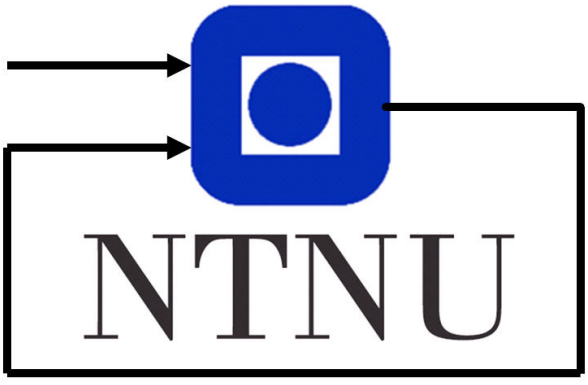
OptReg lab report

Students:
490947
492647

April 11, 2021



Department of Engineering Cybernetics

Contents

1 Optimal Control of Pitch/Travel without Feedback

1.1 Derivation of a continous time state space model

In this part of the exercise we will disregard elevation, therefore we assume $e = 0$ and do include it in the model.

The state-vector, $\boldsymbol{x}$ is defined as:

$$\boldsymbol{x} = \begin{bmatrix} \lambda & r & p & \dot{p} \end{bmatrix}^T, \tag{1.1}$$

where λ is travel, r is the travel rate, p is pitch and $\dot{p}$ is pitch rate.

The dynamic equations for the system was given in the problem description [1]. These following equations were given:

$$\dot{\lambda} = r \tag{1.2a}$$

$$\dot{r} = -K_2 p, \quad K_2 = \frac{K_p l_a}{J_t} \tag{1.2b}$$

$$\dot{p} = \dot{p} \tag{1.2c}$$

$$\ddot{p} = -K_1 V_d = K_1 K_{pd}\dot{p} - K_1 K_{pp} p + K_1 K_{pp} pc, \quad K_1 = \frac{K_f l_h}{J_p} \tag{1.2d}$$

The state-space form of the system therefore becomes:

$$\underbrace{\begin{bmatrix} \dot{\lambda} \\ \dot{r} \\ \dot{p} \\ \ddot{p} \end{bmatrix}}_{\dot{\boldsymbol{x}}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix}}_{\boldsymbol{A}_c} \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix}}_{\boldsymbol{B}_c} \underbrace{p_c}_{u} \tag{1.3}$$

1.1.1 A deeper dive into the state-space model

The state-space form of the system models two part of the whole system, namely:

1. The physics of the helicopter.
2. The proportional-derivative controller for the pitch.

This is shown in fig. 1 where the red dotted box shows what we model with the state space model.

This becomes clear studying the equations in eq. (1.2). They describe the helicopters physics for all states except elevation and elevation rate. λ and r is dependent on the helicopters pitch, p. p and $\dot{p}$ is however dependent on the voltage difference, V_d. The voltage difference is the output of the PD-controller for controlling the pitch.

To summarize, this means that our state space model describes the helicopter's physics through the dynamic equations for λ, r, p and $\dot{p}$, while the equation of V_d describes the PD-controller used to control the pitch angle. In total our state-space model is modelling both the helicopter and the PD controller for pitch.

1.2 Discretizing the continous time model

A discretized model is required for generating an optimal trajectory. *[...] continous time models require quite different solution methods* [2]

We will discretize the model using the forward Euler method, which is given by:

$$\boldsymbol{x}[k+1] = \boldsymbol{I}\boldsymbol{x}[k] + T\boldsymbol{A_c}\boldsymbol{x}[k] + T\boldsymbol{B}_c, \tag{1.4}$$

where T is the sample-time in the discrete model. Reformulating this, we can write:

$$\boldsymbol{x}_{k+1} = \underbrace{(\boldsymbol{I} + T\boldsymbol{A}_c)}_{\boldsymbol{A}_d}\boldsymbol{x}_k + \underbrace{T\boldsymbol{B}_c}_{\boldsymbol{B}_d} u_k \tag{1.5}$$

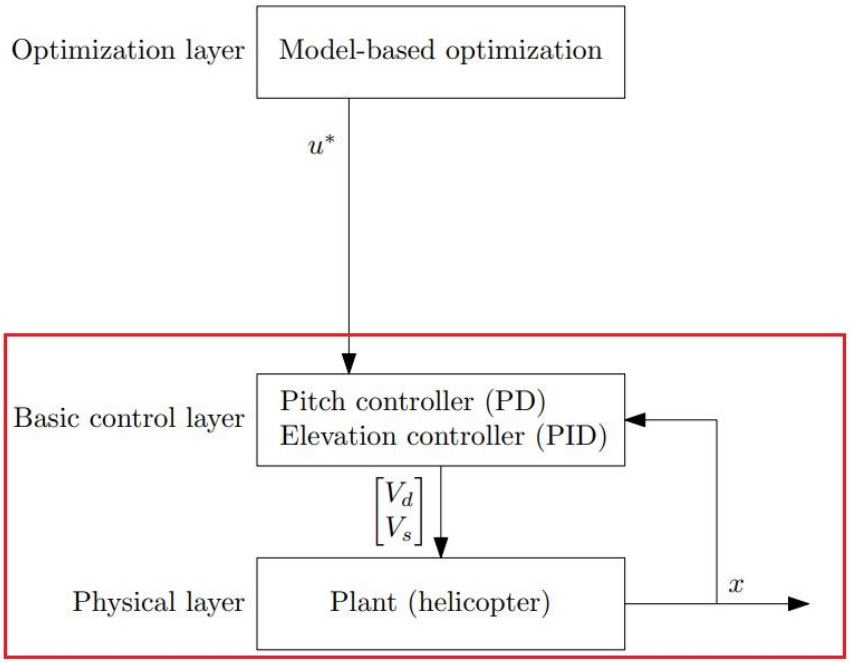


Figure 1: The red box encapsulate what is modelled in the state space model described by eq. (1.3).

On matrix form $\boldsymbol{A}_d$ and $\boldsymbol{B}_d$ becomes:

$$\boldsymbol{A}_d = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & -TK_2 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & -TK_1K_{pp} & 1 - TK_1K_{pd} \end{bmatrix}, \quad \boldsymbol{B}_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ TK_1K_{pp} \end{bmatrix} \tag{1.6}$$

1.2.1 Checking stability

The stability condition for eq. (1.5) is that all eigenvalues of $\boldsymbol{A}_d$ is less than one in absolute values, i.e.:

$$|\lambda_i| \leq 1, \quad \text{for } i = 1, 2, 3, 4 \tag{1.7}$$

, where λ_i is the i'th eigenvalue of $\boldsymbol{A}_d$.

Using the script in listing 1 the eigenvalues became:

$$\lambda_1 = 1 \tag{1.8a}$$
$$\lambda_2 = 1 \tag{1.8b}$$
$$\lambda_3 = 0.55 \tag{1.8c}$$
$$\lambda_4 = 0.55 \tag{1.8d}$$

All eigenvalues fulfills the requirement of eq. (1.7), which means our system is stable! Since two of the eigenvalues are on the unit circle (i.e. has a value of 1), our system is marginally stable.

Listing 1: MATLAB code for calculating eigenvalues for $\boldsymbol{A}_d$

```
clc
clear all

init06;
T = 0.25;
Ad = [1 T 0 0;
    0 1 -T*K_2 0;
    0 0 1 T;
    0 0 -T*K_1*K_pp 1-T*K_1*K_pd];
e = eig(Ad)
```

1.3 The open loop optimization problem

The open loop optimization problem is a constrained quadratic program that can be solved using the Matlab function quadprog. The quadprog function requires the input to be in standard form:

$$\min_x \frac{1}{2}\boldsymbol{x}^t\boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^T\boldsymbol{x} \quad \text{such that} \begin{cases} \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{B}_{eq}, \\ \boldsymbol{lb} \leq \boldsymbol{x} \leq \boldsymbol{ub} \end{cases} \tag{1.9}$$

The next subsections describes this process.

1.3.1 Formulating the cost function

As the problem description states, the cost function used in this exercise is:

$$\phi = \sum_{i=0}^{N-1} \left(\lambda_{i+1} - \lambda_f\right)^2 + qp_{ci}^2, \quad q \geq 0 \tag{1.10}$$

Which can be rewritten with $\boldsymbol{x}$ and u:

$$f(\boldsymbol{x}, u) = \sum_{i=0}^{N-1} \left(\boldsymbol{x}_{i+1} - \boldsymbol{x}_f\right)^T \boldsymbol{Q} \left(\boldsymbol{x}_{i+1} - \boldsymbol{x}_f\right) + qu^2 \tag{1.11}$$

where

$$\boldsymbol{x}_f = \begin{bmatrix} \lambda_f \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{1.12}$$

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.13}$$

In this case we have that $\lambda_f = 0$ and therefore $\boldsymbol{x}_f = \boldsymbol{0}$, and eq. (1.11) can be rewritten as:

$$f(\boldsymbol{x}, u) = \sum_{i=0}^{N-1} \boldsymbol{x}_{i+1}^T \boldsymbol{Q} \boldsymbol{x}_{i+1} + ru^2 \tag{1.14}$$

where

$$r = q \tag{1.15}$$

Defining the the z vector as:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & ... & \boldsymbol{x}_n & u_0 & u_1 & ... & u_{n-1} \end{bmatrix}^T \tag{1.16}$$

we can rewrite summation form in eq. (1.14) to matrix-form:

$$f(\boldsymbol{z}) = \boldsymbol{z}^T \boldsymbol{H} \boldsymbol{z} \tag{1.17}$$

with

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{Q} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \boldsymbol{Q} & \cdots & 0 & 0 & 0 & \cdots & 0 \\ & & \ddots & & & & & \\ 0 & 0 & \cdots & \boldsymbol{Q} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & r & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & r & \cdots & 0 \\ & & & & & & \ddots & \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & r \end{bmatrix} \tag{1.18}$$

1.3.2 The constraints of the optimization problem

There are two separate types of constrains in this problem, the system itself and imposed constraints. The system constrains is described by the physics of the helicopter, which we have modeled in eq. (1.5). Imposed constraints are constraints added by the designer of the system.

In this case we have an imposed constraint on the pitch given by:

$$|p_k| \leq \frac{30}{180}\pi, k \in \{1, ..., N\} \tag{1.19}$$

As the manipulated variable p_c is the setpoint for the p controller, this constraint must also be implemented for the setpoint. Since there is no inequality constraints on the other states, they have lower bounds of $-\infty$ and upper bounds of ∞. This gives us the inequality constraints for the state and the input as:

$$\boldsymbol{x}^{low} = \begin{bmatrix} -\infty \\ -\infty \\ -\frac{30}{180}\pi \\ -\infty \end{bmatrix} \leq \boldsymbol{x} \leq \boldsymbol{x}^{high} = \begin{bmatrix} \infty \\ \infty \\ \frac{30}{180}\pi \\ \infty \end{bmatrix} \tag{1.20a}$$

$$u^{low} = -\frac{30}{180}\pi \leq u \leq u^{high} = -\frac{30}{180}\pi \tag{1.20b}$$

We can now easily expand this to define lower- and upper-bounds for the vector $\boldsymbol{z}$ given in eq. (1.16)

$$\boldsymbol{z}^{low} = \begin{bmatrix} \boldsymbol{x}^{low} \\ \vdots \\ \boldsymbol{x}^{low} \\ u^{low} \\ \vdots \\ u^{low} \end{bmatrix} \leq \boldsymbol{z} \leq \boldsymbol{z}^{high} = \begin{bmatrix} \boldsymbol{x}^{high} \\ \vdots \\ \boldsymbol{x}^{high} \\ u^{high} \\ \vdots \\ u^{high} \end{bmatrix} \tag{1.21}$$

The system constraints can be defined as:

$$\boldsymbol{A}_{eq}\boldsymbol{z} = \boldsymbol{b}_{eq} \tag{1.22}$$

, where

$$\boldsymbol{A}_{eq} = \begin{bmatrix} \boldsymbol{I} & 0 & \cdots & \cdots & 0 & -\boldsymbol{B}_d & 0 & \cdots & \cdots & 0 \\ -\boldsymbol{A}_d & \boldsymbol{I} & \ddots & & \vdots & 0 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 & \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\boldsymbol{A}_d & \boldsymbol{I} & 0 & \cdots & \cdots & 0 & -\boldsymbol{B}_d \end{bmatrix}, \ \boldsymbol{b}_{eq} = \begin{bmatrix} \boldsymbol{A}_d\boldsymbol{x}_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.23}$$

Performing the multiplication (and abusing notation) shows that:

$$\boldsymbol{A}_{eq}\boldsymbol{z} = \boldsymbol{b}_{eq} \implies \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{B}_d u_0 = \boldsymbol{A}_d\boldsymbol{x}_0 \\ -\boldsymbol{A}_d\boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{B}_d u_1 = 0 \\ \vdots \\ -\boldsymbol{A}_d\boldsymbol{x}_{n-1} + \boldsymbol{x}_n - \boldsymbol{B}_d u_{n-1} = 0 \end{bmatrix} \implies \begin{bmatrix} \boldsymbol{x}_1 = \boldsymbol{A}_d\boldsymbol{x}_0 + \boldsymbol{B}_d u_0 \\ \boldsymbol{x}_2 = \boldsymbol{A}_d\boldsymbol{x}_1 + \boldsymbol{B}_d u_1 \\ \vdots \\ \boldsymbol{x}_n = \boldsymbol{A}_d\boldsymbol{x}_{n-1} + \boldsymbol{B}_d u_{n-1} \end{bmatrix} \tag{1.24}$$

Equation (1.24) describes the system in terms of eq. (1.5) for all timesteps, and thereby also describes all system constraints for the system.

1.3.3 Defining the open loop optimization problem

Combining the cost function described in section 1.3.1 and the constraints described in section 1.3.2 the open loop optimization problem can be defined as:

$$f(\boldsymbol{z}) = \boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} \quad \text{s.t.} \quad \begin{cases} \boldsymbol{A}_{eq}\boldsymbol{z} = \boldsymbol{B}_{eq} \\ \boldsymbol{z}^{low} \leq \boldsymbol{z} \leq \boldsymbol{z}^{high} \end{cases} \tag{1.25}$$

, where $\boldsymbol{z}$ is given by eq. (1.16), $\boldsymbol{H}$ is given by eq. (1.18), $\boldsymbol{A}_{eq}$ and $\boldsymbol{B}_{eq}$ is given by eq. (1.23), and $\boldsymbol{z}_{low}$ and $\boldsymbol{z}_{high}$ is given by eq. (1.21).

It is now easy to compare this to the QP formulation given in eq. (1.9), and implement this in MATLAB. This is done in listing 3.

NB: that the variable names are not exactly the same in the code as in the sections above, but the procedure is the same.

1.4 The weights of the optimization problem

$\boldsymbol{Q}$ and r in eq. (1.14) are typically called weight-matrices. Their relative values reflect how the LQ regulator prioritize the objectives (the state and the input) relative ot each other. In this problem, $\boldsymbol{Q}$ is kept constant and the group changed the value of r (or q as it is called in the problem description). Increasing the value of q means that we are placing a higher cost of input - reducing the input usage. In other words, a smaller q will make the "fuel" (or input) expensive. The result is lower input usage and a slower response. This is exactly what is seen in fig. 2.

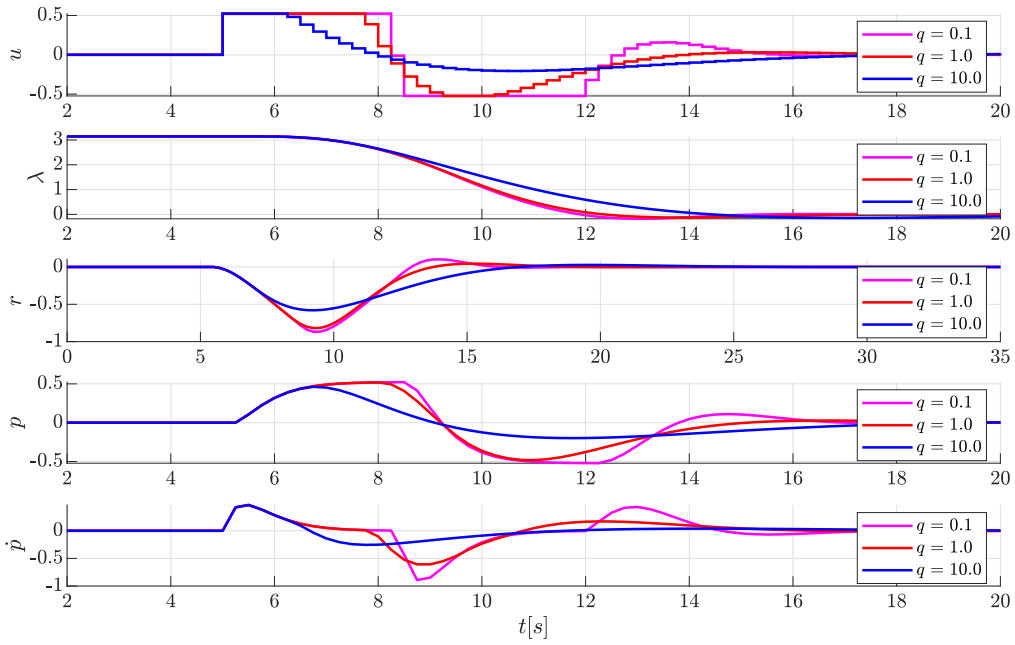


Figure 2: Optimal trajectories for the manipulated variable and outputs with different values of q.

Figure 2 also shows how the constraints work in practice. Studying the plot of u, we can see that no matter how low q is, the input never goes beyond the specified range of $\pm\frac{30}{180}\pi$.

1.5 The objective function

The term $(\lambda_i - \lambda_f)^2$ in eq. (1.10) may cause problems if it becomes too large compared to the term $q \cdot u^2$. What is meant by "too large" is that the solution will actually never reach λ_f in the given time horizon, because it is theoretically impossible due to the problem constraints. Figure 3 shows this. This is the solution of the optimization problem when the group used $\lambda_0 = 40\pi$. It shows that when the error term for the travel is too large, the helicopter will never reach λ_f no matter what. A direct consequence of this is that the helicopter will never stop the rotation before the time horizon ends. Figure 3 also shows that increasing $(\lambda_i - \lambda_f)^2$ has the same effect as increasing $\boldsymbol{Q}$ relative to r in eq. (1.14); it will prioritize to minimize $(\lambda_i - \lambda_f)^2$ and use much input to achieve this. We say that the "fuel" is cheap.

The "solution" to this problem is to increase the time horizon either by adding more steps (increase N) or increase the sampling time. This will give the optimization problem enough time to reach the λ_f.

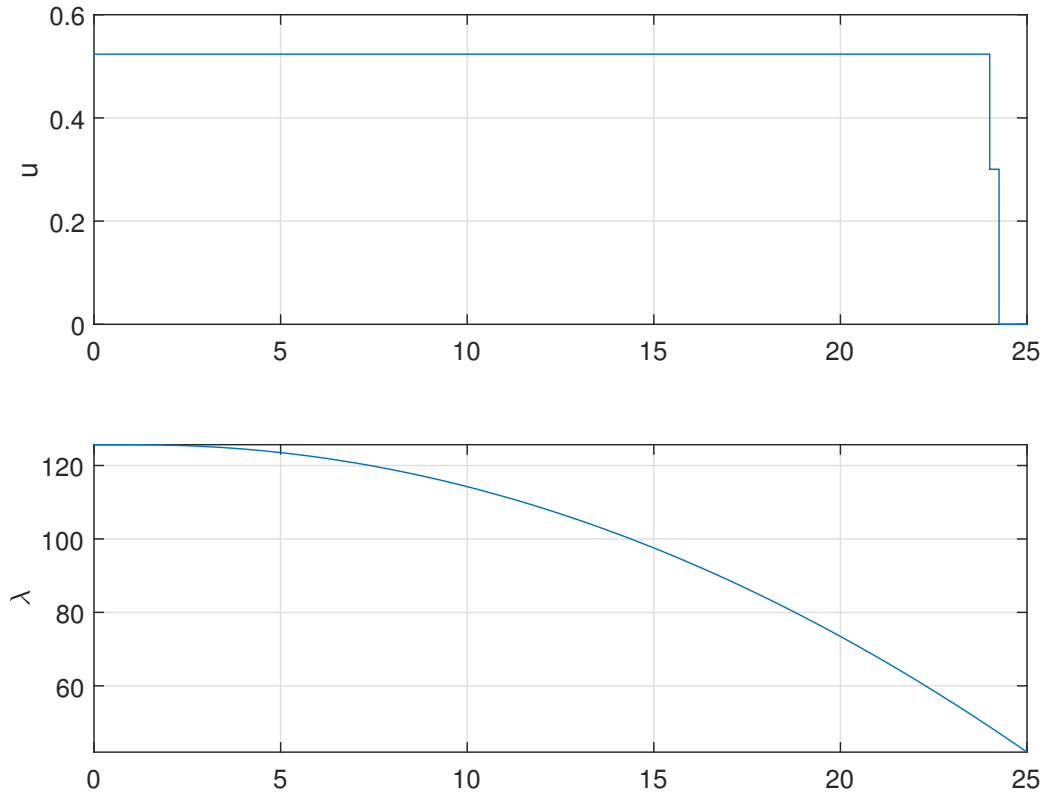


Figure 3: Optimal trajectories using $\lambda_0 = 40\pi$.

This is shown in fig. 4. In this case the sampling time was doubled from 0.25 to 0.5, and the optimization problem had almost enough time to stop rotating at the final value.

1.5.1 Improved λ_0 selection

In the case of the lab setup, it does not really make any sense to have $\lambda_i - \lambda_f > 2\pi$, because the helicopter is always on a circles edge. The result of this is that $\lambda = 0$, $\lambda = 2\pi$ and $\lambda = 4\pi$ and so on, is essentially the same helicopter-configuration.

Also, the way the objective function is set up, it will not choose the always choose the shortest way to λ_f. If $\lambda_0 = \frac{3}{4}\pi$, this means that the helicopter is $\frac{3}{4}\pi$ radians from λ_f in the counter-clockwise direction and $\frac{1}{4}\pi$ radians from λ_f in the clockwise direction. Therefore, it is most logical to go the clockwise direction, because this requires the least amount of input. However, the way the objective function is chosen, it will go in the counter-clockwise direction.

Therefore, a suggested solution from the group, is to use the following formula for λ_0:

$$\lambda_{0,\text{modified}} = \begin{cases} +\text{mod}(\lambda_0 - \lambda_f, 2\pi), & \text{if} \quad \text{mod}(\lambda_0 - \lambda_f, 2\pi) \leq \pi \\ -(2\pi - \text{mod}(\lambda_0 - \lambda_f, 2\pi)) & \text{else} \end{cases} \tag{1.26a}$$

$$\tag{1.26b}$$

, where mod is the modulus operator.

This will insure that $\lambda_i - \lambda_f \leq 2\pi$ for all $i \in (0, 1, 2, ..., N-1)$ and it will also insure that the helicopter will choose the shortest direction.

It is easily implemented in the code, as listing 2 shows.

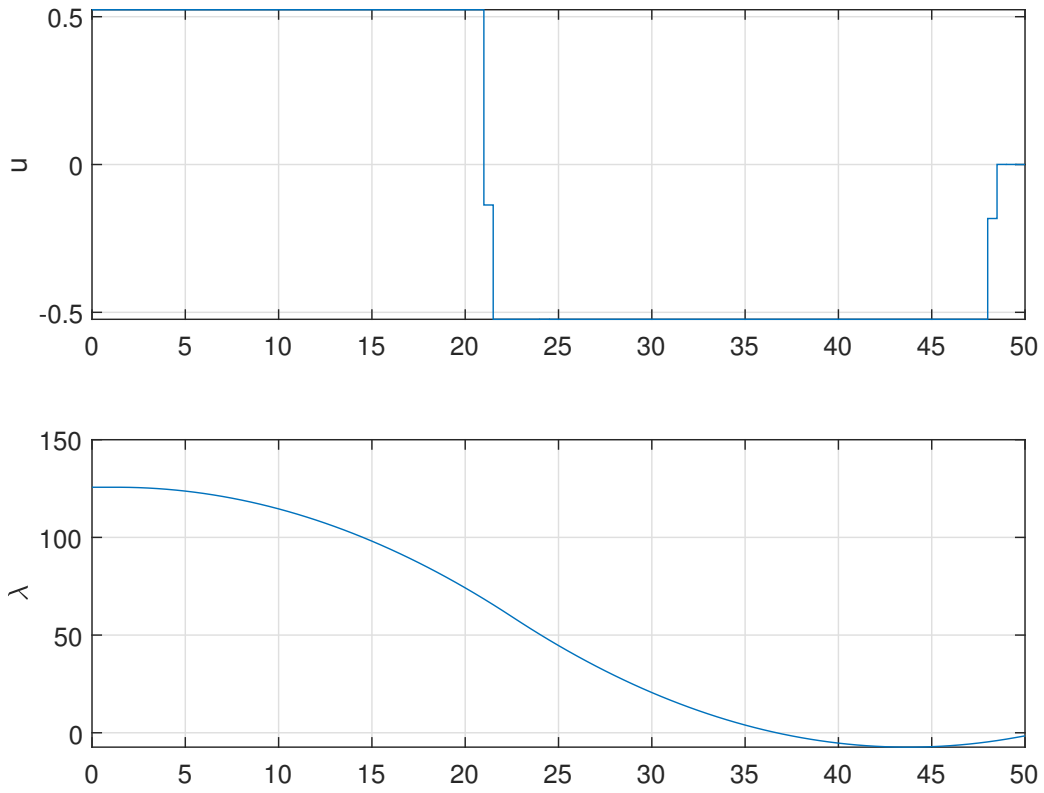


Figure 4: Optimal trajectories using $\lambda_0 = 40\pi$ and $\Delta t = 0.5$.

Listing 2: Improved λ_0 implementation.

```
lambda_0 = 4/3*pi; %Some value
lambda_0 = mod(lambda_0, 2*pi);
if (lambda_0 > pi)
        lambda_0 = -(2*pi - lambda_0);
end
```

1.6 Experimental results

The group performed two experiments with the helicopter:

1. Flight with optimal setpoints: $u_k = u_k^*$
2. Flight without setpoints: $u_k = 0$

Both fligths, along with the optimal trajectory and a compensated trajectory are plotted in fig. 5.

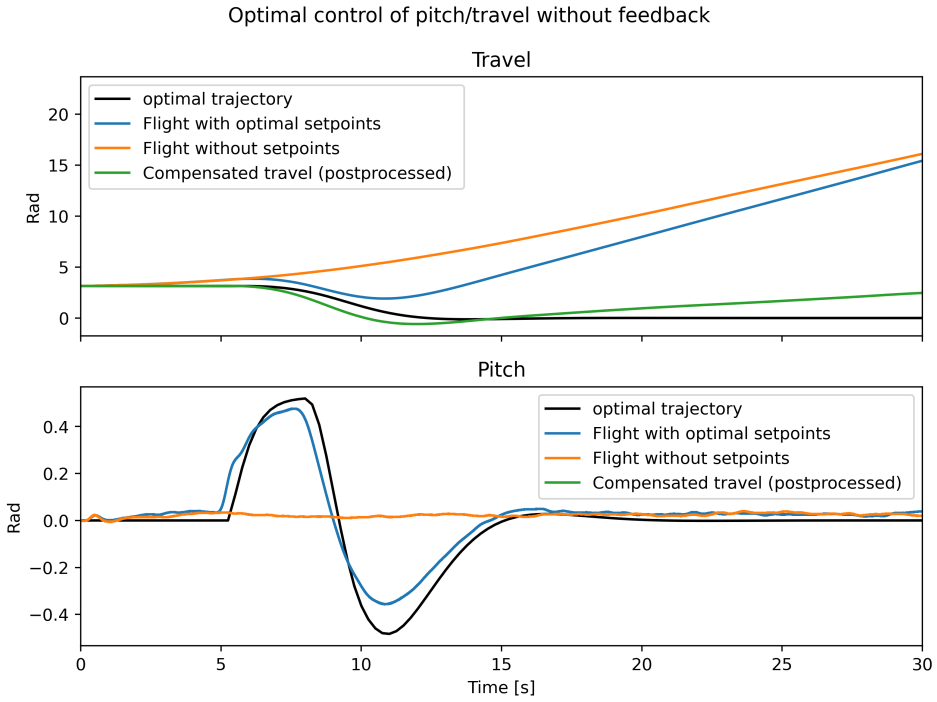


Figure 5: Results of LAB2

1.6.1 Pitch-control

It is clear that both fligths have adequate control of pitch. In the case where $u_k = 0$ the pitch is close to 0. In the case where $u_k = u_k^*$ the pitch is close to the optimal trajectory.

This is expected as pitch-control is a closed loop - the basic control layer has a PD controller that ensures that pitch follows the reference u_k.

1.6.2 Travel-drift

From the flight with $u_k = 0$ it is clear that there is significant travel-drift present in the helicopter. Even when pitch is very close to 0 the helicopter travels quickly far away from the initial point.

There are three main causes of drift:

1. The encoder-measurement of pitch is not precise
2. Helicopter inbalance
3. Disturbances

When the helicopter starts, it reads the initial **encoder**-value for the pitch, and assumes this is equal to zero. However, if the platform is not built precisely, the encoder axis may not be 100 % aligned with the corresponding real world axis. There are many possible reasons for this, but it will not be discussed further. Regardless of the cause, the effect will be the same: The measured encoder-value will have a constant offset compared to real world-values. This will again cause the helicopter to drift when it measures zero pitch. This offset is clearly shown in fig. 9 where the helicopter with feedback pitches approximately 5.7° to keep the travel constant.

The **imbalance** may be that one rotor is stronger than the other, or that the rotors have offsets from the vertical position. These imbalances may generate a sidewards force, even when the helicopter is pitched to 0, which in turn generates travelrate and travel.

The **disturbances** is the most general effect. Air-pressure, wind, temperature effects, walls, and so on. All of these disturbances may cause drifts or noise in the travel.

The group believes that the main cause of the drift observed in the helicopter is the offset in the encoder.

1.6.3 Conclusion

As fig. 5 clearly shows the control sequence did not yield the desired travel-response. This was expected, as there are many causes of drift and disturbances. Without any feedback the helicopter will quickly deviate from the desired response, since it will not have any mechanisms to detect deviations from the optimal trajectories.

Nevertheless it is possible to verify the control sequence by compensating for the drift. The compensation can be done by taking the travel of the flight with optimal setpoints, and subtracting the drift logged in the flight without setpoints (i.e. $u_k = 0$). The result is plotted as compensated travel in fig. 5. This clearly shows that without the major drift the helicopter would be much closer to the optimal trajectory.

1.7 MATLAB and Simulink

1.7.1 MATLAB

Listing 3: MATLAB code for lab 2

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2018, Andreas L. Flten

%% Initialization and model definition
init06; % Change this to the init file corresponding to your helicopter

% Continous time model
Ac = [0, 1,        0,        0;
      0, 0,     -K_2,        0;
      0, 0,        0,        1;
      0, 0, -K_1*K_pp, -K_1*K_pd];

Bc = [     0;
           0;
           0;
      K_1*K_pp];

% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25; % sampling time
A1 = eye(4) + (delta_t.*Ac);
B1 = (delta_t.*Bc);

% 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)

% Trajectory start and end values
lambda_0 = pi;
lambda_f = 0;
% Initial values
x1_0 = lambda_0;                          % Lambda
x2_0 = 0;                                 % r
x3_0 = 0;                                 % p
x4_0 = 0;                                 % p_dot
x0 = [x1_0 x2_0 x3_0 x4_0]';              % Initial values

% Time horizon and initialization
N  = 100;                                 % 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

% Bounds
ul          = -30*pi/180;                     % Lower bound on control
uu          = -ul;                            % 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);
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
P1 = 1;                                     % Weight on input
Q = gen_q(Q1,P1,N,M);                       % Generate Q, hint: gen_q
% The constant linear term is zero in our case
c = zeros(size(Q, 2), 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 = zeros(size(Aeq, 1), mu);                  % Generate b
A0x0 = A1*x0;
beq(1:size(A0x0,1), :) = A0x0;

%% Solve QP problem with linear model
tic
% x = quadprog(H,f,A,b,Aeq,beq,lb,ub);
[z,lambda] = quadprog(2*Q, c,[],[], Aeq, beq, vlb, vub);% hint:
    quadprog. Type 'doc quadprog' for more info
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

%% Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)]; % Control input from solution

x1 = [x0(1);z(1:mx:N*mx)];                   % State x1 from solution
x2 = [x0(2);z(2:mx:N*mx)];                   % State x2 from solution
x3 = [x0(3);z(3:mx:N*mx)];                   % State x3 from solution
x4 = [x0(4);z(4:mx:N*mx)];                   % State x4 from solution

num_variables = 5/delta_t;
zero_padding = zeros(num_variables,1);
unit_padding  = ones(num_variables,1);

u   = [zero_padding; u; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
```

```
x2  = [zero_padding; x2; zero_padding];
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];

%% Plotting
t = 0:delta_t:delta_t*(length(u)-1);

figure(2)
subplot(511)
stairs(t,u),grid
ylabel('u')
subplot(512)
plot(t,x1,'m',t,x1,'mo'),grid
ylabel('lambda')
subplot(513)
plot(t,x2,'m',t,x2','mo'),grid
ylabel('r')
subplot(514)
plot(t,x3,'m',t,x3,'mo'),grid
ylabel('p')
subplot(515)
plot(t,x4,'m',t,x4','mo'),grid
xlabel('tid (s)'),ylabel('pdot')
```

1.7.2 Simulink

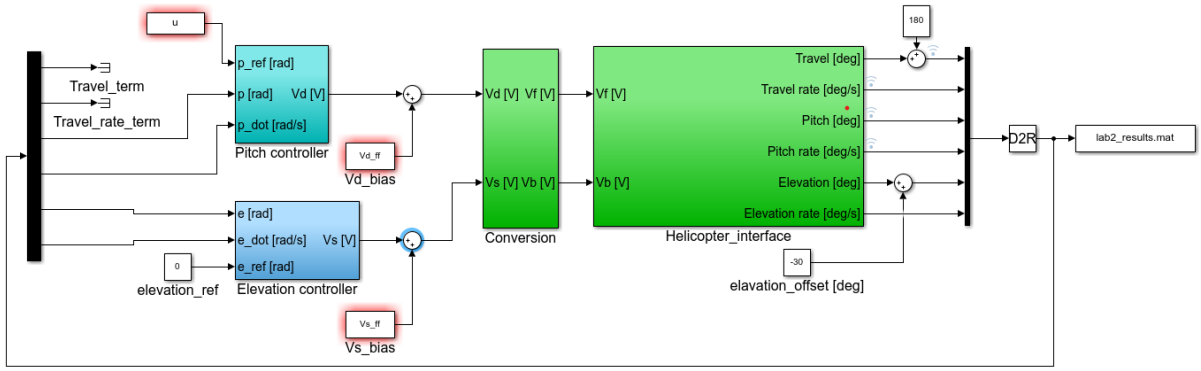


Figure 6: Simulink diagram used in lab 2.

NOTE: Some blocks in the Simulink diagram is red. This is only because the the variables are not in the workspace of MATLAB when the screenshot was taken. Running the MATLAB code in listing 3 will fix this "error". This also goes for the other simulink diagrams in this report.

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

In this task we add feedback to the optimal controller that we developed in section 1.

2.1 Motivation

The experimental results in section 1.6 clearly shows that planning a sequence of inputs does not produce the expected sequence of outputs - as that section explains there are many reasons for travel-drift. In situations with drift without feedback then the output will not follow the planned trajectory, but adding feedback could compensate for the drift and offset and therefore reduce or eliminate the deviation from the planned trajectory.

The results achieved in this section shows that adding feedback greatly improves performance.

2.2 Introducing feedback

There are many approaches to adding feedback, but in this assignment only two will be considered: linear state feedback and model predictive control.

Linear state feedback adds a new control layer below the optimization layer, the advanced control layer, that controls the setpoint to the pitch controller. The setpoint is determined from a control law that considers the optimal trajectory and the current state of the system. If the states deviate from the optimal value the input is changed according to the pitch-control equation:

$$u_k = u_k^* - \boldsymbol{K}^T(\boldsymbol{x}_k - \boldsymbol{x}_k^*) \tag{2.1}$$

where u_k^* and $\boldsymbol{x}_k^*$ are the optimal input and state trajectories predicted in the optimization layer, u_k is the next input and x_k is the current state. K is the linear state feedback gain.

In this exercise the gain matrix K is calculated as an infinite horizon LQ controller, see section 2.3 for more info about the exact implementation.

Model Predictive Control (MPC) is a completely different solution: the optimal trajectory is recalculated at every timestep - instead of compensating for deviation from optimal, the optimal is recalculated to get a new possible trajectory based on the current states. See section 2.4 for more info about how that could be implemented here.

2.3 LQ controller

A Linear Quadratic (LQ) controller minimizes the quadratic objective function:

$$J = \sum_{i=0}^{\infty} \Delta x_{i+1}^\top Q \Delta x_{i+1} + \Delta u_i^\top R \Delta u_i, \quad Q \geq 0, \quad R > 0 \tag{2.2}$$

for a linear model

$$\Delta x = A\Delta x_i + B\Delta u_i \tag{2.3}$$

Here $\Delta x = x - x^*$ and $\Delta u = u - u^*$ are deviations from the optimal trajectory.

This formulation is an infinite horizon linear quadratic regulator which has a solution with a constant linear feedback gain, K - which is exactly what was specified previously! Note that this regulator does not have any constraints.

2.3.1 Choosing the weights

The matrix Q and the scalar R are the weights of the optimalization problem. Q determines how much state-deviations should be penalized, while R determines how much input-deviation should be penalized. What needs to prioritized depends on the application.

In this case the system is a helicopter where the goal is to control the travel of the helicopter. The travel is the most important state, while travelrate, pitch and pitchrate are secondary. Thus it makes sense to prioritize keeping travel close to the planned trajectory - it is of course not possible to keep all states close to the trajectory. This motivates a **state-weight** with relatively high value related to travel.

The LQR regulator controls the pitch-setpoint that in turn controls travel. In the end controlling travel is the most important - a deviation in setpoint from the predicted optimal is completely fine as this is how travel is regulated. Thus a low input-weight is warranted as that allows the setpoint to deviate further from the planned trajectory.

2.3.2 Respecting constraints

The pitch-setpoint in the optimalization layer has a constraint, see section 1.3.2 - but the implementation of LQR controller does not respect this constraint! The result is that the pitch-setpoint could fall outside the constraint imposed in the optimalization layer. This does in fact happen, for instance in fig. 9, where the setpoint is outside the constraint ($\approx 0.52rad$).

The simplest solution to this problem is to respect the constraint by saturating the output of the regulator to that value. However that does makes impossble for the LQ regulator to compensate in cases where the input is already constrained, therefore it would make sense to set the saturation limit slightly higher than the constraint value to allow the regulator to work in such cases (or equivalently set the constraint lower than the saturation point).

2.3.3 Calculating the solution

Calculating the solution to the LQR problem is trivial using Matlab:

```
% The discrete system described as a state space system, Ad and Bd must
    be defined

% W1 is the weigth of travel
% W2 is the weight of travelrate
% W3 is the weight of pitch
% W4 is the weight of pitchrate
Q = diag([W1, W2, W3, W4]);

% W5 is the weight of the input
R = W5;

% The function dlqr is used to solve the LQR problem
[K, S, e] = dlqr(Ad, Bd, Q, R);
```

2.4 Model Predictive Control

Model Predictive Control is another way of introducing feedback to an optimal control system. In an MPC controlled system the optimal response and input is recalculated at every timestep, the input used is simply the first of the optimal input values calculated at every step.

This is a drastically different approach to the LQ-method impemented in this laboratory exercise.

2.4.1 Implementing MPC

Implementing MPC requires a system model, cost function, current state and input control.

One could make a Matlab function that the simulink model uses at every timestep. The input to this function is the current state (either measured or estimated), while the output is connected to the pitch-regulator. The function itself has a model of the system as well as the cost function - with the state from the system the optimalization problem can be solved and the first system-input is returned to the simulink model.

2.4.2 Advantages of MPC

The biggest advantage of MPC compared to the linear state feedback introduced here, is that MPC generates a new optimal trajectory that is more realistic for the helicopter to achieve. Since the optimal trajectory is recalculated based on the current state, the helicopter's response will achieve smaller errors compared to the optimal trajectory.

Another advantage is that MPC allows for feedback with explicitly defined constraints. This is a more intuitive way of keeping the helicopter stable and within the wanted working-range, compared to looking at eigenvalues of the system matrix with linear state feedback.

2.4.3 Disadvantages of MPC

The biggest disadvantage to MPC is complexity and computation time - in a control system there are hard deadlines and if the optimalization problem has not been solved within that deadline, the system has a problem.

First of all the problem must be simple enough or optimized enough to solve at every timestep. If this is not achievable the problem must be simplified/optimized or the timestep must be increased. The time to solve a problem can also vary, which makes this issue more complex.

Another issue is **existence of solutions** must be guaranteed for all situations. It is possible to have problems that have solutions in some configurations, while being undefined for others. Our helicopter is far from perfect, and our model is quite simple. These imperfections may cause the helicopter's configuration to go outside the defined constraints. This could cause the helicopter to freeze mid-air because there was no solution. If these border cases was not taken care of during implementation, this could be fatal.

2.4.4 Modified Control Hierarchy with MPC

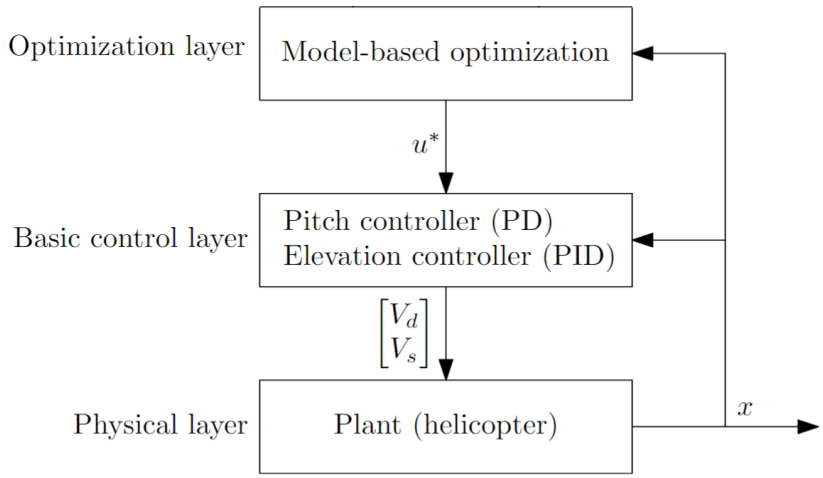


Figure 7: The control hierarchy would look like this using MPC.

2.5 Experimental results

The group performed a host of controlled tests with different tunings within five different targets:

1. Prioritizing input use, see fig. 8
2. Prioritizing travel, see fig. 9
3. Prioritizing travelrate, see fig. 10
4. Prioritizing pitch, see fig. 11
5. Prioritizing pitchrate, see fig. 12

It is very clear that introducing feedback produced results much better than the ones achieved without feedback, almost regardless of the tuning of the LQR regulator.

Prioritizing input usage only has a slight effect, the input usage does get slightly closer to the planned trajectory but not by much. This could of course be pushed further by weighing states lower or input higher, but this was not explored - if the input is regulated very close to the planned trajectory then the result is the same as the previous lab-exercise where u was exactly equal to the planned trajectory.

Prioritizing travel has a great effect on the travel response - resulting in almost perfect travel response. Unfortunately this also introduced oscillations in pitch-setpoint, pitch and pitchrate.

Prioritizing travelrate, pitch and pitchrate results in responses closer to the corresponding planned trajectory, but this is not further discussed as travel is the state most important to control - theese experiments were only to show that it would be possble should another state be valuable.

2.5.1 Possible improvements

The group observed that the state never reached the optimal trajectory, even with very high Q-values. The group believes that adding integration to the LQ-regulator would eliminate the stationary offset between travel and the planned travel trajectory.

Another possible improvement would be to change the cost function in the optimalization layer to generate a trajectory that is easier for the helicopter to follow. There is not guarantee that it is physically possible for the helicopter to follow the trajectory generated here, because the model of the system is not perfect.

2.5.2 Plots

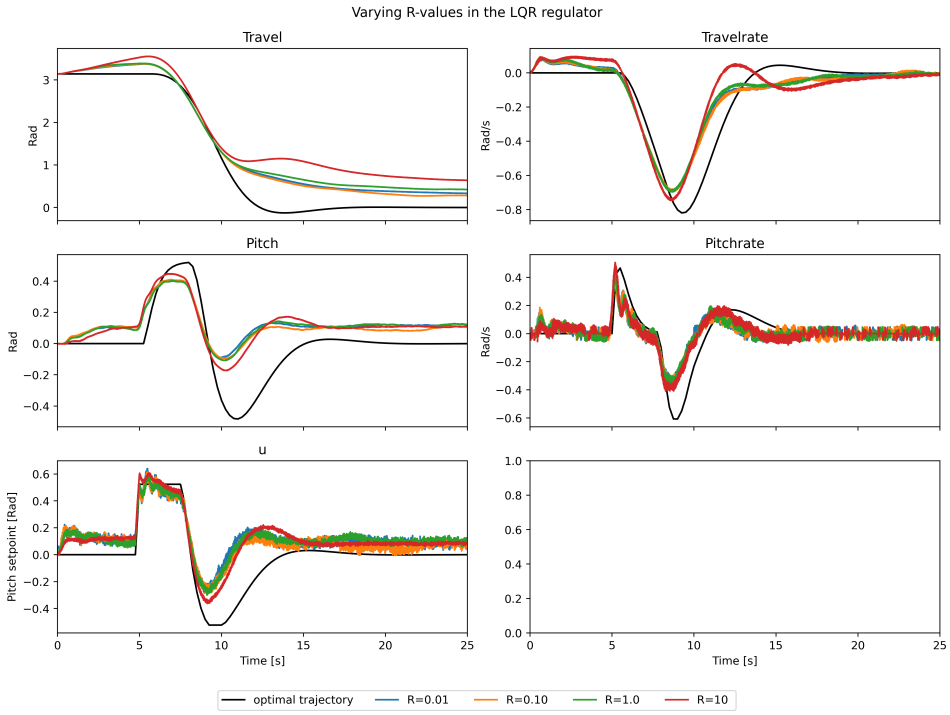


Figure 8: Prioritizing input-usage while keeping Q=diag([1,1,1,1]). This shows a very slight difference between weights. No further experimentation was done because if $u_k = u_k^*$ then that would be the same as the previous exercise.

2.5.3 Final tuning

The final tuning prioritizes travel, as that is the most important state, while comprimizing on the gain - too low results in a large offset while too high results in too much oscillations.

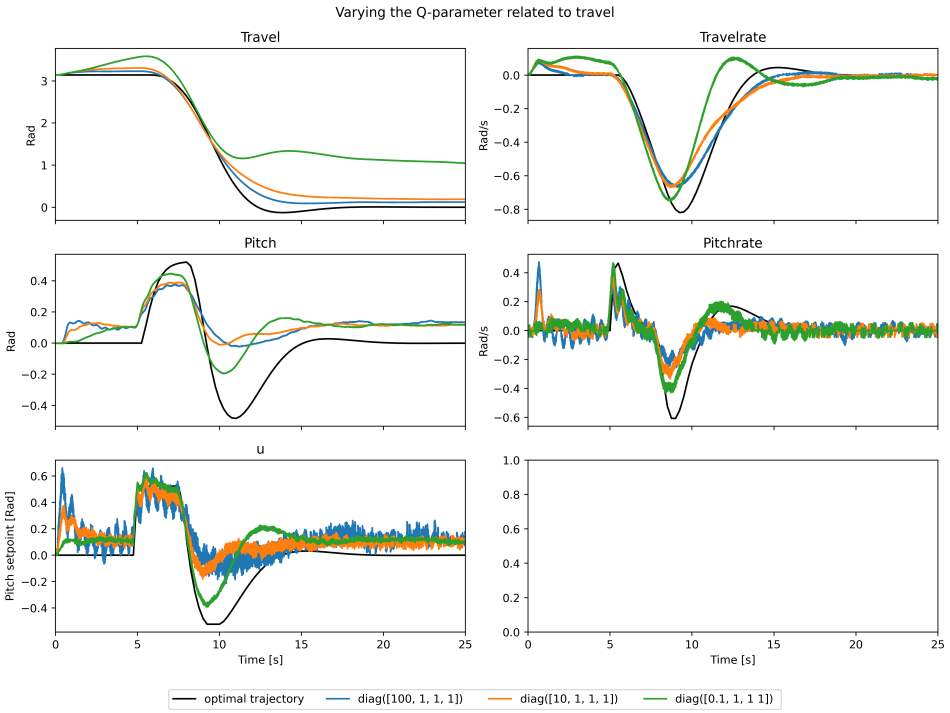


Figure 9: Prioritizing the weight related to travel while keeping $R = 1$. This was very effective in reducing offset between travel and the planned trajectory. Unfortunately at heigher gains there is some offset introducted and even then there is still a constant offset. A regulator with integral action would probably eliminate this offset without introducting oscillations.

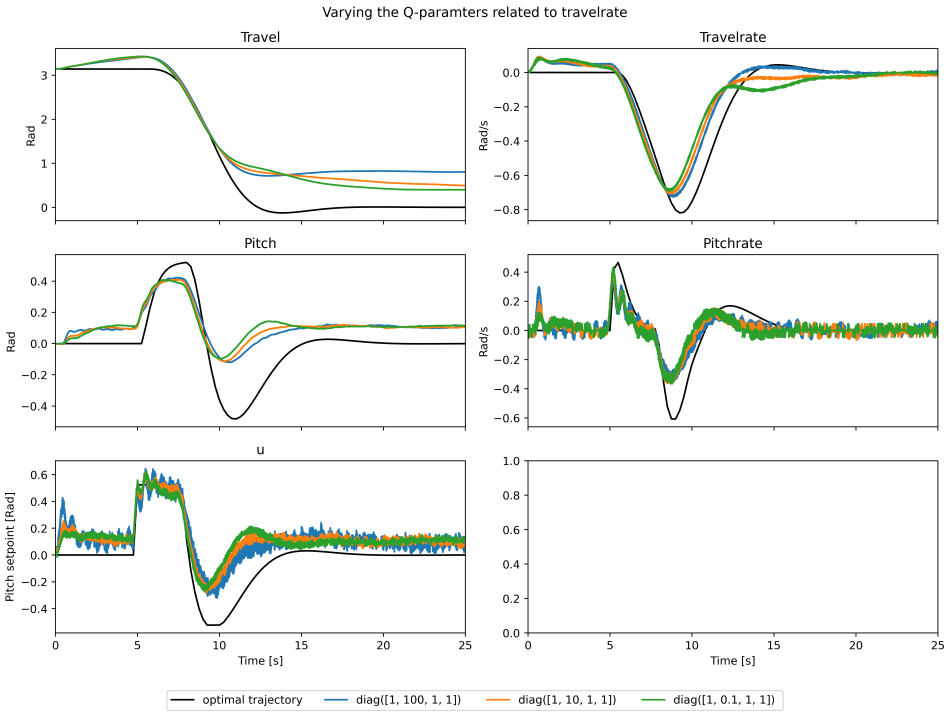


Figure 10: Changing the weight of the travelrate state, $R = 1$. This had only a minor effect with small variations in the response.

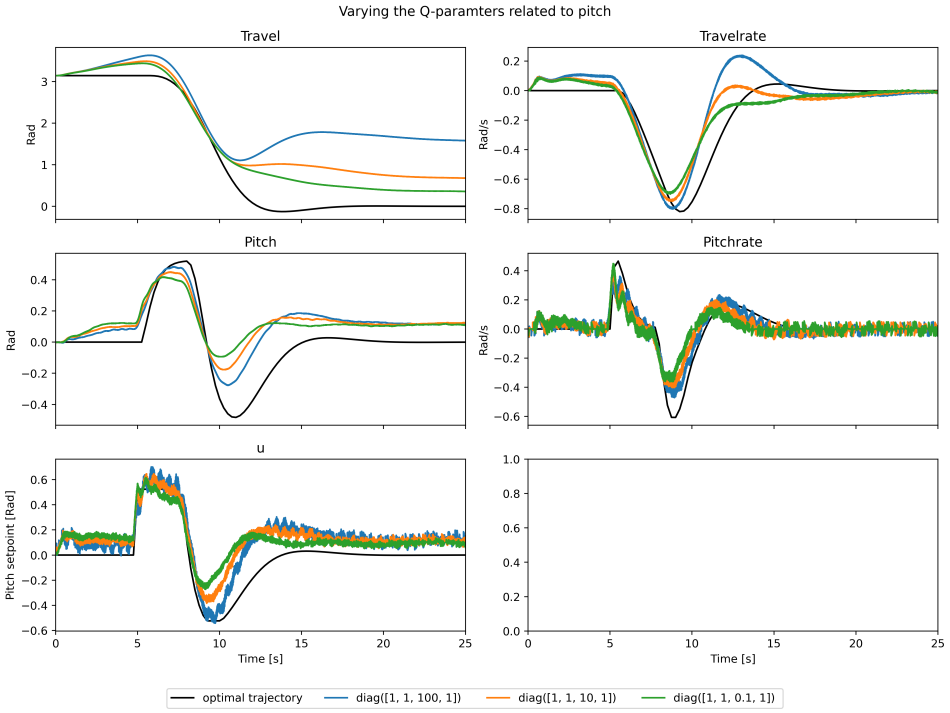


Figure 11: Changing the weight of the pitch state, $R = 1$. This had the effect of getting both pitch, pitch-setpoint and pitchrate closer to the planned trajectory at the expense of travel and travelrate.

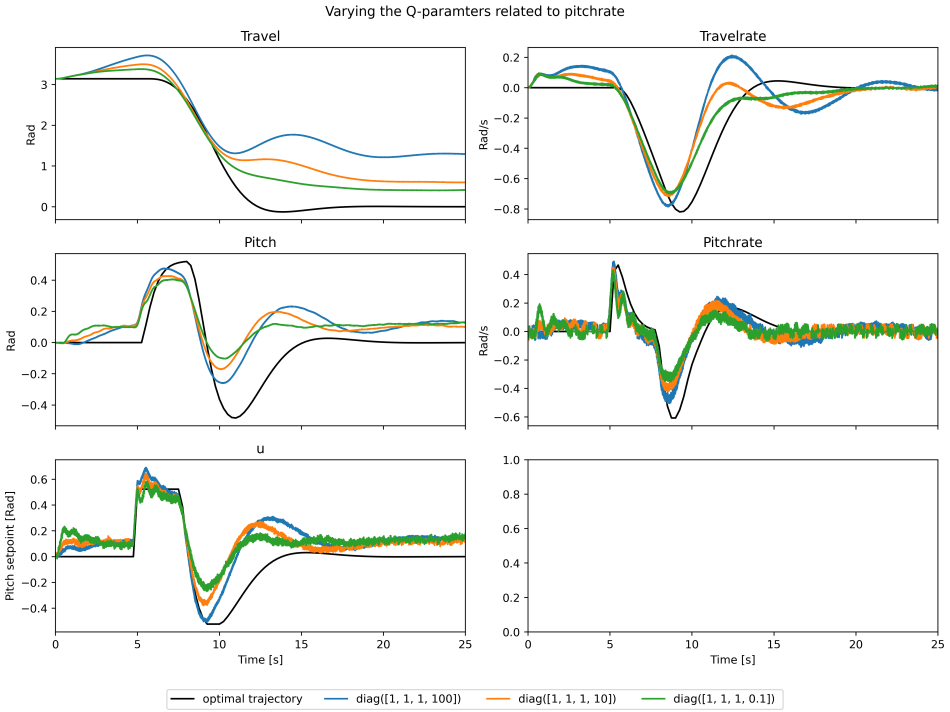


Figure 12: Changing the weight of the pitchrate state, $R = 1$. This was quite similar to fig. 11 but there is less oscillation.

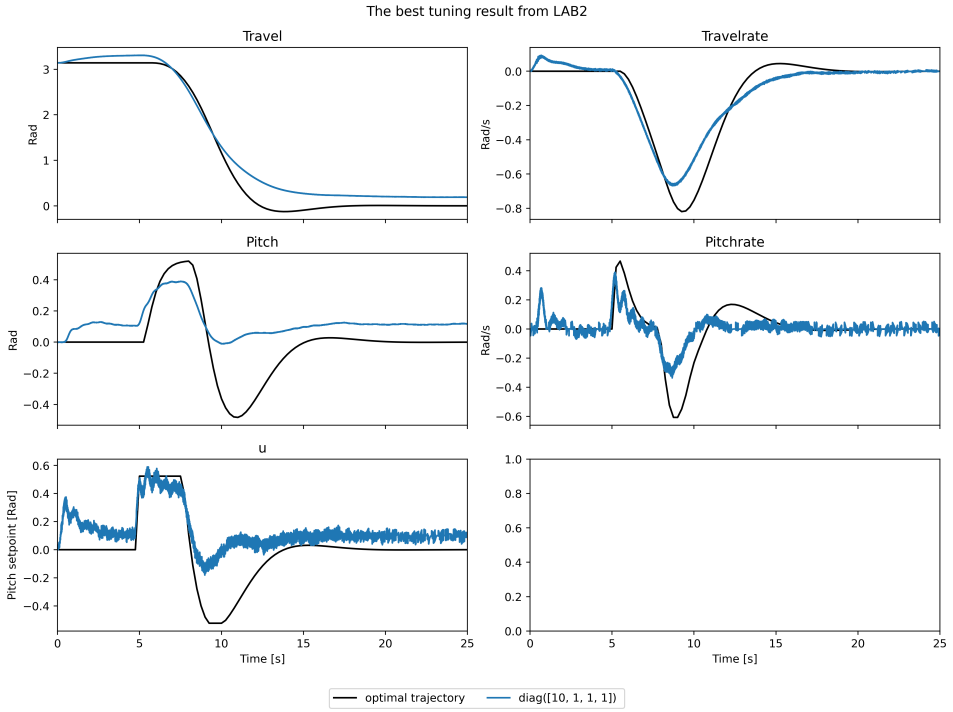


Figure 13: The best tuning of LAB3. This tuning prioritizes travel with a good comprimize between high gain and oscillations.

2.6 MATLAB and Simulink

Listing 4: MATLAB code for lab 3

```
clc
clear all
%% Initialization and model definition
init06; % Change this to the init file corresponding to your helicopter

% Continous time model
Ac = [0, 1,        0,        0;
      0, 0,     -K_2,        0;
      0, 0,        0,        1;
      0, 0, -K_1*K_pp, -K_1*K_pd];

Bc = [     0;
           0;
           0;
      K_1*K_pp];

% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25; % sampling time
A1 = eye(4) + (delta_t.*Ac);
B1 = (delta_t.*Bc);

% 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)

% Trajectory start and end values
lambda_0 = pi;
lambda_f = 0;
% Initial values
x1_0 = lambda_0;                          % Lambda
x2_0 = 0;                                 % r
x3_0 = 0;                                 % p
x4_0 = 0;                                 % p_dot
x0 = [x1_0 x2_0 x3_0 x4_0]';              % Initial values

% Time horizon and initialization
N  = 100;                                 % 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

% Bounds
ul          = -30*pi/180;                    % Lower bound on control
uu          = -ul;                           % 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);
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
P1 = 1;                                 % Weight on input
Q = gen_q(Q1,P1,N,M);                   % Generate Q, hint: gen_q
% The constant linear term is zero in our case
c = zeros(size(Q, 2), 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 = zeros(size(Aeq, 1), mu);              % Generate b
A0x0 = A1*x0;
beq(1:size(A0x0,1), :) = A0x0;

%% Solve QP problem with linear model
tic
% x = quadprog(H,f,A,b,Aeq,beq,lb,ub);
[z,lambda] = quadprog(2*Q, c,[],[], Aeq, beq, vlb, vub);% hint:
    quadprog. Type 'doc quadprog' for more info
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

%% Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)]; % Control input from solution

x1 = [x0(1);z(1:mx:N*mx)];              % State x1 from solution
x2 = [x0(2);z(2:mx:N*mx)];              % State x2 from solution
x3 = [x0(3);z(3:mx:N*mx)];              % State x3 from solution
x4 = [x0(4);z(4:mx:N*mx)];              % State x4 from solution

num_variables = 5/delta_t;
zero_padding = zeros(num_variables,1);
unit_padding  = ones(num_variables,1);

u   = [zero_padding; u; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
x2  = [zero_padding; x2; zero_padding];
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];
```

```
x = [x1 x2 x3 x4];

timesteps = 0:delta_t:delta_t*(length(u)-1);
u_opt = timeseries(u, timesteps);
x_opt = timeseries(x, timesteps);
%% Calculating feedback gain
% This is a simple setup for effective testing of different Q and R
    values.
% It builds and start the simulink model automatically
R_values = [0.01, 0.1, 1, 10];
Q_values = {
    diag([0.1, 1, 1 1]),
    diag([10, 1, 1, 1]),
    diag([100, 1, 1, 1]),
    diag([1, 0.1, 1, 1]),
    diag([1, 10, 1, 1]),
    diag([1, 100, 1, 1]),
    diag([1, 1, 0.1, 1]),
    diag([1, 1, 10, 1]),
    diag([1, 1, 100, 1]),
    diag([1, 1, 1, 0.1]),
    diag([1, 1, 1, 10]),
    diag([1, 1, 1, 100])
};

isR = false; %If true: Test different R values, else test Q values
if (isR == true)
    l = length(R_values)
else
    l = length(Q_values)
end

modelname = 'helicopter'
for i = 1:l
    if (isR == true)
        Q = diag([1, 1, 1, 1]);
        R = R_values(i);
    else
        Q = Q_values{i, 1};
        R = 1;
    end
    [K, S, e] = dlqr(A1, B1, Q, R);
    if (isR == true)
        filename = sprintf('data/data_R%.3f.mat', R);
    else
        filename = sprintf('data/Q_test_%d', i);
    end

    set_param(strcat(modelname,'/To File'), 'Filename', filename);

    set_param(gcs, 'SimulationCommand', 'connect');
    set_param(gcs', 'SimulationCommand', 'start');

    pause(40);
    set_param(gcs', 'SimulationCommand', 'stop');
    set_param(gcs, 'SimulationCommand', 'disconnect');

    input('Press ENTER when you are ready for new test: ')
```

```
161 end
```

2.6.1 Simulink

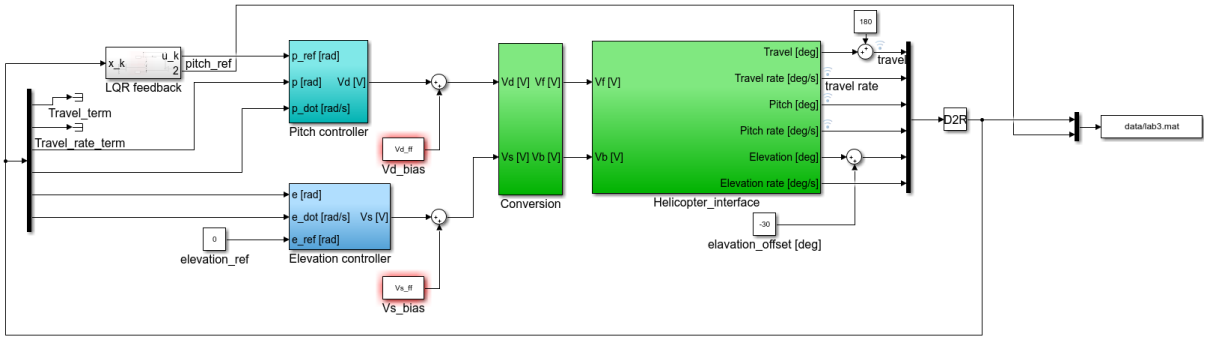


Figure 14: Simulink diagram used in lab 3.

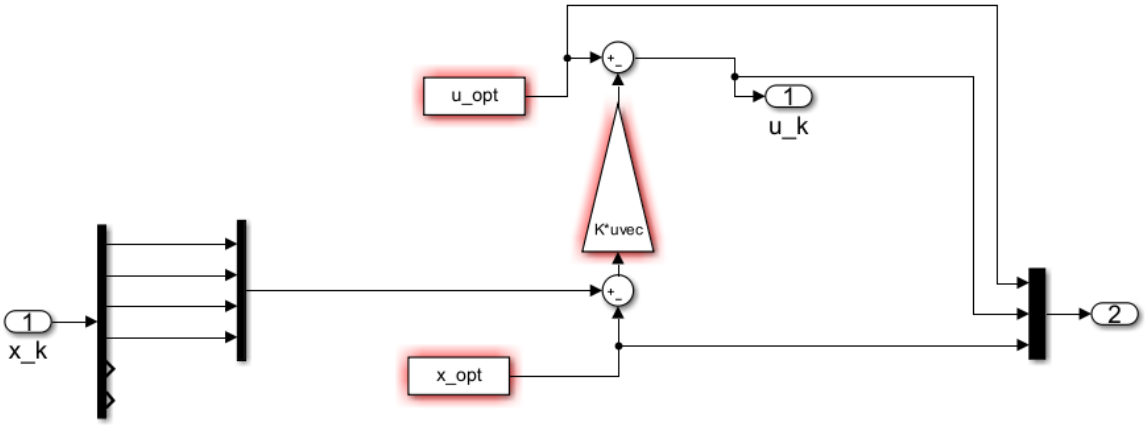


Figure 15: "LQR Feedback" subsystem in fig. 14

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

In the previous labs the elevation has been disregarded, and assumed to be zero. In this lab, elevation is not disregarded, and the group had to calculate an optimal trajectory for the elevation as well.

3.1 Modeling the helicopter with elevation

3.1.1 Continous model

The equation for elevation has been given in the problem description as

$$\ddot{e} + K_3 K_{ed}\dot{e} + K_3 K_{ep} e = K_3 K_{ep} e_c \tag{3.1}$$

where e_c is the elevation setpoint.

Expanding the system defined in eq. (1.3) to include eq. (3.1), gives a new system that includes the elevation:

$$\underbrace{\begin{bmatrix}\dot{\lambda}\\ \dot{r}\\ \dot{p}\\ \ddot{p}\\ \dot{e}\\ \ddot{e}\end{bmatrix}}_{\dot{\boldsymbol{x}}} = \underbrace{\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & -K_2 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed}\end{bmatrix}}_{\boldsymbol{A}_c} \underbrace{\begin{bmatrix}\lambda\\ r\\ p\\ \dot{p}\\ e\\ \dot{e}\end{bmatrix}}_{\boldsymbol{x}} + \underbrace{\begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ K_1K_{pp} & 0\\ 0 & 0\\ 0 & K_3K_{ep}\end{bmatrix}}_{\boldsymbol{B}_c} \underbrace{\begin{bmatrix}p_c\\ e_c\end{bmatrix}}_{\boldsymbol{u}} \tag{3.2}$$

3.1.2 The discretized model

Discretizing the continuous system defined in eq. (3.2) was done using the forward Euler method (see section 1.2 for more information about this method).

The resulting dicretized system became:

$$\boldsymbol{A}_d = \begin{bmatrix}1 & T & 0 & 0 & 0 & 0\\ 0 & 1 & -TK_2 & 0 & 0 & 0\\ 0 & 0 & 1 & T & 0 & 0\\ 0 & 0 & -TK_1K_{pp} & 1-TK_1K_{pd} & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & T\\ 0 & 0 & 0 & 0 & -TK_3K_{ep} & 1-TK_3K_{ed}\end{bmatrix}, \quad \boldsymbol{B}_d = \begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ TK_1K_{pp} & 0\\ 0 & 0\\ 0 & TK_3K_{ep}\end{bmatrix} \tag{3.3}$$

where T is the sample-time.

3.2 The new optimalization problem

The criteria to be minimized was

$$\phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q_1 p_{ci}^2 + q_2 e_{ci}^2 \tag{3.4}$$

and the constraint on the elevation was:

$$e_k \geq \alpha \exp\left(-\beta(\lambda_k - \lambda_t)^2\right) \forall k \in \{1, ..., N\} \tag{3.5}$$

3.3 LQ regulator

As the results from the previous lab shows adding feedback greatly improves performance, see section 1.6 - therefore it makes sense to have feedback for this exercise as well.

The LQ regulator was exactly the same as the one describe in section 2.3, but now with expanded state and input as described in eq. (3.2). This means that $\boldsymbol{Q}$ was now a diagonal matrix of size 6x6, and $\boldsymbol{R}$ was a diagonal matrix of 2x2.

3.4 Experimental results

The group's goal was as in the other labs, to make the helicopter follow the optimal trajectory to λ_f as close as possible. Making the helicopter follow the trajectory consisted of two parts:

1. Tune the LQ regulator to get a good feedback-gain matrix.
2. Find an optimal trajectory the helicopter could follow.

3.4.1 Tuning LQ regulator

Tuning the LQ regulator is a vital part of achieving as good response. Since the elevation is decoupled from the rest of the states, it should been possible to use the tuning from section 2.5 and find a good tuning for the elevation (i.e. tuning Q(5,5), Q(6,6) and R(2, 2)). However, in reality the helicopter's state is not decoupled (see section 3.5), so to achieve good tuning the group had to tune the whole system again.

A good rule of thumb is that the LQ regulator should be tuned with an input trajectory that one know the helicopter actually can follow. It is hard to tune a regulator with an input trajectory that is physically impossible to achieve. Therefore, the group tried to create a moderate input-trajectory which seemed reasonable based on their experience from the lab. However, the task was harder than anticipated, and the group did not have enough time to do this.

Since the group was not able to make a tuning-trajectory for the states and inputs, an optimal trajectory from solving the optimization problem using $q_1 = q_2 = 1$ in the cost function, was used as a replacement. This gave the optimal trajectories shown in fig. 16. The bump in the optimal trajectory for elevation comes from the constraint defined by eq. (3.5).

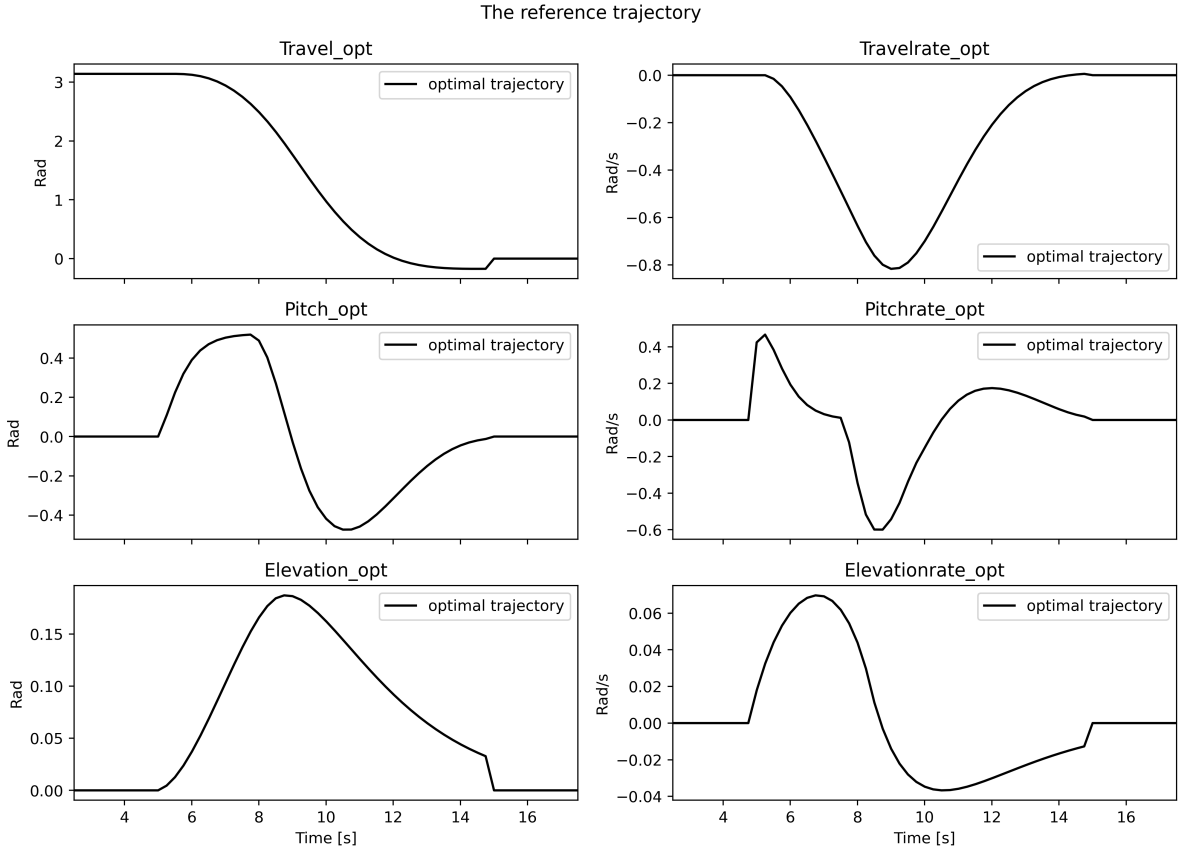


Figure 16: Optimal trajectories using $q_1 = q_2 = 1$ in the cost function.

In the tuning process we decided to keep $\boldsymbol{R}$ and $\boldsymbol{Q}$ as diagonal matrices. $\boldsymbol{R}$ was set constant equal the identity matrix, i.e. $\boldsymbol{R} = \boldsymbol{I}_2$. The diagonal elements of $\boldsymbol{Q}$ was then changed to get a good tuning. The tuning process with the new system, was based on the exact same principles as the tuning process

in section 2: Each diagonal entry of $\boldsymbol{Q}$ and $\boldsymbol{R}$ decides how much the corresponding response of the state/input shall be penalized. Larger values in an entry for $\boldsymbol{Q}$ gives faster response (or equivalently; uses more input). Too large values gives unwanted effects as oscillations. This is shown in fig. 17 where the entry corresponding to the travel response was changed, i.e. Q(1, 1). Low values at Q(1, 1) gives a slower response for travel, and higher values at Q(1, 1) gives faster response. Too high values gives oscillations in pitch and pitch rate. The oscillations are noticeable in the real world as a shaking helicopter, and is an unwanted effect.

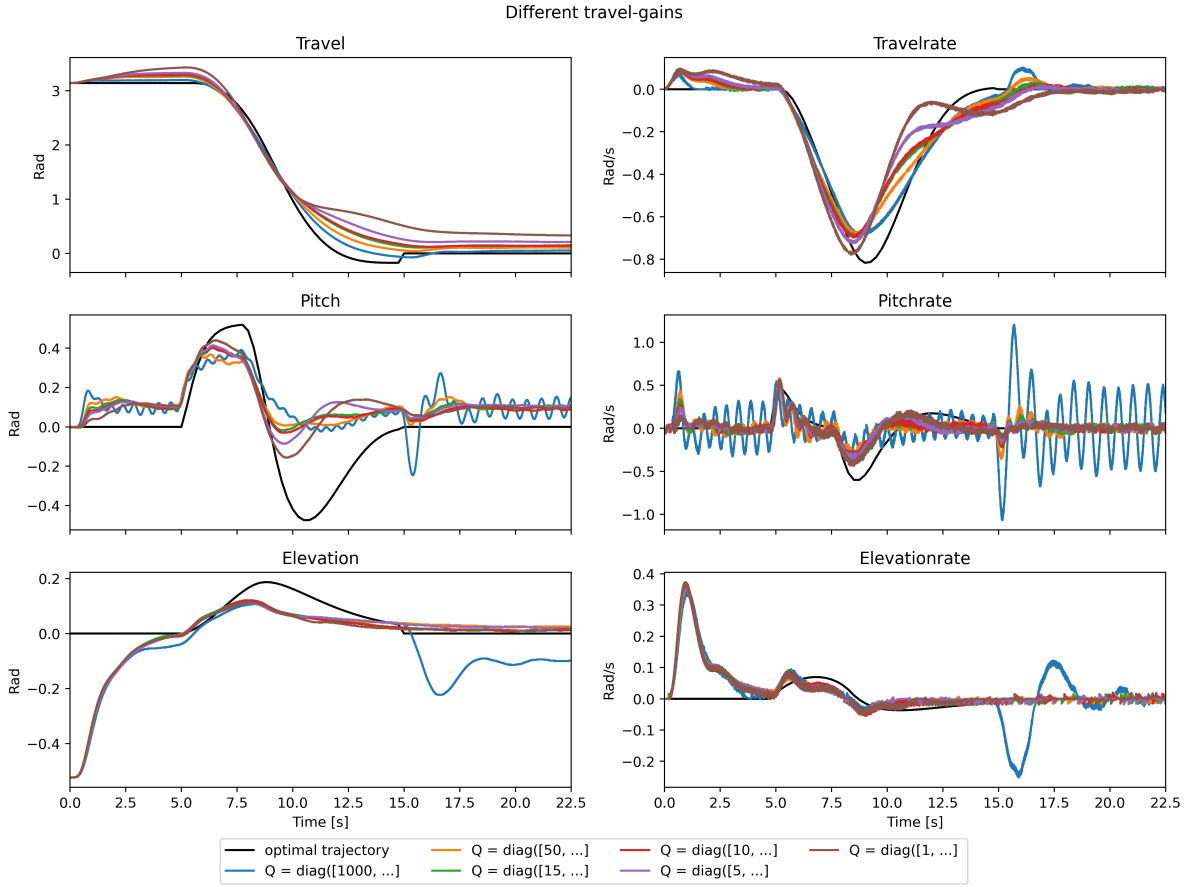


Figure 17: Plot showing the effect of varying the entry in $\boldsymbol{Q}$ related to travel. The rest of the $\boldsymbol{Q}$-values are 1.

As elevation was added to the model, the group also tested the effect of changing the weights for the elevation. This is shown in fig. 18. Prioritizing elevation by increasing Q(5, 5) gave, as expected, a better elevation-response.

Using the same approach as in section 2, the group tuned entry-by-entry in $\boldsymbol{Q}$. In the end the best tuning was achieved using:

$$\boldsymbol{Q} = \begin{bmatrix} 15 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.6}$$

This gave the results in fig. 19

3.4.2 Finding a reasonable trajectory

The group had two criterions on what defined a reasonable trajectory:

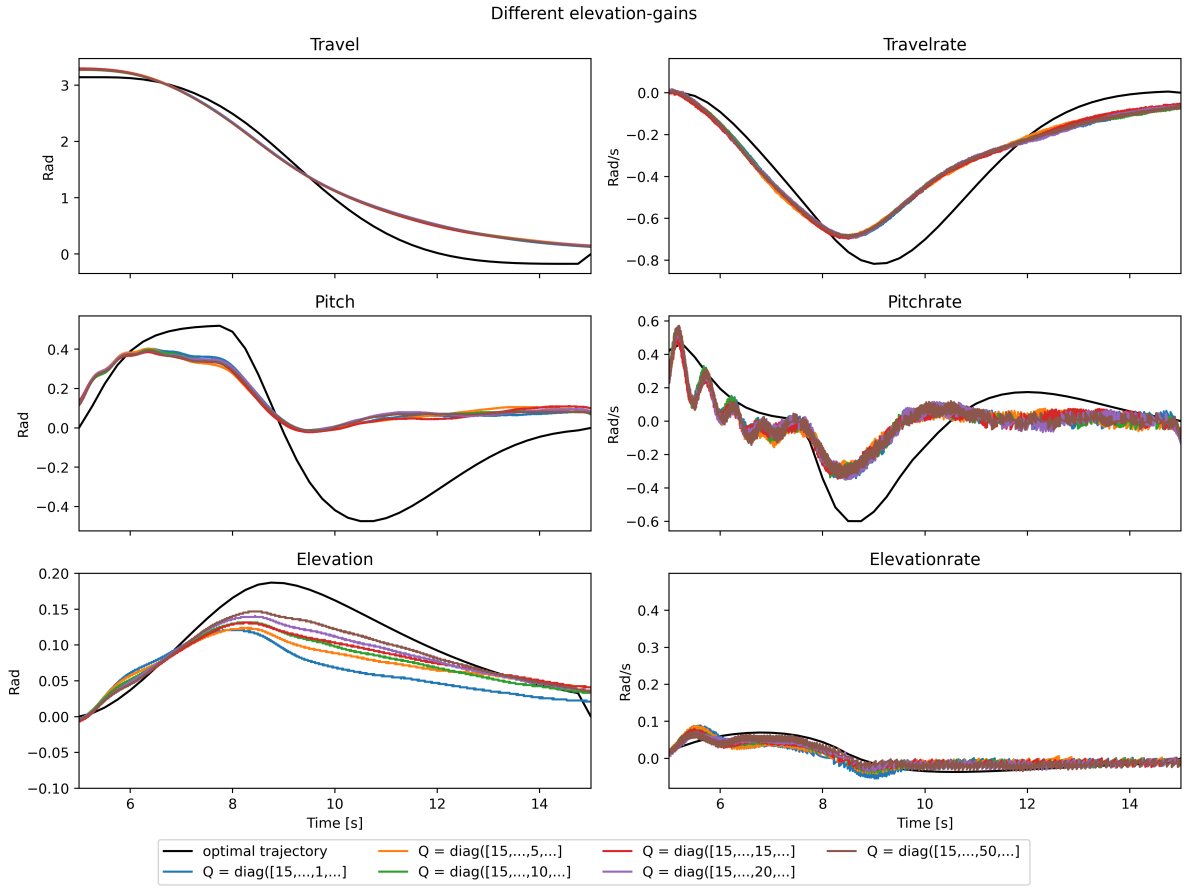


Figure 18: Plot showing the effect of varying the entry in $\boldsymbol{Q}$ related to elevation. The travel-gain in $\boldsymbol{Q}$ is 15, while the other values are 1.

1. The trajectory created should be physical possible (or at least close to possible) for the helicopter to achieve.
2. The time used to go from λ_0 to λ_f should be as small as possible.

The group tested different values for q_1 and q_2 in eq. (3.4). In the end, the result shown in fig. 19 fulfilled both these criterions well compared to other configurations, so the group decided that using $q_1 = q_2 = 1$ was a fair trajectory. Even though there are some error in the pitch and elevation response, the travel response becomes good which was the main focus during tuning.

3.5 Decoupled model

As states in the problem description, the two last states are completely decoupled from the first four states. Or put in other words, the dynamics describing elevation is completely independent on the pitch and travel. This is of course not the case in reality, and illustrates how simplicity of the model!

A simple force analysis will show that the states are not decoupled as described in the model. Assuming the helicopter is horizontal and the pitch is **not** zero, the force from the helicopter rotors will have two components. One horizontal and one vertical. This is shown in fig. 20. A change in pitch will change the vertical component of the rotor forces, and thereby also affecting the elevation. Also there is a lot of rotational forces that have not been considered in the model. E.g. the travel rate r will create a centripetal force going along the helicopter arm, and point inwards to the origin of the arm. If the helicopter arm is not completely horizontal, this centripetal force will have a component in the direction of the elevation. The main point is that our model is a very simplified version of the real physics of the helicopter. In reality the helicopter is a complex system, which we can never define 100 % exact with a mathematical formulation.

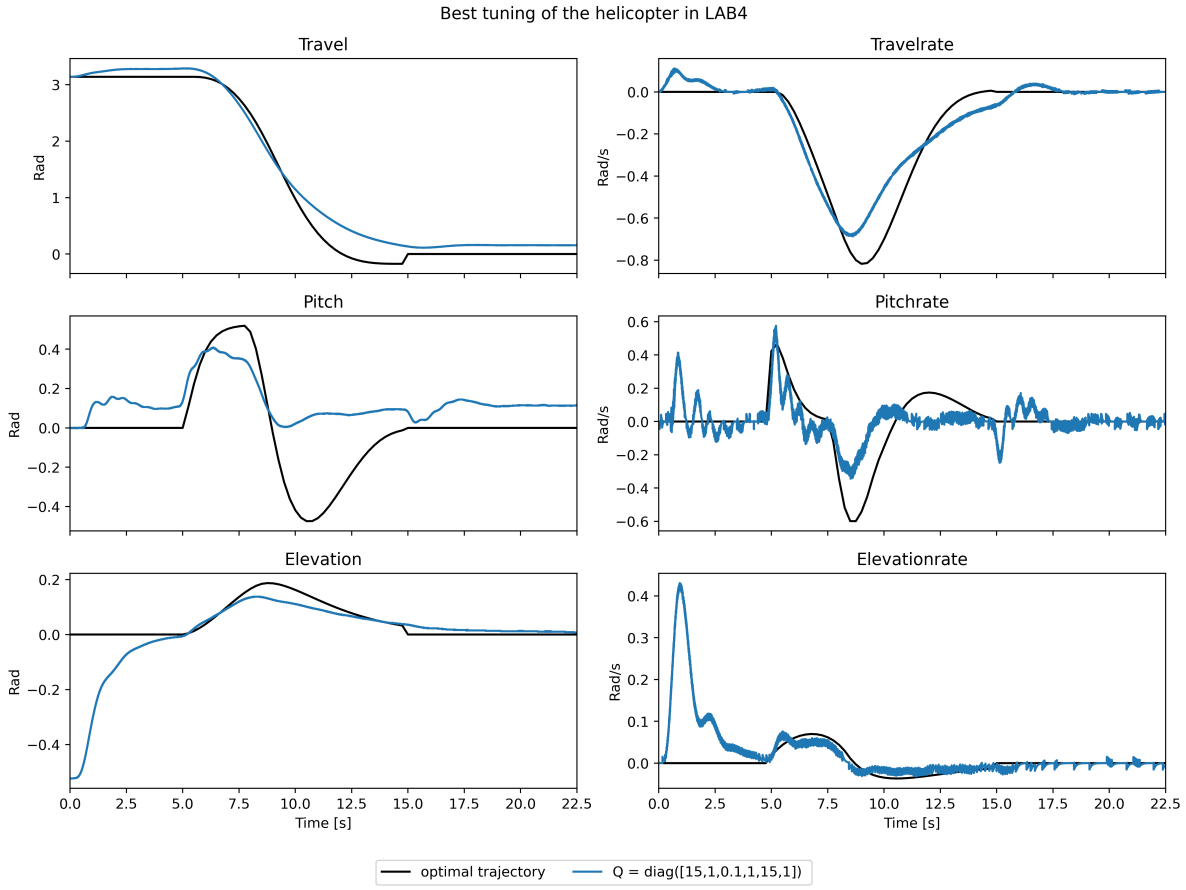


Figure 19: Best tuning of the LQ regulator in lab 4.

The effect this has on our optimal trajectory, is that our solver considers elevation as a system that is completely independent on the other states. This can of course be turned the other way around, as the optimal pitch- and travel-trajectory are also independent on the optimal elevation trajectory. The optimal trajectories are therefore "wrong", since the elevation is clearly not independent on the other states! It is wrong in the sense that the optimal trajectories may be impossible to achieve in the real world. The decoupled model will always model the elevation for a helicopter where pitch, travel and elevation are all zero(the linearization point). I.e. it will always model a helicopter where the rotor forces points directly upwards. Therefore, the further away the helicopter is from the linearization point, the larger the error due to the decoupled model becomes.

3.5.1 Possible solutions

There are no correct solution to this problem, as we will never be able to describe the real-world system exact with mathematical formulations. There are however, some "solutions" that will improve our system and reduce the errors that arises from our decoupled model.

The obvious solution is to couple the model by making it more realistic. The model is linear, but the real helicopter is far from linear. The further away it goes from the linearization point, the more nonlinear it becomes, and the more error-prone the model becomes. There are forces that we can include in our model to make it more accurate. These are for instance the centripetal force mentioned above, friction forces, forces caused by the ground effect, and so on. Of course we can only add simplified equations for these forces, but it would nevertheless increase the accuracy of our model! We will never be able to take into account all forces and nonlinearities, and many forces are actually impossible to model mathematically. Therefore we need to identify the forces that has the biggest impact on the error in our model. This is a hard task in itself and can be considered a downside of this approach. Another downside is that our model will quickly become complex and hard to understand. However, the group thinks that it would not be to much work to take into account the most obvious rotational forces, which could possibly cause

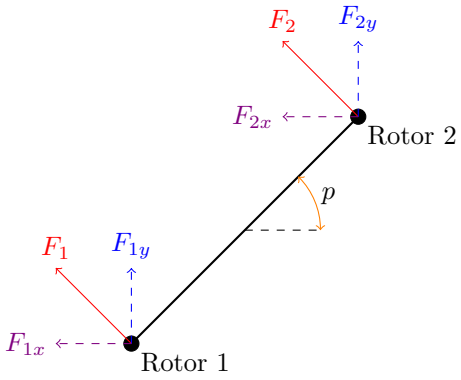


Figure 20: Force diagram of a horizontal helicopter with pitch not zero

a big improvement in the optimal solution.

Another solution is not to change the mathematical model of the helicopter at all, but rather change the regulator for the helicopter. Maybe adding integral-effect in the pitch- and elevation-controller would help decreasing the error? Or maybe other feedback-loop implementations could yield better results? There are many possible solutions to improve the regulation of the helicopter! Using this "solution", our optimal solution will still be wrong, but we can reduce the error by using a better regulator. The advantage of this is that we can spend less time making a more exact mathematical model, and rather focus on making a regulator that is more robust for errors. The disadvantage is of course that our optimal trajectories are still wrong.

3.6 MATLAB and Simulink

3.6.1 MATLAB

Listing 5: MATLAB code for lab 4

```
clc
clear all
%% Initialization and model definition
init06;

Ac = [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; ];

Bc = [     0,      0;
           0,      0;
           0,      0;
      K_1*K_pp,    0;
           0,      0;
           0,   K_3*K_ep];

% Discrete time system model.
delta_t = 0.25; % sampling time
Ad = eye(6) + (delta_t.*Ac);
Bd = (delta_t.*Bc);

%% Initial values and constraints
% Number of states and inputs
mx = size(Ad,2); % Number of states (number of columns in A)
```

```
mu = size(Bd,2); % Number of inputs(number of columns in B)

% Trajectory start and end values
lambda_0 = pi;
lambda_f = 0;
% Initial values
x1_0 = lambda_0;                          % 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+N*mu,1);                  % Initialize z for the whole
    horizon
z0 = z;
z0(1:6) = x0; % Initial value for optimization

% Bounds
ul          = [-30*pi/180; -inf];                   % Lower bound on
    control
uu          = -ul;                             % 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(1);                             % Lower bound on state x3
xu(3)   = uu(1);                             % 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

% We also have a nonlinear constraint as a function: nonlincon
Aeq = gen_aeq(Ad,Bd,N,mx,mu);             % Generate A, hint: gen_aeq
beq = zeros(size(Aeq, 1), 1);                % Generate b
A0x0 = Ad*x0;
beq(1:size(A0x0,1), :) = A0x0;
%% Cost function
q_1 = 1;
q_2 = 1;
Q1 = zeros(mx,mx);
Q1(1,1) = 1;                              % Weight on state x1
P1 = [q_1, 0;
      0,  q_2;];
Q = gen_q(Q1,P1,N,M);                     % Generate Q, hint: gen_q

cost_func = @(z) 0.5*z'*Q*z;
```

```
nonlcon = @nonlincon;

options = optimoptions('fmincon', 'Display', 'iter', 'Algorithm', 'sqp');
[z, fval] = fmincon(cost_func, z0, [], [], Aeq, beq, vlb, vub, nonlcon, options);

%% Extract control inputs and states
u1  = [z(N*mx+1:mu:N*mx+M*mu)]; % Control input from solution
u2  = [z(N*mx+2:mu:N*mx+M*mu)];

x1 = [z(1:mx:N*mx)];              % State x1 from solution
x2 = [z(2:mx:N*mx)];              % State x2 from solution
x3 = [z(3:mx:N*mx)];              % State x3 from solution
x4 = [z(4:mx:N*mx)];              % State x4 from solution
x5 = [z(5:mx:N*mx)];
x6 = [z(6:mx:N*mx)];

num_variables = 5/delta_t;
zero_padding = zeros(num_variables,1);
unit_padding  = ones(num_variables,1);

u1   = [zero_padding; u1; zero_padding];
u2   = [zero_padding; u2; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
x2  = [zero_padding; x2; zero_padding];
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];
x5  = [zero_padding; x5; zero_padding];
x6  = [zero_padding; x6; zero_padding];
x = [x1 x2 x3 x4 x5 x6];

timesteps = 0:delta_t:delta_t*(length(u1)-1);
u_opt = timeseries([u1'; u2'], timesteps);
x_opt = timeseries(x, timesteps);

%% LQR regulator
R_lqr = eye(2);
Q_lqr = diag([15, 1, 10, 1, 15, 1]);
[K, S, e] = dlqr(Ad, Bd, Q_lqr, R_lqr);

%% Plotting
figure();
plot(x)
legend('Travel', 'travelrate', 'pich', 'pitchrate', 'elevation', 'elevationrate');
```

3.6.2 Simulink

The Simulink diagram for lab 4 is almost exactly like the simulink diagram from lab 3, fig. 14. The only difference is that the "LQR Feedback" is changed to also output the elevation reference. This is shown in fig. 21. The outputted elevation reference is then used as input in "e_ref" in the "Elevation controller" subsystem in fig. 14.

3.7 Optional exercise

The group did not have enough time to implement this at the lab, but have discussed afterwards. If any new constraints should be added, the group would add a constraint on the maximum allowed elevation.

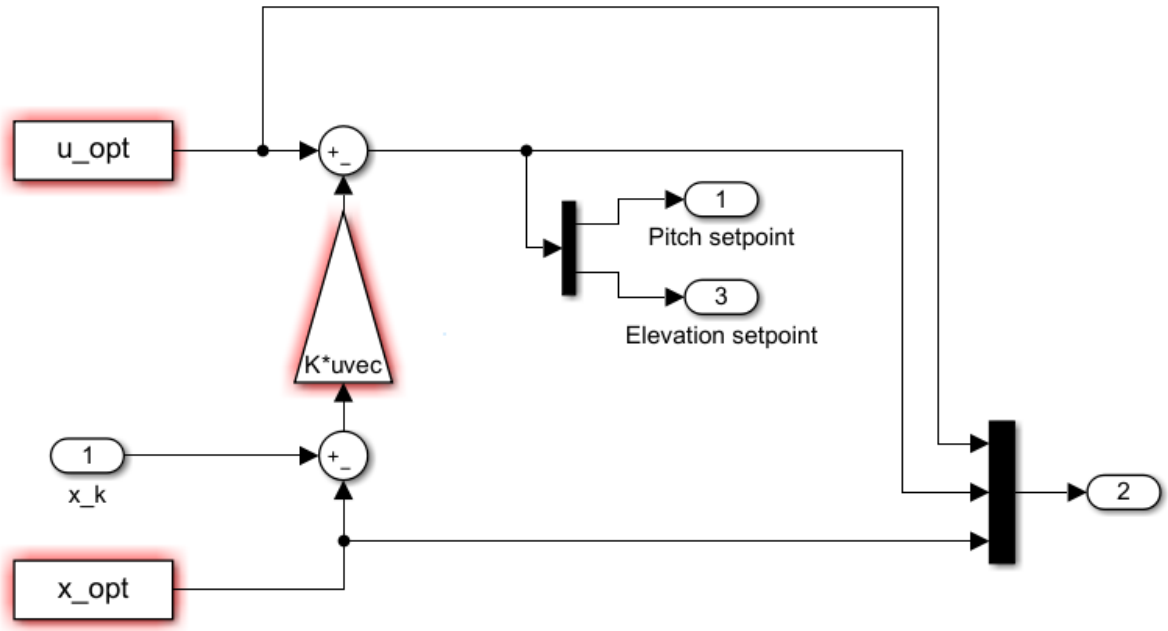


Figure 21: Expanded "LQR Feedback" system in lab 4.

There is no upper bound on the elevation in this task, and it would be reasonable to add this. There are two reasons for this:

1. The real helicopter has bounds on the elevation.
2. The helicopter becomes nonlinear far from the linearization point.

The first point is obvious: The optimal trajectories should not be able to be physical impossible for the helicopter to reach. This can probably be avoided by tuning α and β in the already given constraint in eq. (3.5). The second point is more interesting. When flying far from the linearization point, the helicopter becomes very nonlinear. At these configurations, our model does a bad job at modelling the system. Bounds on the elevation could therefore prevent the helicopter from flying "too far" from the linearization point. "Too far" in the sense that the model has large errors from the real response.

References

[1] NTNU. Ttk4135 optimization and control helicopter lab, 2016.

[2] Bjarne Foss and Tor Aksel N. Heirung. *Merging optimization and control*. 2016.