TTK4135
Optimization and Control

Lab report:

Helicopter lab assignment

Written by:

Håvard Olai Kopperstad 510311

Elias Olsen Almenningen 528497

April 21, 2023

Department of Engineering Cybernetics

Contents

1 10.2 - Optimal Control of Pitch/Travel without Feedback 1
1.1 The continuous model 1
1.2 The discretized model 1
1.3 The open loop optimization problem 1
1.4 The weights of the optimization problem 4
1.5 Experimental results 5
1.6 MATLAB and Simulink 6

2 10.3 - Optimal Control of Pitch/Travel with Feedback (LQ) 9
2.1 LQ controller 9
2.2 Model Predictive Control 9
2.3 Experimental results 11
2.4 MATLAB and Simulink 13

3 10.4 - Optimal Control of Pitch/Travel and Elevation with Feedback 15
3.1 The continuous model 15
3.2 The discretized model 15
3.3 Experimental results 15
3.4 Decoupled model 17
3.5 MATLAB and Simulink 18

References 22

1 10.2 - Optimal Control of Pitch/Travel without Feedback

1.1 The continuous model

From equations [1, eq. (11a), eq. (11b), eq. (11c), eq. (11d)], the continuous time state space model of the system is

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

$$\begin{bmatrix}\dot{\lambda}\\ \dot{r}\\ \dot{p}\\ \ddot{p}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & -K_2 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd}\end{bmatrix}\begin{bmatrix}\lambda\\ r\\ p\\ \dot{p}\end{bmatrix} + \begin{bmatrix}0\\ 0\\ 0\\ K_1K_{pp}\end{bmatrix}p_c \tag{2}$$

The equations used to derive $\boldsymbol{A_c}$ and $\boldsymbol{B_c}$ take into account both the model of the simplified physical properties of the helicopter, as well as the PD- and PID-controller used to control the pitch angle (p) and the elevation angle (e), respectively. Therefore, the model describes the closed loop system of the *Basic control layer* and the *Physical layer* in [1, Figure 7].

1.2 The discretized model

By using the numerical integration scheme called forward Euler's method given as:

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(\boldsymbol{y}_n, t_n) \tag{3}$$

in Egeland and Gravdahl [2], the discrete time state space system can be derived as:

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{A}_d\boldsymbol{x}_k + \boldsymbol{B}_d u_k & (4)\\ &= \boldsymbol{x}_k + \Delta t(\boldsymbol{A}_c\boldsymbol{x}_k + \boldsymbol{B}_c u_k) & (5)\\ &= (\boldsymbol{I} + \Delta t\boldsymbol{A}_c)\boldsymbol{x}_k + \Delta t\boldsymbol{B}_c u_k & (6)\end{aligned}$$

and thus

$$\boldsymbol{x}_{k+1} = \begin{bmatrix}1 & \Delta t & 0 & 0\\ 0 & 1 & -\Delta t K_2 & 0\\ 0 & 0 & 1 & \Delta t\\ 0 & 0 & -\Delta t K_1 K_{pp} & 1-\Delta t K_1 K_{pd}\end{bmatrix}\boldsymbol{x}_k + \begin{bmatrix}0\\ 0\\ 0\\ \Delta t K_1 K_{pp}\end{bmatrix}u_k \tag{7}$$

1.3 The open loop optimization problem

With our discretized model from Section 1.2, the provided cost function

$$\phi = \sum_{i=0}^{N-1}(\lambda_{i+1} - \lambda_f)^2 + qp_{ci}^2, \quad q \geq 0 \tag{8}$$

and the large vector $z = (x_1^T, \dots, x_N^T, u_0^T, \dots, u_{N-1}^T)^T$, the finite horizon optimal control problem can be rewritten to:

$$f(z) = \frac{1}{2} \sum_{k=0}^{N-1} x_{k+1}^T \boldsymbol{Q} x_{k+1} + u_k^T R u_k \tag{9}$$

with

$$\boldsymbol{Q} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad R = 2q \tag{10}$$

With the provided constraint

$$|p_k| \leq \frac{60\pi}{360}, \;\; k \in \{1, \dots, N\} \tag{11}$$

the open loop optimization problem can be cast to

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \sum_{k=1}^{N} \frac{1}{2} \boldsymbol{z}^T \boldsymbol{G} \boldsymbol{z} + \boldsymbol{c}^T \boldsymbol{z} \\ \text{s.t.} \quad & \boldsymbol{A_{eq}} \boldsymbol{z} = \boldsymbol{b_{eq}}, \\ & \boldsymbol{v_{lb}} \leq \boldsymbol{z} \leq \boldsymbol{v_{ub}} \end{aligned}$$

where

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{Q}_1 & 0 & \dots & \dots & \dots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \boldsymbol{Q_N} & \ddots & & \vdots \\ \vdots & & \ddots & R_0 & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & \dots & 0 & R_{N-1} \end{bmatrix}, \quad \boldsymbol{c}^T = \boldsymbol{0}^T \tag{12}$$

in which $\boldsymbol{Q}_1$ is the $\boldsymbol{Q}$ matrix at $k = 1$.

The equality constraint $\boldsymbol{A_{eq}} \boldsymbol{z} = \boldsymbol{b_{eq}}$ contains:

$$\boldsymbol{A_{eq}} = \begin{bmatrix} \boldsymbol{I} & 0 & \dots & \dots & 0 & -\boldsymbol{B}_0 & 0 & \dots & \dots & 0 \\ -\boldsymbol{A}_1 & \boldsymbol{I} & \ddots & & \vdots & 0 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -\boldsymbol{A}_N & \boldsymbol{I} & 0 & \dots & \dots & 0 & -\boldsymbol{B}_N \end{bmatrix} \tag{13}$$

$$\boldsymbol{b_{eq}} = \begin{bmatrix} \boldsymbol{A}_0 \boldsymbol{x}_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{14}$$

The input constraints $\boldsymbol{v_{lb}}$ and $\boldsymbol{v_{ub}}$ must correspond to the z vector and thus becomes:

$$\boldsymbol{v_{lb}} = \begin{bmatrix} \begin{bmatrix} -Inf \\ -Inf \\ -\frac{60\pi}{360} \\ -Inf \end{bmatrix} \\ \vdots \\ \begin{bmatrix} -Inf \\ -Inf \\ -\frac{60\pi}{360} \\ -Inf \end{bmatrix} \\ -\frac{60\pi}{360} \\ \vdots \\ -\frac{60\pi}{360} \end{bmatrix}, \quad \boldsymbol{v_{ub}} = \begin{bmatrix} \begin{bmatrix} Inf \\ Inf \\ \frac{60\pi}{360} \\ Inf \end{bmatrix} \\ \vdots \\ \begin{bmatrix} Inf \\ Inf \\ \frac{60\pi}{360} \\ Inf \end{bmatrix} \\ \frac{60\pi}{360} \\ \vdots \\ \frac{60\pi}{360} \end{bmatrix} \tag{15}$$

1.4 The weights of the optimization problem

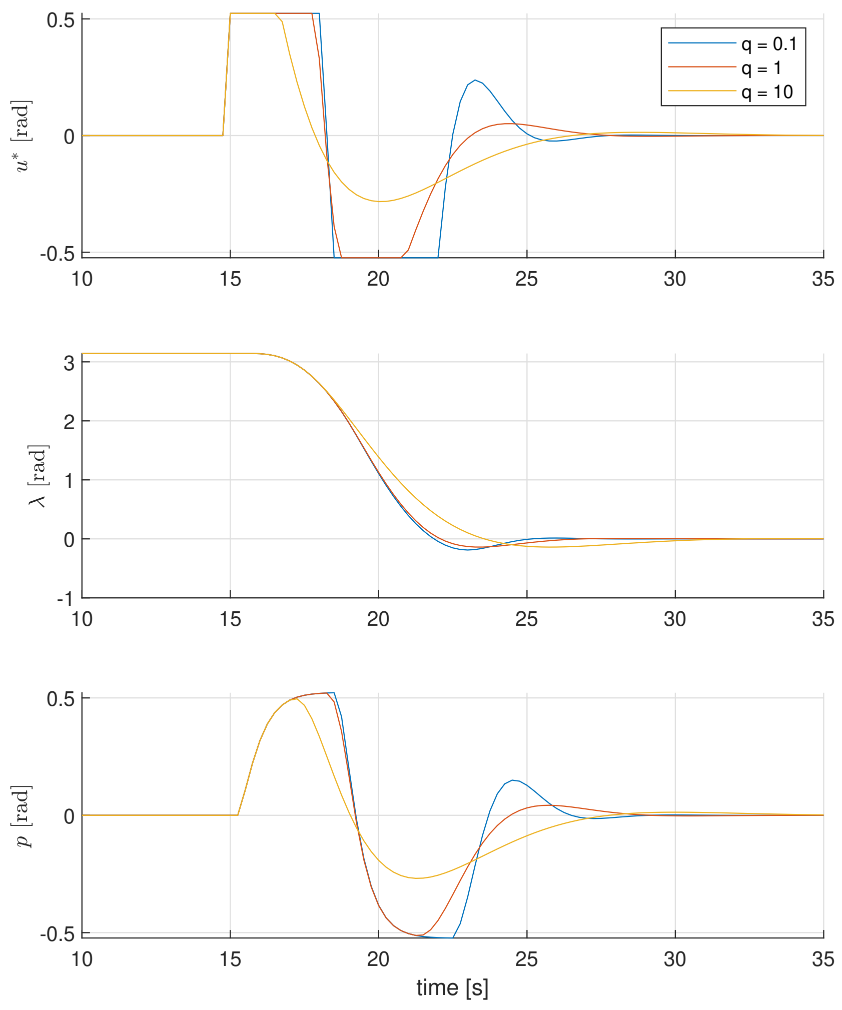

Figure 1: The simulated λ and p with different values for q

In Equation 8 (the cost function), we can choose different values for q to dictate how the system should behave. By setting q to a small value, in our case $q = 0.1$, the open loop optimization problem finds an optimal solution based on our system, the constraints and the bounds. In Equation 8, the other factor of the product qp_{ci}^2 can be "larger" than when q is e.g. 1.0 or 10.0. This means that the size of q dictates the size of $p_k = u_k$. When

$q = 0.1$, the optimal trajectory can use a more aggressive control, making u^* larger. This again means that the simulated helicopter moves from λ_0 to λ_f faster. Setting $q = 10$ would then mean that in order to minimize the cost function, p_k (or u_k) has to be smaller. This again means that the helicopter uses longer time, and less aggressive maneuvers to move from λ_0 to λ_f. This is exactly what happens in the simulations done in the lab preparations, and is shown in Figure 1.

1.5 Experimental results

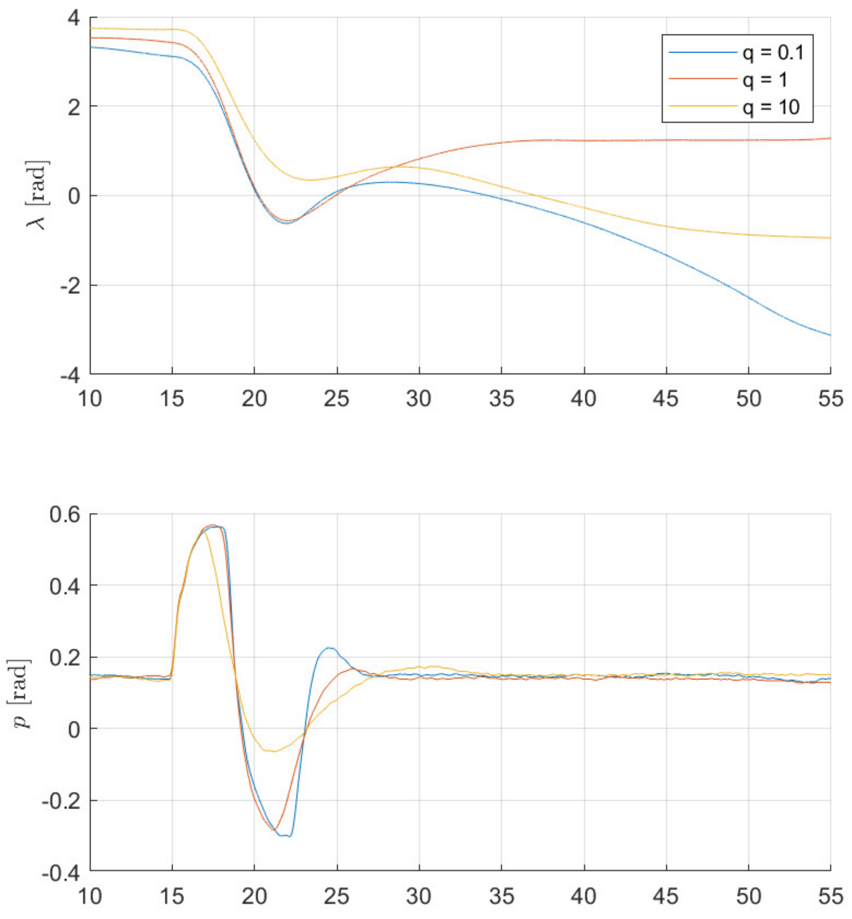

Figure 2: The actual λ and p with different values for q

When performing the simulated tests on the real helicopter, it was hard to visually notice any real difference when changing the value of q. Since this is open-loop control, and only the pre-calculated controller inputs u^* is given to the helicopter, it "doesn't know" whats happening to it since there is no state feedback. It is possible though to see when comparing Figure 1 and 2 that the pitch angle resembles the optimal trajectory quite good.

As seen in [1, Figure 7], the inner control loop only controls the pitch and elevation of the helicopter, not the travel angle, which is what we are optimizing. As seen in Figure 2, the travel angle λ goes towards zero, but then it either under- or overshoots it, and then drifts away in either direction. The pitch controller is fed the optimal controller input, and the pitch angles look similar to the simulated results, but since the stationary angle of the helicopter's rotors are not zero, it drifts after the trajectory is complete.

1.6 MATLAB and Simulink

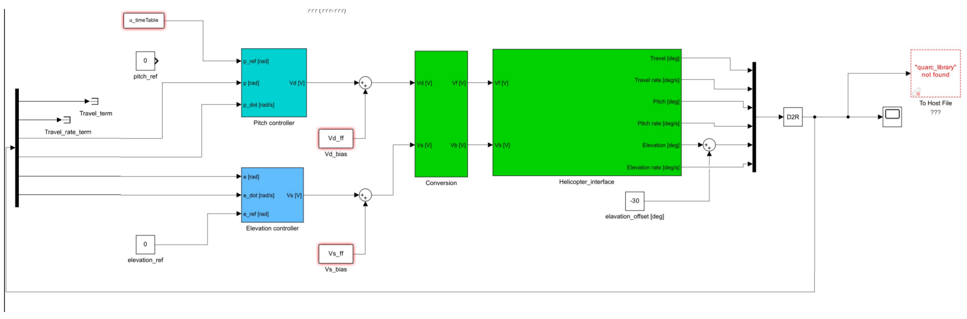

Figure 3: The simulink file used for lab 2

The only change made from the handout file is the *fromWorkspace* block u_timeTable which sends the optimal control input u^* to the pitch controller.

```
init05;
%% Continuous time system model
A = [0      1       0           0;
     0      0     -K_2          0;
     0      0       0           1;
     0      0   -K_1*K_pp -K_1*K_pd];
B = [   0;
        0;
        0;
     K_1*K_pp];

%% Discrete time system model. x = [lambda r p p_dot]'
% Using forward-euler
Δ_t  = 0.25; % sampling time
A1 = eye(4) + Δ_t * A;
B1 = Δ_t * B;

% Initial values
x1_0 = pi;                               % Lambda
x2_0 = 0;                                % r
x3_0 = 0;                                % p
x4_0 = 0;                                % p_dot

% Time horizon and initialization
N  = 100;                                % Time horizon for ...
    states

% Bounds
ul       = -60*pi/360;                   % Lower bound on ...
    control
uu       = +60*pi/360;                   % Upper bound on ...
    control

% Generate constraints on measurements and inputs
[vlb,vub]       = gen_constraints(N,M,xl,xu,ul,uu); % hint: ...
    gen_constraints

% Generate the matrix Q and the vector c (objecitve ...
    function weights in the QP problem)
Q1 = zeros(mx,mx);
q = 0.1;
Q1(1,1) = 2;                             % Weight on state x1
Q1(2,2) = 0;                             % Weight on state x2
Q1(3,3) = 0;                             % Weight on state x3
Q1(4,4) = 0;                             % Weight on state x4
P1 = 2*q;                                % Weight on input
Q = gen_q(Q1,P1,N,M);                    % Generate Q, hint: ...
    gen_q
c = zeros(500,1);                        % Generate c, this ...
    is the linear constant term in the QP

%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);            % Generate A, hint: ...
```

```
    gen_aeq
beq = A1*x0;
beq(5:400) = zeros(396,1);              % Generate b

[z,lambda] = quadprog(Q,c,[],[],Aeq,beq,vlb,vub,x0); % ...
    hint: quadprog. Type 'doc quadprog' for more info

% To simulink
u_timeTable = timetable(u,'TimeStep',seconds(0.25));
```

Listing 1: MatLab code used for Lab 2

Note that the code in Listing 1 are only those that differ from the handout files.

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

2.1 LQ controller

Foss and Heirung in [3, p. 56] writes that the solution of the LQ problem can be written in closed form given as a linear state feedback controller, $u_t = -K_t x_t$, for a finite horizon LQ problem. Thus making the controller a linear time varying controller. The feedback gain matrix K_t is derived by the Riccati equation, the weighting matrices in the objective function and the system matrices. Since the system matrices and the weighting matrices are time-invariant in our case, and the horizon is sufficiently long [3, p. 64], the gain matrix becomes time invariant. Thus the LQ controller takes the form $u_k = -Kx_t$.

The weighting matrices $\boldsymbol{Q}$ and R are a way to tune the controller gain. By increasing the values in the $\boldsymbol{Q}$ matrix deviations in the corresponding states would be punished harder, i.e. making the controller more aggressive. There is a connection between the weighting of the matrix elements of $\boldsymbol{Q}$ and R. Increasing R means we allow the controller to use less input which in turn makes the controller more compliant to deviations in the states. One could in a way say increasing weights of $\boldsymbol{Q}$ and R are opposite operations. One must keep in mind that in reality the connection between the weighting matrices is much more complicated, thus increasing a weight in one matrix would not necessarily be equal to decreasing another weight in the opposite matrix by the same amount.

From the lab assignment we are told to use diagonal weighting matrices and our setup can be seen in Equation (16). Our choice of weights are explained in Section 2.3.

$$\boldsymbol{Q} = \begin{bmatrix} \lambda_w & 0 & 0 & 0 \\ 0 & r_w & 0 & 0 \\ 0 & 0 & p_w & 0 \\ 0 & 0 & 0 & \dot{p}_w \end{bmatrix}, \quad R = r_w \tag{16}$$

2.2 Model Predictive Control

An MPC re-optimizes the trajectory based on the measurements from the state feedback every time step, Δt. To implement the proposed way of using MPC in [1, 10.3.1.3], one would have to apply a structure as can be seen in Figure 4. In the figure the LQ controller is removed and an MPC is used for the Model-based optimization. The MPC takes the newest measured states and calculates x^*_{new}. Then, u^*_{new} can be applied directly to the system, as seen in the figure. To actually implement this proposed MPC scheme on the lab, one way could be to make a Simulink block based on the needed MPC equations that can be written in Matlab. From there one would feed the measured states into the MPC block and feed the calculated output directly to the basic control layer.

One advantage of using an MPC controller compared to the control structure combining prediction and LQ control is that the MPC is continuously updated with measured states (state feedback) and new trajectories for a finite horizon are calculated every time step. In the prediction+LQ scheme the trajectory is calculated once (open loop optimization) and then the LQ controller tries to follow the predicted trajectory the best it can. By continuously calculating a new trajectory every time step using current state of the plant as the inital state, the new solution to the dynamic optimization problem contains the future states too.

Another advantage of using a MPC control scheme is that it opens for implementation of inequality constraints, unlike with LQ control. A disadvantage of implementing the MPC control scheme is that it is more complex and demands of the person implementing it.

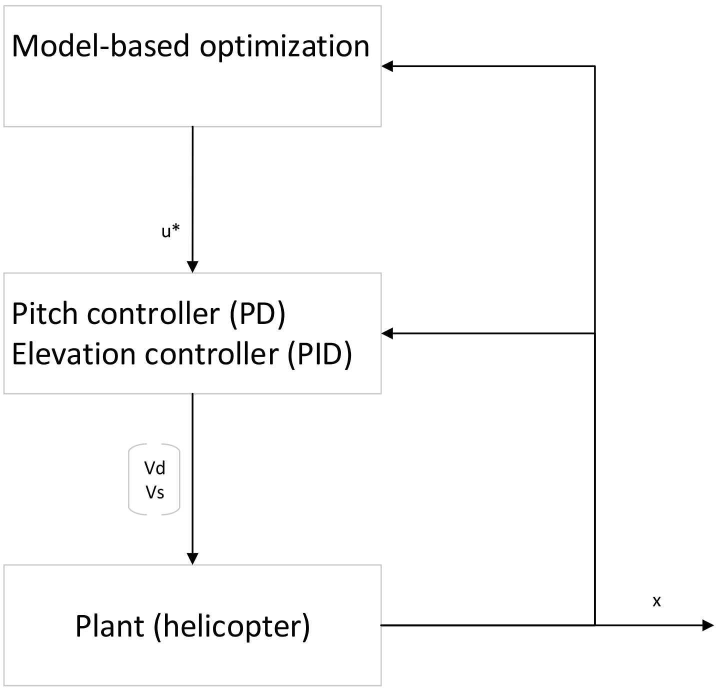

Figure 4: Proposed structure when using MPC

2.3 Experimental results

The weights used during our time at the lab can be seen in Table 1.

Parameters	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7
λ_w	1	1	5	5	20	1	1000
r_w	1	1	1	1	1	1	1
p_w	1	1	1	2	1	20	1
$\dot{p_w}$	1	1	1	1	1	1	1
R	1	2	1	1	1	1	1

Table 1: Different weights used for Q and R during the lab

Test 1 and Test 2 were chosen to see the difference between weighting $\boldsymbol{Q}$ and R the same and weighting the controller input twice as much as the states. Test 3 and 4 were done to see how weighting pitch and travel would affect the time used to travel from λ_0 to λ_f. Test 5, 6 and 7 were also done to test the travel trajectory, but only with much higher weights and an expectation of noticeable behaviour changes to the helicopter. Note that in the legends of Figure 5 and 6 Test x are shortened to Tx.

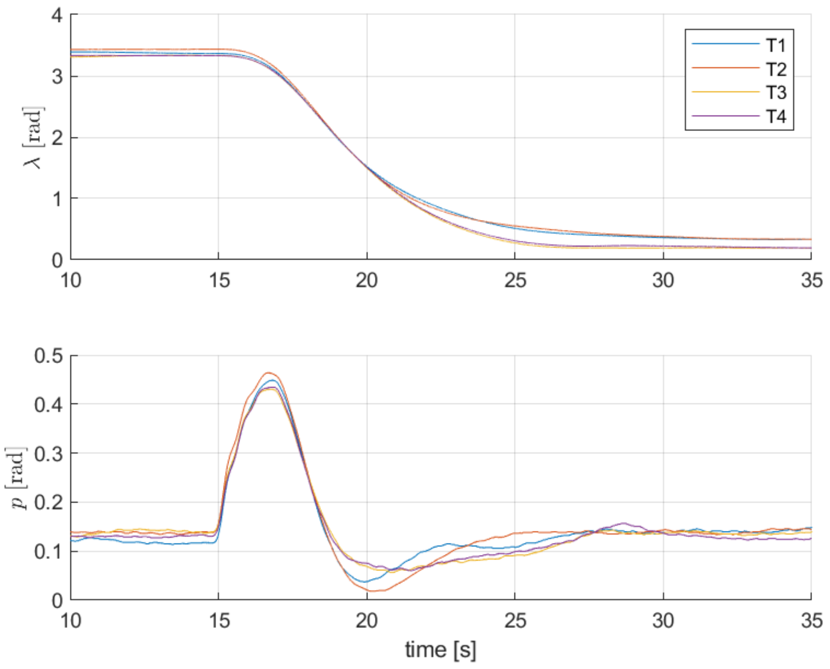

Figure 5: Test 1, 2, 3 and 4 performed at the lab

As seen in Figure 5, Test 1 and 2 are almost identical when looking at travel (λ), but when looking at the pitch angle (p) T2 has a larger angle at around 16 and 21 seconds. Since we chose R to be 2, it should in theory be smaller than T1 but from our testing this does not seem to be the case. One might think the mismatch between theory and practice comes from a too small weight increase in R with respect to the weights in $\boldsymbol{Q}$ or some model error.

Comparing to the results in lab 2 (Figure 2) the trajectories are stationary, and T3 and T4 are closer to zero than T1 and T2. This makes sense, since we now have closed-loop control and the weights on λ are higher in T3 and T4 than T1 and T2. They do not overshoot either, as this is taken care of by the LQ controller. Hence, the pitch angle is much smaller at around 20 seconds than in lab 2.

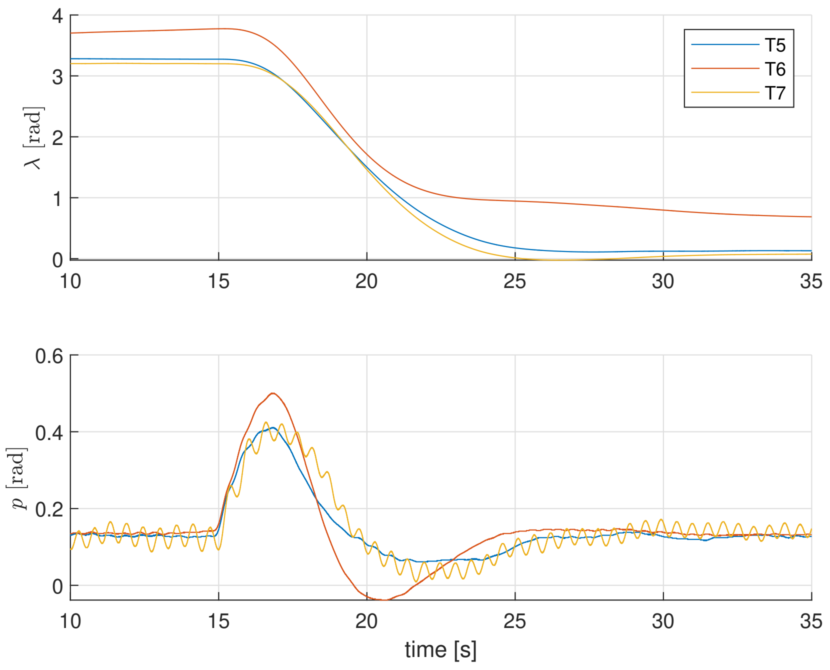

Figure 6: Test 5, 6 and 7 performed at the lab

In Figure 6, T5 and T7s travel angle is weighted much more than in T6, making them almost go to zero, while T6 is quite far away. Its even further away from zero than T1 - T4. This is because the pitch angle is weighted so much more than the other parameters, leading to a closer related and larger angle as in the simulation in Figure 1. In T7, the travel weight is 1000, making the pitch angle oscillate. The oscillations possible comes from the controller trying to follow the optimal trajectory perfectly, thus needing rapid changes in the pitch angle to closely follow it.

The key difference between lab 2 and 3 is that we now have state feedback, making the helicopter system aware of what it's states are. This means that it will try to self correct if it strays too far away from the optimal trajectory, if the LQ controller is tuned to prioritize deviations in the travel.

2.4 MATLAB and Simulink

The Simulink file is updated from lab 2 (Figure 3) with state feedback and an LQ controller. The LQ controller gets the optimal trajectory form the four leftmost *fromWorkspace* blocks and the state from the rightmost multiplexer pointing downwards. The calculated input is then subtracted from

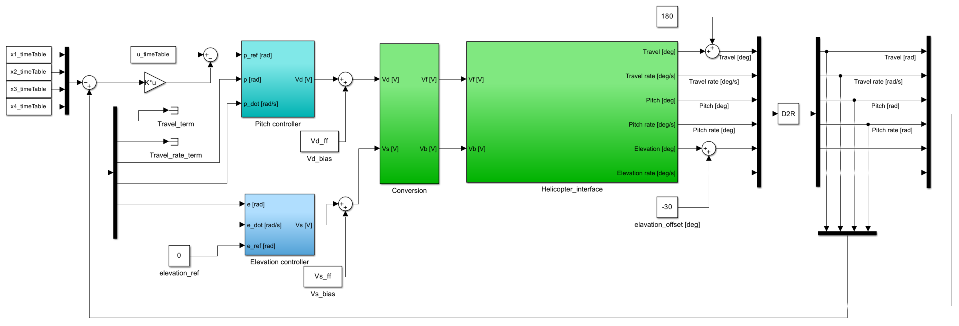

Figure 7: The simulink file used for lab 3

the optimal input u^* and then fed to the pitch controller. We also added a constant offset of 180° to the travel angle such that the real system matches the simulation, which starts in π and goes to zero.

```matlab
% To simulink
u_timeTable = timetable(u,'TimeStep',seconds(0.25));
x1_timeTable = timetable(x1,'TimeStep',seconds(0.25));
x2_timeTable = timetable(x2,'TimeStep',seconds(0.25));
x3_timeTable = timetable(x3,'TimeStep',seconds(0.25));
x4_timeTable = timetable(x4,'TimeStep',seconds(0.25));

%% LQR
Q = diag([1 1 1 1]); %[travel travel rate pitch pitch rate]
R = 1;

K = dlqr(A1,B1,Q,R);
```

Listing 2: MatLab code used for Lab 3

Note that the code in Listing 2 are only those that differ from the handout files and Listing 1.

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

3.1 The continuous model

The continuous time state space model of the system with the two additional states e and $\dot{e}$ is:

$$\begin{bmatrix} \dot{\lambda} \\ \dot{r} \\ \dot{p} \\ \ddot{p} \\ \dot{e} \\ \ddot{e} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed} \end{bmatrix} \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \\ e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep} \end{bmatrix} \begin{bmatrix} p_c \\ e_c \end{bmatrix}$$

3.2 The discretized model

By using Euler's method as in Section 1.2, the resulting discretized model can be stated as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_d\boldsymbol{x}_k + \boldsymbol{B}_d\boldsymbol{u}_k$$

where

$$\boldsymbol{A}_d = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & -\Delta tK_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & -\Delta tK_1K_{pp} & 1-\Delta tK_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & -\Delta tK_3K_{ep} & 1-\Delta tK_3K_{ed} \end{bmatrix}$$

$$\boldsymbol{B}_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \Delta tK_1K_{pp} & 0 \\ 0 & 0 \\ 0 & \Delta tK_3K_{ep} \end{bmatrix}$$

3.3 Experimental results

In Test 1 travel (λ) and pitch (e) are weighted equally and higher than pitch (p). This means that these two tries to follow the planned trajectory, allowing for larger deviations in pitch. In Test 2, travel is weighted significantly higher than the other states. In theory the travel angle should then follow the trajectory closer than any of the other tests, but the other stats "will be free" to deviate much more. This is clearly seen in Figure 8 as being the case. T2 is closest to the travel trajectory and at approximately 26 seconds,

Parameters	Test 1	Test 2	Test 3	Test 4
λ	5	25	1	1
r	1	1	1	1
p	1	1	25	1
$\dot{p}$	1	1	1	1
r	1	1	1	1
e	5	1	25	1
$\dot{e}$	1	1	1	1

Table 2: The parameters used in the last lab day

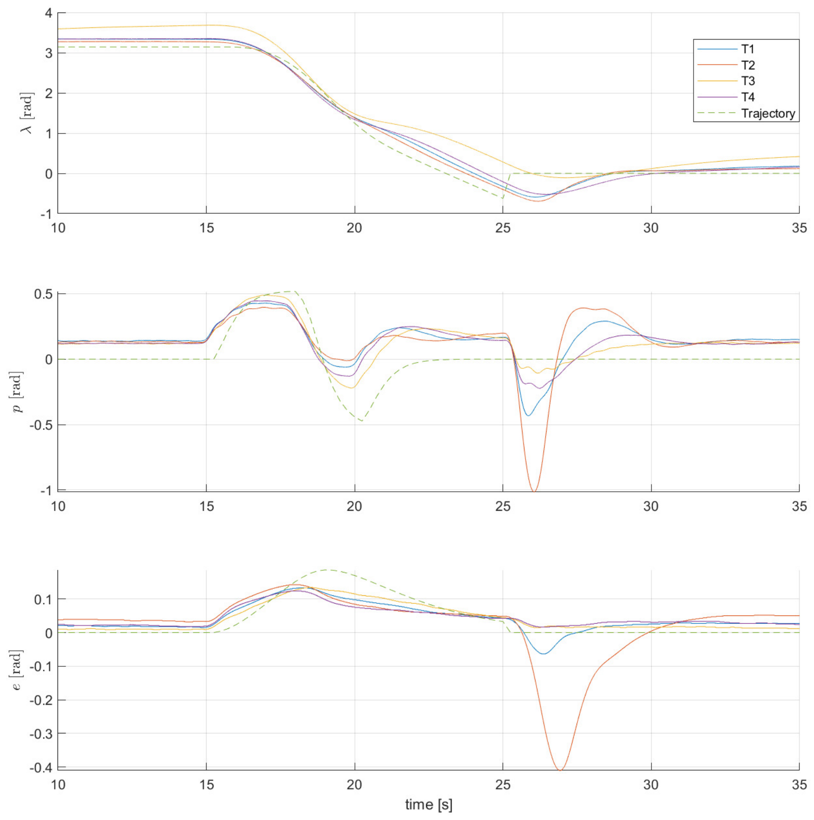

Figure 8: Test 1, 2, 3 and 4 performed at the lab

pitch for T1 and T2 spikes and elevation for spikes violently. For test 3 pitch and elevation have the highest weights, meaning these states should follow the trajectory closest. This is again the case.

As to why the helicopter fails to follow the elevation trajectory, when the elevation angle gets large, the system begins to move far away from the linearization point. When the system strays too far away, the models perception of what is happening to the physical model might be wrong. When the helicopter thinks it reached the top (elevation angle ≈ 0.2), it might be some decimal points of a radian too short. Since the tests performed on the lab always deviate from the trajectory, this is a likely explanation.

3.4 Decoupled model

When we were running the helicopter at the lab it was operating with a constraint on the pitch angle of $\frac{60\pi}{360}\text{rad} = 30^\circ$ in both directions. Having this constraint means the two propellers is always providing some thrust upwards keeping the elevation above zero degrees. So, having state feedback in conjunction with the pitch constraint keeps the helicopter flying. From our testing it seems that the upward thrust was always enough to keep the helicopter flying and we observed that the linearized model is an appropriate model. It also becomes clear that the unmodelled relationship between pitch and elevation is indeed at play in the true nonlinear nature of the helicopter.

Suggestion A of controller improvement suggests to rewrite [1, eq. 2] to include the true sinusoidal relationship between the elevation and pitch angle. Since the true relationship is nonlinear it would not be a viable solution since it would not be possible to solve the optimization problem with the nonlinear system model. One could linearize the new equation to derive a linear state-space model, but then the elevation-pitch relationship would be lost again.

Adding the "$+\sum_{k=1}^{N}(\dot{e}_k - \dot{e}_{k-1} - \Delta t C_p V_s cos(p_k) - \Delta t C_e cos(e_k))^2$" term to the objective function in suggestion B could be done to improve the controller. However the optimization problem increases in complexity due to the nonlinear functions *sin* and *cos*. The problem may end up being nonconvex too, which in turn means there may be several local minima and maxima.

In suggestion C it is proposed to live adjust one weight in $\boldsymbol{Q}$ of the LQR. This can relatively easily be implemented in Matlab and Simulink, but it may not be the best option. Multiplying the elevation rate with $cos(p)$ in $\boldsymbol{Q}$ means that with no pitch angle the elevation rate is weighted "heavily". When the pitch angle increases or decreases towards our pitch constraints the elevation rate is weighted less. I.e. when the pitch angle is high the controller does not punish deviations in the elevation rate as much as it would do with zero pitch.

3.5 MATLAB and Simulink

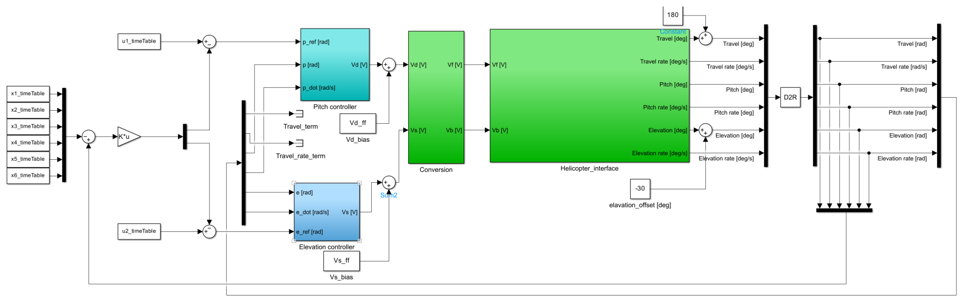

Figure 9: The simulink file used for lab 4

The simulink file is updated from lab 3 (Figure 7) with two more states and a new input to the elevation controller.

```
%% Continuous time system model
A = [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];

B = [   0           0;
        0           0;
        0           0;
     K_1*K_pp       0;
        0           0;
        0        K_3*K_ep];
%% Discrete time system model. x = [lambda r p p_dot e e_dot]'

% Using forward-euler
Δ_t  = 0.25; % sampling time
A1 = eye(6) + Δ_t * A;
B1 = Δ_t * B;

%% Number of states and inputs
mx = size(A1,2); % Number of states (number of columns in A)
mu = size(B1,2); % Number of inputs(number of columns in B)

% Initial values
x1_0 = pi;                              % Lambda
x2_0 = 0;                               % r
x3_0 = 0;                               % p
x4_0 = 0;                               % p_dot
x5_0 = 0;                               % e
x6_0 = 0;                               % e_dot
```

```
x0 = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0]';  % Initial values

% Time horizon and initialization
N  = 40;                               % Time horizon for ...
    states
M  = N;                                % Time horizon for ...
    inputs
z  = zeros(N*mx+M*mu,1);               % Initialize z for ...
    the whole horizon
z0 = z;                                % Initial value for ...
    optimization
z0(1) = x1_0;

%% Bounds
ul       = -60*pi/360;                 % Lower bound on ...
    control
uu       = +60*pi/360;                 % Upper bound on ...
    control

xl       = -Inf*ones(mx,1);            % Lower bound on ...
    states (no bound)
xu       = Inf*ones(mx,1);             % Upper bound on ...
    states (no bound)
xl(3)    = ul;                         % Lower bound on ...
    state x3
xu(3)    = uu;                         % Upper bound on ...
    state x3

%% Generate constraints on measurements and inputs
[vlb,vub]       = gen_constraints(N,M,xl,xu,ul,uu); % hint: ...
    gen_constraints
vlb(N*mx+M*mu)  = 0;                   % We want the last ...
    input to be zero
vub(N*mx+M*mu)  = 0;                   % We want the last ...
    input to be zero

%% Generate the matrix Q and the vector c (objecitve ...
    function weights in the QP problem)
Q1 = zeros(mx,mx);                     % the objective ...
    function is 1/2 x'Qx + scalar shit, so multiply by two
Q1(1,1) = 1;                           % Weight on state x1,
Q1(2,2) = 0;                           % Weight on state x2
Q1(3,3) = 0;                           % Weight on state x3
Q1(4,4) = 0;                           % Weight on state x4
Q1(5,5) = 0;                           % Weight on state x5
Q1(6,6) = 0;                           % Weight on state x6

p1 = 1;                                % Weight on input, ...
    THIS IS q in forarbeid
p2 = 1;
P = diag([p1 p2]);
Q = gen_q(Q1,P,N,M);                  % Generate Q, hint: ...
    gen_q
% c = zeros(N*mx+M*mu,1);               % Generate c, ...
```

```
    this is the linear constant term in the QP

%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);             % Generate A, ...
    hint: gen_aeq
beq = [A1*x0; zeros((mx*N)-mx,1)];

%% Solve QP problem with linear model
alpha = 0.2;
beta = 20;
lambda_t = 2*pi/3;

options = ...
    optimoptions(@fmincon,'Algorithm','sqp','MaxFunctionEvaluations',5*10^4);
params = [N lambda_t alpha beta mx];
tic
z = ...
    fmincon(@(z)fun(z,Q),z0,[],[],Aeq,beq,vlb,vub,@(x)nonlcon(x,params),options);
t1=toc;
% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(N*mx+M*mu,1);
for i=1:N*mx+M*mu
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

%% To Simulink
u1_timeTable = timetable(u1,'TimeStep',seconds(0.25));
u2_timeTable = timetable(u2,'TimeStep',seconds(0.25));
x1_timeTable = timetable(x1,'TimeStep',seconds(0.25));
x2_timeTable = timetable(x2,'TimeStep',seconds(0.25));
x3_timeTable = timetable(x3,'TimeStep',seconds(0.25));
x4_timeTable = timetable(x4,'TimeStep',seconds(0.25));
x5_timeTable = timetable(x5,'TimeStep',seconds(0.25));
x6_timeTable = timetable(x6,'TimeStep',seconds(0.25));

%% LQR
Q_LQR = diag([1 1 1 1 1 1]); %[travel travel_rate pitch ...
    pitch_rate elevation e_rate]
R_LQR = diag([1 1]);

K = dlqr(A1,B1,Q_LQR,R_LQR);

%% Functions
function f = fun(z,Q)
    f = 1/2*(z')*Q*z;
end

function [c,ceq] = nonlcon(x,params)
    N = params(1);
    lambda_t = params(2);
    alpha = params(3);
    beta = params(4);
    mx = params(5);

    c = alpha*exp(-beta*(x(1:mx:mx*N) - lambda_t).^2) - ...
        x(5:mx:mx*N);
    ceq = [];
end
```

Listing 3: MatLab code used for Lab 4

Note that the code in Listing 3 are only those that differ from the handout files and Listing 2 and 1.

References

[1] Ttk4135 optimization and control helicopter lab. `https://ntnu.blackboard.com/bbcswebdav/pid-1963057-dt-content-rid-55524901_1/xid-55524901_1`. Accessed: 2023-02-23.

[2] O. Egeland and J. T. Gravdahl. *Modelling and Simulation for Automatic Control*. Marine Cybernetics AS, first edition, 2002.

[3] B. Foss and T. A. N. Heirung. *Merging Optimization and Control*. Norges teknisk-naturvitenskapelige universitet, Institutt for teknisk kybernetikk, first edition, 2016.