OptReg lab report

Group ??
Vemund Rogne
Jørgen Haaland

April 6, 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 . These following equations were given:

Add ref to page and paper

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

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

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

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

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.

Add ref to the figure in the problem description

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.

Should we add acronym list?

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.

1.1.2 Stability and eigenvalues

The properties of this system is dependent on physical constants $(l_a, J_t, ...)$ and control parameters (K_{pp}, K_{pd}).

Symbolic expressions in Matlab shows that the eigenvalues of A are:

$$\lambda = \pm\frac{1}{2}\left(\sqrt{-K_1(-K_1 K_{pd}^2 + 4K_{pp})} - K_1 K_{pd}\right) \tag{1.4}$$

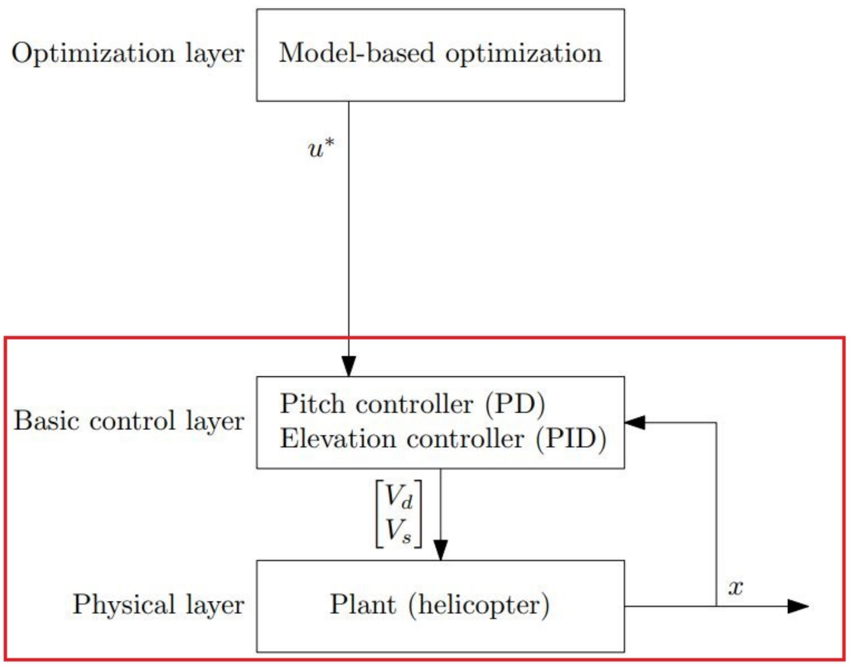


Figure 1: The red box encapsulate what is modelled in the state space model described by eq. (1.3).

The eigenvalues of the continous model, with $K_{pp} = 0,1, K_{pd} = 0,4$ are:

$$\begin{bmatrix} 0 \\ 0 \\ -0.26 + 0.24i \\ -0.26 - 0.24 \end{bmatrix} \tag{1.5}$$

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* [1]

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

Do we need to derive forward euler?

$$\boldsymbol{x}[k+1] = \boldsymbol{I}\boldsymbol{x}[k] + T\boldsymbol{A_c}\boldsymbol{x}[k] + T\boldsymbol{B}_c, \tag{1.6}$$

where T is the sample-time in the discrete model.

Add reference to linsys slides

Reformulating this, we can write:

$$\boldsymbol{x}_{k+1} = \underbrace{(\boldsymbol{I} + T\boldsymbol{A}_c)}_{\boldsymbol{A}_d}\boldsymbol{x}_k + \underbrace{T\boldsymbol{B}_c}_{\boldsymbol{B}_d}u_k \tag{1.7}$$

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

$$\boldsymbol{A}_d = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & -TK_2 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & -TK_1K_{pp} & 1 - TK_1K_{pd} \end{bmatrix}, \quad \boldsymbol{B}_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ TK_1K_{pp} \end{bmatrix} \tag{1.8}$$

1.2.1 Checking stability

The stability condition for eq. (1.7) is that all eigenvalues of $\boldsymbol{A}_d$ is less than one in absolute values, i.e.:

$$|\lambda_i| \leq 1, \quad \text{for } i = 1, 2, 3, 4 \tag{1.9}$$

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

Using the script in listing 1 the eigenvalues became:

$$\lambda_1 = 1 \tag{1.10a}$$
$$\lambda_2 = 1 \tag{1.10b}$$
$$\lambda_3 = 0.55 \tag{1.10c}$$
$$\lambda_4 = 0.55 \tag{1.10d}$$

All eigenvalues fulfills the requirement of eq. (1.9), 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 more specifically marginally stable. More information on this subject can be found .

Add cite to wikipedia

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

How is it formulated? As the problem description states, the cost function that must be minimized is described by:

$$\phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q p_{ci}^2, \quad q \geq 0 \tag{1.11}$$

We can solve this using the MATLAB function quadprog, but need to formulate the optimization problem as a QP problem in standard form to do so. This requires the open loop optimization problem to be on the form:

Add cite to quadprog docs?

$$\min_{x} \frac{1}{2} \boldsymbol{x}^t \boldsymbol{H} \boldsymbol{x} + \boldsymbol{f}^T \boldsymbol{x} \quad \text{such that} \begin{cases} \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{B}_{eq}, \\ \boldsymbol{lb} \leq \boldsymbol{x} \leq \boldsymbol{ub} \end{cases} \tag{1.12}$$

The next subsections describes this process.

1.3.1 Formulating the cost function

How to you formulate a difference? Im trying to understand that before I write this section

Equation (1.11) can be rewritten with $\boldsymbol{x}$ and u:

$$f(\boldsymbol{x}, u) = \sum_{i=0}^{N-1} (\boldsymbol{x}_{i+1} - \boldsymbol{x}_f)^T \boldsymbol{Q} (\boldsymbol{x}_{i+1} - \boldsymbol{x}_f) + q u^2 \tag{1.13}$$

, where

$$\boldsymbol{x}_f = \begin{bmatrix} \lambda_f \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{1.14}$$

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.15}$$

However, in this case, we have that $\lambda_f = 0$ and therefore $\boldsymbol{x}_f = \boldsymbol{0}$, and eq. (1.13) can be rewritten as an finite-horizon discrete LQR:

Add cite to wikipedia description

$$f(\boldsymbol{x}, u) = \sum_{i=0}^{N-1} \boldsymbol{x}_{i+1}^T \boldsymbol{Q} \boldsymbol{x}_{i+1} + ru^2 \tag{1.16}$$

, where

$$r = q \tag{1.17}$$

Combining all variables that should be optimized in one vector, we can get rid of the summation in eq. (1.16). This means all states and inputs for the N timesteps in one vector. This is described by:

Er timesteps et feil ord?

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_n & u_0 & u_1 & \dots & u_{n-1} \end{bmatrix}^T \tag{1.18}$$

Equation (1.16) can then be rewritten as:

$$f(\boldsymbol{z}) = \boldsymbol{z}^T \boldsymbol{H} \boldsymbol{z} \tag{1.19}$$

, where

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{Q} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \boldsymbol{Q} & \cdots & 0 & 0 & 0 & \cdots & 0 \\ & & \ddots & & & & & \\ 0 & 0 & \cdots & \boldsymbol{Q} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & r & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & r & \cdots & 0 \\ & & & & & & \ddots & \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & r \end{bmatrix} \tag{1.20}$$

Maybe fix this H. I'm not sure of the convention here

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. (1.7). Imposed constraints are constraints added by the designer of the system.

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

$$|p_k| \leq \frac{30}{180}\pi, k \in \{1, ..., N\} \tag{1.21}$$

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:

$$\boldsymbol{x}^{low} = \begin{bmatrix} -\infty \\ -\infty \\ -\frac{30}{180}\pi \\ -\infty \end{bmatrix} \leq \boldsymbol{x} \leq \boldsymbol{x}^{high} = \begin{bmatrix} \infty \\ \infty \\ \frac{30}{180}\pi \\ \infty \end{bmatrix} \tag{1.22a}$$

$$u^{low} = -\frac{30}{180}\pi \leq u \leq u^{high} = -\frac{30}{180}\pi \tag{1.22b}$$

We can now easily expand this to define lower- and upper-bounds for the vector $\boldsymbol{z}$ given in eq. (1.18)

$$\boldsymbol{z}^{low} = \begin{bmatrix} \boldsymbol{x}^{low} \\ \vdots \\ \boldsymbol{x}^{low} \\ u^{low} \\ \vdots \\ u^{low} \end{bmatrix} \leq \boldsymbol{z} \leq \boldsymbol{z}^{high} = \begin{bmatrix} \boldsymbol{x}^{high} \\ \vdots \\ \boldsymbol{x}^{high} \\ u^{high} \\ \vdots \\ u^{high} \end{bmatrix} \tag{1.23}$$

The system constraints can be defined as:

$$\boldsymbol{A}_{eq}\boldsymbol{z} = \boldsymbol{b}_{eq} \tag{1.24}$$

, where

$$\boldsymbol{A}_{eq} = \begin{bmatrix} \boldsymbol{I} & 0 & \cdots & \cdots & 0 & -\boldsymbol{B}_d & 0 & \cdots & \cdots & 0 \\ -\boldsymbol{A}_d & \boldsymbol{I} & \ddots & & \vdots & 0 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 & \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\boldsymbol{A}_d & \boldsymbol{I} & 0 & \cdots & \cdots & 0 & -\boldsymbol{B}_d \end{bmatrix}, \; \boldsymbol{b}_{eq} = \begin{bmatrix} \boldsymbol{A}_d\boldsymbol{x}_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.25}$$

Performing the multiplication (and abusing notation) shows that:

$$\boldsymbol{A}_{eq}\boldsymbol{z} = \boldsymbol{b}_{eq} \implies \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{B}_d u_0 = \boldsymbol{A}_d\boldsymbol{x}_0 \\ -\boldsymbol{A}_d\boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{B}_d u_1 = 0 \\ \vdots \\ -\boldsymbol{A}_d\boldsymbol{x}_{n-1} + \boldsymbol{x}_n - \boldsymbol{B}_d u_{n-1} = 0 \end{bmatrix} \implies \begin{bmatrix} \boldsymbol{x}_1 = \boldsymbol{A}_d\boldsymbol{x}_0 + \boldsymbol{B}_d u_0 \\ \boldsymbol{x}_2 = \boldsymbol{A}_d\boldsymbol{x}_1 + \boldsymbol{B}_d u_1 \\ \vdots \\ \boldsymbol{x}_n = \boldsymbol{A}_d\boldsymbol{x}_{n-1} + \boldsymbol{B}_d u_{n-1} \end{bmatrix} \tag{1.26}$$

Equation (1.26) describes the system in terms of eq. (1.7) for all timesteps, and thereby also describes all system constraints for the system.

Again: timesteps illegal?

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:

$$f(\boldsymbol{z}) = \boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} \quad \text{s.t.} \quad \begin{cases} \boldsymbol{A}_{eq}\boldsymbol{z} = \boldsymbol{B}_{eq} \\ \boldsymbol{z}^{low} \leq \boldsymbol{z} \leq \boldsymbol{z}^{high} \end{cases} \tag{1.27}$$

, where $\boldsymbol{z}$ is given by eq. (1.18), $\boldsymbol{H}$ is given by eq. (1.20), $\boldsymbol{A}_{eq}$ and $\boldsymbol{B}_{eq}$ is given by eq. (1.25), and $\boldsymbol{z}_{low}$ and $\boldsymbol{z}_{high}$ is given by eq. (1.23).

It is now easy to compare this to the QP formulation given in eq. (1.12), 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

Try using the values 0.1, 1 and 10 as weights q. Plot the manipulated variable and the output. Comment the results with respect to the different weights chosen.

$\boldsymbol{Q}$ and r in eq. (1.16) 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 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 $\pm\frac{30}{180}\pi$.

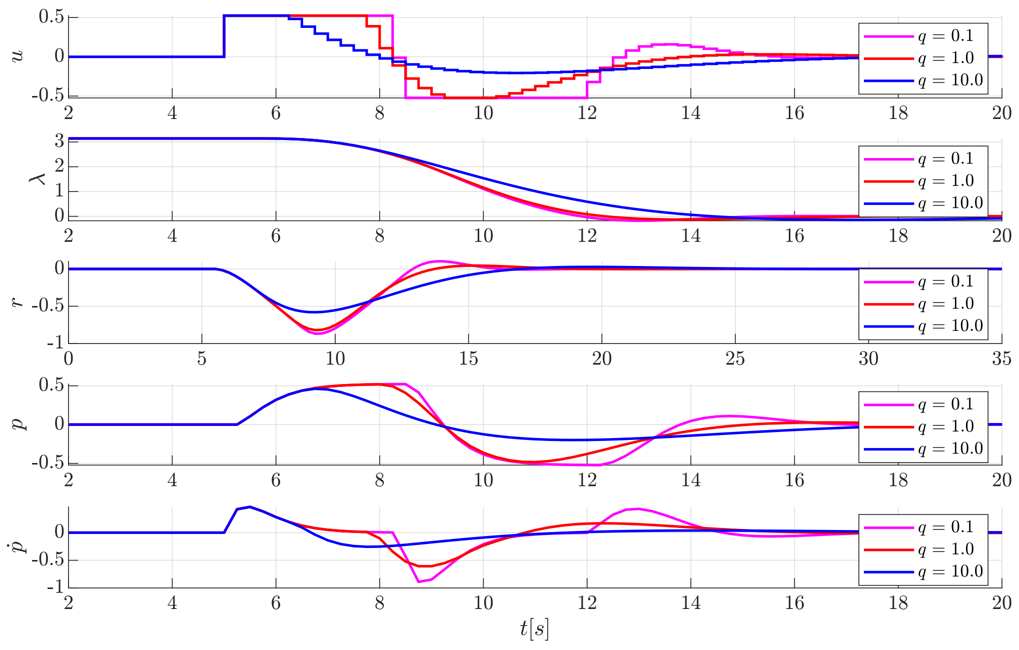


Figure 2: Optimal trajectories for the manipulated variable and outputs with different values of q.

1.5 The objective function

Furthermore, discuss the objective function (15) (in the lab assignment text) in particular the term $(\lambda_i - \lambda_f)^2$. For instance, could any unwanted effects arise from steering the helicopter to $\lambda = \lambda_f$ with this objective function?

The term $(\lambda_i - \lambda_f)^2$ in eq. (1.11) 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. (1.16); 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. 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.

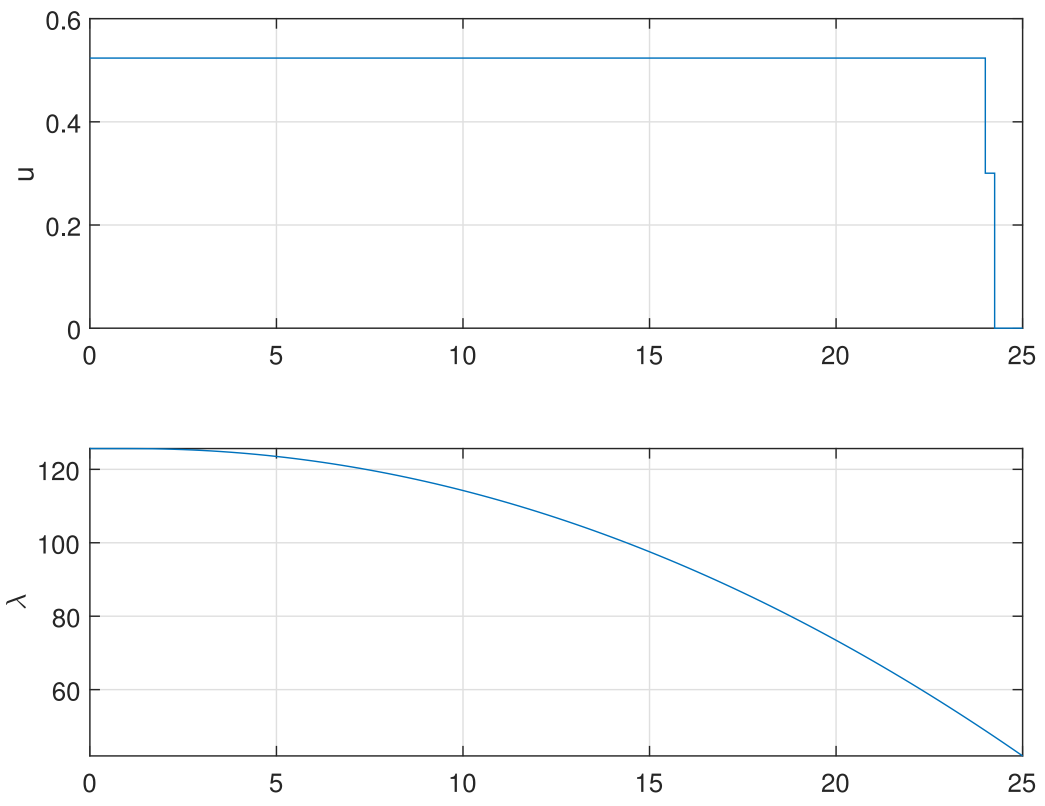


Figure 3: Optimal trajectories using $\lambda_0 = 40\pi$.

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

$$\lambda_{0,\text{modified}} = \begin{cases} +\text{mod}(\lambda_0 - \lambda_f, 2\pi), & \text{if} \quad \text{mod}(\lambda_0 - \lambda_f, 2\pi) \leq \pi \\ -(2\pi - \text{mod}(\lambda_0 - \lambda_f, 2\pi)) & \text{else} \end{cases} \tag{1.28a}$$

(1.28b)

, 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.

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
```

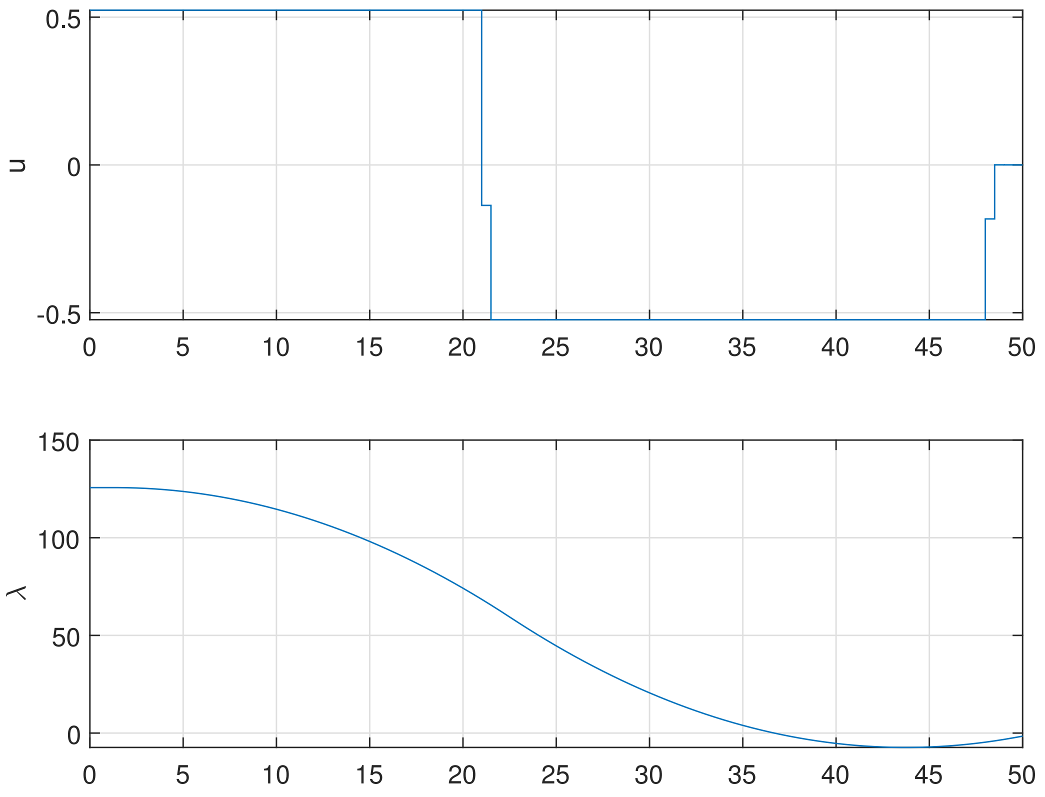


Figure 4: Optimal trajectories using $\lambda_0 = 40\pi$ and $\Delta t = 0.5$.

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.

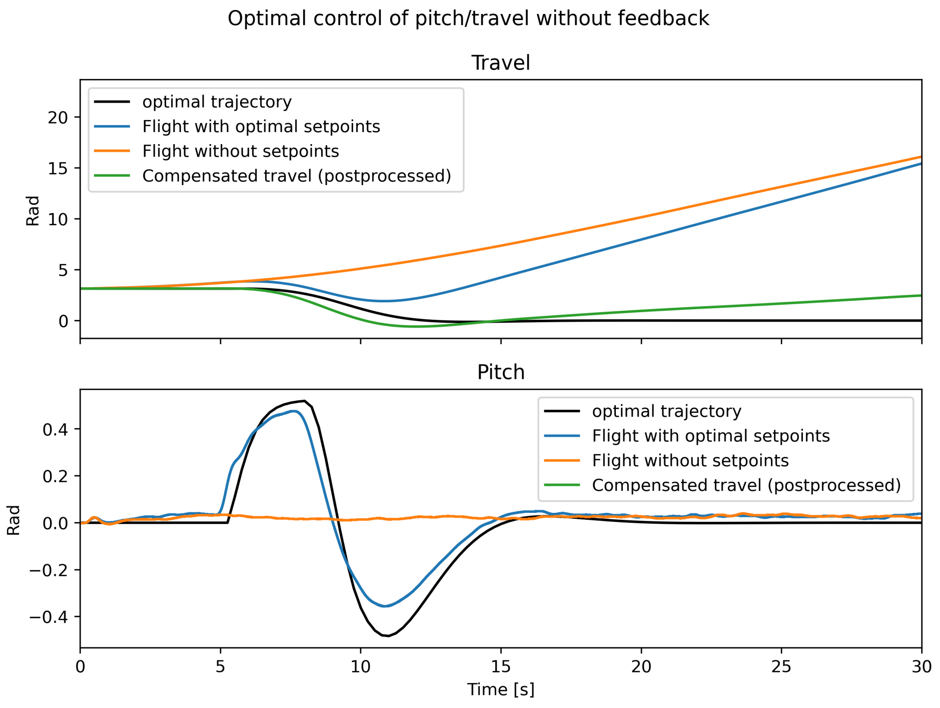


Figure 5: Results of LAB2

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

The **encoder** does not measure pitch at the helicopter in relation to gravity, rather it measures the angle between the helicopter blades and the platform that holds them. Because that platform is not build to be precisely the same as gravity (and because of wear-and-tear) the measured value 0 is not exactly 0. This causes the helicopter to have a pitch offset, which in turn generates travelrate and travel.

The **imbalance** may be that one rotor is stronger than the other, or that the rotors have offsets from the vertical position. Theese imbalances may generate a sidewards force, even when the helicopter is pitced 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 theese 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: there are as explained in section 1.6.2 many causes of drift and without any feedback the helicopter will quicly deviate from the desired response.

Nevertheless it is possible to verify the control sequence by compensating for the drift. The compensation is done by taking the travel of the flight with optimal setpoints and subtracing the drift logged in the flight without setpoints. 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: MATALB code for lab 2

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

%% Initialization and model definition
init05; % 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(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

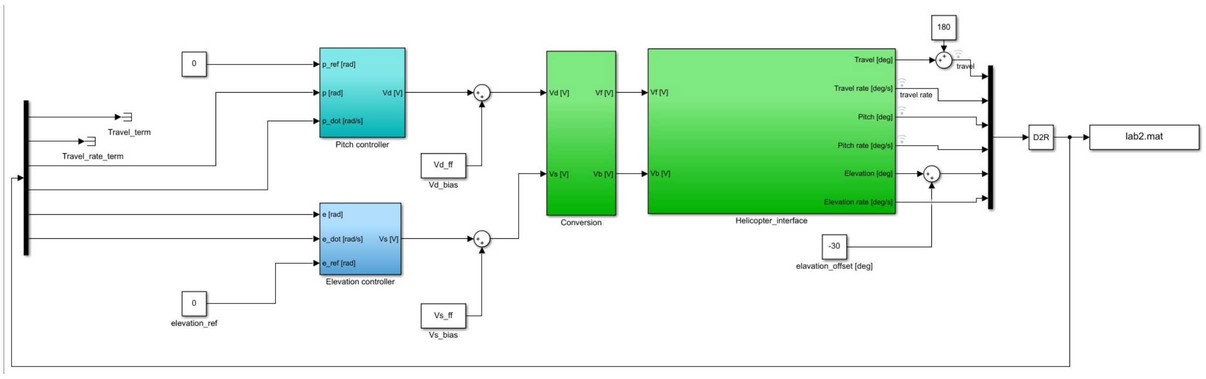


Figure 6: Simulink diagram used in lab 2.

Need a much better image of this.

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. Feedback is clearly needed as the experimental results in section 1.6 show that the helicopter response quickly deviates from the optimal trajectory. The results of this laboratory exercise shows that adding feedback greatly improves the perfomance of the helicopter.

2.1 Introducing feedback

There are many ways of adding feedback to a system. In this case the assignment states that we are to add feedback between the calculated optimal trajectory and the actual trajectory. In other words the feedback introduced here modifies the input u to get closer to the optimal trajectory x^*.

The pitch-control input is now governed by the 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.

This feedback equation modifies the input u_k to the pitch-controller if the state x_k deviates from the optimal value x_k^*. The feedback gain K is calculated using an LQ controller.

2.2 LQ controller

As explained in the problem description, an LQ (or linear-quadratic) controller, solves the quadratic objective function given by cref

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

for a linear model

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

without including inequality constraints. Here $\Delta x = x - x^*$ and $\Delta u = u - u^*$ are deviations from the optimal trajectory.

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 punished. This allows the designer to optimize the regulator to the specific implementation: Is it more important to have low input-deviation than state-deviation?

In this system there is only one constraint on the input; that it shall not exceed 30 degrees. Therefore a small R-value compared to Q is warranted. This will produce a regulator that tries harder to minimize state-deviation than input-deviation.

2.3 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.

2.3.1 Modified Control Hierarchy with MPC

Rather than introducing the Advanced Control Layer with the LQ-controller; introducing MPC would introduce the feedback to the Optimization Layer instead. The optimization layer would use the current state value and generate an optimal trajectory to the target, outputting pitch-setpoints to the Basic control layer at every timestep.

2.4 Experimental results

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. This is of course what was expected.

The group performed experiments with different values of Q and R to analyze the performance of the helicopter with different tunings.

Figure 7 shows that changing the R-value did not drastically change the response, nevertheless it was observed that a larger R-value produces greater state-offset and smaller input-offset.

Figure 8 shows that increasing the Q-value greatly reduces the offset from the optimal and actual travel response. It was also observed that this caused oscillations in the input-values.

The experimental results are consistent with the theory of the LQ controller laid out in section 2.2.

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- and reduce the dynamic offset. This was however not explored further.

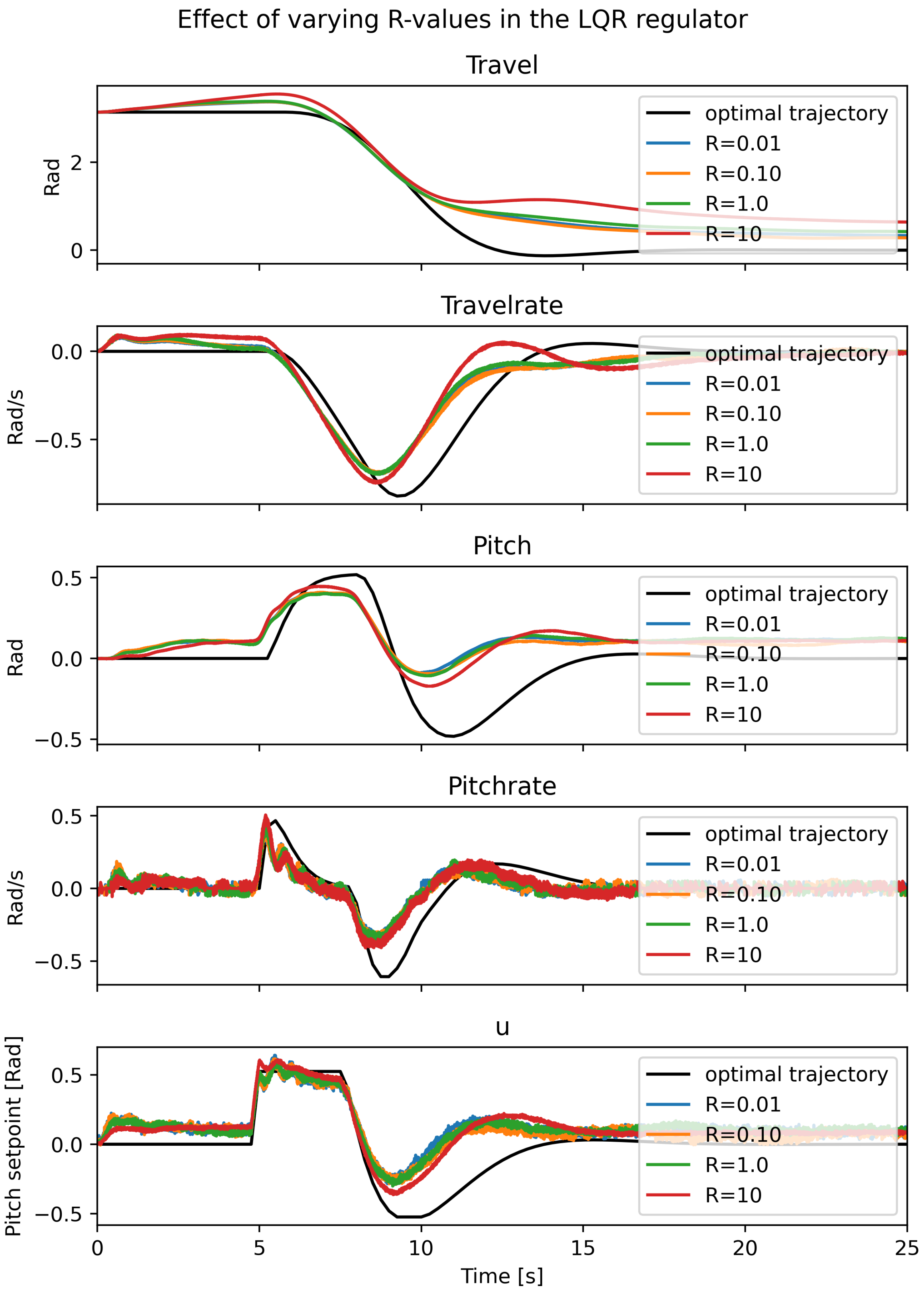


Figure 7: Results of varying R-values wihle keeping Q=diag([1,1,1,1])

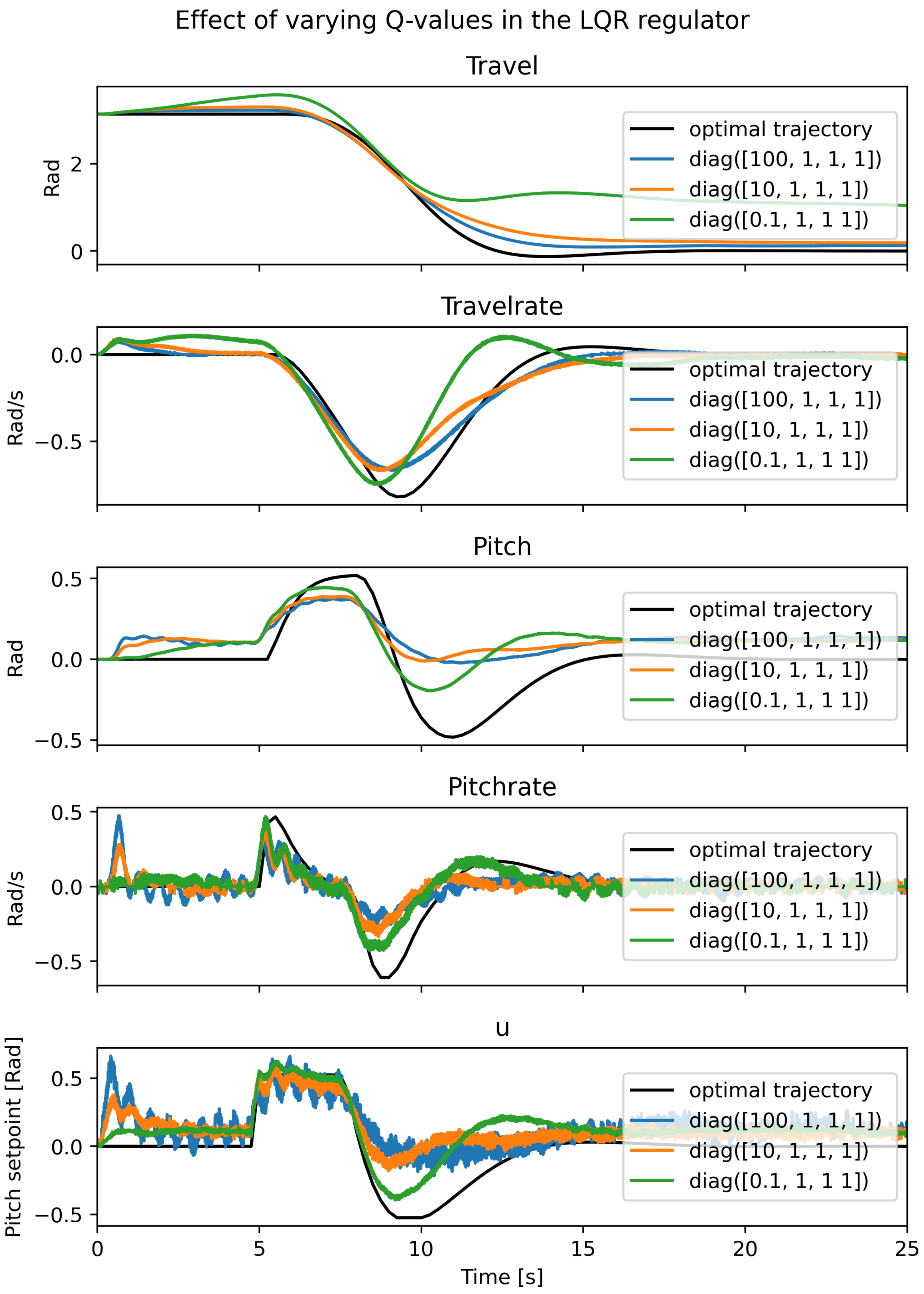


Figure 8: Results of varying Q-values while keeping $R = 1$

2.5 MATLAB and Simulink

Code and diagrams go here

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

We have to add the equation for elevation

$$\ddot{e} + K_3K_{ed}\dot{e} + K_3K_{ep}e = K_3K_{ep}e_c$$

to the system defined in eq. (1.3)

$$\underbrace{\begin{bmatrix}\dot{\lambda}\\ \dot{r}\\ \dot{p}\\ \ddot{p}\\ \dot{e}\\ \ddot{e}\end{bmatrix}}_{\dot{\boldsymbol{x}}} = \underbrace{\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & -K_2 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed}\end{bmatrix}}_{\boldsymbol{A}_c}\begin{bmatrix}\lambda\\ r\\ p\\ \dot{p}\\ e\\ \dot{e}\end{bmatrix} + \underbrace{\begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ K_1K_{pp} & 0\\ 0 & 0\\ 0 & K_3K_{ep}\end{bmatrix}}_{\boldsymbol{B}_c}\underbrace{\begin{bmatrix}p_c\\ e_c\end{bmatrix}}_{u} \tag{3.1}$$

3.1 The continuous model

Answer 10.4.1.1

3.2 The discretized model

Answer 10.4.1.2

3.3 Experimental results

Printouts of data from relevant experiments (plots). Discussion and analysis of the results. Answer 10.4.2.6 here. 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.

3.3.1 Tuning LQ regulator

Before the group started testing the optimal trajectory found using the SQP-algorithm, the LQ regulator used to find the feedback-gain matrix $\boldsymbol{K}$ had to be tuned. Since the feedback-loop was generated using both the optimal trajectory for both the state and the input, we used the optimal trajectories in as our reference in the tuning process. However, there is a disadvantage doing this; we don't quite know if the helicopter is even possible to follow these trajectories. Therefore, an optimal procedure would been to generate a trajectory the helicopter had physical possibilities to follow, and use this for tuning. In this way we could've decoupled the tasks of tuning the helicopter and finding an optimal trajectory. However, the group did not have enough time to produce such a tuning process. It could also be argued that this is not necessary, since we have already implemented constraint based on the physical limitations of the helicopter in the optimal trajectory (AND IN THE LQR??). "Men det er god skikk å tune på et annet referanse signal slik helikopteret blir tuned mer generelt"

Since the group was not able to make a tuning-trajectory for the states and inputs, we used the optimal trajectory with $q_1 = q_2 = 1$. This gave the optimal trajectories shown in **??**

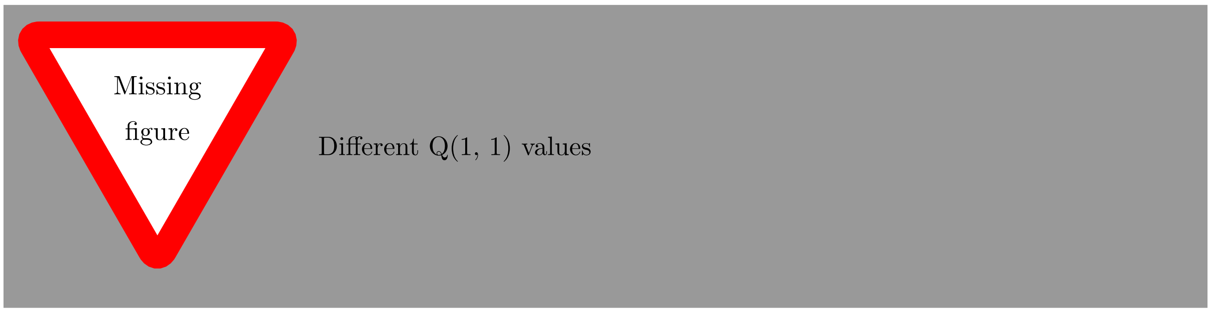
add cref to cost func

In the tuning process we decided to keep R and Q as diagonal matrices. R was set constant equal the identity matrix, i.e. $R = I_2$. The diagonal elements of Q was then changed to get a good-tuned system. Each diagonal entry in Q corresponds to the corresponding stat in eq. (3.1) , so the tuning process was done by tuning state-by-state.

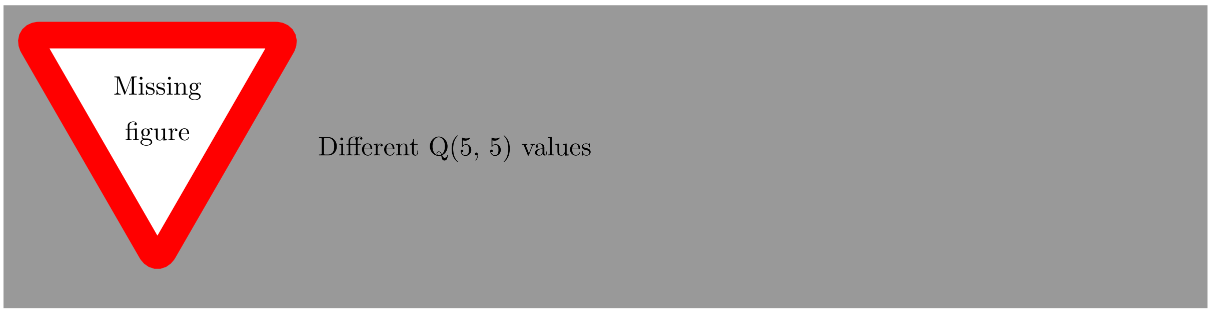
REFORMUL

Since we are interested in following the optimal trajectory for the travel, we started by tuning this state, i.e. changing Q(1, 1) . The figure below shows the helicopter states to the optimal trajectory for different Q(1, 1) values. As expected, a higher value of Q made the system use more fuel to follow the travel. Using too large value caused the helicopter's pitch to oscillate quite much, and too low resulted in a bad response.

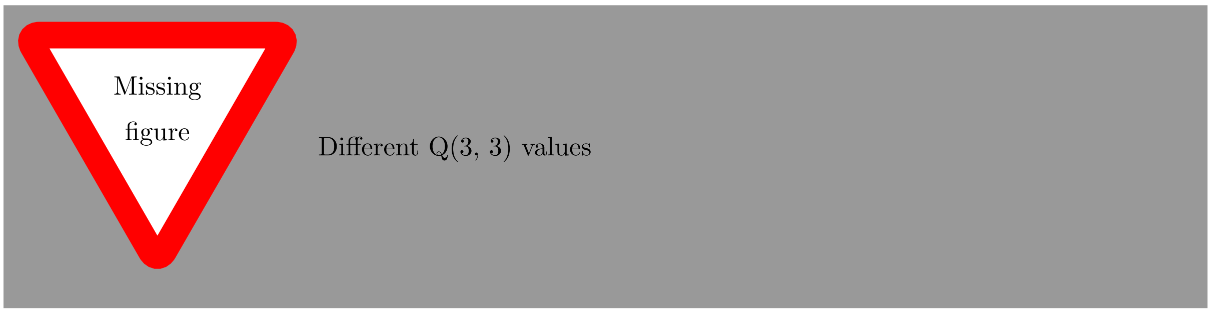
add ref to matlab script



The group then proceeded to tuning the elevation. Since the helicopter did not have a good response for the elevation (SEE PLOT...) we decided to increase Q(5, 5) to use more fuel to get a response that followed the optimal elevation...



We also tried changing Q(3, 3) to see if we could get a better pitch response.



In the end we chose this configuration as out best tuning:

$$\boldsymbol{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.2}$$

3.4 Finding a reasonable trajectory

With a good

3.5 Decoupled model

Answer 10.4.2.7

3.6 MATLAB and Simulink

Code and diagrams go here

3.7 Optional exercise

Which constraints did you add? What was the results? Plots? Discussion?

References

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