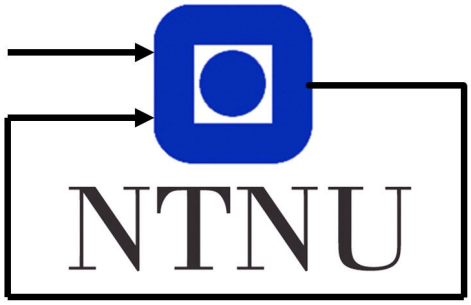
TTK4135 Optimization and Control
Helicopter Lab

Student 544941, Marius Vårdal
Student 544946, Simon Lervåg Breivik

April 21, 2023



Department of Engineering Cybernetics

Contents

1 10.2 - Optimal Control of Pitch/Travel without Feedback

1.1 The continuous model

The continuous model looks like the following:
This uses the state vector $x = \begin{bmatrix}\lambda & r & p & \dot{p}\end{bmatrix}^T$ and $u = p_c$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 \cdot K_{pp} & -K_1 \cdot K_{pd} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -K_1 \cdot K_{pp} \end{bmatrix}$$

We are disregarding the elevation here. Then we are sending in an optimal trajectory input for the pitch-controller. However we are mot using any feedback. This makes our controller very poor. We are modeling the travel, pitch and their rates of change. The model was discovered by setting up moment balances and using pid-controllers. We assume that the pitch is small $0 < p << 1 \implies sin(p) \approx p$. Even though we assume that there is minimal interactions between elevation and pitch/travel, these actually do exist. But we don't include them in our model.

1.2 The discretized model

Using forward Euler with $\Delta t = 0.25$ we get the following model:

$$\begin{aligned} A_d &= I_4 + A \cdot \Delta t \\ &= \begin{bmatrix} 1 & 1/4 & 0 & 0 \\ 0 & 1 & -K_2/4 & 0 \\ 0 & 0 & 1 & 1/4 \\ 0 & 0 & -(K_1 \cdot K_{pp})/4 & -(K_1 \cdot K_{pd})/4 \end{bmatrix} \end{aligned}$$

$$B_d = B \cdot \Delta t = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 - (K_1 \cdot K_{pp})/4 \end{bmatrix}$$

This uses the fact that $\dot{x} \approx \frac{x_{k+1} - x_k}{T} = A \cdot x_k + B \cdot u$

1.3 The open loop optimization problem

We want to move the helicopter travel from $\lambda_0 = pi$ to $\lambda_f = 0$.
The objective function to be minimized is:

$\sum_i^{N-1}(\lambda_{i+1})^2 + q \cdot p_{ci}^2, \qquad 0 \leq q$
This means that all our lambda values must be squared in addition to our input variable. We have our z (optimization variable) like this:
$z = \begin{bmatrix}\lambda(1) & r(1) & p(1) & \dot{p}(1) & \lambda(2) & \dots & u(1) & u(2) & \dots\end{bmatrix}^T$
So in the input to the quadprog will be a large diagonal matrix with 1 first every 4 and after all the states everything to the end will be 1 (for the inputs). There are no linear terms, so $c = 0$. In terms of the equality constrains we have something like this.

$$A_eq = \begin{bmatrix} I & 0 & \dots & 0 & -B & 0 & \dots & 0 \\ -A & I & \ddots & \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & -A & I & 0 & \dots & \dots & -B \end{bmatrix}, beq = \begin{bmatrix} Ax_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

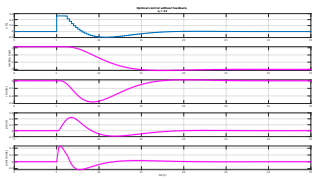
This way :
$x1 = Ax_0 + Bu_0$ (left side is $x_1 - Bu_0$).
And for everything after:
$x_i - Ax_{i-1} - Bu_{i-1} = 0$
This means that the state space equations are satisfied. In addition we have inequality constraints on the solution. This is only for the input and the pitch. Their absolute value should be inside $[0, \pi/6]$ This is fixed in the last inputs to quadprog.

1.4 The weights of the optimization problem

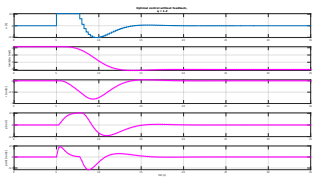
As we can see from Figure 1, with a smaller q we can tolerate higher inputs values and therefore reach the target faster. This gives a more aggressive system. We also can see that the q actually is a weighing between using large values for the travel versus using little "fuel". The value of the q decides the relative weighting of how much we are to penalize fuel vs travel (being away from the final point). As hinted in the "for the daring" task, it is not optimal that the travel part is squared, since it penalizes the helicopter more at the start than when it is closer to the optimum.

1.5 Experimental results

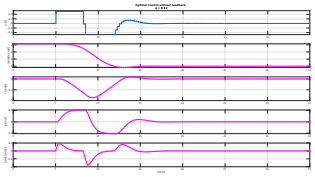
As can bee seen from Figure 2, The solution is not at all satisfactory. It does not solve our problem. This is because the model (Open Loop) doesn't take into account the disturbances via some form of feedback loop. This could be done for example with Model Predictive Control (MPC), Kalman filter or LQ-controller. We can conclude that the helicopter does not end up in the desired point λ_f, because it does not take into account the disturbances and the noise in the system, in addition to the model being a simplification



(a) $q = 12$



(b) $q = 1.2$



(c) $q = 0.12$

Figure 1: Showing the trajectories for three values of q. This is not from an actual flight, but only a simulation using the handed out code.

of reality. The helicopter drifts, after it is supposed to be at the final point. It does not stop moving in the travel direction.

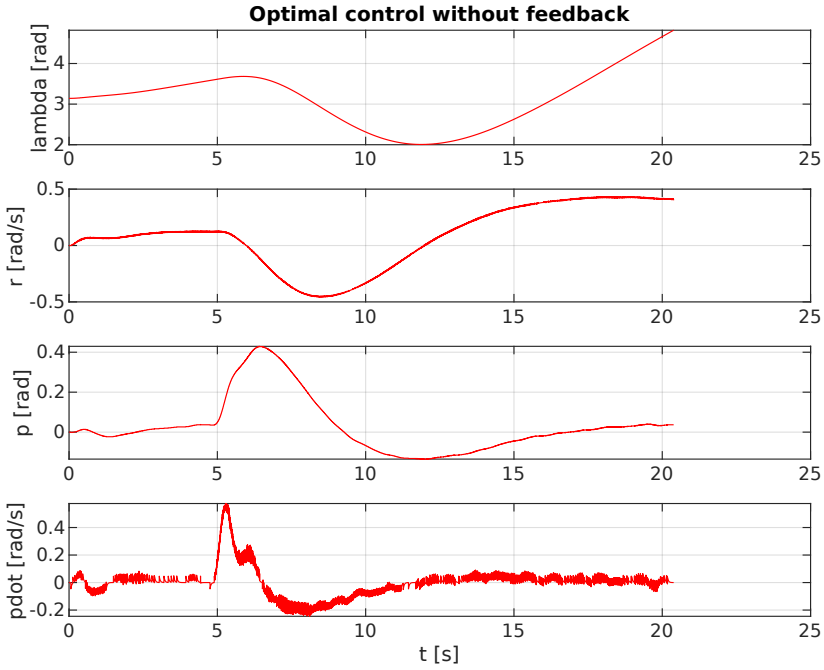


Figure 2: Experimental result from day 2. Here you can see the different states in the statevector. That is travel, pitch and their rates of change. The optimal path is not plottet, since it was not followed in a good way.

1.6 MATLAB and Simulink

```
%% Defining model (not shown here)
Aeq = gen_aeq(A1,B1,N,mx,mu);
beq = [zeros(N*mx, 1)];
beq(1:4, 1) = A1*x0;
%% Solve QP problem with linear model
[z,lambda] = quadprog(Q, c, [], [], Aeq, beq, vlb,
    vub, x0);
% Calculate objective value
phi1 = 0.12;
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

%% Extract control inputs and states
```

```
16 u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)]; % Control
       input from solution
17
18 x1 = [x0(1);z(1:mx:N*mx)];              % State x1
       from solution
19 x2 = [x0(2);z(2:mx:N*mx)];              % State x2
       from solution
20 x3 = [x0(3);z(3:mx:N*mx)];              % State x3
       from solution
21 x4 = [x0(4);z(4:mx:N*mx)];              % State x4
       from solution
22
23 num_variables = 5/delta_t;
24 zero_padding = zeros(num_variables,1);
25 unit_padding  = ones(num_variables,1);
26
27 %% adding padding (not shown here)
28
29 t = 0:delta_t:delta_t*(length(u)-1);
30 simU = timeseries(t, u)
```

As for the simulink-program, it was only the template handout with a 'from workspace'-block giving in the timeseries of the optimal input to the pitch reference.

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

2.1 LQ controller

The LQ-controller is a linear quadratic controller aiming to minimize a certain function involving all the states and inputs. It uses a Q matrix for penalizing deviation in states and a R matrix for penalizing using much fuel (input). The relative weighting between the elements is what decides the output of the system. if the r value on the diagonal specifying input to pitch controller is high, then the system will prioritize using small values there. The LQ approach is deciding a feedback matrix to stabilize the system. That means, given matrices A, B, Q and R, decide the optimal matrice K, which decides the feedback in the loop. Creating stability is easier with LQR controller than with pole-placement, because the numbers you put in gives more intuitive sense. The objective function to be minimize is as following.

$$\min_{\Delta u} \sum_{i=0}^{\infty} \frac{1}{2} \left(\Delta x_{i+1}^T Q \Delta x_{i+1} + \Delta u_i^T R \Delta u_i \right), \qquad Q \geq 0, \quad R > 0$$

Where $\Delta u = u - u^* \quad and \quad \Delta x = x - x^*$
It should also be stated that you find the optimal gain matrix (K), by solving the riccati-equation and inserting into a formula. The equations are shown below:

$$\begin{aligned} P &= A^T P A - A^T P B (R + B^T P B)^{-1} B^T P A + Q \\ K &= (R + B^T P B)^{-1} B^T P A \\ u_k &= u_k^* - K(x_k - x_k^*) \end{aligned}$$

This way we achieve feedback control. With some trial and error we ended up including these two choices for Q and R:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R = 1 \implies K = \begin{bmatrix} -0.6488 \\ -2.4747 \\ 1.3122 \\ 0.4625 \end{bmatrix}^T$$

Just to see how good a basic controller is. And:

$$Q = \begin{bmatrix} 2.6 & 0 & 0 & 0 \\ 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 0.2 \end{bmatrix}, R = 1 \implies K = \begin{bmatrix} -1.2607 \\ -3.6876 \\ 1.7091 \\ 0.4531 \end{bmatrix}^T$$

This was found to be a good choice, because it weights the travel higher. Thereby avoiding too fast oscillations in the pitch. The lqr is responsible

for correcting model deviations and other disturbances etc, as mentioned in the previous task. This means that the optimization puts the system close enough to the optimum, so that the lq controller can "finish" the job.

2.2 Model Predictive Control

We could instead of a lq-controller, just send the newest measurement from the helicopter, then use it as the starting point in the optimization problem, then return the opdated first step (u*) of the solution. This way we introduce feedback in the system. However an mpc would use much more computational effort, and it would have to do calculations on the fly. With the open loop solution and the lq-controller, everything of computation is already done when starting the helicopter. But, the fact that the mpc always recalculates the optimal path, makes it better at handling the types of errors that can't easily be foreseen. For example if the optimal path ends up changing alot, the lq controller wants to return to the same path, while the mpc could calculate a new and cheaper path. Lastly the mpc is regarded more safe in terms of constraints. Because, even though the open loop solution takes constraints into consideration, the lq-controller does not.
The MPC would be implemented like in the scheme in Figure 3.

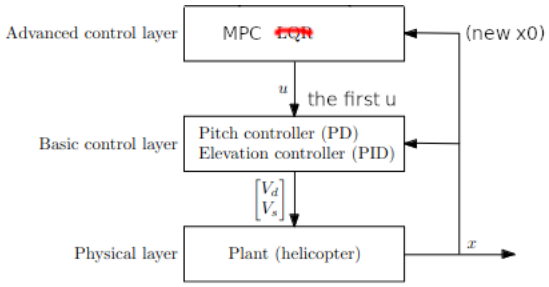
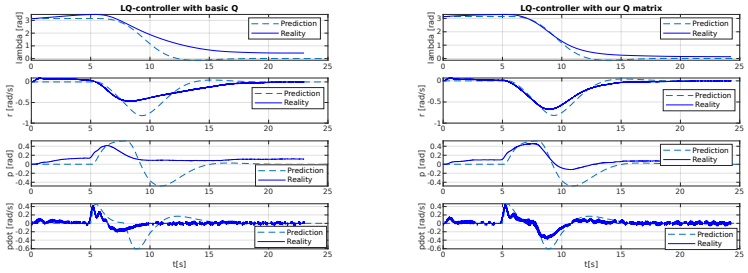


Figure 3: Schematics of MPC. It is a modified image from the document containing the exercises.

2.3 Experimental results

The plots from the lq-controller are shown in Figure 4. We observe that our weighting creates a better result which is better at following the optimal path. From the experiment the path observed is plotted alongside the optimum trajectory from theory, in dashed lines. The same state vector: travel, pitch and their rates of change, was used. The response was much better than in the previous task, and the target $\lambda_f = 0$ was reached.



(a) With diagonal Q with ones and where r is 1

(b) With specified Q. See commentet out code.

Figure 4: Plots from the helicopter with LQ-controller and two different Q matrices. The precalculated optimal paths is also plottet in dashed lines.

2.4 MATLAB and Simulink

Below you can see some of our code, and our setup in Figure 5. Most of the code is the same as in the previous task.

```
%% LQR
Q_e = eye(4);
% our Q_e = [2.6 0 0 0; 0 0.6 0 0; 0 0 0.3 0; 0 0 0
    0.2];
R_e = 1;

[K, S, e] = dlqr(A1, B1, Q_e, R_e);

t = 0:delta_t:delta_t*(length(u)-1);
simU = timeseries(u, t);
x = [x1 x2 x3 x4];
simX = timeseries(x, t);
```

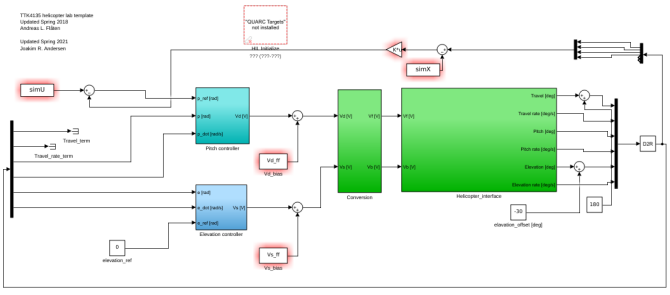


Figure 5: Day3 Simulink

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

3.1 The continuous model

This uses the state vector $x = \begin{bmatrix}\lambda & r & p & \dot{p} & e & \dot{e}\end{bmatrix}^T$ and $u = \begin{bmatrix}p_c & e_c\end{bmatrix}^T$. We are therefore also including elevation dynamics in our model.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1 \cdot K_{pp} & -K_1 \cdot K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3 \cdot K_{ep} & -K_3 \cdot K_{ed} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -(K_1 \cdot K_{pp})/4 & 0 \\ 0 & 0 \\ 0 & (K_3 \cdot K_{ep})/4 \end{bmatrix}$$

3.2 The discretized model

With $\Delta t = 0.25$
Using once again the forward Euler method

$$A_d = I_6 + A \cdot \Delta t$$

$$= \begin{bmatrix} 1 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 1 & -K_2/4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1/4 & 0 & 0 \\ 0 & 0 & -(K_1 \cdot K_{pp})/4 & 1-(K_1 \cdot K_{pd})/4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1/4 \\ 0 & 0 & 0 & 0 & -(K_3 \cdot K_{ep})/4 & 1-(K_3 \cdot K_{ed})/4 \end{bmatrix}$$

$$B_d = B \cdot \Delta t = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -(K_1 \cdot K_{pp})/4 & 0 \\ 0 & 0 \\ 0 & (K_3 \cdot K_{ep})/4 \end{bmatrix}$$

3.3 Experimental results

The result of the full model is shown in Figure 6. As we can see the control was a little bit poor. We are not entirely sure as to why our results doesn't follow the optimal trajectory. This may be faults in the code, the tuning or

in the helicopter. Nevertheless it almost follows the trajectory. From the results it looks like the elevation measurement starts off at a funky place, by being negative. Here we should also mention the effect of thinking we have a decoupled model, when it in fact is more complicated. For example pitch will result in less lift.

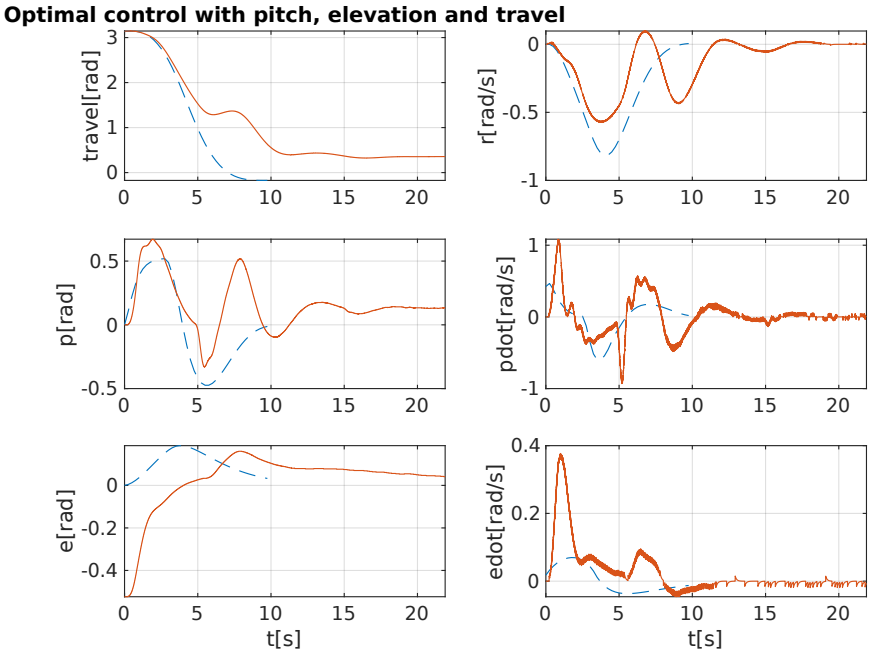


Figure 6: Results with feedback for both elevation and pitch

3.4 Decoupled model

If one imagine the helicopter being sideways, then it cannot produce force to get neither up or down. If one were to improve the model, there is many options. One would be to just use the nonlinear equations. However this would prevent us from using an linear quadratic controller. Another option would be to add more non-linearity in the objective function. This would probably make it avoid pitch values where the model is poor. But it is only in calculating the optimum trajectory. Our linear quadratic controller would still have the same problem. Therefore the solution we view as the best is to update the q-matrix every once in a while, based on the pitch. But we may be wrong.

3.5 MATLAB and Simulink

Underneath you may have a look at our code and our simulink model (see Figure 7).

```
N  = 40;
M  = N;
z  = ones(N*mx+M*mu,1);
z0 = z;

ul           = [-30/180*pi; -inf];
uu           = [30/180*pi;  inf];

xl       = -Inf*ones(mx,1);
xu       = Inf*ones(mx,1);
xl(3)    = ul(1);
xu(3)    = uu(1);

[vlb,vub]        = gen_constraints(N,M,xl,xu,ul,uu);
vlb(N*mx+M*mu)   = 0;
vub(N*mx+M*mu)   = 0;

%% everything not shown

q = zeros(mx);
q(1) = 1;
Q = gen_q(q,diag([1 1]),N,N);
fun = @(z) z'*Q*z;
options = optimoptions('fmincon', 'Algorithm', 'sqp'
   , 'MaxFunEvals', 10e4);
z = fmincon(@(z) z'*Q*z, z0, [], [], Aeq, beq, vlb,
   vub, @mycon, options);

u_p = z(mx*N+1:2:(mx+mu)*N);
u_e = z((mx*N+2):2:(mx+mu)*N);

Rlqr = eye(2);
Qlqr = eye(6);
[K, S, e] = dlqr(A1, B1, Qlqr, Rlqr);

% constraint function
function [c,ceq] = mycon(x)
N = 40;
alpha = 0.2;
beta = 20;
```

```
lambda_t = 2*pi/3;
c = zeros(N, 1);
mx = 6;
mu = 2;
for i = 1:N
    c(i) = alpha*exp(-beta*(x(mx*(i-1)+1)-lambda_t)
       .^2)-x(mx*(i-1)+5);
end
ceq = [];
end
```

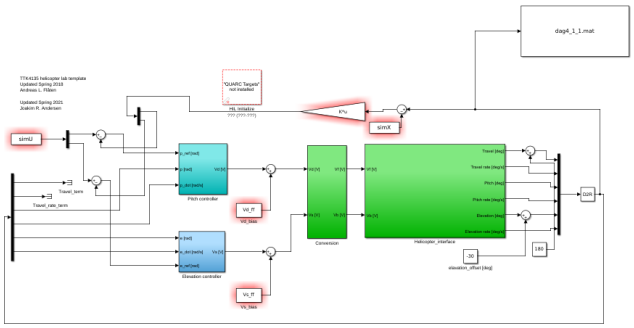


Figure 7: Day4 Simulink