term in $\mathbf{Q}$ corresponding to ζ considerably to reduce the steady state error in $\dot{e}$. Even with a high integral effect like this we still had a steady state error. We contemplated increasing it even more, but we had to make a compromise with regards to overshoot and stability. The final tuning

we ended up with was

$$\mathbf{Q} = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 100 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Which gave us the feedback gain

$$\begin{bmatrix} \mathbf{K} & \mathbf{K_a} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 16.87 & 0 & 10.00 \\ 11.52 & 7.32 & 0 & 2.23 & 0 \end{bmatrix},$$

From the tests we have done, we can conclude that adding integral effect was quite helpful at removing steady state errors, especially in the pitch output, as it compensates for constants disturbances and modelling errors. Now, a question remains; how does integral action influence how can we choose the **F** matrix? We hypothesised that since what the **F** matrix is supposed to do is make the output converge on the reference as time tends towards infinity, and since now we have integral effect to do this job for us, the **F** was redundant. We set **F** to zero and tested our system. After some experimentation we found that even though the output moved towards the reference, the response was too slow for all practical purposes. We believe this is because when we used reference feedforward, we instantly gain the reference signal as to give the correct output. But when we are using integral action, it takes time for the error to accumulate in the integrator and thus it takes some time before the correct bias is set. To accommodate for this, the terms in **Q** corresponding to γ and ζ could be set higher to get more integral action, but this would still not be the same as reference feed forward. For us it seemed like the best solution was to bring back the reference gain as to get a quicker response, and use it in combination with integral action to remove the steady state error.

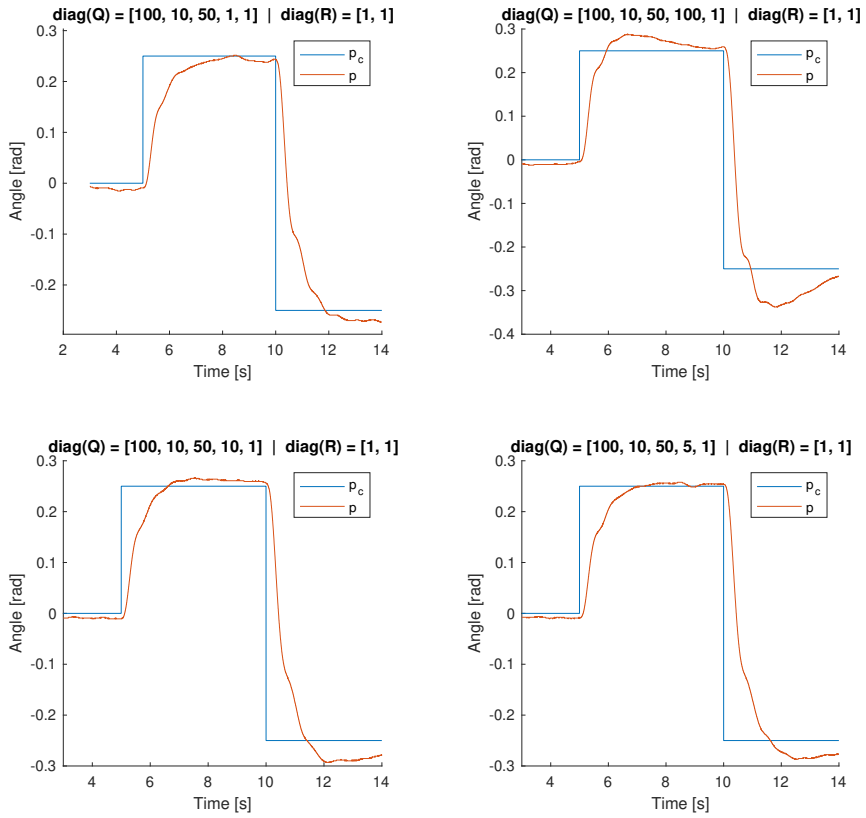


Figure 8: Pich step responses for different LQR optimizations with integral action

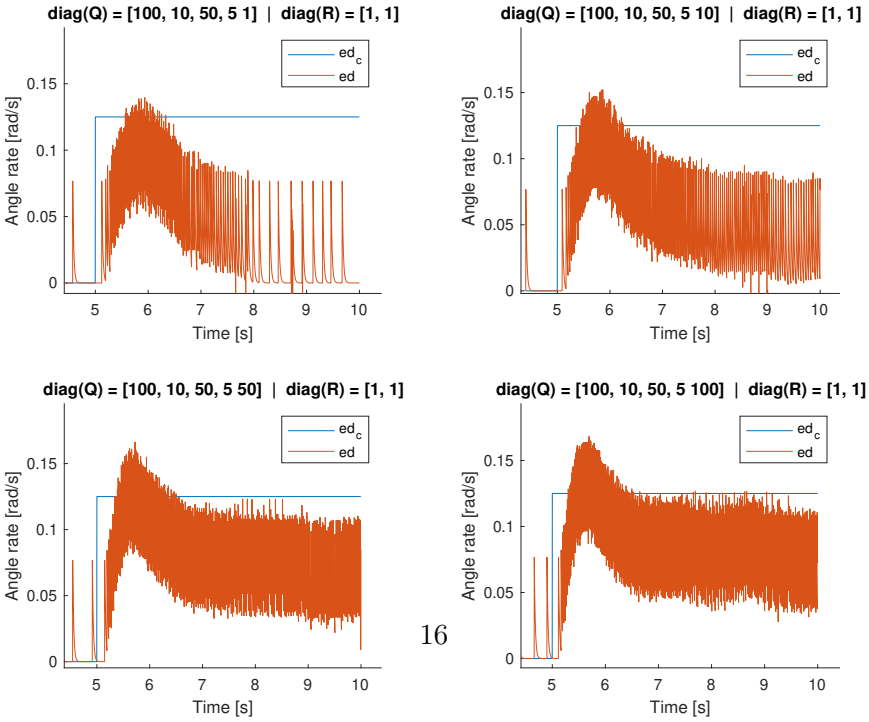


Figure 9: Elevation rate step responses for different LQR optimizations with integral action

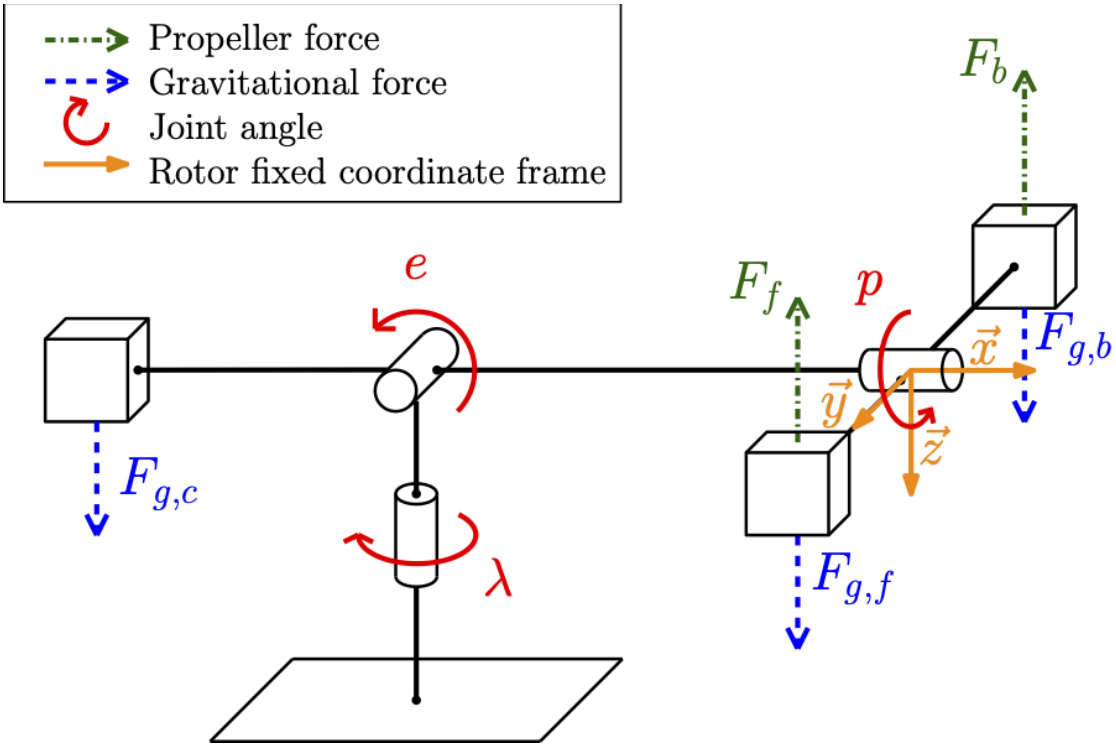


Figure 10: The two different reference frames. Found in the lab preparation [2]

3 Part III - Luenberger Observer

3.1 IMU characteristics

In this task we looked at the output of the IMU when we manually moved the helicopter and compared them to the data from the encoders. It is important to note that the encoders and IMU does not operate within the same reference frame so a deviation is expected. The two different reference frames are shown in figure 10.

Despite of this, we found that the rates from the gyro matched those from the encoders well around the equilibrium point. However, we saw big discrepancies between the gyro and encoders in travel and elevation when pitch was rotated maximally. This also happened when we rotated elevation, but to a smaller extent because we were not physically able to rotate this angle too much. Rotating travel did not have any effect that we noticed. In addition, when we rotated the helicopter about one axis and then rotated about a second axis, we found that when moving pitch as well as elevation or travel, there were big deviations in two latter axis. However, moving just travel and elevation did not result in big discrepancies, and pitch seemed to be correct no matter the rotation.

We can try to explain this by again referring to figure 10. Here we see that pitch follows the IMU and the p angle will always be around the $\vec{x}$ axis on the IMU. However

when elevation is rotated, the IMU should pick up rotations around travel as pitch and create a discrepancy. Surprisingly this did not seem to be a big problem, most likely because elevation is restrained from going too high and create any big problems. Also, when pitch equals zero, the y and z axes align with λ and e. This explains our observation that travel and elevation rates followed the encoders well when we held pitch horizontal. Be that as it may, there are big errors when pitch is rotated, so a coordinate transformation is needed.

When it comes to the accelerometer output we noticed how the norm of the components equal 9.81, as the accelerometer measures the acceleration in relation to free fall. This allows us to use accelerometer readings a_x, a_y and a_z to calculate the angles e and p. However we see that moving around the λ angle will not change the any of the acceleration components, meaning that the accelerometer cant directly give us this angle. This makes sense since the gravitational force will always be parallel to λ's axis of rotation.

3.2 Transformations

To get the p and e angles from the accelerometer readings, we implemented the following equations in Simulink as seen in figure 11 and 12.

$$p = \arctan \frac{a_y}{a_z} \tag{15}$$

$$e = \arctan \frac{a_x}{\sqrt{{a_y}^2 + {a_z}^2}}. \tag{16}$$

In the previous task we noticed how having two different frames of reference caused errors when the helicopter was moved away from its equilibrium point. To account for this the following coordinate transformation was applied to the IMU rates

$$\mathbf{T} = \begin{bmatrix} 1 & \sin p \tan e & \cos p \tan e \\ 0 & \cos p & -\sin p \\ 0 & \sin p / \cos e & \cos p / \cos e \end{bmatrix}.$$

From this we notice a couple of things. Firstly we see that when e and p is equal to zero, we get the identity matrix and the gyro outputs aren't transformed at all. This works well with our observation from the previous task that there where little discrepancies around the equilibrium point. Secondly we notice that when $e \to \frac{\pi}{2}$ we get a singularity. This problem is also present in equations 15 and 16. We are not sure how to fix this, except from perhaps using something other than Euler angles to represent rotation. However, when experimenting at the lab we found that this wasn't an issue because the helicopter is physically restrained from moving too close to this position. Also this transformation seems to make our system non-linear, but for reasons beyond our knowledge this seemed to have no impact.

With the transformations applied we found that now both the angles and rates follow those from the encoders nicely. However, the signals from the IMU are noisy, so

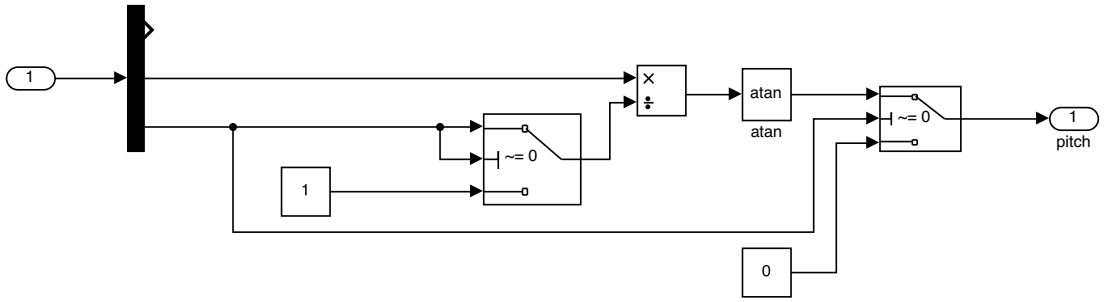


Figure 11: Simulink implementation of equation 15. The input a is a vector containing a_x, a_y and a_z.

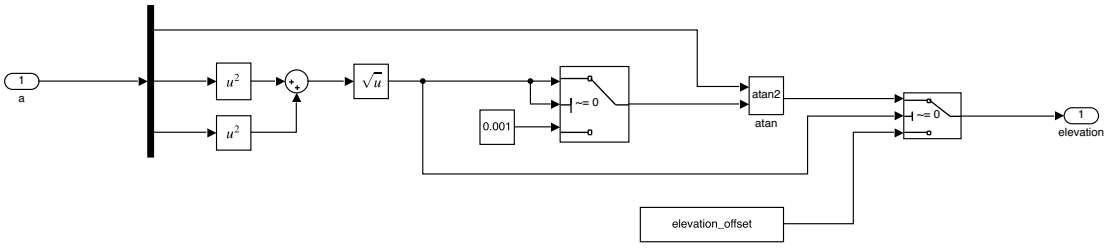


Figure 12: Simulink implementation of equation 16.

some filtering is necessary. There also seems to be a wrong bias in some of the IMU measurements, and that also needs to be corrected.

3.3 Theoretical discussion

We hypothesized that for a state to be observable, there has to be a connection between the given state and the output. Examining the observability of our system, given by the formula

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{C}\,\mathbf{A} \\ \mathbf{C}\,\mathbf{A}^2 \\ \vdots \\ \mathbf{C}\,\mathbf{A}^{n-1} \end{bmatrix},$$

we found that the minimal set of measured states that makes the system observable is

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

which corresponds to measuring elevation e, and travel rate $\dot{\lambda}$. Looking at our new system, we see that measuring these two states, we can in principle calculate all other states by differentiating e and $\dot{\lambda}$ a number of times. However, differentiating will amplify

noise with high frequencies resulting in poor measurements. This means that even though the system is observable with this minimal set, it will be beneficial to measure more of the states. We also see that we cannot only measure the rates, since integration of the rates gives us undefined constants for pitch and elevation.

Considering the accelerometer, we think that the angle measurements will be correct as long as we do not accelerate the helicopter too much. When we move the helicopter, the IMU will read the free fall acceleration as well as the acceleration caused by the helicopter moving, causing an error. Obviously it is necessary to move the helicopter, but this makes us aware off the fact that too large accelerations will introduce errors. It is important to note that we fed the transformation from the last section with the angles calculated with the accelerometer values, as to be completely independent of the encoders. This created some serious discrepancies when the helicopter was moving fast around its travel axis. We believe this is because when the helicopter is moving in a circular motion around this axis, there is a acceleration pointed towards the center of the helicopter that the IMU picks up as acceleration in the $\vec{x}$ direction, that results in a error in the elevation angle, which then permeates the other measurements through the transformation.This centripetal acceleration will also be a flaw of the whole state space model, as we have observed that the helicopter falls down to the ground much slower when it is rotating around the travel axis.

3.4 State Estimator

In this part we wanted our system states to be $x = \begin{bmatrix} p & \dot{p} & e & \dot{e} & \dot{\lambda} \end{bmatrix}^T$. Studying equations 3a, 3b and 3a we found our new system to be

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ K_3 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & K_1 \\ 0 & 0 \\ K_2 & 0 \\ 0 & 0 \end{bmatrix}.$$

All measurements available from the IMU was used, such that $\mathbf{C} = \mathbf{I}$. Furthermore, the following state observer was used to calculate the state estimate $\hat{\mathbf{x}}$,

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}(\mathbf{y} - \mathbf{C}\hat{\mathbf{x}}), \tag{17}$$

Our Simulink implementation of this state estimator can be found in figure 13. Equation 17 can be rewritten as

$$\dot{\hat{\mathbf{x}}} = (\mathbf{A} - \mathbf{L}\mathbf{C})\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}\mathbf{y}, \tag{18}$$

and our controller used this new estimate such that

$$\mathbf{u} = \mathbf{F}\mathbf{r} - \begin{bmatrix} \mathbf{K} & \mathbf{K_a} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}} \\ \mathbf{x}_a \end{bmatrix} \tag{19}$$

Because of the separation property we can design our estimator and controller separately

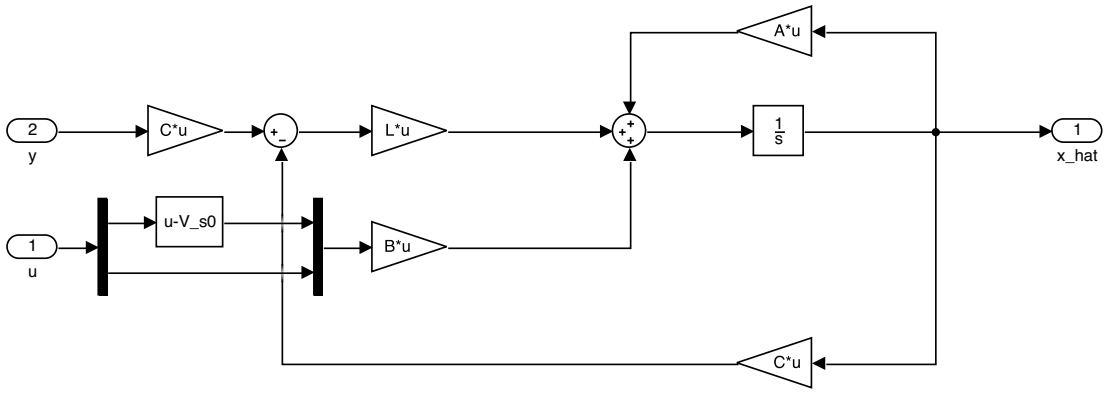


Figure 13: Simulink implementation of the state estimator in equation 17.

[1], p.307, so we used the same LQR controller with integral action as from day 2. However, we had to accommodate for the new states and find a satisfactory tuning to be able to use the controller. We arrived at

$$\mathbf{Q} = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The task ahead was to find an observer gain $\mathbf{L}$ such that our Luenberger observer would give us accurate estimates. We ran the controller on the values from the encoder, ensuring us a well behaving helicopter, letting us focus on the performance of the estimator.

The method used to achieve a good tuning for the estimator involved placing the poles and then comparing the response with the ground truth from the encoders. To place the poles of the observer we used the MATLAB command $L = place(A', C', p)'$ where p was a vector that contained our poles. As described in [1], p. 302, it is probably best to place the poles evenly along a circle in the left hand plane with a maximum angle not too high up from the real line. Because of this, the eigenvalues of $(\mathbf{A} - \mathbf{LC})$ where evenly dispersed around a half circle with angles ranging from 150° to 210° as shown in figure 14. This choice is somewhat arbitrary and we see in hindsight that it might have been beneficial too experiment with different angles. We then tested varying the magnitude of the half circle containing the poles in the range $[0.5, 200]$, while we fed the controller a square wave as a pitch reference. While running these tests, the output from the IMU, encoders and estimator was observed and logged. We used the MATLAB command $MSE = mean((p_{hat} - p_{enc}).^2)$ to calculate the mean square error between the pitch angle from the encoder and estimator. This was then used as a measure of the performance of the estimator. We also observed that all the other estimated states matched well with the encoder values.

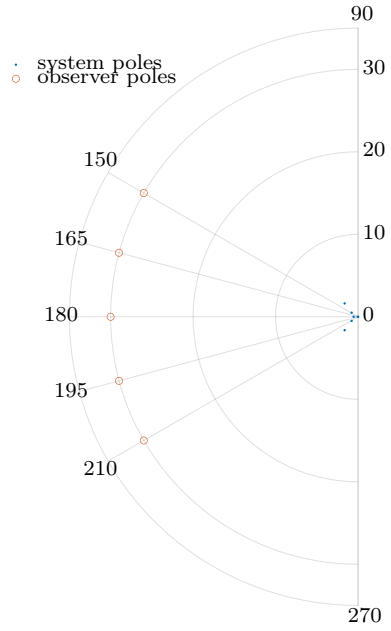


Figure 14: Poles of observer that gave the best results.

An excerpt of the results from this experimentation can be seen in figure 15. Furthermore, the mean square errors between the pitch angle from the encoder and estimate from all the configurations of $\mathbf{L}$ tried can be found in figure 16. The lowest mean square error achieved in these test was found when the magnitude of the eigenvalues of $(\mathbf{A} - \mathbf{LC})$ were placed at 30. This final pole configuration can be seen in figure 14. We can see from figure 15 that when we moved the poles further into the left half plane our estimator tended to follow the noisy IMU measurements closer. As we moved them closer to origin, we got more of a low pass filter effect and a smoother signal. To explain this in basis of theory we can see from equation 18 that by moving the poles of $(\mathbf{A} - \mathbf{LC})$ further into the left half plane our observer will have a faster time constant and thus follow the measurement closer. When overdone we noticed the helicopter became jittery and hard to control. Inversely, when we move the poles closer to the origin, our time constant drops and the observer wont be able to follow the noise as much. This accounts for the low pass filter effect that we observed. This low pass effect was beneficial because we had a noisy signal, but it must not be overdone. Notice for example the first plot in figure 15 when we set the magnitude of the poles to 0.5 how the observer is slower than the vehicle dynamics and thus cant keep up at all. Furthermore when $\mathbf{L}$ is low such that $(\mathbf{A} - \mathbf{LC})$ is close to $\mathbf{A}$, the correction term $\mathbf{L}(\mathbf{y} - \mathbf{C}\hat{\mathbf{x}})$ looses its influence and our system model dominates in the estimate. This is fine as long as we have a perfect model, but we don't. From this we can conclude that neither an open loop estimator nor complete trust in

the IMU is optimal, but a weighted combination of both is whats needed to get the best results. This can be confirmed in figure 16, where we see that the fitted error curve appears to be convex with a minimum.

When we had arrived at a satisfactory tuning for the estimator, we ran the helicopter on the estimator to confirm that our chosen tuning behaves well in practice. In summary we had good results with using the estimator, however, we experienced some drifting in elevation which seemed to stem from a bias in the elevation rate from the IMU. To compensate for this, we made a calibration program which we could run to calculate the necessary offsets to subtract from the IMU measurements. This can be seen in figure 17. We made sure to first calculate the offsets in the pitch and elevation angle, such that the transformation used to get the correct rates, received the correct angles before calibrating the rates. After implementing this calibration program the estimator behaved much more nicely. As we were made aware of later from our teachers, our system becomes nonlinear because of the transformation done to the rates, as we feed in the elevation and pitch angle from the IMU. This might make examining the whole system more problematic, but as we wrote earlier, when pitch and elevation angles are equal to zero, no transformation is done to the gyro rates, so this should not be an issue around the equilibrium point.

We were also made aware of the fact that adding an offset to the pitch and elevation derived from the equations using the acceleration vector, might not be optimal, as these equations also are nonlinear, and one should optimally find the necessary offsets to add to the acceleration vector to get a correct pitch and elevation angle over the whole angle range. However, the offsets added to the angles were small and we guess that the deviation will be small.

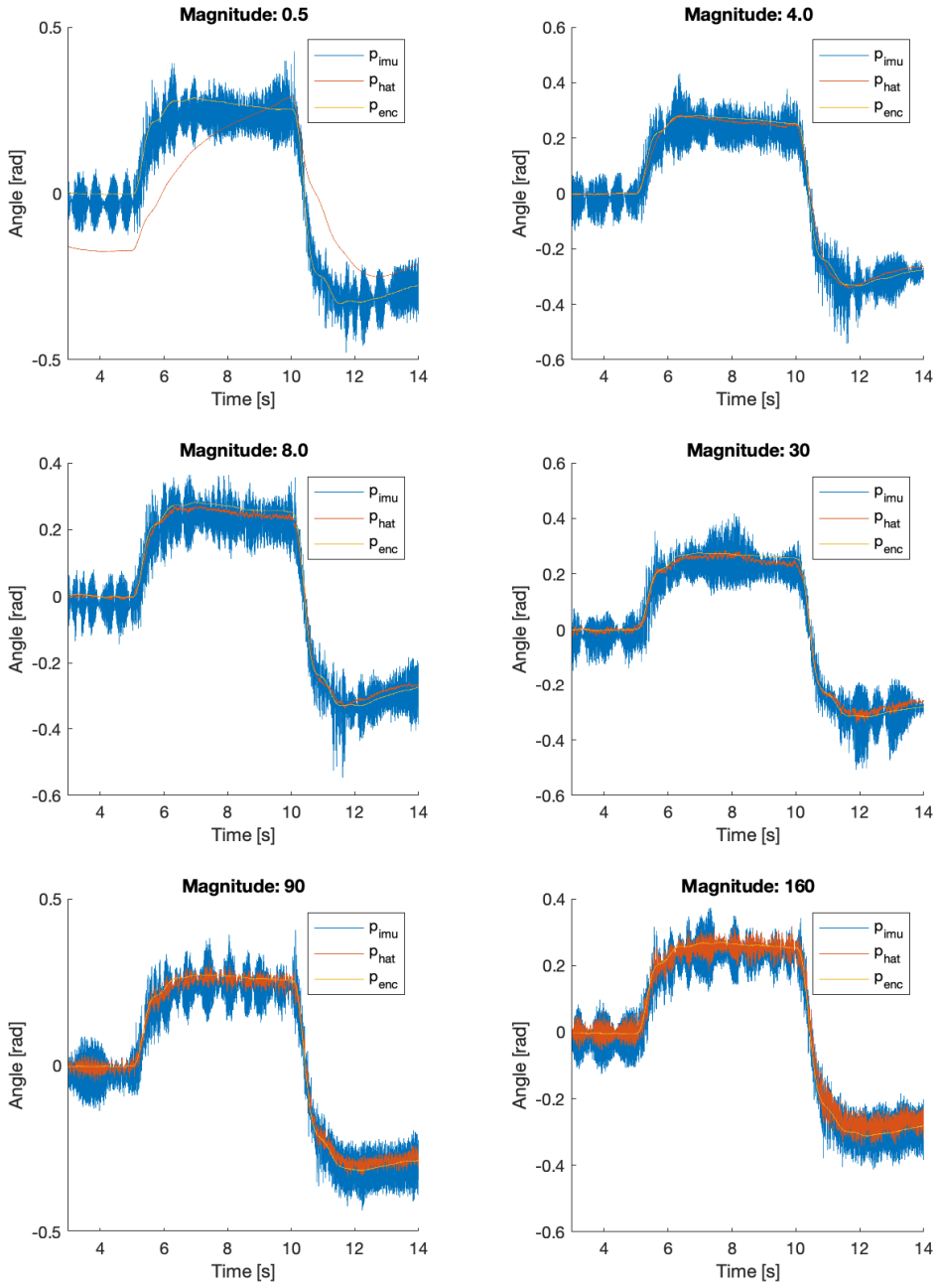


Figure 15: Each plot, with a corresponding pole configuration, shows IMU data P_{imu}, encoder data P_{enc} and estimation P_{hat} for a pitch step response. The the poles of $A-LC$ where evenly dispersed around a half circle in the left half plane with a varying magnitude as shown in each title and a fixed range of angles.

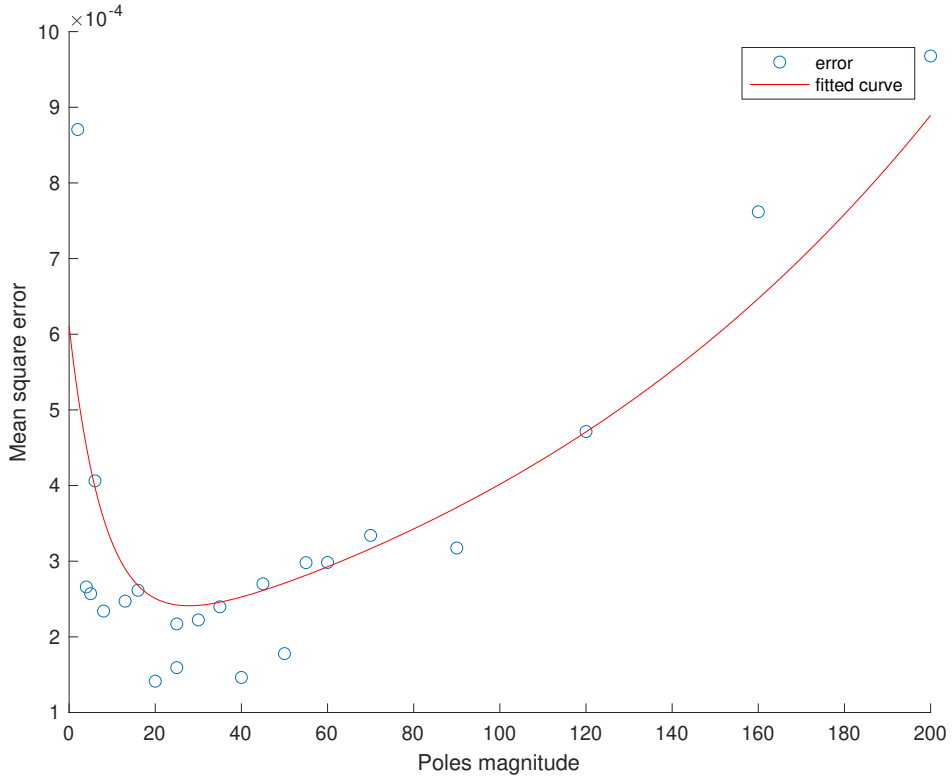


Figure 16: Mean square error between the pitch angle from the encoder and estimator for a pitch step response with different values of $\mathbf{L}$. The the poles of $(\mathbf{A} - \mathbf{LC})$ where evenly dispersed around a half circle in the left half plane with a varying magnitude as shown on the x-axis.

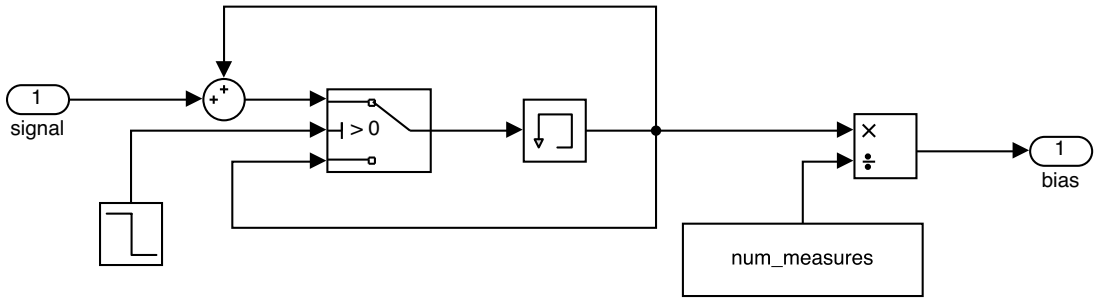


Figure 17: Simulink implementation of a calibrator to set the correct IMU bias.

4 Part IV - Kalman Filter

4.1 Noise estimate

In this final part, we implemented a discrete time Kalman filter to improve our state estimation with the help of the Kalman gain. The Kalman gain will give us the optimal estimate, which will minimize the mean-square error of our estimate. Our state-space system can be described by the following discrete-time model expressed in the lab preparation [2].

$$\mathbf{x}[k+1] = \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_d\mathbf{u}[k] + \boldsymbol{\omega}_d[k] \tag{20a}$$

$$\mathbf{y}[k] = \mathbf{C}_d\mathbf{x}[k] + \boldsymbol{\upsilon}_d[k] \tag{20b}$$

$$\boldsymbol{\omega}_d \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_d),\ \boldsymbol{\upsilon}_d \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_d) \tag{20c}$$

$\mathbf{A}_d$, $\mathbf{B}_d$ and $\mathbf{C}_d$ is the discretized state space system which is described in the next section. $\boldsymbol{\omega}_d[k]$ and $\boldsymbol{\upsilon}_d[k]$ describe the random disturbances which acts upon our system, and noise, which affects our measurements. As the Kalman filter assumes that the measurement and plant is affected by zero mean white Gaussian noise/disturbances, we need to make an estimate of the measurement covariance, $\mathbf{R}_d$. Generating a time-series with the IMU measurements while the helicopter is laying still at the ground, and while it is at the linearization point, we can inspect the noise in the measurements. We can also get an estimate for the autocovariance matrix $\mathbf{R}_d$ by calculating the covariance between the measurements. This was done with the MATLAB command $cov(x)$.

Laying still on the ground, the assumption of the noise being white seems reasonable, as we see no bias (after we have calibrated the IMU), the measurements does not drift and there does not seem to be any correlation between the values. However, when the motors are running and we are at the equilibrium point, we see a lot of low frequency oscillations in the signals, and they are much more noisy, which can be seen in figure 18. This indicates that the assumption of white noise might not be entirely correct, as one can see some correlation between the values at each time step, but the assumption is still sensible for this lab. It is also worth noting that we had to physically hold the helicopter at the travel axis to prevent it from spinning while at the equilibrium point, as the helicopter in theory should not be moving here, but either small differences in the motors thrust or unbalanced weight, or a pitch encoder slightly off from horizontal resulted in rotation around the travel axis. If we had not kept the helicopter still, we might have had a unrealistically high covariance between the travel rate measurement and the other signals.

Comparing the two estimated covariance matrices, we see that the covariances while running are many orders of magnitudes larger than the one we got with the helicopter laying still. This agrees well with the fact that the signals looked much more noisy while the helicopter was running. Our experiments gave us this matrix for $\mathbf{R}_d$ which stems from measurements while at the equilibrium point

$$\mathbf{R}_d = 10^{-4} \times \begin{bmatrix} 58 & 50 & 39 & -58 & -3 \\ 50 & 55 & 47 & -53 & -2 \\ 39 & 47 & 207 & -64 & 3 \\ -58 & -53 & -64 & 78 & -2 \\ -3 & -2 & 3 & -2 & 9 \end{bmatrix}.$$

Continuing the discussion on whether it is a good assumption to call the measurement noise white or not, we later investigated a little bit further. As we mentioned earlier, the measurements from the IMU showed some oscillatory behaviour while at the equilibrium point. We figured out that we could get an estimate for the power spectral density of the different signals with the help of the MATLAB function $periodogram(x)$, which can be seen in figure 18. This revealed that most of the signals seemed to be affected by a frequency of around 80Hz while at the equilibrium point, which could indicate that noise shaping would be beneficial. However, we do not know enough about this noise to make any conclusions. Firstly, we have not checked if this frequency stays constant when controlling the helicopter, as we do not know if the disturbance stems directly from vibrations caused by the motors and unbalanced propellers, or if the frequency only stems indirectly from the motors, and there is some sort of resonance in the helicopter structure which causes the frequency, or some other phenomenon. It does however not look like the spikes in spectral density functions are multiples of each other, as the next significant peak is at around 190Hz, which might rule out resonance, as this tends to give smaller peaks, that are a multiple of the first peak. Another possibility is that there is some disturbance like the 3P-frequency disturbance that wind turbines experience. If this is the case, a rough estimate suggest that the motor has to spin around with 1700rpm. We do not know if this is reasonable, and further testing is necessary to draw any conclusions. For now we will assume that the measurement noise is white.

4.2 Discretization & Implementation

We now utilize all states giving us $\mathbf{x} = [p, \dot{p}, e, \dot{e}, \lambda, \dot{\lambda}]$, but since λ is not measurable when using the IMU, we get the following matrices

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ K_3 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & K_1 \\ 0 & 0 \\ K_2 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

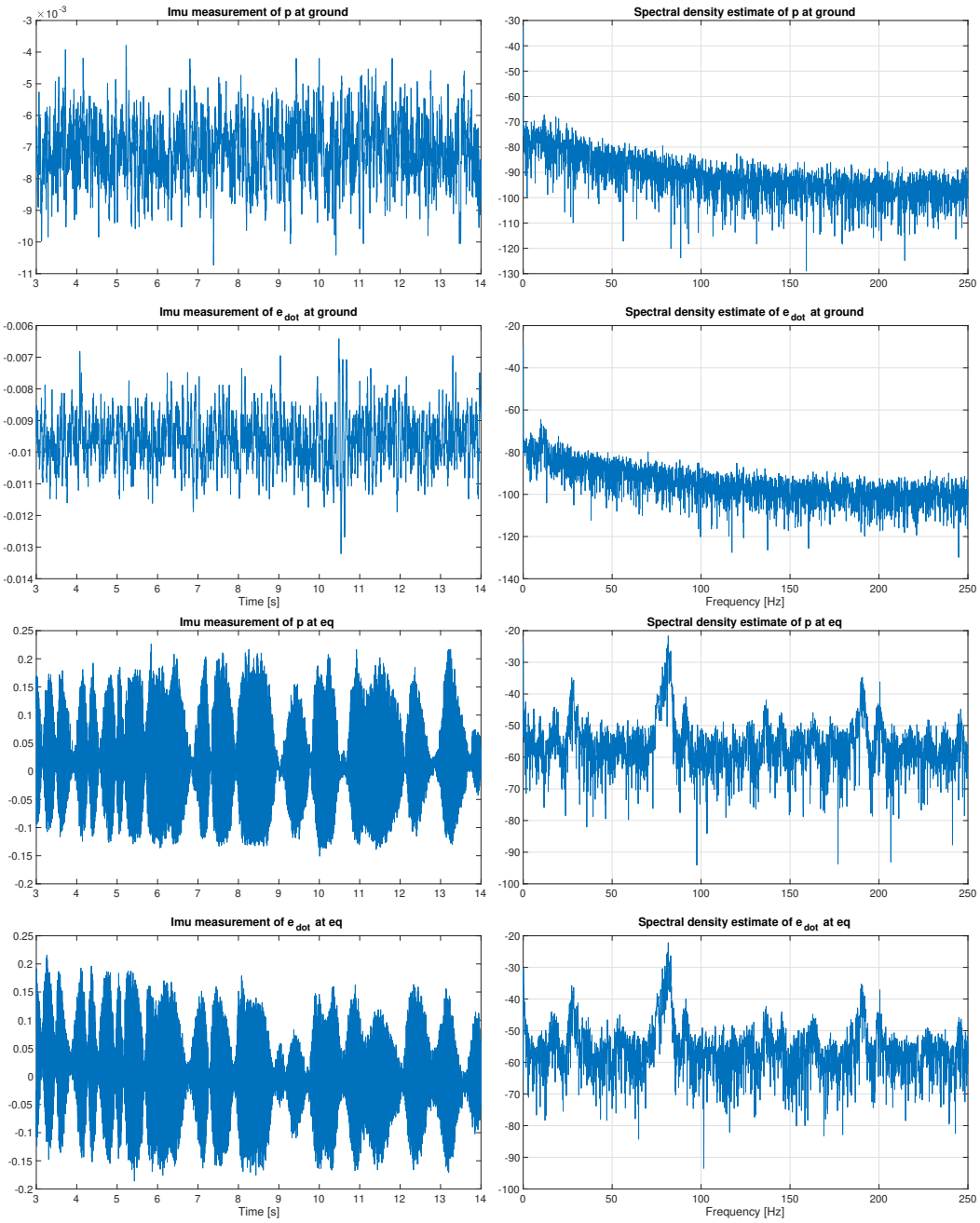


Figure 18: Plot showing pitch and elevation rate from IMU measurements after transformation, using estimated pitch and elevation as input to the transformation, laying still on the ground and at the equlibrium point. Measurements are in rad and rad per seconds, while the spectral density estimates are in decibels.

Furthermore, as we wish to implement a discrete Kalman filter, we need to discretize our state space system using a time-step T of 0.002s, as is used in the Simulink program. This we achieved by using the MATLAB function $c2d(sys, Ts)$. We then realized our discrete Kalman filter by implementing a correction and prediction functions according to the equations found in the lab preparation [2]. This can be seen in listing 1 and 2. The Simulink implementation can be seen in figure 19 and 20. The only difference from a standard discrete time Kalman filter is that we only correct our predictions when the IMU has new data, as it runs on a slower frequency.

When we first performed the lab exercises, we were not sure about what the initial states $\overline{\mathbf{x}}[0]$ and $\overline{\mathbf{P}}[0]$ should be set to. Therefore both variables were left as zeros for most of the time. The consequences of this will be discussed later.

Listing 1: Code used in correction function block in Simulink system in figure 19

```
%correction function
function [x_hat, P_hat] = fcn(x_bar, P_bar, y, new_data, C_d, R_d)
n=6;

if new_data == 1
    L = P_bar * C_d' / (C_d * P_bar * C_d' + R_d);

    x_hat = x_bar + L * (y - C_d * x_bar);
    P_hat = (eye(n) - L * C_d) * P_bar * (eye(n)
          - L * C_d)' + L * R_d * L';

else
    x_hat = x_bar;
    P_hat = P_bar;

end
```

Listing 2: Code used in prediction function block in Simulink system in figure 20

```
%prediction function
function [x_bar, P_bar]= fcn(x_hat, P_hat, u, A_d, B_d, Q_d)

x_bar = A_d * x_hat + B_d * u;
P_bar = A_d * P_hat * A_d' + Q_d;
```

4.3 Experimentation

After implementing the discrete time Kalman filter, we started experimenting as before by setting the values of the diagonal of $\mathbf{Q}_d$ to 1 and used the encoders for feedback for our regulator and inspected the estimate from the Kalman filter. We also used the pitch and elevation angle from the IMU in the transformation of the gyro rates, as we also did

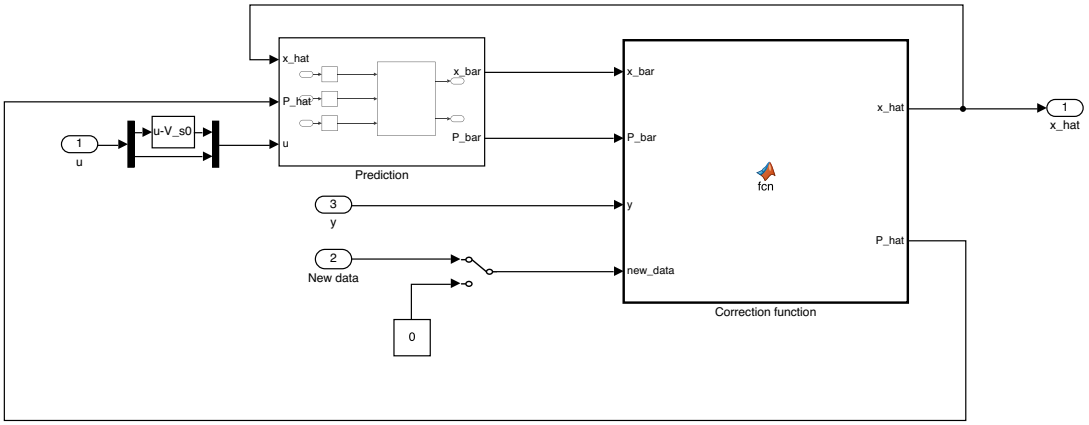


Figure 19: Simulink implementation of Kalman filter.

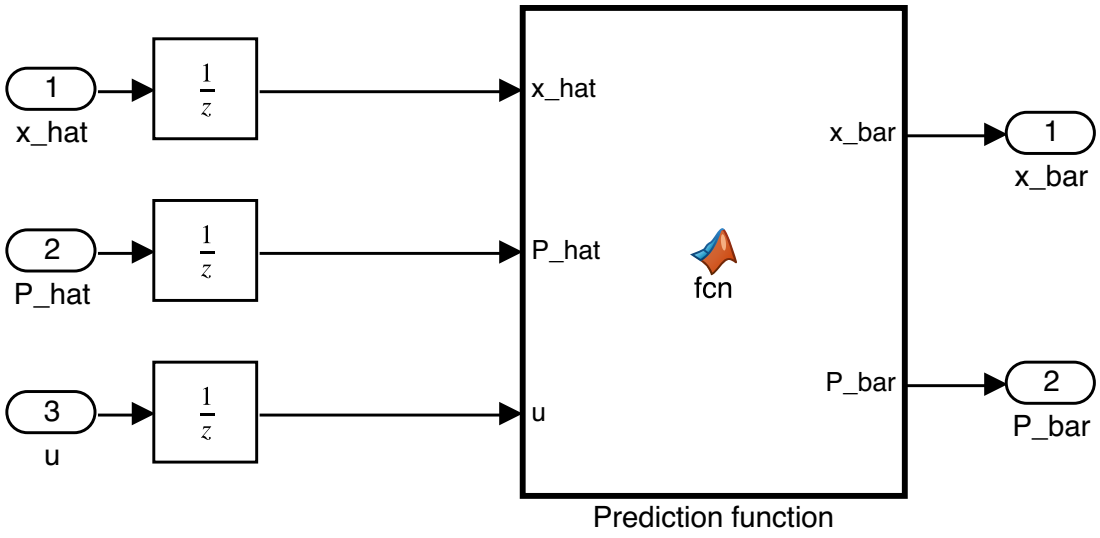


Figure 20: Simulink implementation predction step.

in part 3. The estimate was at first very noisy, which indicates that our model has a high uncertainty compared to the uncertainty in the measurements. As $\mathbf{Q}_d$ describes the variances for the disturbance acting on the model, it seems that setting the diagonal equal to infinity, would be the same as completely ignoring the model, as the Kalman filter would not be able to get anything useful out of the model with such high uncertainty. This would then make our estimate rely only on the measurements. Inversely, setting the diagonal of $\mathbf{Q}_d$ to 0, would be the same as saying that there is no disturbance acting on our system, which would make our estimate perfectly correct relying only on the model, having no need for corrections from the measurements. This was confirmed by experiments and helped us further improve our tuning.

We then tested turning the IMU new data signal off while running. This made the estimated states run off from the true values. This makes sense since disconnecting the new data signal, is basically just turning the correction step off, leaving us with an open loop estimator. Turning the signal on again, we can see that the Kalman filter recovers nicely, as long as $\mathbf{Q}_d$ is reasonable. It is also worth noting that since we only correct our predictions when the IMU has new data, we will in principle rely more on the model, compared to what the $\mathbf{Q}_d$ and $\mathbf{R}_d$ signify. However we do not think this will be a problem as long as the IMU is not significantly slower.

The main difference between the Luenberger observer and the discrete time Kalman filter, besides the obvious difference between a continuous and discrete system, is that the Kalman filter models and takes into account the noise affecting the measurements and the disturbance acting upon the model, producing an optimal Kalman gain that minimizes the trace of the error covariance matrix $\mathbf{P}$. This again results in the Kalman gain being time dependent until it converges, while the Luenberger gain is constant. If we were to only compute the steady state kalman gain however, we would in theory be able to achieve the same performance with a Luenberger observer, if we were able to match the steady state kalman gain with pole placement. The Kalman filter also has the benefit of making it easier to decide how much to rely on each measurement, by tuning the corresponding $\mathbf{R}_d$ and $\mathbf{Q}_d$, compared to dealing with pole placements which feels more abstract.

Looking at figure 21 it appears that the Luenberger observer is tuned to be slightly faster, which again results in more noise. To compare the two estimators, we calculated the mean squared error of the estimator versus the encoder measurements. This gave us that the MSE of the Luenberger is 3.8006×10^{-3}, and 6.3093×10^{-3} for the Kalman filter. Surprisingly, the Luenberger observer performs better than the Kalman filter, even when correct for the startup phase of the Kalman filter, as we used wrong inital values. Our best explanation for the better performance of the Luenberger, is simply that we spent a lot more time tuning the Luenberger observer, than we did with the Kalman filter, and of course, that used the wrong startup values.

$$MSE = \frac{1}{N_d} \sum_{k=1}^{N_d} \|\mathbf{C}_d \hat{\mathbf{x}}[k] - \mathbf{y}_{enc}[k]\|_2^2$$

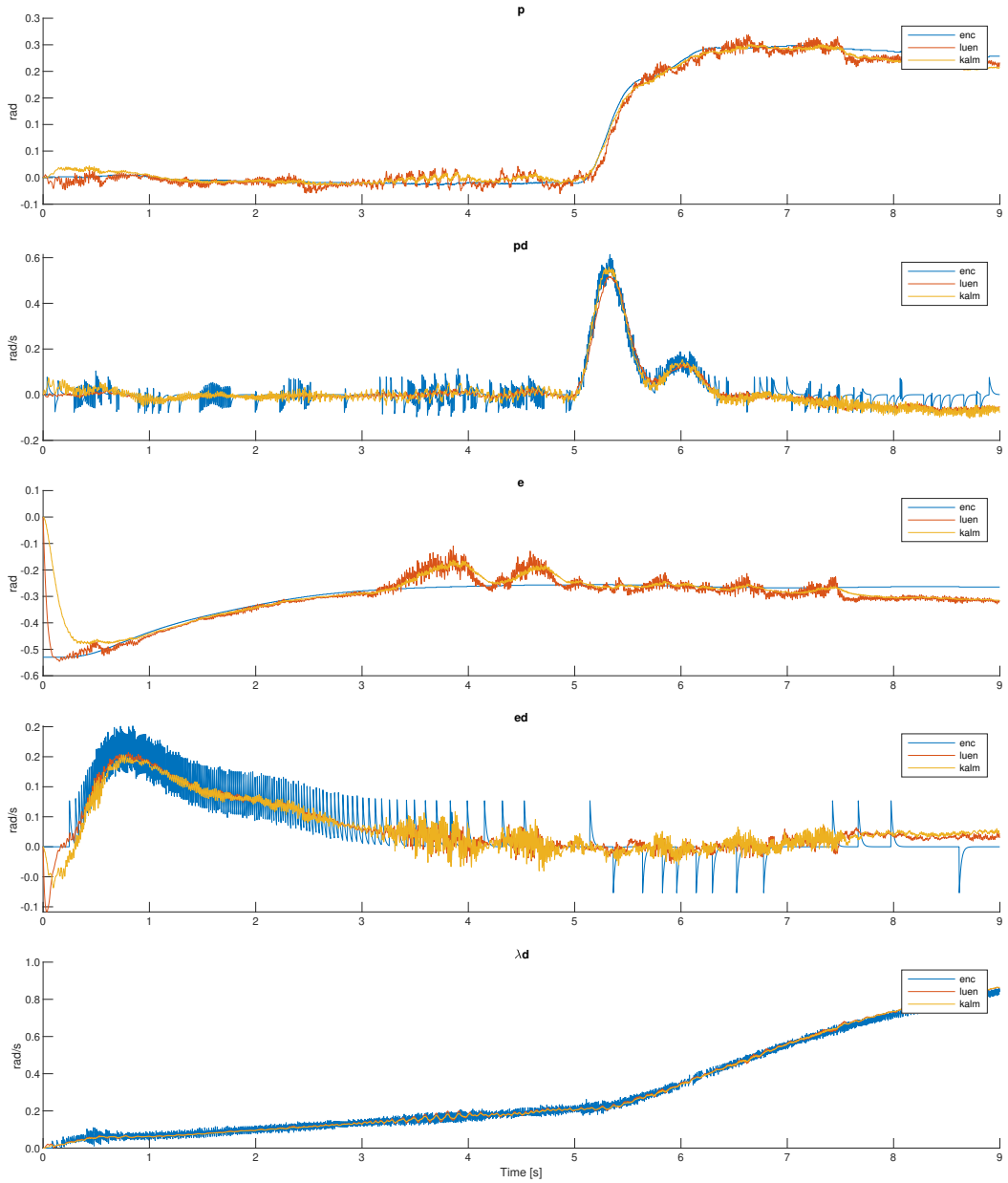


Figure 21: Plot comparing the performance of our best experimentally tuned Kalman filter and Luenberger observer, plotted against the encoder values. Please note that during experimentation we failed to initialize the initial values of the Kalman filter correctly.

4.4 Tuning

Tuning the values on the diagonal in the $\mathbf{Q}_d$ matrix felt a bit fiddly, and it was hard to observe if a tuning was better than another. Nevertheless, we ended up with a somewhat satisfactory tuning, meaning that we could try to run the regulator on the estimate, which went surprisingly well. We could then further tweak our parameters until we were satisfied. Finally we ended up with

$$\mathbf{Q}_d = 10^{-5} \times \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}.$$

Later on, when analyzing the data we captured during the testing, we realized that since we had captured both the encoders values, IMU values, and the output from the regulator while the helicopter was running on the encoders, we could simulate the Kalman filter, feeding in captured data and comparing with the encoder data. This would allow us to much more easily test out different values for the $\mathbf{Q}_d$ matrix, and calculate the MSE, mean squared error, of the estimate versus the encoder values as a measure of performance. Even better, since we are in a position were we have encoder measurements, which has much more desirable characteristics, we could try to "fit" our estimator to the encoder, and use MSE as a way of tuning $\mathbf{Q}_d$.

Starting with the $\mathbf{Q}_d$ matrix found during experimentation, we could in a rather brute-force way test out combinations of up and down scalings of each of the values in the diagonal in matrix in a script. We could then continue this search process by selecting the $\mathbf{Q}_d$ matrix giving the minimum mean squared error and then repeat the search process by scaling the values in this new matrix up and down. Each time the program preferred its original $\mathbf{Q}_d$, we reduced the scaling factor. Eventually when the script no longer chose a new $\mathbf{Q}_d$ over its starting matrix, we had in principle found a local minimum, which hopefully resulted in an optimal tuning for the discrete Kalman filter. Whether this is true or not we dare not to say, however, the fact that the script did not diverge towards 0 or infinity gives us a little confidence in the script.

In the end the script gave us

$$\mathbf{Q}_{d,sim} = 10^{-5} \times \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3162 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0316 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.0032 \end{bmatrix},$$

which is not far off from the one we started with. This gives us reason to believe that our tuning is not half bad, if we trust the simulation values. We have sadly not had

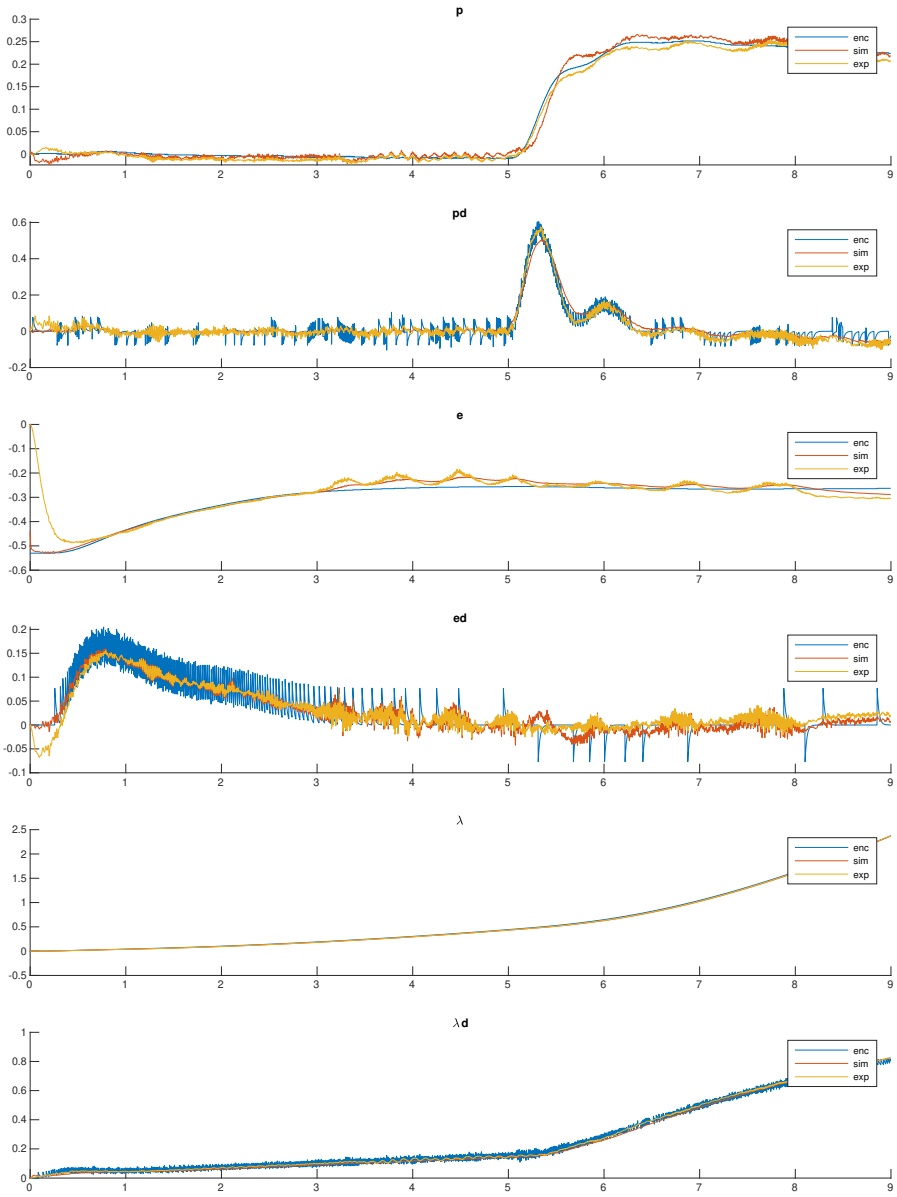


Figure 22: Plot showing the performance of our best experimentally tuned Kalman filter, plotted against the encoder values and as well our best tuning found from simulations. Please note that the during experimentation we failed to initialize the initial values of the Kalman filter correctly.

the opportunity to test our optimal $\mathbf{Q}_d$ matrix from the simulations, which gives us no information about how well the helicopter flies when running on the estimator with this $\mathbf{Q}_d$ matrix. As our simulation does not take into account the new data signal, there is also a reason to believe that the $\mathbf{Q}_d$ found in the simulations are not directly transferable to reality, however this could easily be fixed. The best measure of performance we have, is 22 figure, which shows the encoder values compared to the kalman filter with our experimental $\mathbf{Q}_d$, which is values gathered at the lab, and our simulated kalman filter with the optimal $\mathbf{Q}_d$. In most of the states, the simulation looks less noisy, which makes sense since $\mathbf{Q}_d$ from the simulation either opted for the same $\mathbf{Q}_d$ value as from experimentation or a smaller one, giving less variance for the disturbances acting on the model, which in turn give the kalman filter more trust in the model. This might be bad for flight performance as the kalman filter will be slower.

Earlier we said that we said that we failed to initialize the starting values correctly during experimentation, however, when we started testing out simulating the discrete time Kalman filter we realized that we had made a mistake. We now understood that the covariance matrix $\overline{\mathbf{P}}[0]$ represented the variance of our initial state $\overline{\mathbf{x}}[0]$, in other words, the uncertainty in our initial state. We also became aware of the fact that elevation should be set to -0.53 in our initial state $\overline{\mathbf{x}}[0]$. This became clear when we plotted our best tuning from simulation up against the experimental data as seen in 22. Here we see that the kalman filter from the lab uses a little less than a second to get its elevation estimate back on track, while the simulation starts out with the correct start value.

As the the helicopter always starts firmly lying on the table, the diagonal of the covariance could be initialized with fairly small estimated variances. Nevertheless, an uncertainty of 0 seemed to work during experimentation, but we guess that failing to initialize the covariance matrix results in worse performance of the estimator until it converges, as the filter wrongly assumes that the model is much more reliable than it really is. It is still interesting to see that the Kalman filter works so well, even when initialized wrong.

We also think that the correct initialization of $\overline{\mathbf{x}}[0]$ is critical if one tries to achieve a good tuning. This we can see from the table 1, were the best tuning found during simulation with correct initialization, actually performs worse than than our experimental tuning, if it is initialized wrongly. This can be explained by the fact that the $\mathbf{Q}_d$ matrix found during simulation arrived at lower values, which in turn means that the Kalman filter has a higher trust in the model compared to the measurements. If you then initialize the Kalman filter wrongly, it will use longer time getting back on track with the measurements.

To give our $\mathbf{Q}_d$ found at the lab a fair chance competing against the $\mathbf{Q}_d$ found during simulation, we opted to simulate the experimentally found $\mathbf{Q}_d$ as well with the correct initial values. For comparison, the MSE of the wrongly initialized Kalman filter recorded during lab is 5.4544×10^{-3}, which is not far off from the simulation. In table 1 you can see the MSE of the error between the measurements from the encoder and the two Kalman filter simulations receiving IMU measurements.

Table 1: MSE for different Q_d matrices, simulated

$diag(\mathbf{Q}_d)$	uncorrected MSE	corrected MSE
$10^{-5} \times [0.1, 10, 1, 1, 0.01, 0.1]$	5.2149×10^{-3}	2.4428×10^{-3}
$10^{-5} \times [0.1, 0.3162, 0.0316, 1, 0.01, 0.0032]$	18.1545×10^{-3}	2.1989×10^{-3}

Table 2: Table showing the achieved MSE values during simulation, both for the the experimentally found $\mathbf{Q}_d$ (top one), and the one found during using a script and simulation. We show the MSE when we use the wrong initial values to be able compare better with our data collected at the lab, as well as the MSE when correctly initialized to show what we could have achieved.

5 Conclusion

During this helicopter lab we have examined different techniques for controller and observer design. We began with a simple pd-regulator for pitch and learned how different pole placements caused different dynamics. We found that poles with high magnitudes and high angles corresponded to shorter settling time, but also introduced more oscillations. Then we wanted to control a system with multiple inputs and multiple outputs, so we put our system on state space form. This allowed us to use state feedback to control the helicopter. We saw, however, how it was difficult to choose the right feedback gain matrix. The LQR algorithm was used to set the feedback gain as to minimize a quadratic performance index. We learned how we could increase a weight in the $\mathbf{Q}$ matrix to prioritize the corresponding state, or increase a term in $\mathbf{R}$ to limit the control signal. All in all both pole placement and LQR required considerable trial and error to achieve a good tuning. Integral action was added to make up for modeling errors and constant disturbances, and successfully removed steady state errors.

The rest of the time on the lab was used to create different state estimators that used our model and an IMU as input to generate an estimate of the states. Similar to what we did with the controller in the first half of this lab, the poles of our state estimator was first set manually and then in an optimal fashion. For the manual case, we saw how placing the poles further out into the left hand plane correspond with more trust in the measurement, and how keeping them close correspond to more trust in the model. We then calculated the mean square error between the estimator and encoder for different pole placements, and found a minimum that gave good results. Lastly the Kalman filter algorithm was utilised. We began this part by analysing the measurement noise to see if it was applicable for the assumptions of white noise. We found that the assumption of the noise being white seemed reasonable, but when the helicopter was running at its equilibrium point, there were considerable spikes in the noises power spectral density function. Noise shaping was discussed but not implemented as a solution to this. When experimenting with the Kalman filter model uncertainty matrix $\mathbf{Q}_d$, we found that high values correspond with more trust in the measurement and low values more trust in the model. Trial and error was utilised to find a good tuning of the $\mathbf{Q}_d$ matrix. A simple script was also utilised to find the $\mathbf{Q}_d$ that minimized the mean square errors between the encoder and estimated values based on collected data. The obtained value matched well with our trial and error tuning suggesting that it was reasonable. The Kalman filter ended up giving us a bit worse performance than pole placement with the Luenberger observer, at least in terms of mean square errors between the encoder and estimated values, but it was considerably easier to tune and a lot more time was spent fine tuning the Luenberger observer.

References

[1] Chi-Tsong Chen. *Linear System Theory and Design*. Oxford University Press, Incorporated, 2014.

[2] NTNU Department of Engineering Cybernetics. *Helicopter lab preparation*.