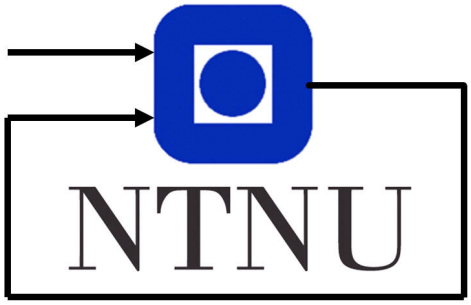
Helicopterlab TTK4135

Student 592084
Student 592082

March 21, 2025



Department of Engineering Cybernetics

Contents

1 10.2 - Optimal Control of Pitch/Travel without Feedback

1.1 The continuous model

To get the system on continuous time state space form:

$$\dot{\mathbf{x}} = A_c\mathbf{x} + B_c u. \tag{1}$$

One must look at the model derivation section in the lab exercise. We are given:

$$\mathbf{x} = \begin{bmatrix} \lambda & r & p & \dot{p} \end{bmatrix}^T, \tag{2}$$

$$u = p_c \tag{3}$$

and from the model derivation section:

$$\ddot{p} + K_1K_{pd}\dot{p} + K_1K_{pp}p = K_1K_{pp}p_c, \tag{4}$$

$$\dot{\lambda} = r, \tag{5}$$

$$\dot{r} = -K_2 p. \tag{6}$$

This will result in the system matrices:

$$\mathbf{A}_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} \end{bmatrix}, \qquad \mathbf{B}_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp} \end{bmatrix}. \tag{7}$$

We are modeling the pitch and travel of the helicopter in addition to modeling the optimal input. This relates to figure 7 in the exercise as u go from optimization into the regulators which then go into the plant [1].

1.2 The discretized model

To write the system on discrete-time state space form:

$$\dot{\mathbf{x}}_\mathbf{k} = A_k\mathbf{x}_\mathbf{k} + B_k u_k \tag{8}$$

It is necessary to discretize it. The Euler forward method is given by:

$$x_{k+1} = x_k + T_s f(x) \tag{9}$$

$$x_{k+1} = x_k + T_s(A_c x_k + B_c u_k) \tag{10}$$

$$x_{k+1} = (I + T_s A_c)x_k + T_s B_c u_k. \tag{11}$$

It is then possible to model $\boldsymbol{A_k}$ and $\boldsymbol{B_k}$ as:

$$A_k = I + T_s A_c \tag{12}$$
$$B_k = T_s B_c. \tag{13}$$

This was done in line 3 and 4 of 1. And became:

$$\mathbf{A}_k = \begin{bmatrix} 1 & 0.250 & 0 & 0 \\ 0 & 1 & -0.1414 & 0 \\ 0 & 0 & 1 & 0.250 \\ 0 & 0 & -0.8100 & 0.1000 \end{bmatrix}, \qquad \mathbf{B}_k = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.8100 \end{bmatrix}. \tag{14}$$

1.3 The open loop optimization problem

The open loop optimization problem is formulated as:

$$\phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q p_{ci}^2, \quad q \geq 0. \tag{15}$$

This is the cost function that should be minimized. It is therefore clear that a larger q will make the system prioritize low energy usage. While a smaller q will make the system prioritize getting the travel to the reference.

1.4 The weights of the optimization problem

The experiment was carried out with weights of 0.12, 1.2, and 12. The results can be seen in figure 1, 2 and 3 respectively. It can be noted from equation 15 in the exercise sheet that the cost function to be minimized makes q 'punish' power usage [1]. An increase in q will therefore make the system use less power. This can be noted from the results where there is considerably less overshoot for larger q-values.
For the daring, questions:
Starting the helicopter away from it reference is an approach to get the system to the reference. Still, since we are minimizing the squared of the difference from the reference, the same effect will appear if we increase the weight for the travel so that it "punishes" not reaching the reference. In other words, starting the helicopter away from the reference is not a surefire way of making the system reach the reference faster. Since it is the relative weight between the error and the input that decides how fast the convergence should be.
The scaling between the error term and input term is where the weights are relevant. Since the weight is 1 on the error, a larger error will make the helicopter reach the reference faster, depending on how large the q value for the input is, because it is the relative weight between the error and the input, which decides how fast the system converges.

Furthermore, the system could be created in other ways, it could introduce terminal constraints to make sure it doesn't go outside the boundaries set. The use of modulo to solve the problem of being one extra way around could lead to two errors, if you are close to N*360 degrees, you would perform a large input even though you are close to the reference. In addition, it could lead to discontinuities, which might be a problem if the solver is gradient-based.

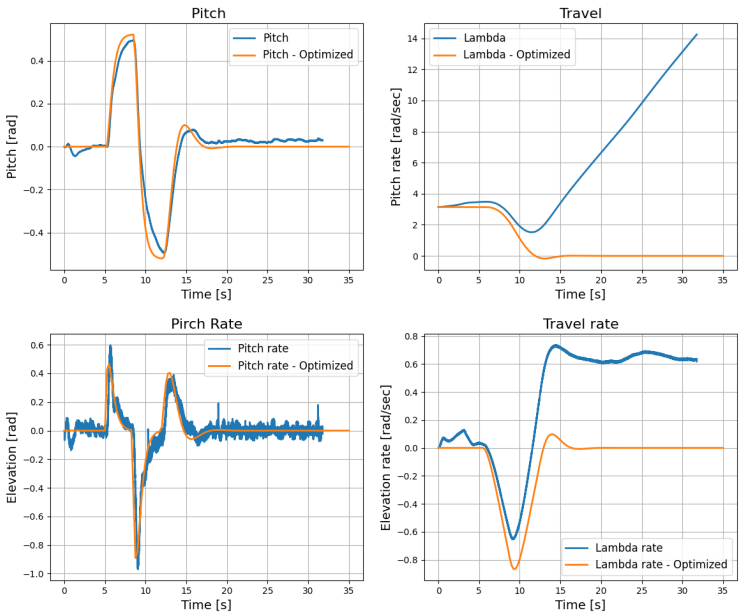


Figure 1: q=0.12

1.5 Experimental results

As can be seen from the figures 1, 2 and 3, the real values from the experiment follows the optimized to some degree. It is quite clear, however, that the travel and travel rate of the real system do not follow the optimized path very well. This is a result of the lack of feedback in the system. A small deviation in the real system from the model will make the helicopter spin at an almost constant rate. Causing an increased deviation over time. As for the pitch, the real values follow the optimized path to a much larger degree than the travel. This is expected due to the feedback of the pitch.

1.6 MATLAB and Simulink

Listing 1: Code for day 2

```
1
2  %Discrete system model
3  A1 = eye(4) + delta_t*[[0 1 0 0];[0 0 -K_2 0]; [0 0 0
       1];[0 0 -K_1*K_pp -K_1*K_pd]];
4  B1 = delta_t*[0 0 0 K_1*K_pp]';
5
6  % Initial values
7  x1_0 = pi;                                  % Lambda
8  x2_0 = 0;                                   % r
9  x3_0 = 0;                                   % p
10 x4_0 = 0;                                   % p_dot
11 x0 = [x1_0 x2_0 x3_0 x4_0]';                % Initial values
12
13 % Time horizon and initialization
14 N  = 100;                 % Time horizon for states
15
16 % Bounds
17 ul = -(60*pi)/360;        % Lower bound on control
18 uu = (60*pi)/360;         % Upper bound on control
19
20 % Generate constraints on measurements and inputs
21 [vlb,vub]  = gen_constraints(N,M, xl, xu, ul, uu);
22
23 % Generate the matrix Q and the vector c (objective
       function weights in the QP problem)
24 Q1 = zeros(mx,mx);
25 Q1(1,1) = 1;             % Weight on state x1
26 Q1(2,2) = 0;             % Weight on state x2
27 Q1(3,3) = 0;             % Weight on state x3
28 Q1(4,4) = 0;             % Weight on state x4
29 P1 = 12;                % P1 - Weight on inputs
30 Q = gen_q(Q1, P1, N,M);  % Generate Q, hint: gen_q
31 c = 0 ;                  % Generate c
32
33 %% Generate system matrixes for linear model
34 Aeq = gen_aeq(A1, B1, N, mx,mu); % Generate A, hint:
       gen_aeq
35 beq = zeros(N*mx,1);             % Generate b
36 beq(1:mx) = A1*x0;
37
38 %% Solve QP problem with linear model
39 [z,lambda] = quadprog(Q,[],[],[],Aeq,beq,vlb,vub) ;
40
41 %To Simulink
42 ts_lambda_ref = timeseries(u,t);
```

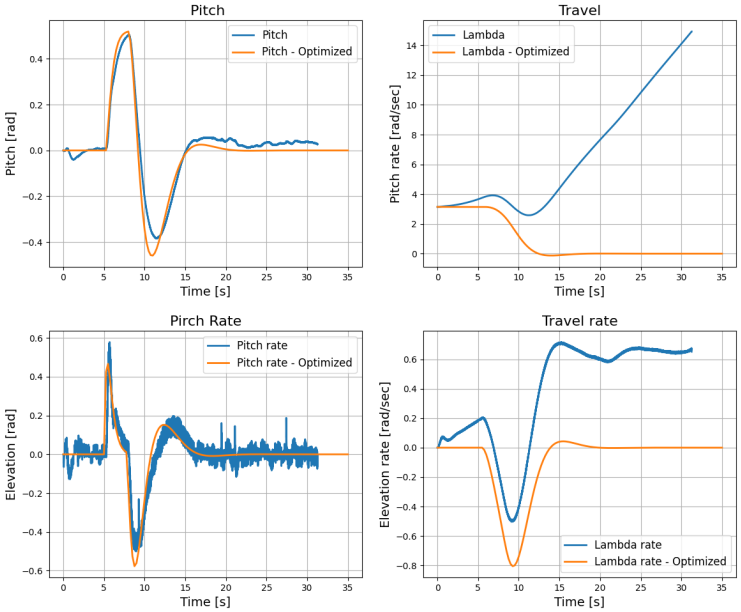


Figure 2: q=1.2

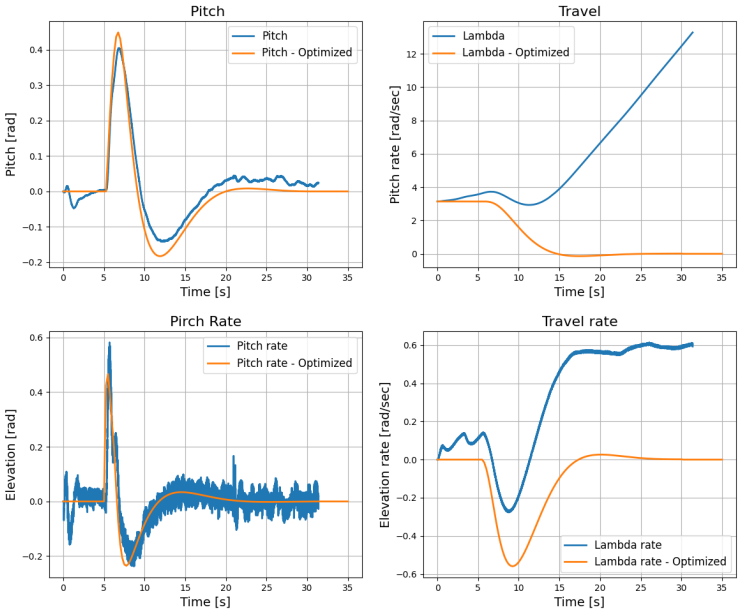


Figure 3: q=12

2 10.3 - Optimal Control of Pitch/Travel with Feedback (LQ)

2.1 LQ controller

For this section of the experiment a linear quadratic(LQ) controller was implemented. An LQ controller aims to minimize a quadratic cost function:

$$J = \sum_{i=0}^{\infty} \left(\Delta x_{i+1}^{\top} Q \Delta x_{i+1} + \Delta u_i^{\top} R \Delta u_i \right), \quad Q \geq 0, \quad R > 0, \tag{16}$$

where Δx and Δu is:

$$\Delta x = x - x^*, \tag{17a}$$
$$\Delta u = u - u^*. \tag{17b}$$

When minimizing this cost function, one can prove that the solution given by:

$$u_t = K_t x_t. \tag{18}$$

Where the feedback matrix is derived by the Discrete Ricatti equation:

$$K_t = R_t^{-1} B_t^T P_{t+1} (I + B_t R_t^{-1} B_t^T P_{t+1})^{-1} A_t, \quad t = 0, \ldots, N-1 \tag{19}$$
$$P_t = Q_t + A_t^T P_{t+1} (I + B_t R_t^{-1} B_t^T P_{t+1})^{-1} A_t, \quad t = 0, \ldots, N-1 \tag{20}$$
$$P_N = Q_N [2]. \tag{21}$$

In this cost function 16, an increase in Q will increase the contribution of Δx. Causing the system to punish deviation from the optimized path. An increase in R will, on the other hand, increase the contribution of ΔU, causing the system to punish a deviation of the power usage. It is the ratio of the Q and R matrices that matter. It was therefore tested how the system would react with equal Q and R matrices and with a ratio difference of 10. This was done to observe the different behavior of the system when prioritizing

2.2 Model Predictive Control

To further investigate effective ways to optimize the helicopter path and model predictive control(MPC) regulator was implemented. An MPC optimizes an input sequence, but unlike an open loop system, it can take feedback and re-calculate the input sequence based on the real system. The system can then use the first calculated input, then measure the state, and then re-optimize. This can lead to much better performance because the feedback takes inaccuracies in the model into account. On the other hand, an MPC implemented this way takes a lot of computational power.

Alternative: To implement the MPC, one would then solve the optmization problem many times, and use the first input of the input sequence which the optimization problem created. The positives of using this approach over LQ + predicition is that it is easy to handle constraints, it is robust for disturbance since we reoptimize in every time step. On the other hand, this comes at a price of heavy computation and a fast optimizing.
The physical implementation wouldnt be very different from the current block diagram. One would need to add a loop from the plant and into the optimization step to be able to always use the newest state.
The other approach is based on planning once, and then using another MPC to make sure we can stay close to the predicted trajectory. This would need fewer calculations, but it might also be less robust for disturbances, if we our model, predicted path, or MPC is badly designed.
One way to use both of these could be to do periodic planning. Reoptimize the full trajectory as in A, fewer times. Then one could use the other MPC in between the planning, as in B, to make sure we dont stray away from the predicted trajectory.

2.3 Experimental results

The results from the experiment can be seen in figures 4 to 9. Test 1 was done with Q = R= 1, test 2 was Q = 1 and R = 10, test 3 was Q = 10 and R = 1. As mentioned this was done to see the difference in system behavior when punishing system error or power usage. The system acted as expected, as it punished deviation from the Δu for a larger R and Δx for larger Q. The difference is, however, smaller than expected. This is due to the small changes in K. For test 1: $K = [-0.648, -2.447, 1.312, 0.4625]$, while for test 2: $K = [-0.2808, -1.3760, 0.7313, 0.2156]$.

2.4 MATLAB and Simulink

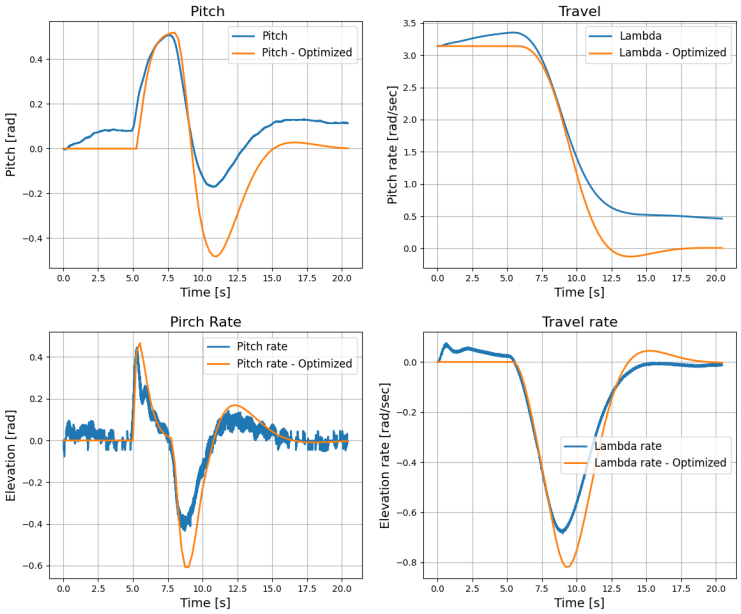


Figure 4: LQ test 1

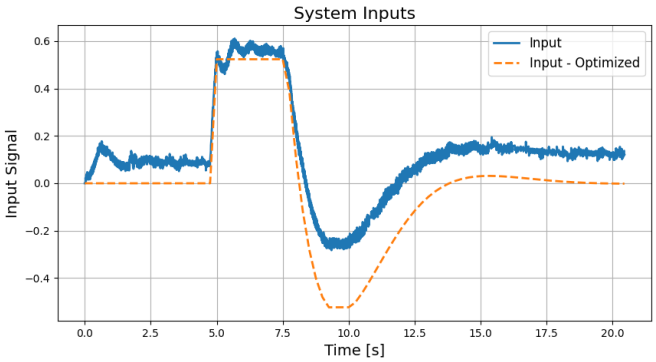


Figure 5: LQ test 1, inputs

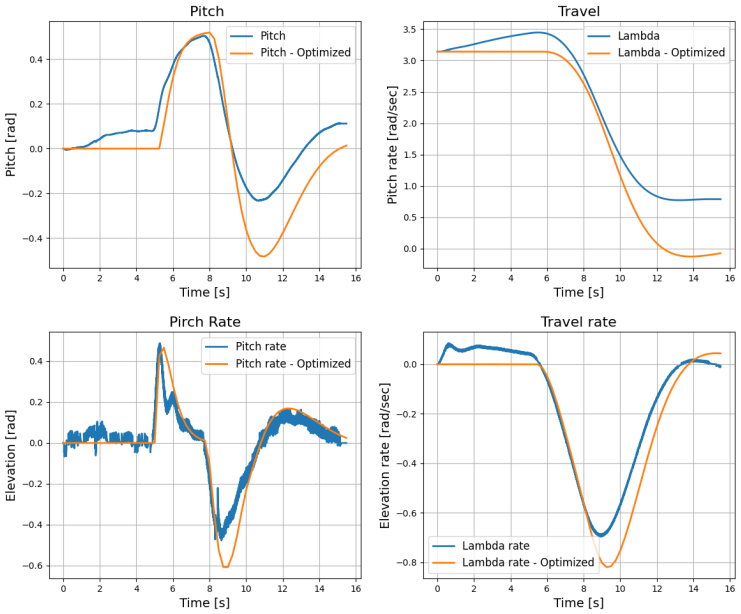


Figure 6: LQ test 2

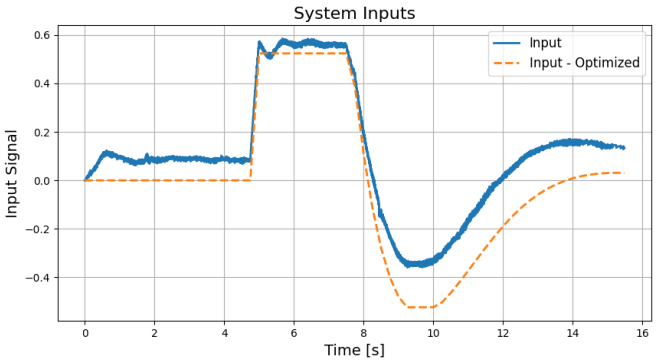


Figure 7: LQ test 2 inputs

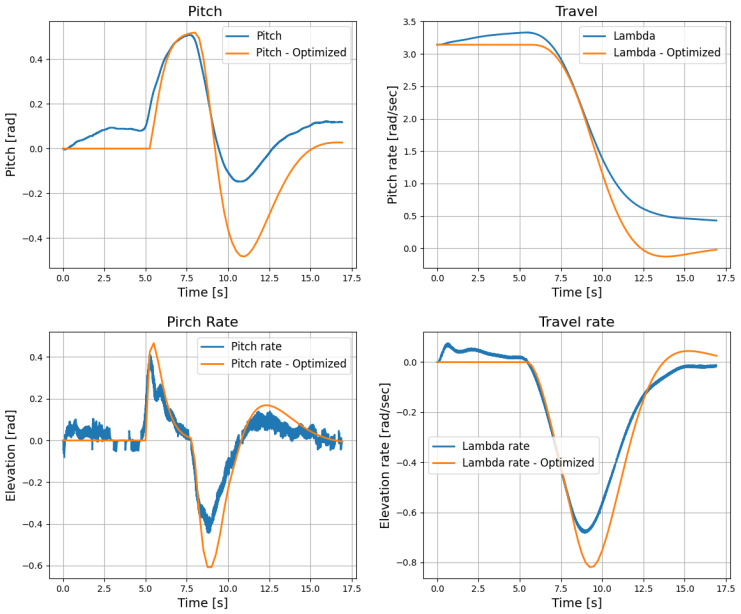


Figure 8: LQ test 3

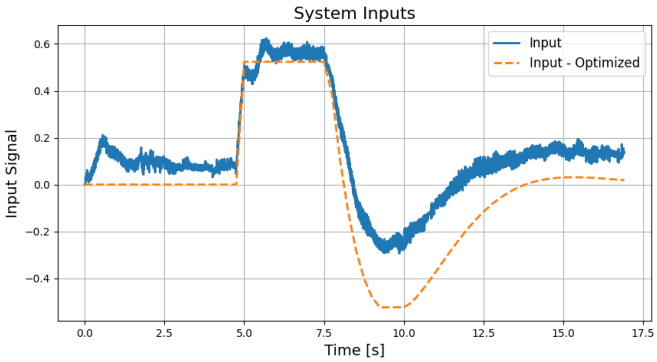


Figure 9: LQ test 3 inputs

Listing 2: Code for day 3

```
%Oppgave 10.3.1 - 1
delta_t = 0.25; % sampling time

A1 = eye(4) + delta_t*[[0 1 0 0];[0 0 -K_2 0]; [0 0 0
    1];[0 0 -K_1*K_pp -K_1*K_pd]];
B1 = delta_t*[0 0 0 K_1*K_pp]';

q=[10 10 100 100];
r=[1];

Q_raaa=diag(q);
R_raaa=diag(r);


K=dlqr(A1,B1,Q_raaa,R_raaa);
```

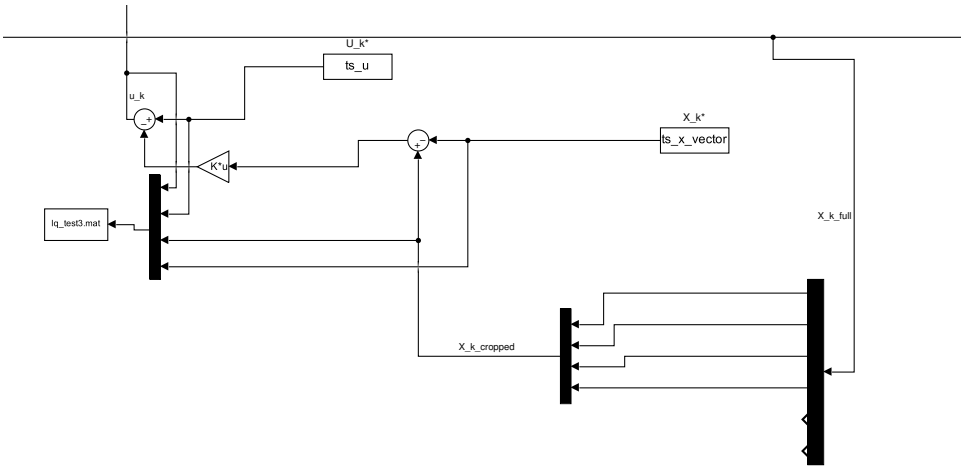


Figure 10: Changes to Simulink diagram from day 3

3 10.4 - Optimal Control of Pitch/Travel and Elevation with Feedback

3.1 The continuous model

To get the system on continious time state space form one have to look at the model derivation section in the lab exercise. We are given:

$$\mathbf{x} = \begin{bmatrix} \lambda & r & p & \dot{p} & e & \dot{e} \end{bmatrix}^T, \tag{22}$$

and from the model derivation section:

$$\ddot{p} + K_1 K_p d\dot{p} + K_1 K_p p = K_1 K_p p_c, \tag{23}$$

$$\dot{\lambda} = r, \tag{24}$$

$$\dot{r} = -K_2 p \tag{25}$$

$$\ddot{e} + K_3 K_{ed} \dot{e} + K_3 K_{ep} e = K_3 K_{ep} e_c. \tag{26}$$

This will result in the system matrices:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3 K_{ep} & -K_3 K_{ed} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1 K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3 K_{ep} \end{bmatrix}, \tag{27}$$

with

$$\mathbf{U} = \begin{bmatrix} p_c & e_c \end{bmatrix}^T. \tag{28}$$

3.2 The discretized model

To find the discretized model it is possible to use the exact same procedure as in subsection 1.2. This results in:

$$A = I + T_s A \tag{29}$$

$$B = T_s B \tag{30}$$

The resulting discretized model:

$$\mathbf{A} = \begin{bmatrix} 1 & -0.2500 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0.1415 & 0 & 0 & 0 \\ 0 & 0 & 1 & -0.2500 & 0 & 0 \\ 0 & 0 & -0.81000 & 1.9000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -0.2500 \\ 0 & 0 & 0 & 0 & 0.1600 & 1.4000 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.8100 & 0 \\ 0 & 0 \\ 0 & 0.4000 \end{bmatrix}, \tag{31}$$

3.3 Experimental results

The results from the experiment can be seen in figure 11. The Q and R matrices were:

$$\mathbf{Q} = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{32}$$

The matrices were chosen like this to emphasize that the system should strictly follow the requirements of the pitch. This did however cause the pith to get a noisy behavior. The weight on the pitch was chosen too large. Because of this, the pitch did not follow the calculated trajectory well. One can see that the travel followed well, in addition to the elevation. The elevation rate is also affected because the pitch did not have enough angle to move as fast as the optimal path calculated. One can see from the plot that even though the pitch is badly tuned, the elevation does not seem to be affected which means that the pitch and elevation are decoupled in some way. In conclusion, the Q-matrix should have been tuned better and with a little more emphasize on the states following the correct path, and not only that it completes the full rotation.

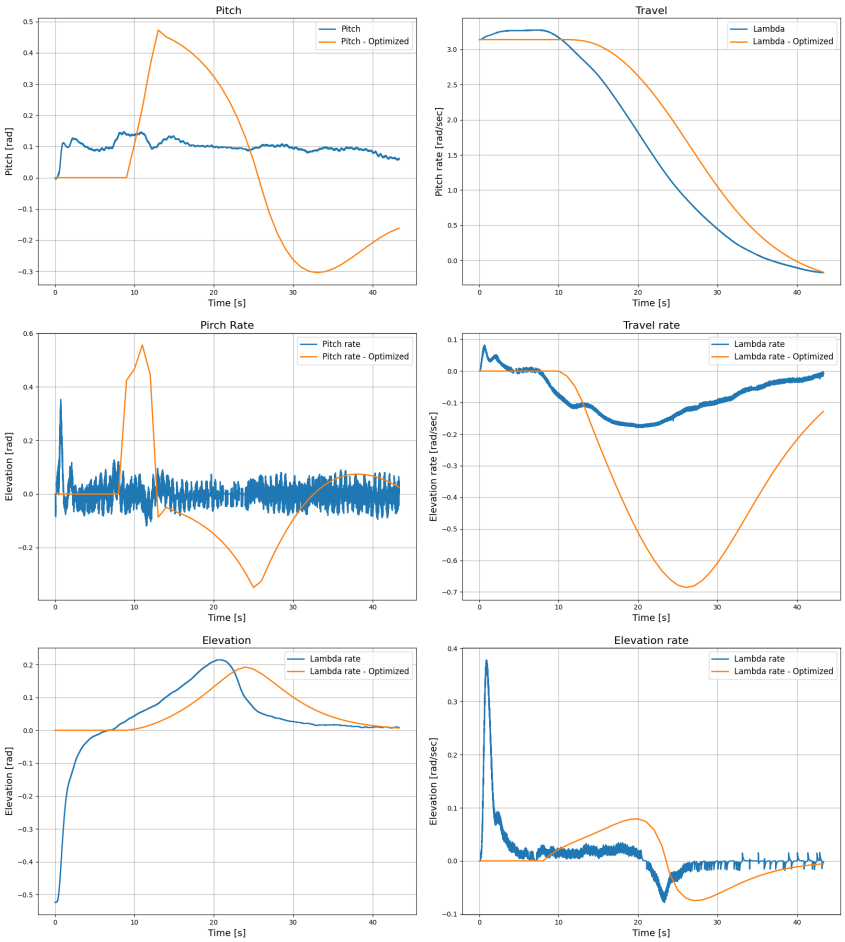


Figure 11: States with feedback day 4

3.4 Decoupled model

From the model, the pitch and elevation is decoupled. The helicopter still works because it was implemented a constraint on the pitch so that it could not be larger than $\pi/2$.

To improve the system three different approaches were suggested. Approach A was to introduce nonlinear terms in the state equations. This would not affect the performance of the system around the linearized point. If you were to get rid of the pitch constraints and let it fly freely without being connected. The linearized model would not work optimally since we could get pitch angles that are far away from the linearized point. In addition, a LQ controller would not work, since the state equations aren't linear anymore, which means we would have to implement a quadratic controller of sorts, which probably would be more computationally heavy. Suggestion

B looks like the same non-linear state equation implemented in the cost function. This would make it possible to use the LQ-controller since we keep the state equations linear. Since there are no weights on the elements, it could calculate wrong values if one value were to fall way lower than the other ones. This approach would probably work well if you added weights so that you could tune each element. The last approach looks like an MPC approach at first glance, which seems like it would work. It would again require more computational work since we calculate new values every few time-steps. It also introduces non-linear terms in the Q-matrix which possibly could change how the solution is calculated and might make the method not work. Therefore each of the 3 different methods has its pros and cons, but it seems like suggestion B would be the easiest to implement without fundamentally changing the different methods used.

3.5 MATLAB and Simulink

Listing 3: Code for day 4

```
%some values
dt=0.25;
N_aug = 40;  % shorter horizon for augmented model
lambda_f = 0;
alpha = 0.2;
beta = 20;
lambda_t = 2*pi/3;

x0_aug = [pi; 0; 0; 0; 0; 0];  % initial state for
    augmented model
x=zeros(N_aug*6,1);
x(1:6)=x0_aug;
mx_aug = size(A,2);  % 6 states
mu_aug = size(B,2);  % 2 inputs
nz = mx_aug + mu_aug;  % 8 variables per time step

% Discretized system
A_d_aug = eye(mx_aug) + dt * A;
B_d_aug = dt * B;

Aeq_aug = gen_aeq(A_d_aug, B_d_aug, N_aug, mx_aug, mu_aug
    );
beq_aug = zeros(N_aug*mx_aug, 1);
beq_aug(1:mx_aug) = A_d_aug * x0_aug;

%create lower bound
vlb_aug=-inf*ones(N_aug*nz,1);
vlb_aug(3:mx_aug:mx_aug*N_aug)=-(60*pi)/360;
vlb_aug(mx_aug*N_aug+1:mu_aug:nz*N_aug)=-(60*pi)/360;

```

```
%create upper bound
vub_aug=inf*ones(N_aug*nz,1);
vub_aug(3:mx_aug:mx_aug*N_aug)=(60*pi)/360;
vub_aug(mx_aug*N_aug+1:mu_aug:nz*N_aug)=(60*pi)/360;

q1 = 1;
q2 = 1;

% Create Q
Q_aug = zeros(mx_aug,mx_aug);
Q_aug(1,1) = 1;  % Weight on lambda
Q_aug(2,2) = 0;
Q_aug(3,3) = 0;
Q_aug(4,4) = 0;
Q_aug(5,5) =0;
Q_aug(6,6)=0;

%create G
P_aug = [q1 0;
          0 q2];
G = gen_q(Q_aug, P_aug, N_aug, N_aug);

%create z
z_aug = zeros(nz * N_aug, 1);
z0_aug=z_aug;

% Cost function:
cost_function = @(z) 1/2*z_aug'*G*z_aug;

%non linear constraints
nonlincon = @(z_aug) nonlinear_constraints(z_aug, N_aug,
    alpha, beta, lambda_t, mx_aug);

%run fmincon
opt=optimoptions('fmincon','Algorithm','sqp', '
    MaxFunEvals', 40000);
[z_opt, fval] = fmincon(cost_function, z0_aug, [], [],
    Aeq_aug, beq_aug, vlb_aug, vub_aug, nonlincon,opt);

function [c,ceq] = nonlinear_constraints(z,N,alpha,beta,
    lambda_t,nx_aug)
     c=zeros(N,1);
     for k=1:N
         c(k)=alpha*exp(-beta*(z(1+(k-1)*nx_aug)-lambda_t)
             ^2)-z(5+(k-1)*nx_aug);
     end
     ceq=[];
end
```

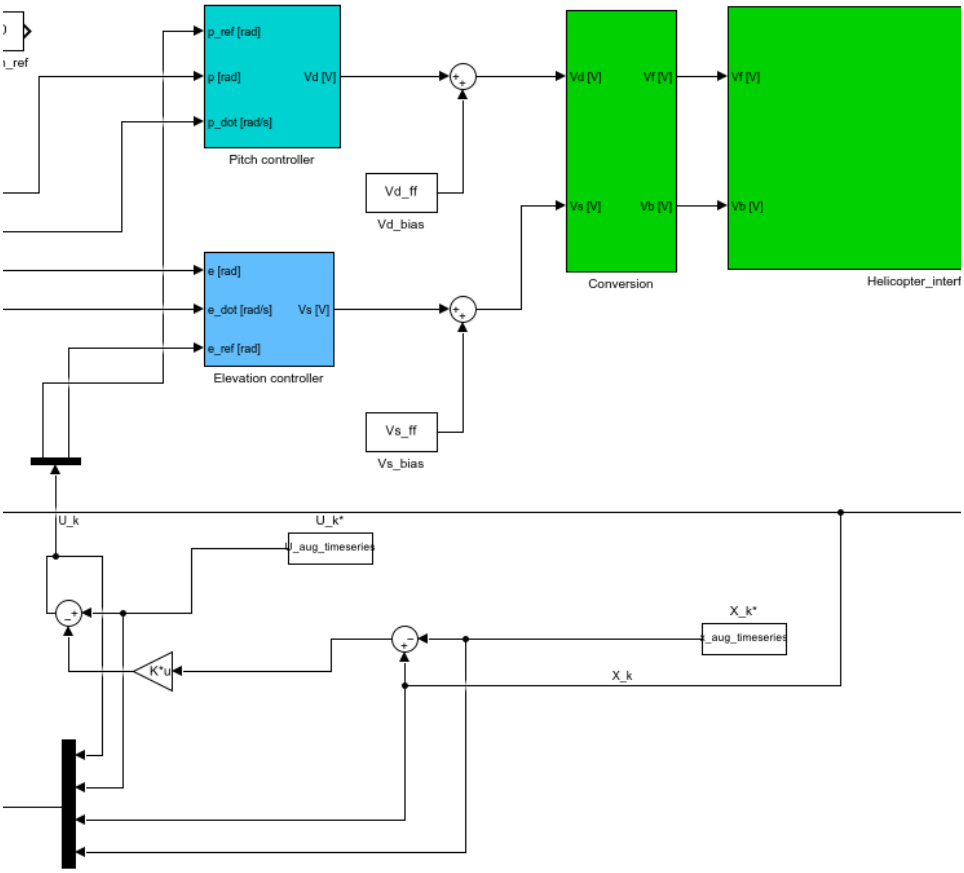


Figure 12: Changes to Simulink diagram from day 4

References

[1] Norwegian University of Science and Technology Department of Engineering Cybernetics. Ttk4135 optimization and control helicopter lab, 2025. Accessed: 03-03-25.

[2] Bjarne Foss and Tor Aksel N. Heirung. Merging optimization and control. NTNU, 2016. Accessed: 24-02-25.